ANDRÉ LUIZ DA SILVA SOLINO

TESTE BASEADO EM DEFEITOS PARA *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 Vergilio

CURITIBA

2008

ANDRÉ LUIZ DA SILVA SOLINO

TESTE BASEADO EM DEFEITOS PARA *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 Vergilio

CURITIBA

2008

AGRADECIMENTOS

A toda a minha família, que sabe do meu esforço para conquistar mais esse objetivo, e principalmente a minha mãe, pelo seu exemplo de vida, determinação e por ser sempre minha grande incentivadora, e sempre ter me ajudado e confiado em minha capacidade.

A professora Dra. Silvia pela orientação nesse trabalho, sempre disponível para ajudar de todas as formas que eram possíveis, fosse através de conversas em sua sala ou no laboratório, ou através do telefone e e-mails e até mesmo através do envio de correções por SEDEX.

A UFPR, pela formação acadêmica que me proporcionou. Em especial aos amigos que fiz durante o curso.

A todos os professores que tive durante a vida, e que possibilitaram que eu chegasse até esse ponto. Professores que não apenas me ensinaram diversas disciplinas, mas que foram verdadeiros mestres em minha vida.

A Milano e a Jalmaratan pela enorme contribuição na fase do experimento. Foram eles que inseriram os defeitos para que o experimento pudesse ser realizado.

A Ricardo Pantaleão, que me incentivou a iniciar o mestrado. À época eu estava trabalhando no MPF em Paranaguá, e acreditava ser inviável fazer o mestrado em Curitiba, na UFPR. Além do incentivo, ainda me orientou para solicitar o horário diferenciado para estudante e autorizou o mesmo. Sem esse horário diferenciado seria impossível frequentar as aulas do mestrado.

A Rogério Gehring, companheiro de trabalho no MPF em Curitiba. Após um ano em Paranguá fui transferido para o MPF em Curitiba e tive o privilégio de trabalhar com o Rogério. Ele, como meu chefe imediato, autorizou que fosse mantido o horário diferenciado, permitindo que eu continuasse a assistir as aulas do mestrado.

As pessoas do kitesurf no Paraná, que se transformaram em grandes amigos e sem eles a minha passagem pelo Paraná teria sido muito difícil. Nessa amizade, que surgiu através do esporte, sempre íamos para a praia, o que de certa forma matava um pouco as

saudades da minha cidade, com exceção dos velejos no frio paranaense. Mesmo nesse frio insuportável, os momentos eram sempre prazerosos.

A todos os amigos, que com certeza contribuíram de algum forma para que eu chegasse até esse ponto.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE CÓDIGOS

GLOSSÁRIO

ADA	*Alternative Data Instances Analysis*
AM	*Análise de Mutantes*
CORBA	*Common Object Request Broker Architecture*
DCOM	*Distributed Component Object Model*
DCP	*Data Communication Pertubation*
DVP	*Data Value Pertubation*
HTTP	*Hipertext Transfer Protocol*
JVM	*Java Virtual Machine*
RMI	*Remote Method Invocation*
RPC	*Remote Procedure Call*
SOA	*Service Oriented Architecture*
SOAP	*Simple Object Access Protocol*
SQL	*Structured Query Language*
UDDI	*Universal Description, Discovery, and Integration*
W3C	*World Wide Web Consortium*
WS	*Web Services*
WSeTT	*Web Service Testing Tool*
WSDL	*Web Service Definition Language*
XML	*Extensible Markup Language*

RESUMO

Atualmente é crescente o número de aplicações *Web* que utilizam a tecnologia de Web Services (WS), os quais são freqüentemente utilizados para resolver problemas de integração. O teste é uma atividade fundamental para assegurar a qualidade do software. Devido a esse fato, existe atualmente uma demanda por ferramentas e técnicas de teste adequadas ao teste de aplicações *Web* e que considerem as características peculiares dos WS. Alguns trabalhos da literatura abordam essa questão, entretanto eles não têm o objetivo de oferecer um critério que permita tanto a geração quanto a avaliação dos dados de teste. Para auxiliar nessas tarefas este trabalho explora o teste baseado em defeitos no contexto de WS, que tem se mostrado em diversos experimentos um dos tipos de teste mais eficaz em termos do número de defeitos revelados. O trabalho propõe duas abordagens de teste que têm como foco o documento WDSL. A primeira introduz um conjunto de consultas específicas para revelar defeitos nos documentos WSDL, e a segunda propõe novos operadores de mutação que consideram a semântica do documento WSDL. Para aplicar as abordagens propostas foi desenvolvida uma ferramenta, chamada WSeTT, para auxiliar na atividade de testes. Ambas abordagens foram implementadas na ferramenta. Na primeira não é necessário se conectar ao *Web Service* em teste, este é realizado através de consultas XQuery ao documento WSDL que define o *Web Service*. Na segunda, é utilizada a emulação de clientes para o *Web Service* em teste. Por fim, são realizados experimentos e, a eficácia e o custo das abordagens são analisados.

ABSTRACT

Web Services (WS) have been intensively used to solve integration problems in the development of Web applications. This fact propiciates a crescent demand for testing techniques and tools that consider the particular characteristics of WS. Testing is a fundamental activity to ensure the quality of software. Some works in the literature address this subject. However, most of them do not offer criteria to be used for generation and evaluation of test sets. To aim at these tasks, this work explores fault-based test in the context of WS. This kind of test has been considered by many authors as the most effective, in terms of the number of revealed faults. This dissertation proposes two approaches, which focus on WSDL documents. The first approach introduces a set of queries specific to reveal fault in such documents. The second one introduces a set of mutation operators that consider the semantics of the WSDL document. A tool, called WSeTT, was implemented to apply the proposed approaches. The first approach does not require connection with to the Web Service under test. To apply the second one emulation of clients is used. Some experimental results are also presented, obtained by using the implemented tool.

CAPÍTULO 1

INTRODUÇÃO

O processo de desenvolvimento de software envolve uma série de atividades. Atividades agregadas sob o nome de Garantia da Qualidade de Software têm sido introduzidas ao longo de todo o processo de desenvolvimento, entre estas, atividades de V & V (Verificação e Validação), com o objetivo de minimizar a ocorrência de erros e riscos associados. Dentre as atividades de V & V, o teste é considerado uma das mais importantes [DMJ07]. Isso porque ele é fundamental para assegurar a qualidade do produto. Geralmente a falta de equipes preparadas e de ferramentas adequadas contribui para um aumento significativo do custo e esforço gastos na atividade de teste.

Existem diversas técnicas para realizar o teste de um software, todas com o mesmo objetivo: revelar um defeito ainda não descoberto. Essas técnicas geralmente estão classificadas em três grupos: técnica estrutural, funcional e baseada em defeitos. A técnica estrutural utiliza informações da implementação do software para realizar o teste. A técnica funcional, também conhecida como caixa preta, testa os requisitos funcionais do software sem levar em consideração detalhes de implementação do mesmo. Já a técnica de teste baseada em defeitos utiliza certos tipos de defeitos conhecidos e comuns para derivar requisitos de teste.

Em geral, sabe-se que é impraticável utilizar o conjunto completo do domínio de entrada para um software, pois esse conjunto pode ser muito grande ou até mesmo infinito. Além disso, é necessário saber quando um software foi suficientemente testado. Para auxiliar nessas questões foram propostos critérios de teste. Tais critérios podem ser classificados em critério de seleção e critério de adequação. Os critérios de seleção auxiliam a escolher os casos de teste que tenham uma maior probabilidade de revelar defeitos. Os critérios de adequação estabelecem predicados que devem ser satisfeitos para que a atividade de teste seja considerada satisfatória ou mesmo encerrada. Existem vários critérios

para cada uma das técnicas citadas anteriormente. Como exemplos de critérios para a técnica funcional podem ser citados os critérios particionamento em classes de equivalência, análise do valor limite e grafo de causa efeito. Critérios baseados na complexidade, em fluxo de controle e em fluxo de dados são exemplos de critérios utilizados na técnica estrutural. Análise de mutantes e semeadura de erros são dois critérios típicos da técnica baseada em defeitos.

O critério análise de mutantes foi proposto por DeMillo et al [DLS78] e está baseado em duas hipóteses principais: (i) A hipótese do programador competente assume que os programadores experientes escrevem códigos corretos ou muito próximos de estarem corretos e (ii) A segunda hipótese diz respeito ao efeito do acoplamento e afirma que defeitos complexos são compostos por defeitos menores e mais simples. Um conceito fundamental desse critério são os operadores de mutação. Cada operador é responsável por inserir uma única modificação no programa em teste. Assim para o programa original P, em teste, são gerados diversos programas mutantes, que diferem de P apenas devido a uma mudança sintática simples. O objetivo é gerar um dado de teste que seja capaz de diferenciar o comportamento de P de seus mutantes. Se a saída do mutante estiver correta, P contém o defeito descrito pelo operador de mutação que gerou o mutante.

Com o advento da Internet, e o crescente número de aplicações *Web* atualmente, existe uma grande demanda por ferramentas e técnicas adequadas ao teste dessas aplicações. Essas aplicações possuem características peculiares que as diferenciam do software tradicional, o que dificulta a atividade de teste [OX04]. Tais aplicações estão cada vez mais complexas e são utilizadas em áreas críticas na grande maioria das empresas. A falha dessas aplicações pode provocar grandes prejuízos. Por isso, a garantia da qualidade e da confiabilidade é crucial, e a demanda por metodologias e ferramentas para realizar o teste de aplicações *Web* é crescente. Alguns trabalhos dedicam-se a esse tema [EVJ07],[Eme07].

Dentre esse trabalhos, destaca-se a abordagem ADA (*Alternative Data Instance Analysis*) proposta por Emer [Eme07]. Essa abordagem tem por objetivo detectar erros em esquemas de dados. No caso de aplicações *Web* que manipulem documentos XML, esses esquemas de dados estão relacionados aos esquemas que definem tais documentos. O

teste é realizado através de consultas a documentos XML válidos em relação ao esquema em teste. A idéia central dessa abordagem é testar o esquema de dados para garantir a integridade dos dados definidos por ele e manipulados por uma aplicação de software.

Geralmente as aplicações *Web* utilizam diferentes tecnologias, que estão em constante evolução e por isso existe a necessidade de que o teste acompanhe essa evolução. Uma dessas tecnologias é a de serviços *Web* (*Web Services* - WS).

WS podem ser vistos como um estágio recente no desenvolvimento de software; são componentes que permitem a integração entre aplicações com baixo nível de acoplamento através do envio e recebimento de mensagens em formato XML, mais especificamente mensagens SOAP. Os WS conjuntamente com protocolos da *Web* tornam as instâncias de componentes facilmente acessíveis dentro e fora de uma empresa e são bastante utilizados para resolver problemas de integração.

Segundo definição do W3C[1], um *Web* Service é um sistema de software projetado para permitir interoperabilidade máquina a máquina em uma rede. Ele possui uma interface descrita em um formato possível de ser processado por máquinas (especificamente o WSDL[2]). Outros sistemas interagem com o *Web Service* da maneira prescrita em sua descrição usando mensagens SOAP[3], tipicamente transportadas utilizando o protocolo HTTP[4] com serialização de XML[5] em conjunto com outros padrões *Web* relacionados.

Para garantir a qualidade e a confiabilidade dos WS, o teste é fundamental e devido as características próprias dos WS, são necessários métodos e técnicas específicos. Alguns trabalhos têm sido desenvolvidos com o objetivo de testar esse tipo de software. Entretanto, o teste de WS ainda é considerado um desafio [Luc05]. Eles são mais amplamente distribuídos e heterogêneos que o software tradicional [OX04]. Além disso, a ausência de interfaces com o usuário torna difícil aplicar procedimentos de teste, devido à perda de controle [Dav02].

Os trabalhos encontrados na literatura procuram testar diferentes aspectos dos WS.

[1] http://www.w3.org/
[2] http://www.w3.org/TR/wsdl
[3] http://www.w3.org/TR/2003/REC-soap12-part1-20030624/
[4] http://www.w3.org/Protocols/
[5] http://www.w3.org/XML/

Bloomberg [Blo04] expõe alguns desses aspectos: teste do mecanismo de requisição e resposta do *Web Service* através de mensagens SOAP; utilização das informações contidas nos documentos WSDL para o planejamento dos casos de teste; teste da publicação, busca e associação de WS, que se trata de uma nova característica sob a perspectiva de teste de software e por isso são necessárias novas técnicas; emulação do cliente de um *Web Service*, assim como emulação de um *Web Service* para um cliente.

Considerando esses aspectos, alguns autores introduziram abordagens de teste específicas para o teste de WS dentre estas: teste baseado em cenários [TPSC02], em contratos [HL06], em perturbação de dados [AV06, FV07, OX04, XOL05]. No teste baseado em perturbação de dados [AV06, FV07, OX04, XOL05], mensagens XML e SOAP são modificadas (perturbadas) e utilizadas como dados de entrada para testar a interação entre dois WS. Siblini e Mansour [SM05] propõem uma abordagem que utiliza Análise de Mutantes, e definem operadores de mutação específicos para documentos WSDL. O propósito dos operadores definidos é encontrar defeitos nos documentos WSDL como também na lógica de programação do próprio *Web Service*. Para realizar o teste, Siblini e Mansour [SM05] aplicam um dos operadores definidos em um documento WSDL e a partir desse documento mutante geram os dados de entrada para o *Web Service*, analisando as mensagens de retorno, que são comparadas com a especificação.

Dentre esses tipos de abordagem, destacam-se as duas últimas que são abordagens baseadas em defeitos. Esse tipo de abordagem tem se mostrado bastante promissor e eficaz em termo do número de defeitos revelados. Os trabalhos citados anteriormente, com exceção do trabalho de Siblini e Mansour [SM05], não levam em consideração aspectos semânticos do documento WSDL que define os WS. O trabalho de Siblini e Mansour [SM05] leva em consideração tais aspectos, porém em poucos de seus operadores propostos. Operadores de mutação que sejam baseados nos aspectos semânticos do documento WSDL podem ser mais específicos e podem revelar diferentes tipos de defeito.

1.1 Motivação

Diante do exposto, são apresentadas as principais motivações para este trabalho.

1. A linguagem XML atingiu uma grande importância para aplicações que utilizam troca de dados, entre essas aplicações estão os WS, tecnologia que vem sendo utilizada cada vez mais nos últimos anos.

2. O critério Análise de Mutantes tem se apresentado como um dos mais eficazes e promissores em revelar defeitos, no entanto os operadores de mutação encontrados na literatura precisam ser refinados, e a grande maioria deles não leva em consideração aspectos semânticos dos WS.

3. Poucos trabalhos usam o teste específico de um documento WSDL, todos presupõem que os WS estão presentes. Realizar o teste considerando o documento WSDL pode ser bastante útil e simples. Pode ser efetuado mesmo que os WS não estejam disponíveis.

4. Existe uma carência de ferramentas que auxiliem na atividade de teste de WS, para que essas técnicas sejam aplicadas, na prática, uma ferramenta é necessária.

1.2 Objetivos

Este trabalho tem como objetivo contribuir para a atividade de teste de WS utilizando a técnica baseada em defeitos. Duas abordagens de teste baseadas em defeitos são exploradas: o critério Análise de Mutantes e a Análise de Instâncias de Dados Alternativas (ADA).

Na primeira abordagem novos operadores de mutação são propostos. Esses operadores consideram a semântica do documento WSDL. Baseado no documento WSDL, mensagens SOAP mutantes sao geradas e clientes do *Web Service* em teste são emulados e seus comportamentos comparados com o original. O comportamento é analisado através das respostas de retorno enviadas através de mensagens SOAP.

Na segunda, a abordagem ADA é adaptada para o contexto de documentos WSDL. As classes de defeitos para documentos XML propostas por Emer et al. são utilizadas e refinadas. A partir dessas classes de defeitos, consultas XQuery são efetuadas no docu-

mento WSDL. Os resultados são então avaliados e defeitos no documento WSDL poderão ser revelados.

As duas abordagens foram implementadas em uma ferramenta chamada WSeTT (**W**eb **Se**rvice **T**esting **T**ool) que foi utilizada para conduzir um experimento de comparação com a abordagem de perturbação de dados.

1.3 Organização do Trabalho

O trabalho está organizado da seguinte forma: o Capítulo 2 contém informações e conceitos necessários para o entendimento básico das tecnologias utilizadas durante o desenvolvimento deste trabalho. No Capítulo 3 é fornecida uma revisão teórica da atividade de teste. São apresentadas as principais técnicas de teste, com destaque para a técnica baseada em defeitos, que é utilizada neste trabalho. No Capítulo 4 são apresentados trabalhos relacionados ao teste de WS. No Capítulo 5 são propostos novos operadores de mutação para documentos WSDL, como também uma proposta de aplicação da abordagem ADA que utiliza consultas XQuery. Nesse capítulo também é descrita a ferramenta WSeTT. Os estudos de caso são apresentados no Capítulo 6. No Capítulo 7 são apresentadas as conclusões e trabalhos futuros. O trabalho contém três apêndices. No Apêndice A são descritos os operadores de mutação. No Apêndice B são descritas as consultas XQuery. No Apêndice C são descritos os defeitos inseridos nos estudos de caso.

CAPÍTULO 2

CONCEITOS BÁSICOS

Neste capítulo são descritas tecnologias que dão suporte ao desenvolvimento deste trabalho, e são apresentados os conceitos subjacentes para o entendimento de WS.

2.1 XML

A XML (*Extensible Markup Language*) [Con06a], assim como a HTML (*HyperText Markup Language*) [Con99], derivam da SGML (*Standard Generalized Markup Language*), que é um padrão internacional (ISO 8879) publicado em 1986. SGML é uma metalinguagem através da qual podem ser definidas linguagens de marcação para documentos. Enquanto a HTML é uma aplicação da SGML, a XML é um subconjunto específico da SGML, projetado para ser mais simples de ser analisado gramaticalmente e de ser processado do que SGML.

A XML é uma linguagem de marcação de dados definida pelo W3C (*World Wide Web Consortium*) [Con04b] que provê uma maneira para descrever dados estruturados. A XML define um formato de texto simples e muito flexível. Originalmente foi projetada para atender aos desafios da publicação eletrônica em larga escala. XML assumiu um importante papel na troca de uma grande variedade de dados na *Web* e em muitos outros lugares [Con06a]. Entre os outros usos da XML podem ser citados: documentação, desenvolvimento *Web* e desenvolvimento de bancos de dados. Os principais objetivos da XML são:

- Prover o intercâmbio de documentos através da *Web* de forma independente de sistemas operacionais ou formatos de arquivos;
- Dar suporte a uma grande gama de aplicações, permitindo a definição de elementos pelo usuário (ou aplicação) para estruturar o documento;

- Facilitar a análise de documentos XML por programas;
- Ser legível tanto por humanos quanto por máquinas;
- Separar conteúdo e a formatação;
- Possibilitar a criação de *tags* sem limitação;
- Ter uma especificação formal para a marcação de documentos;

Um documento XML bem formado consiste de um prólogo e de um elemento raiz. O prólogo contém metadados a respeito do resto do documento e é composto da declaração XML, das instruções de processamento e das definições de DTD ou XML *Schema*. Para ser bem formado o documento XML deve obedecer a algumas regras:

- Só pode haver um único elemento raiz no documento XML;
- Todas as *tags* abertas devem ser fechadas;
- Os elementos devem estar aninhados de forma correta;
- Elementos podem conter atributos, e esses devem estar sempre entre aspas;
- É sensível à caixa (alta/baixa);

Os documentos XML podem estar associados a alguma gramática que define a sua estrutura. Essa gramática geralmente esté definida em DTD - *Document Type Definition* ou em XML *Schema*. Quando um documento obedece à gramática ao qual ele está associado, o documento é dito válido. O Código 2.1 é um exemplo de um documento XML.

```
1 <? xml version="1.0" encoding="UTF-8" ?>
2 <person>
3         <name first="André" last = "Solino" />
4         <profession value="Bacharel em Computação"/>
5         <profession value="Tecnólogo em Informática"/>
6 </person>
```

Código 2.1: Exemplo de documento XML

O documento mostrado no Código 2.1 é bem formado, obedece às regras citadas anteriormente. A Figura 2.1 mostra a estrutura do documento em forma de árvore. Pode ser percebida claramente a presença de um único elemento raiz.

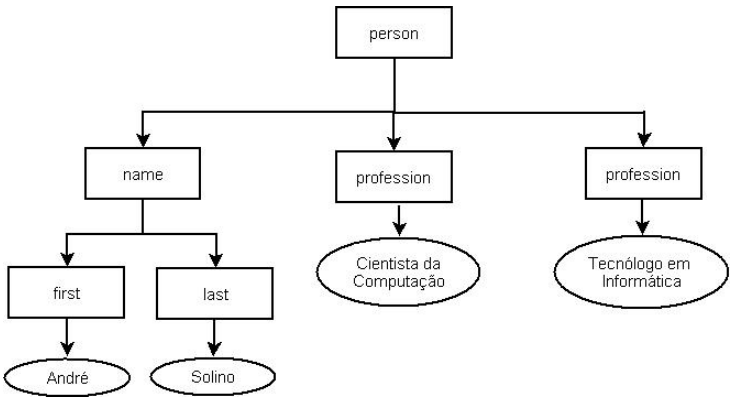

Figura 2.1: Diagrama do exemplo do Código 2.1

2.1.1 DTD - *Document Type Definition*

A DTD - *Document Type Definition* foi o primeiro mecanismo usado para definir a estrutura de um documento XML. Ele é escrito em uma sintaxe formal que explica precisamente quais elementos e entidades podem aparecer em qual lugar do documento, e o que são os elementos e atributos [ERH02].

Os *parsers* de DTD são geralmente classificados em *parsers validating* e *nonvalidating*. Um *parse validating* lê um documento DTD e determina se um dado documento XML está ou não de acordo com ele. Se o documento XML está de acordo com a DTD, o documento é dito válido. Se o documento não estiver de acordo com a DTD, mas estiver correto sintaticamente, ele é um documento bem formado, porém não válido. Por definição, um documento válido é também um documento bem formado. Um *parse nonvalidanting* lê um documento XML mas não é capaz de checar se um documento está de acordo com a DTD.

A DTD é capaz de descrever aspectos básicos dos seguintes itens em um documento XML:

- Aninhamento de elementos;
- Restrições nas ocorrências de elementos;

- Atributos permitidos;
- Tipos de atributos e valores padrão;

O processo de validação de um documento XML nem sempre é obrigatório. Algumas vezes é importante fazer a validação, como por exemplo, em uma aplicação no qual os dados contidos no documento XML serão inseridos em um banco de dados. Caso um elemento que seja obrigatório não esteja no documento, e caso esse elemento também seja obrigatório no banco de dados, no momento da inserção no banco ocorrerá um erro. Em outras aplicações, como na "renderização" de uma página *Web*, a validação do documento XML não é tão crítica, devendo apenas verificar se o mesmo está bem formado.

Um documento XML que esteja de acordo com a DTD do Código 2.2 deve ser formado por um elemento raiz *person*. O elemento *person* deve ter no mínimo uma ocorrência do elemento *name* que possui as propriedades *first* e *last* e zero ou mais ocorrências do elemento *profession* que possui a propriedade *value*. O documento XML do Código 2.1 é um exemplo de documento XML que obedece as definições da DTD apresentada no Código 2.2

```
1 <?xml version="1.0" ?>
2 <!DOCTYPE person [
3         <!ELEMENT person (name+, profession*)>
4         <!ELEMENT name EMPTY>
5         <!ATTLIST name  first CDATA #REQUIRED
6                         last CDATA #REQUIRED>
7         <!ELEMENT profession EMPTY>
8         <!ATTLIST profession value CDATA #REQUIRED>
9 ]>
```

Código 2.2: Exemplo de documento DTD

2.1.2 XML Schema

Na seção anterior foi apresentada a linguagem DTD de definição de esquemas. Esta seção apresenta uma alternativa ao DTD, o XML *Schema*. Enquanto o primeiro descreve o documento XML com regras básicas, o XML *Schema* descreve os elementos e atributos com restrições complexas.

Um documento XML *Schema* é um documento XML que contém uma descrição formal que descreve o que é um documento XML válido. Um arquivo contendo as definições na linguagem XML *Schema* é chamado de XSD (XML *Schema Definition*). O termo XSD é muitas vezes utilizado como referência à linguagem XML *Schema*.

Um documento XML descrito por um esquema é chamado documento de instância. Se o documento satisfizer todas as restrições especificadas pelo esquema, ele é considerado válido para o esquema (*schema-valid*) [ERH02].

A linguagem XML está cada vez mais presente em aplicações que necessitam de um maior controle sobre os dados (como por exemplo, o protocolo SOAP [Con03] utilizado em WS). O padrão XML *Schema* provê maior controle sobre os tipos de elementos e atributos e apresenta as seguintes características:

- Tipos de dados simples e complexo;
- Tipos derivados e herança;
- Restrições nas ocorrências de elementos;
- Suporte a *namespaces*;

Um documento XML que esteja de acordo com o XML *Schema* do Código 2.3 deve ser formado por um elemento raiz *person*. O elemento *person* deve ter no mínimo uma ocorrência do elemento *name*, pois como não foi definido o valor *minOccurs*, a definição assume o valor padrão 1. O elemento *person* também possui zero (*minOccurs* igual a zero) ou várias (*maxOccurs* igual a *unbounded*) ocorrências do elemento *profession*. Por sua vez, o elemento *name* possui dois atributos (*first* e *last*) do tipo *string* e requerido (uso igual a *required*). O elemento *profession* possui o atributo *value* do tipo *string* e de uso requerido. O documento XML do Código 2.1 obedece ao XML *Schema* apresentado no Código 2.3.

2.2 *Web Services*

Uma dificuldade na pesquisa científica é a grande quantidade de definições para o termo *Web Service* [Com00], algumas delas são contraditórias ou imprecisas. Segundo definição

```
<schema
  xmlns='http://www.w3.org/2000/10/XMLSchema'
  targetNamespace='http://www.w3.org/namespace/'
  xmlns:t='http://www.w3.org/namespace/'>

 <element name='person'>
  <complexType>
   <sequence>
    <element ref='t:name' maxOccurs='unbounded'/>
    <element ref='t:profession' minOccurs='0' maxOccurs='unbounded'/>
   </sequence>
  </complexType>
 </element>

 <element name='name'>
  <complexType>
   <attribute name='first' type='string' use='required'/>
   <attribute name='last' type='string' use='required'/>
  </complexType>
 </element>

 <element name='profession'>
  <complexType>
   <attribute name='value' type='string' use='required'/>
  </complexType>
 </element>
</schema>
```

Código 2.3: Exemplo de documento XML Schema

do W3C [Con04a], que é um consórcio destinado a desenvolver tecnologias interoperantes, de domínio público, para a World Wide Web, um *Web Service* é um sistema de software projetado para permitir interoperabilidade máquina a máquina em uma rede. Ele possui uma interface descrita em um formato possível de ser processado por máquinas (especificamente o WSDL). Outros sistemas interagem com o *Web Service* da maneira prescrita em sua descrição usando mensagens SOAP, tipicamente transportadas utilizando o protocolo HTTP com serialização de XML em conjunto com outros padrões *Web* relacionados [Con04a].

Um *Web Service* é uma abstração de uma determinada operação que deve ser implementada por um agente concreto. O agente é a peça do software ou hardware que envia e recebe mensagens e implementa efetivamente a operação. Para ilustrar essa diferença, suponha que seja implementado um agente em uma determinada linguagem de programação para um determinado *Web Service*. Em um momento posterior é implementado

um agente em uma outra linguagem de programação para o mesmo *Web Service* anterior. Embora os agentes tenham mudado, e tenham sido implementados em linguagens de programação diferentes, o *Web Service* continua sendo o mesmo.

O agente concreto, que implementa a funcionalidade do *Web Service*, pode ser visto como uma "caixa preta", que pode ser reutilizada sem a preocupação de como o serviço/funcionalidade foi implementado. De maneira simplificada, um *Web Service* é uma forma de expor funcionalidade para usuários *Web* através de protocolos da Internet padrão.

2.2.1 Arquitetura de um *Web Service*

Os WS utilizam a arquitetura orientada a serviços (SOA - *Service Oriented Architecture*). Porém, WS e SOA são conceitos distintos. Pode-se dizer que os WS são 'instanciações' da arquitetura orientada a serviços. No entanto, podem ser criados componentes SOA utilizando outras tecnologias e que não sejam WS. A Figura 2.2 ilustra a arquitetura orientada a serviço - SOA (*Service Oriented Architecture*).

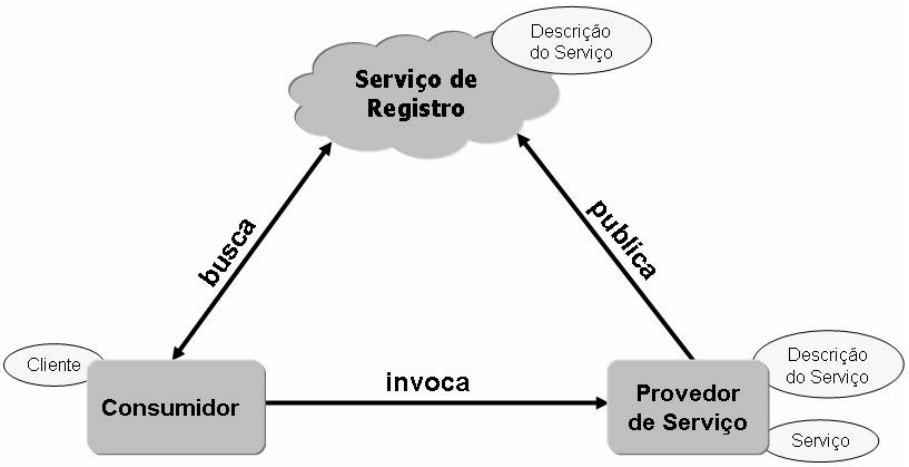

Figura 2.2: Arquitetura Orientada a Serviço - SOA

SOA tem por objetivo criar módulos funcionais chamados de serviços, com baixo acoplamento e permitindo a reutilização de código. Uma arquitetura orientada a serviços é basicamente uma coleção de serviços. Esses serviços se comunicam entre si, e a comunicação pode ser uma simples troca de dados, ou envolver a comunicação com outros serviços, para que juntos e de maneira coordenada, executem alguma tarefa.

2.2.2 SOAP - *Simple Object Access Protocol*

O SOAP [Con03] foi inicialmente criado para possibilitar a invocação remota de métodos por meio da Internet. As outras alternativas na época do surgimento do SOAP eram: DCOM (*Distributed Component Object Model*), CORBA (Common Object Request Broker Architecture) ou RMI (*Remote Method Invocation*). O SOAP utiliza como protocolo o HTTP, que é um protocolo bastante difundido, e a linguagem XML para comunicação de paramêtros. Atualmente o SOAP é um padrão regulado pelo W3C.

O protocolo SOAP provê capacidade de comunicação para que os WS se comuniquem uns com os outros e com os programas clientes, por meio da Internet e de outras redes. Ele é independente de sistema operacional, linguagem de programação ou ambiente de computação distribuída.

A Figura 2.3 ilustra a interoperabilidade de comunicação obtida através do uso do SOAP. Os sistemas conectados podem ser implementados em diversos tipos de software e hardware que tenham suporte de acesso a Internet. No exemplo da Figura 2.3 é mostrada a interconexão de um sistema que implementa seus WS na plataforma .NET, com outro sistema implementado na plataforma J2EE.

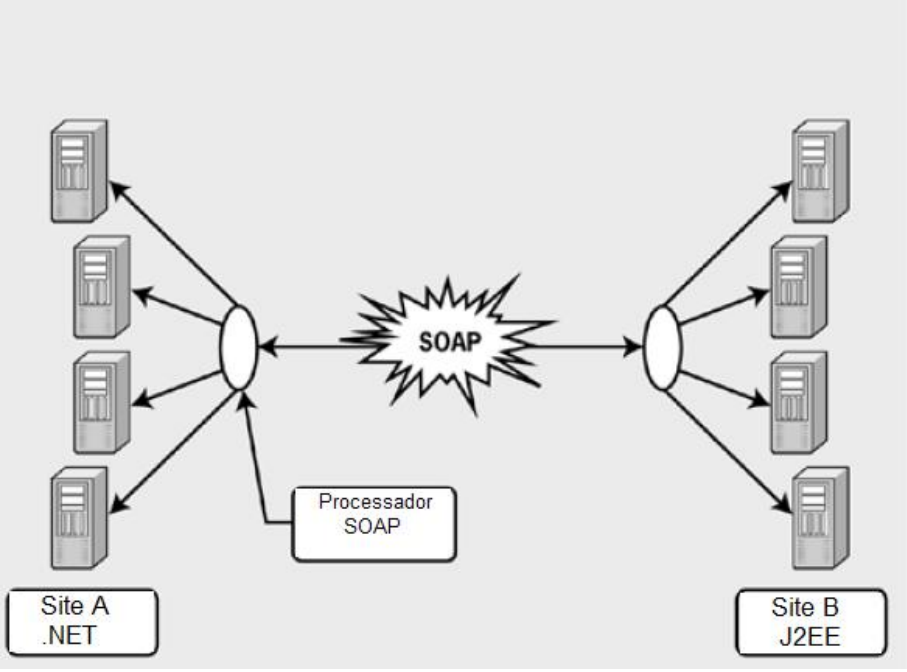

Figura 2.3: Interconexão de sistemas através de SOAP

Existem duas maneiras de um cliente se comunicar com o *Web Service*. Ele pode invocar um método e aguardar uma resposta, forma conhecida como RPC (*Remote Procedure Call*), ou pode enviar um documento para processamento, sem a necessidade de esperar pela resposta do *Web Service* de maneira síncrona, forma conhecida como *Document Centric*.

As principais partes do SOAP são:

- *Envelope*: define o início e o fim da mensagem;
- *Header*: informações de controle da mensagem XML, normalmente de contexto de configuração;
- *Body*: contém o corpo da mensagem, tanto da mensagem que está sendo enviada quanto a que está sendo recebida;
- *Attachment*: consiste de um ou mais documentos anexados a mensagem principal;
- *RPC Interaction*: define qual o modelo de interação a ser utilizado;
- *Encoding*: define como representar dados simples e complexos que serão transmitidos na mensagem;
- *Fault*: quando ocorre algum erro, esse elemento indica qual o erro que aconteceu;

A Figura 2.4 mostra a organização das principais partes que compõem uma mensagem SOAP.

2.2.3 WSDL - *Web Service Description Language*

O WSDL [Con01] é um documento em formato XML que descreve serviços de rede de forma padronizada e independente de plataforma. Através dele os serviços podem ser conectados e utilizados através da rede. Os documentos WSDL são divididos em dois níveis: um abstrato e outro concreto. No nível abstrato, o *Web Service* é definido em termos de mensagens que são enviadas e recebidas. No nível concreto define a implementação, especificando o protocolo de comunicação utilizado no serviço. O W3C [Con04a]

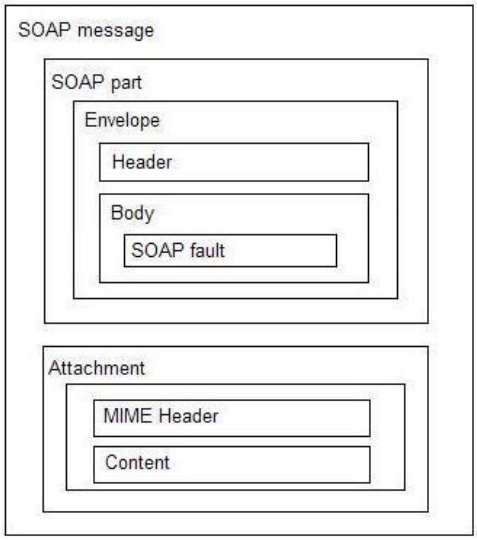

Figura 2.4: Partes que compõem uma mensagem SOAP[a]

[a] adaptado de http://www.builderau.com.au/i/s/soap_message.jpg

define os elementos que compõem o WSDL. Abaixo são citados esses elementos, os quatro primeiros elementos compõem a parte abstrata, enquanto os demais a parte concreta.

- *Types:* definição dos dados utilizados nas mensagens, usando algum sistema de definição de dados, como por exemplo o XML *Schema*;
- *Message:* definição abstrata dos dados que serão trocados entre um *Web Service* e um cliente desse serviço;
- *Operation:* definição de uma operação/serviço disponibilizado pelo *Web Service*;
- *Port Type:* um conjunto abstrato de operações implementadas em uma ou mais portas (*endpoints*);
- *Binding:* uma especificação concreta de protocolo e formato de dados para um *Port Type*. Mapeia as operações e as mensagens definidas em uma porta para um protocolo e um formato de dados;
- *Port:* um único *endpoint* formado pela combinação de um *Binding* e um endereço de rede;
- *Service:* uma coleção de *endpoints* (serviços) relacionados;

A Figura 2.5 mostra a organização dos elementos no documento WSDL.

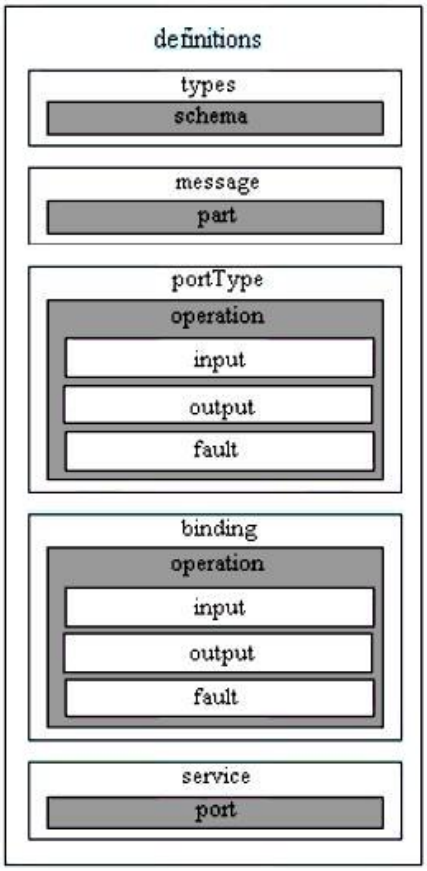

Figura 2.5: Elementos que compõem o WSDL[a]

[a]fonte: http://www.imsglobal.org/gws/gwsv1p0/images/bindFig2p01_WSDLStructure.gif

2.2.4 UDDI - *Universal Description, Discovery and Integration*

A especificação UDDI [Dra04] define um método padrão para publicação, descoberta e integração de componentes de software baseados na comunicação através de uma rede que utiliza a arquitetura orientada a serviços - SOA.

O propósito funcional do UDDI é a representação de dados e metadados sobre um *Web Service*. Esses WS podem ser públicos, ou estar disponíveis apenas internamente na rede de uma empresa. O UDDI define métodos para classificar, catalogar e gerenciar WS, de forma que o mesmo possa ser descoberto e utilizado por outras aplicações.

A especificação UDDI pode ser comparada com um catálogo telefônico composto das seguintes seções:

- Páginas Brancas (*White Pages*): possui informações sobre nomes, endereços, números de telefone, descrição e outras informações referentes aos fornecedores de serviço;

- Página Amarelas (*Yellow Pages*): incluem dados de classificação geral para uma empresa ou serviço oferecido. Essa informação pode conter dados como produtos, códigos geográficos, tipo de uma indústria, entre outros;

- Páginas Verdes (*Green Pages*): contém informações técnicas que descrevem o modo de interação com o serviço;

2.3 XQuery

Com o aumento da quantidade de dados que são armazenados e trocados através de documentos XML, tornou-se necessário um mecanismo para consultar tais dados de maneira eficiente. A linguagem de consulta XQuery foi desenvolvida pelo W3C [Con04b] com o objetivo de realizar consultas de maneira padronizada a qualquer documento XML. Fazendo uma analogia, a linguagem XQuery está para a XML assim como a SQL está para bancos de dados relacionais.

As construções na linguagem XQuery são expressões que são avaliadas para algum valor. Um *script* ou programa XQuery é uma expressão, que pode ter opcionalmente a definição de algumas funções. Dessa forma, a expressão *"2 + 2"* é um programa completo e válido na linguagem XQuery que tem como retorno o valor *"4"* .

Os tipos da XQuery são baseados em XML *Schema*. Existem dois conjuntos de tipos na XQuery: os tipos primitivos (*build-in*), que estão disponíveis em qualquer consulta, e os tipos importados de algum esquema específico [KCD+03].

Uma das principais características da linguagem XQuery são as expressões FLWOR (pronuncia-se *flower*), que são um acrônimo para *"For, Let, Where, Order By, Return"*. Essas expressões permitem ao usuário percorrer sequências de valores, calcular valores intermediários, e opcionalmente filtrar valores. O Código 2.4 mostra um programa simples que utiliza a expressão FLWOR. Na linha 1 é realizado um laço, onde é atribuído para a variável x os valores de 1 até 10. Na linha 2 é feito um filtro e são selecionados apenas

os valores de x onde o resto da divisão inteira por 3 é igual a 0 (zero). É atribuído a variável y o valor da variável x - 2. Por fim, na linha 4 é retornado o valor de y para cada iteração, e serão retornados os valores *"1, 4, 7"* .

```
1 for $x in ( 1 to 10 )
2 where $x mod 3 = 0
3 let y := $x -2
4 return $y
```

Código 2.4: Exemplo de expressão FLWOR

2.4 Considerações Finais

Este capítulo apresentou o conceito de XML e seus objetivos. Foram apresentadas regras que um documento XML deve obedecer, assim como a gramática que ele deve seguir. Essa gramática pode estar definida em um documento XML *Schema* ou DTD.

Também foi apresentado o conceito de WS, incluindo sua arquitetura (SOA), protocolos de comunicação (SOAP), arquivos de definição de serviços (WSDL) e serviço de descoberta de WS (UDDI). WS possibilitam a interoperabilidade entre vários componentes, facilitando a integração de sistemas. O arquivo de definição de serviço, WSDL, é um componente de fundamental importância na arquitetura de WS, pois é através dele que o cliente que deseja se conectar a um *Web Service* pode obter informações do procedimento necessário. Além disso, o WSDL define também o esquema que a mensagem XML a ser enviada pelo cliente deve seguir. Por fim, foi apresentada a linguagem de consulta XQuery que permite realizar consultas a documentos XML, e no contexto deste trabalho será utilizada para construir consultas que serão aplicadas aos documentos WSDL.

CAPÍTULO 3
TESTE DE SOFTWARE

O processo de desenvolvimento de software envolve uma série de atividades. Atividades agregadas sob o nome de Garantia de Qualidade de Software têm sido introduzidas ao longo de todo esse processo, dentre essas estão as atividades de Verificação e Validação (V & V). A verificação refere-se ao conjunto de atividades que garantem que o software implementa corretamente uma função específica. A validação refere-se a um conjunto diferente de atividades que garante que o software que foi construído é rastreável às exigências do cliente [Pre02]. Uma definição clássica, que esclarece melhor o conceito, é dada por Boehm [Boe81], que explica que para identificar a atividade de verificação deve ser feita a pergunta: "Estamos construindo certo o produto?", enquanto para a atividade de validação a pergunta deve ser: "Estamos construindo o produto certo?".

O teste de software é uma atividade de V & V vista como fundamental para garantia da qualidade. Para padronizar a terminologia associada a esta atividade existe um esforço da comunidade científica. Neste trabalho utiliza-se o padrao IEEE 610.12-1990 [IEE90] que define os seguintes termos:

- *Fault* (defeito): passo, processo ou definição de dados incorreto, como por exemplo, uma instrução ou comando incorreto;
- *Failure* (falha): produção de uma saída incorreta em relação à especificação;
- *Error* (erro): diferença entre o valor obtido e o valor esperado, ou seja, a causa para o software produzir um resultado incorreto;
- *Mistake* (engano): ação humana que produz um resultado incorreto, como por exemplo, uma ação incorreta tomada pelo programador;

O software deve ser testado para se obter a confiança que o mesmo trabalhará conforme a sua especificação no ambiente pretendido. O teste de software deve ser o mais eficaz possível na detecção de defeitos, deve apresentar um bom desempenho, os testes

devem ser executados de maneira rápida ao menor custo operacional possível. Em geral é impraticável, quase impossível, encontrar todos os defeitos de um software [Mye04].

Segundo Pressman [Pre02], a atividade de teste é um elemento crítico da garantia de qualidade de software e pode consumir cerca de até 40% do esforço despendido no desenvolvimento de software. Quanto mais tarde um defeito for identificado, mais caro é para corrigí-lo. Os custos para a correção do software aumentam exponencialmente na proporção que o desenvolvimento evolui através das fases do projeto [Boe81].

O teste de produtos de software envolve basicamente quatro etapas [Pre02]:

- Planejamento de testes: define as etapas e categorias de teste a serem realizadas. Inclui requisitos a serem testados, critérios de aceitação, regras, ferramentas de auxílio ao teste e o cronograma para a atividade de teste.

- Projeto de casos de teste: transforma os requisitos do sistema e comportamentos esperados pela aplicação em casos de teste documentados.

- Execução dos casos de teste: executa os casos de teste. Essa etapa gera os resultados para serem analisados pelo testador.

- Avaliação dos resultados dos testes: análise dos resultados produzidos pela execução dos casos de teste. Essa análise pode ser utilizada para avaliações de eficiência do processo, cobertura dos testes, custos entre outras características.

3.1 Objetivos da Atividade de Teste

O principal objetivo da atividade de teste é revelar defeitos.

> "O teste de software pode parecer uma anomalia no processo de desenvolvimento de software. Durante as fases iniciais de definição e desenvolvimento, o engenheiro tenta construir um software a partir de conceitos abstratos em uma implementação concreta. Na fase de teste o engenheiro tenta construir uma série de casos de teste que visam a "destruir" o software que foi construído" [Pre92].

Glenford Myers [Mye04] enuncia três regras que servem como objetivos para o teste de software:

1. A atividade de teste é o processo de executar um programa com a intenção de descobrir um defeito.

2. Um bom caso de teste é aquele que tem uma elevada probabilidade de revelar um defeito ainda não descoberto.

3. Um teste bem sucedido é aquele que revela um defeito ainda não descoberto.

Construir casos de teste que sejam capazes de cobrir diferentes classes de defeitos com o mínimo de tempo e esforço também é um dos objetivos do teste. Caso o teste seja realizado com sucesso, defeitos ainda não revelados serão descobertos. É importante destacar que o teste não pode mostrar a ausência de defeitos em um software, ele pode apenas mostrar que defeitos estão presentes [Pre92].

3.2 Técnicas de Teste

Existem muitas maneiras de se testar um software. Embora as estratégias sejam diferentes, o objetivo de todas é o mesmo: encontrar defeitos em um software. As técnicas de teste estão geralmente classificadas em três grupos: técnica estrutural, técnica funcional e técnica baseada em defeitos.

Sabe-se que é impraticável utilizar o conjunto completo do domínio de entrada para um software, pois esse conjunto pode ser muito grande ou até mesmo infinito. Além disso, é necessário saber quando um software foi suficientemente testado. Para auxiliar nessas questões foram propostos critérios de teste. Tais critérios podem ser classificados em critério de seleção e critérios de adequação. Os critérios de seleção auxiliam a escolher os casos de teste que tenham uma maior probabilidade de revelar defeitos. Os critérios de adequação estabelecem predicados que devem ser satisfeitos para que a atividade de teste seja considerada satisfatória ou mesmo encerrada.

Na técnica estrutural, ou caixa branca, o teste é desenvolvido utilizando informações da implementação do software. O testador analisa a estrutura interna do software, o

código propriamente dito, para derivar os dados de teste. São exemplos de critérios de teste associados a essa técnica: critérios baseados em complexidade [MB89], baseados em fluxo de controle [GG75], baseados em fluxo de dados [EVJ07].

A técnica funcional, também conhecida como caixa preta, testa os requisitos funcionais do software. A técnica leva em consideração a análise das entradas e saídas do software em teste, sem levar em consideração informações sobre a sua estrutura interna, o software é visto como uma caixa preta. São exemplos de critérios dessa técnica [Pre92]: particionamento em classes de equivalência, análise do valor limite, técnicas de grafo de causa-efeito e teste de comparação.

A técnica de teste baseada em defeitos utiliza certos tipos de defeitos conhecidos e comuns para derivar requisitos de teste. O foco dessa abordagem está nos possíveis erros que o programador ou projetista podem cometer durante o desenvolvimento do software. Semeadura de Erros e Análise de Mutantes são dois critérios típicos dessa técnica. Os casos de teste gerados são específicos para demonstrar a presença ou ausência dos defeitos.

O critério Análise de Mutantes foi proposto por DeMillo et al. [DLS78]. O critério baseia-se na hipótese do programador competente, que assume que programadores experientes escrevem códigos corretos ou muito próximos de estarem corretos. Considerando essa hipótese, pode-se afirmar que defeitos são introduzidos nos programas através de pequenos desvios sintáticos, que alteram a semântica do programa conduzindo-o a um comportamento incorreto [EVJ07]. O critério Análise de Mutantes identifica os defeitos sintáticos mais comumente cometidos pelos programadores para identificar erros nos programas.

Uma outra hipótese proposta por DeMillo et al. [DLS78] diz respeito ao efeito de acoplamento. Essa hipótese afirma que defeitos complexos são compostos por defeitos menores e mais simples, então, um conjunto de casos de teste capaz de revelar um defeito simples é capaz de revelar defeitos complexos também.

Para realizar o teste de um programa P o testador deve aplicar operadores de mutação a P. Esse operadores modificam o programa P original dando origem a um programa P' mutante, que difere de P pela inserção de uma única modificação. O programa P original

e os vários programas P' mutantes devem ser executados com o mesmo conjunto de casos de teste T. Os resultados obtidos devem ser analisados e os programas mutantes são classificados em dois grupos:

- Mutantes Mortos: para algum caso de teste t pertencente a T o resultado do programa original P e do programa P' são diferentes;
- Mutantes Vivos: para todos os casos de teste o programa mutante P' apresenta o mesmo resultado do programa P, então, deve ser analisado para verificar se o mutante é equivalente a P, ou se o conjunto de casos de teste T precisa ser melhorado para que seja possível matar P'.
- Mutantes Equivalentes: não existe caso de teste que diferencie o programa P do programa mutante P', para todo e qualquer caso de teste t os programas apresentam sempre o mesmo resultado.

Após a execução dos mutantes, deve ser verificada a adequação dos casos de teste por meio do escore de mutação. O escore de mutação fornece uma medida de quanto o conjunto de casos de testes aproxima-se da adequação. O escore de mutação possui um valor no intervalo de [0..1] e, quanto mais próximo de 1 mais adequado é o conjunto de casos de teste. O escore é obtido pela fórmula:

$$ms(P,T) = \frac{DM(P,T)}{M(P) - EM(P)}$$

Onde:

DM(P,T): número de mutantes mortos.

M(P): número total de mutantes gerados.

EM(P): número de mutantes equivalentes a P.

Uma das maiores desvantagens do critério Análise de Mutantes é o seu alto custo de execução, pois é gerada uma grande quantidade de mutantes, até mesmo para programas pequenos, e todos os programas mutantes devem ser executados com o conjunto de casos de teste, o que demanda grande tempo computacional para execução.

Existem ferramentas para o apoio à aplicação do critério Análise de Mutantes. A ferramenta Proteum [Del93] desenvolvida no ICMP-USP, fornece apoio ao teste de programas escritos na linguagem C. A JAVAMUT [PP03] é uma ferramenta que implementa operadores de mutação para programas estruturados e orientados a objetos escritos na linguagem Java. SQLMutation [TSCdlR06] é uma ferramenta que auxilia no teste de aplicações que utilizam banco de dados, gerando automaticamente mutantes para consultas SQL a banco de dados.

3.3 Considerações Finais

A atividade de teste é uma das mais importantes dentre as técnicas de V & V. Neste capítulo foram apresentados os três principais grupos de técnicas de teste: técnica estrutural, técnica funcional e técnica baseada em defeitos.

A técnica estrutural deriva os casos de teste baseados em informações da estrutura interna do software a ser testado. Já a técnica funcional não considera a estrutura interna, e sim a especificação e análise das entradas e saídas do software. A técnica baseada em defeitos deriva os casos de teste a partir de erros comumente cometidos pelo programador durante o desenvolvimento do software.

Semeadura de Erros e Análise de Mutantes são dois critérios da técnica baseada em defeitos. Uma maior atenção foi dispensada ao critério Análise de Mutantes, por ele ser utilizado como base na proposta deste trabalho. Os critérios de teste baseados em defeitos têm se mostrado em experimentos descritos na literatura como os mais eficazes em termos de defeitos revelados [WMM94]. A análise de mutantes tem sido aplicada em diferentes contextos e serviu como base para o estabelecimento do teste baseado em perturbação, aplicado com sucesso para o teste de WS. Esse tema será abordado no capítulo seguinte.

CAPÍTULO 4
TRABALHOS RELACIONADOS

O teste de WS é importante tanto para o provedor do serviço, quanto para o cliente do mesmo. Ele ainda é considerado um desafio de pesquisa [Luc05]. Os WS são mais amplamente distribuídos que os sistemas de software tradicionais [OX04], característica que adiciona complexidade ao teste.

Várias pesquisas ([XOL05], [OX04], [AV06], [HL06], [SM05]) têm sido realizadas com o intuito de aperfeiçoar o teste de WS, garantindo-lhe uma maior confiabilidade. O presente capítulo apresenta um resumo dessas pesquisas. Serão apresentados trabalhos como o de Siblini e Mansour [SM05] que propõe técnicas de teste que são aplicadas diretamente aos WS, e trabalhos como o de Emer [Eme07], no qual é proposta uma abordagem chamada Análise de Instância de Dados Alternaticas (ADA - *Alternative Data Intances Analysis*), aplicada a uma tecnologia específica (XML *Schema*) dentre as tecnologias que integram um WS. Dessa forma, o teste de XML *Schema* acaba por contribuir para o teste de WS como um todo.

4.1 Análise de Instância da Dados Alternativas (ADA - *Alternative Data Intances Analysis*)

Como citado anteriormente, XML *Schema* é uma, dentre um conjunto de tecnologias utilizadas para a implementação de WS. No trabalho de Emer é proposta uma abordagem de teste denominada ADA (Análise de Instância de Dados Alternativas) para o teste de esquemas [Eme07]. Tal técnica é baseda em defeitos e identifica as classes de defeitos que podem estar presentes nos esquemas em teste. Os defeitos são revelados através de consultas aos esquemas associados às classes de defeitos propostas [Eme07]. Essa abordagem foi aplicada com sucesso em dois tipos de esquema: esquemas para bases de dados relacionais [EVJ08] e em esquemas escritos em XML *Schema* [EVJ05]. Nesse trabalho é de especial importância o segundo tipo de esquema e a abordagem será descrita considerando o contexto de XML *Schemas*.

A abordagem de Emer et al. [EVJ05] considera que os documentos e esquemas que serão testados são válidos. Os defeitos detectados em um esquema estão relacionados a enganos cometidos durante o desenvolvimento do esquema ou durante sua evolução. No contexto de XML *Schema* são propostas três classes de defeitos: elementos, atributos e restrições nos dados. Cada uma dessas classes é subdivida de acordo com os tipos de defeitos. A seguir são apresentadas cada uma dessas classes propostas, e os defeitos por elas descritos [EVJ05].

1. Defeitos em Elementos: defeitos relacionados com a definição do elemento no esquema.

   (a) *Ordem Incorreta*: testa a ordem em que os elementos filhos aparecem em um elemento complexo;

   (b) *Valor Incorreto*: verifica o valor *default* ou *fixed* para um elemento;

   (c) *Ocorrência Incorreta*: consulta o número mínimo e máximo de vezes que um elemento pode ocorrer;

2. Defeitos em Atributos: defeitos relacionados à definição de atributos de um elemento no esquema.

   (a) *Uso Incorreto*: verifica se o atributo é do tipo opcional ou requerido (*required*);

   (b) *Valor Incorreto*: testa o valor *default* ou *fixed* de um atributo;

3. Defeiros nas Restrições nos Dados: defeitos associados aos dados que um elemento pode assumir.

   (a) *Tipo de Dado Incorreto*: testa os tipos de dados mais comuns em documentos XML *Schema*: *string, decimal, integer, boolean, date and time;*

   (b) *Valor Incorreto de Tipo Enumerado*: verifica a ocorrência de valores aceitáveis para um elemento;

   (c) *Valores Máximo e Mínimo Incorretos*: testa os limites máximo e mínimo de valores numéricos.

(d) *Padrão Incorreto*: consulta a existência de uma sequência incorreta de caracteres para um elemento;

(e) *Caracteres de Espaço em Branco Incorretos*: testa como o processador do arquivo XML deve manipular os espaços em branco (preservar, remover ou reduzir);

(f) *Tamanho Incorreto*: testa o tamanho de um elemento em relação ao número de caracteres;

(g) *Digitos Incorretos*: verifica a quantidade total de dígitos que um número pode ter, ou a quantidade total de dígitos decimais;

Para cada tipo de defeito em cada classe de defeito é realizada uma consulta que seja capaz de revelar o defeito descrito pela classe. O objetivo é detectar defeitos, os quais podem estar relacionados à ausência de restrições ou a uma definição incorreta de restrições para um elemento [EVJ05].

Para realizar o teste de um dado esquema XML *Schema* uma instância de dados (documento XML) é fornecida e modificada gerando instâncias (documentos) alternativas. Essas instâncias são geradas com base em padrões especificados nas classes de defeito previamente identificadas, ou seja, representam possíveis defeitos no esquema.

Consultas às instâncias são geradas e executadas. Seus resultados são avaliados de acordo com a instância original e com o com o resultado esperado. Assim, possíveis defeitos no esquema podem ser revelados.

Nazar [Naz07] apresenta uma ferramenta que provê suporte automático a abordagem ADA. A ferramenta XTool (*XML and Relational Database Schema Testing Tool*), gera dados de teste baseados em dados pertencentes à aplicação e seu esquema.

Para realizar as consultas a documentos XML *Schema* é utilizada a implementação Qexo [Qex06] da linguagem de consulta XQuery [Con06b]. A seguir será apresentado um exemplo para ilustrar a abordagem.

Seja o documento XML esquema para uma loja de CD's apresentado no Código 4.1. A ferramenta XTool irá gerar várias instâncias de documentos XML baseados nas classes

de defeitos previamente descritas. A ferramenta pode utilizar uma determinada classe de defeito, como por exemplo "Tipo de Dado Incorreto", e gerar um documento XML que possua CD's cujo o atributo ano é descrito de acordo com o trecho mostrado no Código 4.2. É realizada uma consulta XQuery, apresentada no Código 4.3, ao documento XML gerado. A consulta gera como retorno o dado contido no Código 4.4, que possui o valor " -1234" para o valor de ano do CD, o que é um valor numérico inválido para a representação de um ano. Dessa forma, o testador pode concluir que o esquema possui um defeito na definição do elemento "ano" descrito pela classe de defeito "Tipo de Dado Incorreto", pois um documento XML com um ano inválido foi considerado válido pelo esquema.

```
<schema>
  <element name="cds">
    <complexType>
      <sequence>
        <element name="cd" maxOccurs="unbounded">
          <complexType> <sequence>
            <element name="titulo" type="string"
               minOccurs="1" maxOccurs ="1" />
            <element name="artista" type="string"
               minOccurs="1" maxOccurs="unbounded" />
            <element name="musica" type="string"
              minOccurs="1" maxOccurs="unbounded"/>
            <element name="ano" type="string"
              minOccurs="1" maxOccurs="unbounded"/>
            <element name="gravadora" type="string"
            ...
            <attribute name="duracao" type="string"
              use="optional"/> </sequence>
          </complexType>
        </element>
      </sequence>
    </complexType>
  </element>
</schema>
```

Código 4.1: Trecho de documento XML *Schema* para loja de CD's

```
<cds>
  ...
  <ano>-1234</ano>
  ...
</cds>

```

Código 4.2: Trecho de documento de instância para o XML Schema do Código 4.1

```
1  let $doc := document("cd.xml")
2  for $i in $doc\\cds\cd\ano
3  return <result>{$i}</result>
```

Código 4.3: Consulta XQuery

```
1  <result>
2    <ano>-1234</ano>
3  <result>
```

Código 4.4: Retorno da consulta XQuery aplicado ao Código 4.2

4.2 Teste de WS

WS podem ser descritos como componentes de software que disponibilizam suas funcionalidades através de interfaces SOAP, e, por meio delas trocam mensagens XML [XOL05]. Essa definição, apesar de correta, oculta a complexidade existente em um *Web Service*. WS possibilitam a interoperabilidade entre vários componentes de software, essa característica inerente acrescenta grande complexidade aos componentes e consequentemente ao teste dos mesmos.

WS são mais distribuídos e heterogêneos, envolvem múltiplos padrões e protocolos, admitem mudanças no fluxo de dados entre os componentes em tempo de execução, e além disso, a ausência de uma interface com o usuário dificulta a aplicação do procedimento de teste.

Bloomberg [Blo04] expõe alguns aspectos que envolvem o teste de WS, entre eles: teste do mecanismo de requisição e resposta do *Web Service* através de mensagens SOAP; utilização das informações contidas nos documentos WSDL para o planejamento dos casos de teste; teste da publicação, busca e associação de WS, que se trata de uma nova característica sob a perspectiva de teste de software e por isso são necessárias novas técnicas; emulação do cliente de um *Web Service*, assim como emulação de um *Web Service* para um cliente.

Considerando esses aspectos, diversas abordagens têm sido propostas para o teste de WS. O teste de WS através da técnica de Projeto por Contrato (*Design by Contract*) é proposto por Heckel e Lohmann [HL06]. O objetivo é adicionar informações compor-

tamentais à especificação do WS. São apresentadas regras de transformação de grafos para a visualização dos contratos no nível de modelos gráficos. São definidos dois tipos de contratos. Os contratos fornecidos (*provided contracts*) descrevem o comportamento oferecido pelo WS, enquanto os contratos requeridos (*required contracts*) determinam o comportamento necessário pelo WS [HL06].

Testar o comportamento de WS é também o objetivo de Bultan et al [BFHS03] com a utilização de máquinas de estado. Alguns trabalhos têm o objetivo de gerar testes a partir de WSDL. Em Tsai et al [TPSC02] um framework chamado Coyote converte especificações WSDL em cenários de teste. Siblini e Mansour [SM05] aplicam o teste de mutação em documentos WSDL. Os trabalhos de Xu, Offutt e Luo [XOL05], Offutt e Xu [OX04], Almeida e Vergilio [AV06], e, Cruz Filho e Vergilio [FV07] exploram a abordagem de perturbação de dados para o teste de WS. Essa abordagem tem o objetivo de alterar mensagens XML (SOAP) através de operadores de perturbação e tem apresentado bons resultados em termos de número de defeitos revelados. As mensagens modificadas são utilizadas como dados de teste para a interação entre pares de WS. Os dois últimos grupos de trabalho são os mais relacionados com a proposta e são descritos em detalhe a seguir.

4.2.1 Teste Baseado em Perturbação

Offutt e Xu[OX04] propõem uma técnica de perturbação de dados que modifica os valores das mensagens de requisição e analisa os valores das mensagens de resposta. Dois métodos de perturbação de dados são apresentados: perturbação no valor dos dados, que modifica os valores nas mensagens SOAP em relação aos tipos de dados e perturbação na interação, que modifica mensagens RPC e comunicações de dados [OX04].

A perturbação no valor dos dados (DVP - *Data Value Pertubation*) [OX04] modifica o valor dos dados em mensagens SOAP de acordo com regras definidas nos tipos de valores e tendo por base as idéias definidas na Análise de Valor Limite. Os casos de teste são derivados a partir de valores limites dos documentos XML *Schema*. Os testes são criados pela substituição de cada valor limite, um após o outro.

A perturbação na comunicação RPC [OX04] tem o foco no teste de uso dos dados, e

está subdividida em dois tipos: uso de dados normais, que são utilizados em programas, e uso de dados SQL, que ocorrem quando *strings* são utilizadas como entrada em um banco de dados e a perturbação tenta gerar um SQL *Injection*. A seguir são apresentados os operadores propostos, onde os elementos n_1, ..., n_i pertencem a um dado conjunto de instâncias.

1. Uso de dados normais

   - *Divide* modifica o valor n para $1/n$;
   - *Multiply* altera o valor de n para $n * n$;
   - *Negative* troca o valor de n para $-n$;
   - *Absolute* modifica o valor de n para $| n |$;

2. Uso de dados SQL

   - *Exchange* substitui o valor de n_1 pelo valor de n_2 e vice versa quando ambos são do mesmo tipo;
   - *Unauthorized* modifica uma *string str* para *str' OR '1' ='1*;

O propósito da comunicação de dados é transferir dados, então, geralmente este inclui mais dados que as mensagens RPC. É comum, por exemplo, incluir relacionamentos e restrições de um banco de dados. A perturbação na comunicação de dados (DCP - *Data Communication Pertubation*) [OX04] tem o propósito de testar os relacionamentos e as restrições nos dados utilizados na comunicação.

Almeida e Vergilio [AV06] desenvolveram um conjunto complementar de operadores de mutação baseados em perturbação de dados e nos trabalhos mencionados acima. Os operadores definidos por Almeida e Vergilio perturbam as mensagens SOAP, que é vista como uma árvore, e são descritos a seguir, onde n é um nó dessa árvore fornecida.

1. *Null*: configura para null o valor determinado para o nó n na mensagem SOAP. Se o nó n não possuir nenhum valor, o operador não faz nenhuma ação;

2. *Incomplete*: apaga o nó n e os nós filhos de n da mensagem SOAP;

3. *Inversion*: inverte a ordem dos nós dentro do nó n em uma mensagem SOAP;

4. *ValueInversion*: inverte a ordem dos valores associados aos nós filhos de n em uma dada mensagem XML;

5. *Mod_Len*: modifica o tamanho do valor associado ao nó n em uma dada mensagem XML;

6. *Space*: configura como ' ' (espaço em branco) o valor associado ao nó n;

Para auxiliar na aplicação e avaliação dos operadores de perturbação apresentados anteriormente, Almeida desenvolveu uma ferramenta de teste denominada SMAT-WS [Alm06] que implementa tais operadores. É objetivo da ferramenta reduzir a necessidade de entendimento da tecnologia de WS por parte dos testadores, que não precisam ter conhecimentos avançados sobre WS para prosseguirem nas suas atividades.

O trabalho proposto por Xu et al. [XOL05] deriva mensagens perturbadas utilizando esquemas. O trabalho define alguns operadores de perturbação de dados para o esquema que define a mensagem XML. O objetivo é perturbar os documentos XML *Schema* para que sejam criadas mensagens XML inválidas.

Xu et al. [XOL05] definem um modelo formal para descrever o documento XML *Schema*, que está dividido em três elementos: nós, tipos de dados e arestas. Os operadores de perturbação fazem modificações sistematicamente nesses elementos, porém eles não devem perturbar o nó raiz.

O primeiro operador definido por Xu et al. [XOL05] insere um novo nó na hierarquia que define o documento XML a ser perturbado. Para isso, além do novo nó, é removida uma aresta e são criadas duas novas arestas. O segundo operador de mutação apaga um nó da árvore que define o documento XML, e apaga também a aresta que ligava o nó a ser apagado ao seu nó pai, reorganizando as demais arestas de maneira conveniente. O terceiro operador é semelhante ao primeiro, mas leva em consideração o tipo de dado do nó a ser inserido e o quarto operador é semelhante ao segundo, mas da mesma forma que o anterior leva em consideração o tipo de dado do nó a ser removido.

Além dos quatro operadores primitivos citados anteriormente, Xu et al. [XOL05] definem três outros operadores que podem ser derivados dos operadores primitivos. O primeiro insere uma nova sub-árvore na árvore que define o documento XML, o segundo remove uma sub-árvore da árvore e o terceiro muda a aresta que liga o nó e o seu tipo de dado.

4.2.2 Análise de Documentos WSDL Mutantes

Siblini e Mansour [SM05] propõem uma abordagem que utiliza Análise de Mutantes, e definem operadores de mutação específicos para documentos WSDL. O propósito dos operadores definidos é encontrar defeitos nos documentos WSDL como também na lógica de programação do próprio WS. São definidos três grupos de operadores: grupo de troca, grupo especial e grupo de ocorrência.

Os operadores classificados no grupo de troca, substituem a sequência de um elemento. São definidos seis operadores nesse grupo:

- *SwitchTypesComplexTypeElement*: troca os elementos que sejam do mesmo tipo dentro de um elemento *complexType*;
- *SwitchTypesComplexTypeAttribute*: troca os atributos que sejam do mesmo tipo dentro de um elemento *complexType*;
- *SwitchTypesSimpleTypeElement*: troca os elementos que sejam do mesmo tipo dentro de um elemento *simpleType*;
- *SwitchTypesSimpleTypeAttribute*: troca os atributos que sejam do mesmo tipo dentro de um elemento *simpleType*;
- *SwitchMessagePart*: troca partes de um elemento do mesmo tipo na mensagem;
- *SwitchPortTypeMessage*: troca mensagens do mesmo tipo no elemento *operation* que é definido no *PortType*;

O segundo grupo, grupo especial, inclui os operadores que modificam o valor de um elemento, no entanto, o valor modificado deve ser do mesmo tipo de dado do elemento. É definido um operador nesse grupo:

- *SpecialTypesElementNil*: configura o atributo *nil* para verdadeiro em um elemento do tipo *complexType*;

Os operadores classificados no terceiro grupo, grupo de ocorrência, apagam ou adicionam uma ocorrência de um elemento. São dois os operadores definidos nesse grupo:

- *OccurrenceTypesComplexTypeElement*: adiciona ou apaga a ocorrência de um elemento em um tipo *complexType*;

- *OccurrenceTypesComplexTypeAttribute*: adiciona ou apaga um atributo opcional em um tipo *complexType*;

Para realizar o teste de WS, Siblini e Mansour [SM05] aplicam um dos operadores definidos em um documento WSDL e a partir desse documento mutante geram os dados de entrada para o WS, analisando as mensagens de retorno. As mensagens de retorno são comparadas com a especificação, ou seja, com a mensagem que é esperada como retorno. Com essa análise pode-se verificar quais mutantes estão mortos e quais são mutantes equivalentes.

Em seu artigo, Siblini e Mansour [SM05] demonstram o uso dos operadores através de um exemplo que será descrito a seguir.

O *Web Service* testado é um verificador de cartão de crédito, que contém uma função para checar o cartão de crédito e recebe como entrada o número do cartão de crédito e a data de expiração do mesmo, nesta ordem. Para o número de cartão de crédito *4111111111111111* e data de expiração *01/01/2004*, o *Web Service* retorna que o cartão é valido. Após aplicar o operador *SwitchTypesComplexTypeElement*, que trocará a ordem dos parâmetros de entrada, a entrada para o *Web Service* passa a ter os valores na ordem *01/01/2004 4111111111111111*, ao invés de *4111111111111111 01/01/2004*. Essa entrada quando passada ao *Web Service* em teste retorna uma resposta de cartão válido. No entanto esse mutante revela um defeito, já que para o número de cartão *01/01/2004* e data de expiração *4111111111111111* o *Web Service* deveria retornar que o cartão é inválido. Utilizando um outro operador de mutação, *SpecialTypesElementNil*, que irá

alterar a data de expiração para *null*, é obtida como resposta uma mensagem que indica um erro interno do servidor, revelando outro defeito no *Web Service* em teste.

4.3 Considerações Finais

Esse capítulo apresentou algumas técnicas para o teste de WS. A técnica de Projeto por Contrato proposta por Heckel e Lohmann [HL06] visualiza o WSDL como um contrato que define um *Web Service*, onde os contratos fornecidos descrevem o comportamento oferecido pelo WS e os contratos requeridos determinam o comportamento necessário pelo WS. Bultan et al [BFHS03] buscam o teste através de máquinas de estado.

As técnicas que utilizam a abordagem baseada em pertubação têm sido propostas e aplicadas com sucesso ao teste de WS. Essa abordagem tem a vantagem de não necessitar conhecimento da implementação do *Web Service*. No entanto, ela não utiliza o aspecto semântico do WSDL para derivar os casos de teste, as perturbações representam, em geral, desvios sintáticos. Considerar o aspecto semântico do WSDL pode levar à revelação de outros tipos de defeitos. Além disso o WSDL é o ponto de ligação de um cliente a um determinado *Web Service*. O cliente confia na descrição contida no WSDL, no entanto defeitos podem estar presentes tanto na definição do documento WSDL, quanto na própria implementação do *Web Service*.

Abordagens que se utilizam do documento WSDL tais como a de Siblini e Mansour [SM05] são muito importantes. Entretanto os operadores de mutação propostos por Siblini e Mansour [SM05] podem ser refinados. Tais operadores podem ser refinados, dando maior importância ao aspecto semântico do documento WSDL. Outro aspecto relevante é a necessidade de se conectar ao *Web Service* para realizar o teste. No procedimento descrito por Sibline e Mansour [SM05] são aplicados operadores de mutação aos documentos WSDL. Os clientes enviam mensagens modificadas de acordo com o WSDL gerado após aplicar os operadores de mutação. Como na análise de mutantes, é aplicado apenas um operador por vez ao documento WSDL.

Uma outra maneira de se testar o documento WSDL, mesmo sem a necessidade de se conectar ao *Web Service*, é considerar o documento WSDL como um esquema e aplicar

a abordagem ADA, utilizando apenas consultas, alguns defeitos no documento WSDL podem ser revelados.

Diante do exposto, refinar os operadores de mutação e avaliar diferentes formas de aplicação do teste baseado em defeitos, incluindo a geração de consultas XQuery propostas na abordagem ADA sem a necessidade de se conectar ao *Web Service* em teste, são o assunto do próximo capítulo, que trata da proposta deste trabalho.

CAPÍTULO 5
PROPOSTA DE TESTE BASEADO EM DEFEITOS PARA WS

Técnicas de teste baseadas em pertubação têm obtido sucesso quando aplicadas ao teste de WS. No entanto, essas técnicas não utilizam uma característica peculiar que define os WS: os documentos WSDL. Tomando o WSDL como base para orientar as pertubações realizadas a fim de testar WS, aspectos semânticos podem ser considerados.

O presente capítulo apresenta a proposta deste trabalho. Levando em consideração o embasamento teórico apresentado nos capítulos anteriores, são propostos operadores de mutação que sejam capazes de revelar classes de defeito no contexto de documentos WSDL. Esses operadores estão baseados no trabalho de Siblini e Mansour [SM05], mas os operadores aqui propostos dão uma maior atenção aos aspectos semânticos do documento WSDL, e espera-se que com isso sejam revelados outros tipos de defeitos.

São criados clientes que enviam mensagens ao *Web Service* de acordo com o especificado no documento WSDL modificado pelos operadores. A resposta enviada pelo *Web Service* é analisada pelo testador. A abordagem ADA também é explorada. Consultas são aplicadas aos documentos WSDL e o retorno dessas consultas é analisado e confrontado em relação à especificação do *Web Service* em teste.

A Seção 5.1 apresenta os operadores de mutação propostos por este trabalho. Na seção seguinte são apresentadas as consultas XQuery para os operadores apresentados na seção anterior, assim como consultas relacionadas as classes de defeitos definidas para o contexto de XML *Schema*. A Seção 5.3 mostra o procedimento a ser utilizado para realizar o teste. São apresentadas duas abordagens, a primeira utilizando o *framework* Qexo para realizar as consultas XQuery aos documentos WSDL e a segunda abordagem consiste no teste de WS através da emulação de clientes.

5.1 Operadores de Mutação

Os operadores aqui propostos foram desenvolvidos levando em consideração as especificidades presentes nos documentos WSDL. Uma breve descrição é feita na Tabela 5.1. No

Apêndice A os operadores são descritos mais detalhadamente por um nome, descrição do que o mesmo se propõe a realizar, a base obtida dos trabalhos relacionados e operadores apresentados no Capítulo 4, e um exemplo para melhor ilustrar o funcionamento dos operadores.

Tabela 5.1: Operadores de Mutação propostos

Operador	**Descrição**
mudaAssinatura Serviço	troca a ordem dos parâmetros de um serviço disponibilizado no WSDL. O operador troca somente a ordem dos elementos responsáveis por definir a assinatura de um serviço disponibilizado pelo *Web Service*.
mudaTipoAssinatura Serviço	semelhante ao operador apresentado anteriormente. A diferença é que ao invés de modificar a ordem dos parâmetros na assinatura de um serviço, esse operador modificará apenas o tipo entre dois parâmetros.
mudaTipoElemento Mensagem	modifica o tipo de um elemento definido em uma mensagem por um elemento de outro tipo definido em outra mensagem também presente no mesmo documento WSDL.
mudaInputOutput	modifica o tipo da mensagem de *input* pelo tipo da mensagem de *output* de uma mesma operação definida no documento WSDL.

5.2 Consultas XQuery

Além dos operadores mostrados anteriormente, são realizadas consultas aos documentos WSDL que são responsáveis por definir a interface dos WS. As consultas são realizadas utilizando a linguagem de consulta XQuery [Con06b] e são baseadas nos operadores propostos e nas classes de defeitos propostas por Emer et al. [EVJ05]. Através dessas consultas é verificado se o retorno das mesmas coincide com a especificação do serviço. Com essa abordagem, espera-se revelar os defeitos sem a necessidade de ter que gerar os vários documentos WSDL e os respectivos clientes do *Web Service*, resultando em uma redução no custo operacional da atividade de testes.

As consultas estão sumarizadas na Tabela 5.2, que além do nome e descrição, apresenta em qual operador ou classe de defeito a consulta foi baseada. No Apêndice B as consultas são descritas detalhadamente.

Tabela 5.2: Consultas XQuery

Operador	**Descrição**	**Operador / Classe de defeito**
consultaAssinaturaServiço	consulta os elementos modificados pelo operador de mutação mudaAssinaturaServiço	operador mudaAssinaturaServiço
consultaTipoAssintatura Serviço	retorna os tipos dos parâmetros definidos na assinatura dos serviços	operador mudaTipoAssinaturaServiço
consultaTipoElemento Mensagem	retorna os tipos das mensagens definidos no *Web Service*	operador mudaTipoElementoMensagem
consultaInputOutput	retorna a mensagem de *input* e *output* de uma dada operação	operador mudaInputOutput
Incorrect Order	retorna a ordem definida nos elementos *complexType*	Classe de defeito *Incorrect Order* [EVJ05]
Incorrect Value	busca por valores *default* e *fixed* de um elemento	Classe de defeito *Incorrect Value* [EVJ05]
Incorrect Occurrence	busca o número mínimo e máximo de ocorrências de um elemento	Classe de defeito *Incorrect Occurrence* [EVJ05]
Incorrect Use	consulta pelo tipo de uso de um atributo.	Classe de defeito *IncorrectUse* [EVJ05]
Incorrect Value (atributos)	consulta se um atributo possui um valor *default* ou *fixed*	Classe de defeito *Incorrect Value* [EVJ05]
Incorrect Data Types	consulta pelos tipos de dados mais comuns no XML esquema	Classe de defeito *Incorrect Data Types* [EVJ05]
Incorrect Enumerated Values	consulta os valores aceitáveis para um elemento	Classe de defeito *Incorrect Enumerated Values* [EVJ05]
Incorrect Maximum and Minimum Values	consulta os valores máximo e mínimo de valores numéricos	Classe de defeito *Incorrect Maximum and Minimum Values* [EVJ05]
Incorrect Pattern	consulta o padrão definido para um elemento	Classe de defeito *Incorrect Pattern* [EVJ05]
Incorrect White Spaces Characters	consulta como o processador XML deve processar os espaços em branco	Classe de defeito *Incorrect White Space Characters* [EVJ05]
Incorrect Length	consulta a restrição de tamanho de um elemento	Classe de defeito *Incorrect Length* [EVJ05]
Incorrect Digits	consulta a quantidade total de dígitos que um tipo numerico pode conter	Classe de defeito *Incorrect Digits* [EVJ05]

5.3 Ferramenta WSeTT e Procedimento de Teste

Definidos os operadores e as consultas, serão aplicadas as duas abordagens para realizar o teste. A primeira abordagem realiza consultas aos documentos WSDL com a finalidade de detectar erros na especificação do mesmo. A segunda abordagem consiste em gerar mensagens SOAP modificadas de acordo com os operadores propostos. Essas mensagens serão enviadas ao *Web Service* em teste e as mensagens de retorno serão analisadas. Essa abordagem tem a desvantagem de necessitar se conectar ao *Web Service*.

Com o propósito de auxiliar na aplicação da abordagem proposta, foi desenvolvida a ferramenta chamada WSeTT (*Web Service Auxiliary Testing Tool*) para auxiliar o testador durante o processo de teste. A ferramenta em questão não tem por objetivo ser utilizada em ambiente de produção, mas servir como apoio para aplicar as propostas deste trabalho.

A ferramenta foi desenvolvida utilizando a tecnologia Java SE [SM08b], por esta apresentar uma grande portabilidade, permitindo que a ferramenta seja executada em qualquer sistema operacional que disponha de uma JVM (*Java Virtual Machine*) [SM08b]. A interface gráfica com o usuário foi desenvolvida utilizando o pacote Swing [SM08a] que disponibiliza um conjunto de componentes para construir aplicações gráficas em Java. As mensagens SOAP são construídas com o auxílio do Velocity [Fou07a], que é um template engine, ou seja, ele interpreta e renderiza templates. Após criadas, as mensagens SOAP são enviadas, recebidas e processadas com o auxílio do Axis2 [Fou07b]. Para realizar as consultas XQuery[Con06b] é utilizado o framework Qexo[Qex06], que implementa parcialmente a linguagem de consultas XQuery [Con06b].

5.3.1 Uso do Qexo para realizar consultas XQuery

O testador pode utilizar a abordagem ADA através da ferramenta. O resultado de cada consulta será confrontado com a especificação do serviço, analisando se os mesmos estão em conformidade com a especificação. A Figura 5.1 ilustra de maneira simplificada esse processo.

As consultas a serem realizadas não levam em conta novos *namespaces* definidos no

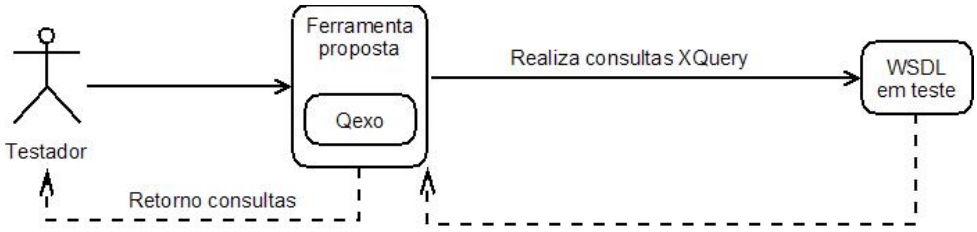

Figura 5.1: Procedimento para realizar as consultas XQuery

documento WSDL. Isso é devido à limitação da implementação Qexo [Qex06] não permitir a importação de esquemas. Foram testadas outras implementações da linguagem XQuery, como por exemplo a Qizx/Open [Sof07] e Saxon [SAX07]. A única implementação que anuncia dar suporte a importação de esquemas é a versão paga da implementação Saxon, denominada Saxon-SA 8.9.

5.3.2 Teste de WS Através de Emulação de Clientes de Serviço

Uma das formas de testar um WS, segundo Bloomberg [Blo04], é através da emulação de um cliente para o WS. Esses clientes de teste são construídos a partir da especificação do documento WSDL após serem aplicados os operadores de mutação propostos. Foram analisadas duas formas para a geração desses clientes de teste.

A primeira forma de teste analisada consiste em gerar os clientes a partir da ferramenta *WSDL2Java* disponível na Distribuição Binária Padrão (Standard Binary Distribution) do Axis2 [Fou07b] versão 1.2. Essa ferramenta cria *Stubs* para o acesso ao WS. Esses *Stubs*, que são classes Java [SDN07b], possuem todas as classes necessárias definidas no documento WSDL. O desenvolvedor acessa o WS de maneira simples com o uso desses *Stubs*. Essa forma é mais simples para o desenvolvedor, que não precisa conhecer a estrutura das mensagens SOAP, pois os *Stubs* encapsulam esses detalhes. A desvantagem é a falta de controle detalhado sobre a mensagem que é enviada e a que é recebida, pois os *Stubs* farão todo o tratamento antes de apresentar para o desenvolvedor, o que pode impossibilitar certos testes, devido à necessidade de verificar a mensagem SOAP sem ter passado por tratamentos.

A segunda forma para realizar o teste possui um menor nível de abstração, manipulando mais diretamente as mensagens SOAP. Para criar as mensagens SOAP é utilizado

o *Java Web Services Developer Pack (Java WSDP) 2.0* [SDN07a]. Através desse pacote o desenvolvedor possui um maior controle sobre as mensagens SOAP que envia e recebe do WS.

A Figura 5.2 mostra de maneira simplificada o procedimento de teste. O testador fornece para a ferramenta proposta o documento WSDL que define o *Web Service* a ser testado. A ferramenta aplica os operadores de mutação propostos gerando as mensagens SOAP modificadas. A ferramenta envia cada uma dessas mensagens SOAP modificadas ao *Web Service* em teste, recebe o retorno do *Web Service* e apresenta esse retorno ao testador para que o mesmo possa analisar.

Figura 5.2: Procedimento para emular clientes do *Web Service*

5.4 Ilustrando o Procedimento de Teste

O procedimento para realizar os testes é ilustrado através de um exemplo que consiste em verificar se um cliente é o dono de determinada conta. Para o exemplo, o serviço irá retornar *true* para os valores apresentados na Tabela 5.3, devendo retornar *false* para todas as outras combinações.

Tabela 5.3: Valores de retorno do WS

idCliente	**idConta**	**Retorno do WS**
1	2	true
1	3	true
2	1	true

O trecho do arquivo WSDL que apresenta a ordem dos parâmetros do serviço *verificaCliente* é mostrado no Código 5.1.

```
1  <message name="verificaClienteMessage">
2      <part name="idCliente" nillable="true" type="xs:int" />
3      <part name="idConta" nillable="true" type="xs:int" />
4  </message>
```

Código 5.1: Trecho arquivo WSDL

Como pode ser observado, o documento WSDL apresenta os parâmetros *idCliente* e *idConta* ordenados. A consulta XQuery *Assinatura Serviço* irá retornar a ordem especificada no WSDL, a Figura 5.3 mostra a tela da ferramenta que apresenta o resultado dessa consulta. Essa consulta apresenta também o que é esperado como mensagem de retorno do WS.

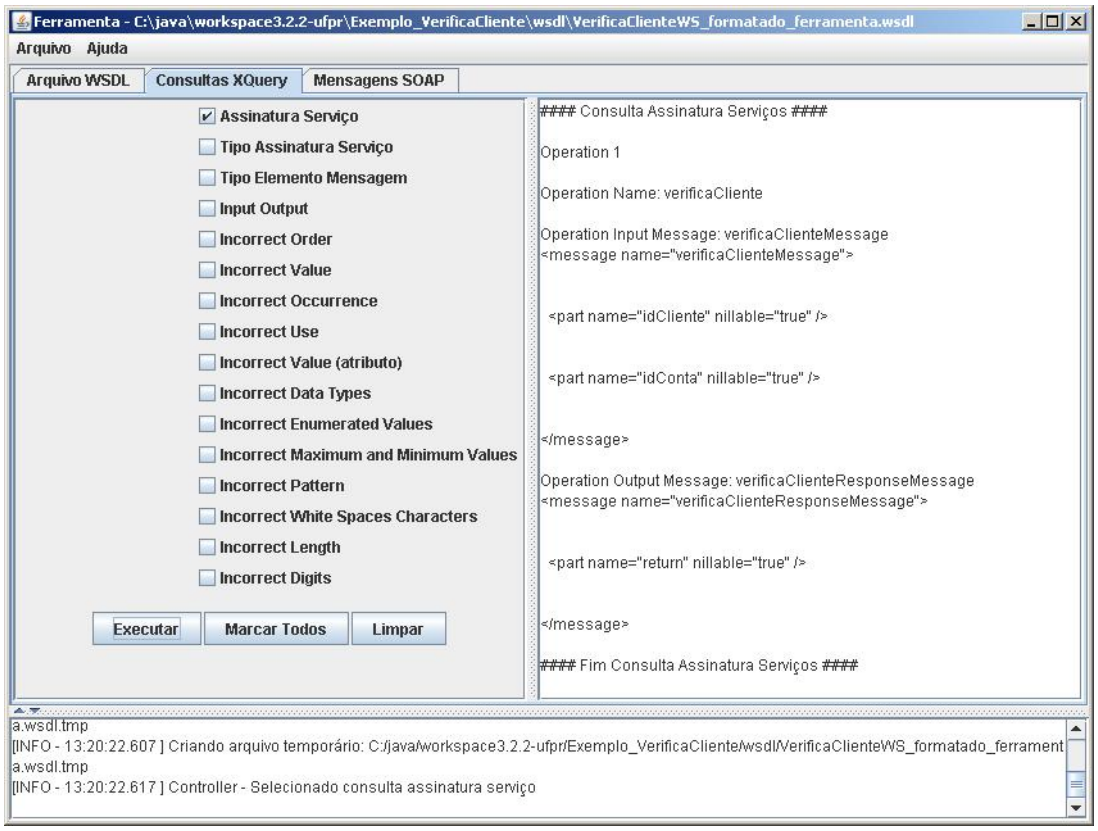

Figura 5.3: Resultado da consulta Assinatura Serviço

Com o resultado da consulta, o testador deve comparar e verificar se o documento WSDL está descrito de acordo com a especificação. Caso não esteja, um defeito no documento WSDL foi revelado. É válido observar que a ferramenta não necessita ter acesso ao WS para realizar as consultas XQuery. Dessa forma, apenas defeitos referentes à definição do documento WSDL em relação à especificação do serviço podem ser revelados.

A segunda abordagem da ferramenta consiste em gerar mensagens SOAP e enviá-las para o *Web Service* em teste. Para o envio das mensagens, estas devem ser previamente configuradas pelo testador. A configuração consiste em informar o serviço a ser invocado no *Web Service*, a URL onde o serviço está localizado, o *target namespace*, além dos parâmetros do serviço a ser invocado e o valor dos mesmos. A maioria dessas configurações são feitas de forma automática pela ferramenta, mas o testador pode alterar qualquer uma delas a seu critério.

Para o mesmo exemplo usado anteriormente, a ferramenta auxiliaria a construir a mensagem a ser enviada para o WS. Essa mensagem é construída de acordo com as especificações do documento WSDL. A Figura 5.4 apresenta a mensagem a ser enviada e o retorno da mesma pelo WS.

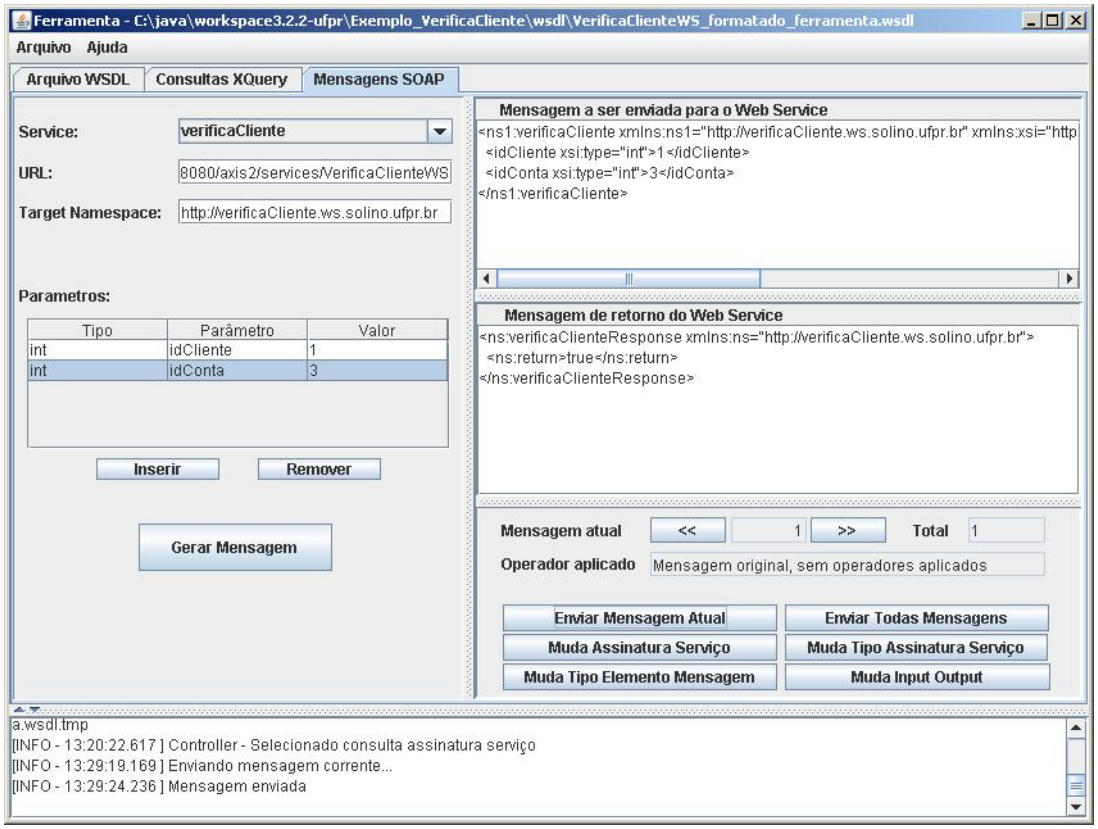

Figura 5.4: Envio de mensagem para o WS e retorno do mesmo

O testador pode enviar essa mensagem original, sem a aplicação de qualquer dos operadores de mutação, para verificar se foi gerada uma mensagem SOAP válida, e se existe o acesso ao WS. Confirmado o acesso ao WS, os operadores podem ser aplicados

na mensagem original e enviados para o WS. O retorno de cada uma das mensagens deve ser analisado pelo testador, que já sabe previamente qual o resultado esperado e pode confrontar com o resultado obtido.

Para ilustrar melhor, suponha que o programador tenha invertido a ordem dos parâmetros no método que implementa o serviço. Ao invés da assinatuda *boolean verificaCliente(int idCliente, int idConta)* teríamos a seguinte assinatura: *boolean verificaCliente(int idConta, int idCliente)*. Dessa forma, quando o testador configurar uma mensagem a ser enviada com os valores *idCliente* igual a 1 e *idConta* igual a 3, que segundo a Tabela 5.3 deveria retornar *true*, irá retornar o valor *false*, como mostra a Figura 5.5.

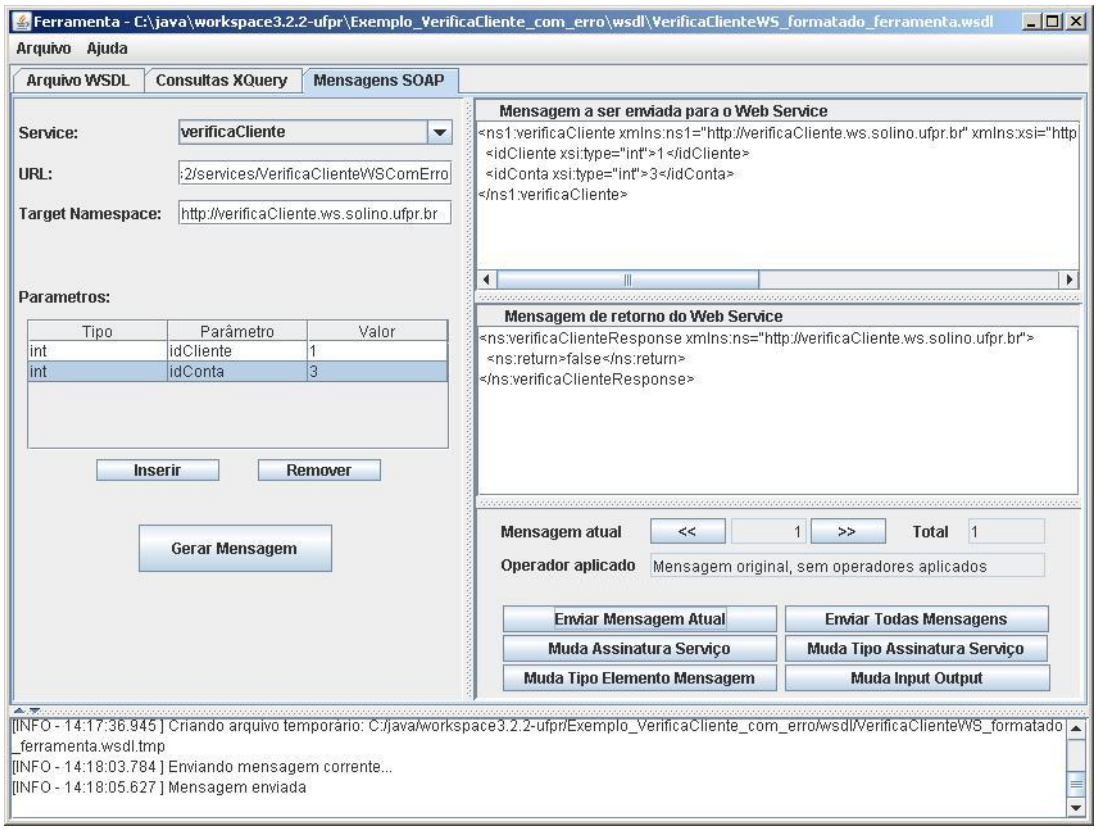

Figura 5.5: Envio de mensagem para o WS que contém um erro na implementação

Ao aplicar o operador *Muda Assinatura Serviço*, a ordem dos parâmetros *idCliente* e *idConta* será invertida. Com isso a inversão que o programador fez erroneamente na implementação seria desfeita, e o resultado, como pode ser visto na mensagem de retorno apresentado na Figura 5.6, é *true*.

Com a consulta XQuery realizada previamente o testador consegue verificar que o

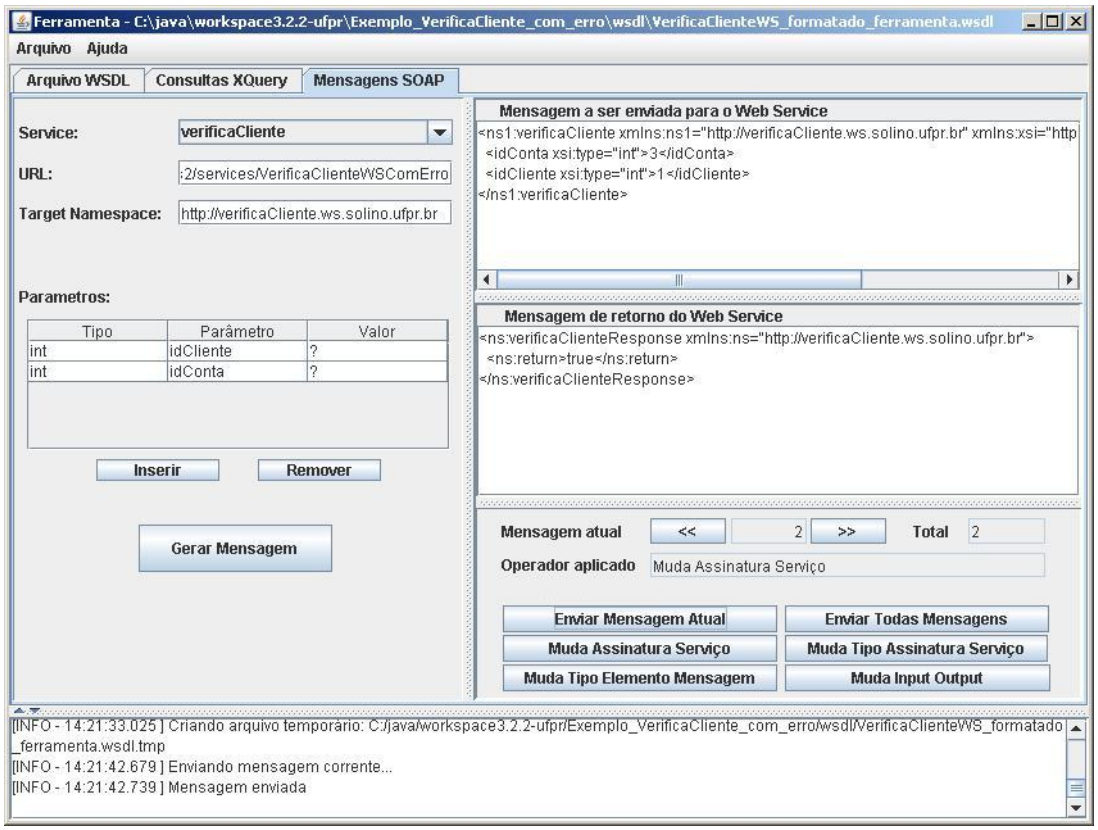

Figura 5.6: Envio de mensagem após aplicar operador Muda Assinatura Serviço para o *Web Service* que contêm um erro na implementação

documento WSDL está de acordo com a especificação. No entanto, o resultado esperado difere do resultado obtido quando é enviada a mensagem original. Quando é aplicado o operador *Muda Assinatura Serviço* o resultado obtido com o valor *true*, leva a crer que ocorreu um erro de implementação, o que de fato ocorreu.

5.5 Considerações Finais

Neste capítulo foram exploradas duas abordagens de teste baseadas em defeitos que têm como foco o documento WSDL. Este documento é de fundamental importância para a tecnologia de WS. É através dele que o cliente obtêm informações a respeito do serviço disponibilizado e os procedimentos necessários para realizar a conexão e executar os serviços. Por isso, dados de teste que consideram esse documento, como é o caso dos dados gerados pela proposta deste trabalho, podem levar à descoberta de outros tipos de defeitos.

Novos operadores de mutação foram propostos. Esses novos operadores adicionam uma carga semântica em relação ao teste de perturbação e aos operadores propostos

na literatura. Essa carga semântica é devida à inclusão do documento WSDL, que é o responsável pela definição do *Web Service*.

Neste capítulo também foi proposta uma adaptação da abordagem ADA para o contexto de WS, introduzindo-se um conjunto de consultas específicas para revelar defeitos nos documentos WSDL.

Ambas abordagens foram implementadas na ferramenta WSeTT e o procedimento de uso foi descrito e ilustrado. O primeiro procedimento consiste em utilizar consultas XQuery ao documento WSDL em busca de defeitos. Tem vantagem de não necessitar se conectar ao *Web Service* para realizar o teste, entretanto erros de implementação do *Web Service* não podem ser detectados. A segunda abordagem necessita se conectar ao *Web Service* em teste. Isso pode ser uma grande desvantagem quando não se tem acesso ilimitado ao mesmo, no entanto, erros de implementação podem ser detectados. As abordagens são utilizadas de forma a se complementarem, buscando uma maior qualidade para o teste. Ambas abordagens implementadas serão avaliadas em um estudo de caso descrito no próximo capítulo.

CAPÍTULO 6
ESTUDO DE CASO

O estudo de caso tem por objetivo verificar a eficácia e aplicabilidade das abordagens propostas. Foram utilizados no experimento quatro *Web Services* que foram submetidos às ferramenta WSeTT e SMAT-WS [AV06].

6.1 Metodologia

Devido à dificuldade em se encontrar um conjunto de WS com o código disponível para realizar os testes, os mesmos foram desenvolvidos, com exceção do *Web Service* de consulta a CEP, que está disponível para acesso livre na URL "http://www.byjg.com.br/ xmlnuke-php/webservice.php/ws/cep".

Os seguintes WS foram utilizados para a realizacão do experimento:

- WS 1 - Verificador de cliente: o usuário deve fornecer o identificador do cliente e o identificador de conta, o *Web Service* verifica então se o cliente é o proprietário da conta.

- WS 2 - Conversor de temperatura: esse *Web Service* possui dois serviços disponibilizados. O primeiro realiza a conversão de temperatura de Celsius para Farenheit, e o segundo faz a operação inversa, converte de Farenheit para Celsius.

- WS 3 - Verificador de CPF: realiza a verificação de um determinado CPF. O usuário fornece um número de CPF em formato *string*, e o WS retorna um valor boleano que indica se o CPF é válido.

- WS 4 - CEP: serviço de consulta a CEP. É um *Web Service* de acesso público desenvolvido pelo site ByJG [ByJ08].

Para cada *Web Service* desenvolvido foram inseridos defeitos. Tais defeitos foram inseridos por outros programadores que colaboraram nessa fase do experimento. Para isso, todos tiveram acesso ao documento WSDL original, e ao código que implementa

o *Web Service*. Os programadores podiam inserir os defeitos no documento WSDL, ou na implementação do *Web Service*. No entanto, eles só deveriam inserir um defeito de cada vez. Para refletir a mudança no WSDL eles poderiam modificar a implementação do *Web Service* também, e, apesar de ter modificado o WSDL e a implementação, seria considerado como uma única modificação.

Os programadores colaboradores não tinham conhecimento das consultas XQuery, nem dos operadores propostos neste trabalho. Após serem aplicadas as modificações, foi gerado um novo conjunto de WS modificados para cada *Web Service* original. Esse conjunto contém WS que possuem o WSDL modificado, a implementação modificada, ou ambos.

A Tabela 6.1 apresenta um resumo dos defeitos inseridos em cada *Web Service*. O Apêndice C contém tabelas com a descrição de cada defeito inserido.

Tabela 6.1: Resumo defeitos inseridos

Web Service	**Defeito no WSDL**	**Defeito no código**	**Código e WSDL**	**Total**
WS1	4	3	3	10
WS2	3	3	2	8
WS3	3	5	3	11
WS4	0	0	0	0

A ferramenta foi utilizada em cada um dos WS, e foi analisado se os defeitos inseridos foram revelados. Para o *Web Service* do CEP o objetivo é verificar se a abordagem proposta consegue revelar algum defeito no *Web Service* em produção. Lembrando que a ferramenta desenvolvida contempla as duas abordagens propostas: consultas XQuery e emulação de clientes do *Web Service*. A ferramenta SMAT-WS [AV06] também foi utilizada no mesmo conjunto de WS para que fosse possível comparar a proposta com o teste baseado em perturbação.

6.2 Resultados

As Tabelas 6.2, 6.3, 6.4, e 6.5 apresentam um comparativo entre o número de consultas geradas e o número de mensagens geradas, relacionadas às abordagens propostas neste trabalho e, o número de mensagens perturbadas, relacionado ao teste de perturbação,

respectivamente, para cada *Web Service* testado.

Tabela 6.2: Comparativo entre abordagens - WS Verifica Cliente - WS 1

Web Service	**Consultas XQuery** (número de consultas geradas)	**Emulação de clientes** (número de mensagens geradas)	**Ferramenta SMAT-WS** (número de mensagens perturbadas)
WS Verifica Cliente 1.1	16	6	16
WS Verifica Cliente 1.2	16	6	15
WS Verifica Cliente 1.3	16	4	7
WS Verifica Cliente 1.4	16	4	6
WS Verifica Cliente 2.1	16	6	16
WS Verifica Cliente 2.2	16	6	16
WS Verifica Cliente 2.3	16	6	16
WS Verifica Cliente 3.1	16	6	16
WS Verifica Cliente 3.2	16	6	26
WS Verifica Cliente 3.3	16	6	16

Tabela 6.3: Comparativo entre abordagens - WS Temperatura - WS 2

Web Service	**Consultas XQuery** (número de consultas geradas)	**Emulação de clientes** (número de mensagens geradas)	**Ferramenta SMAT-WS** (número de mensagens pertubadas)
WS Temperatura 1.1	16	14	7
WS Temperatura 1.2	16	14	7
WS Temperatura 1.3	16	14	7
WS Temperatura 2.1	16	14	7
WS Temperatura 2.2	16	14	7
WS Temperatura 2.3	16	14	7
WS Temperatura 3.1	16	14	7
WS Temperatura 3.2	16	14	7

Tabela 6.4: Comparativo entre abordagens - WS CPF - WS 3

Web Service	**Consultas XQuery** (número de consultas geradas)	**Emulação de clientes** (número de mensagens geradas)	**Ferramenta SMAT-WS** (número de mensagens pertubadas)
WS CPF 1.1	48	96	34
WS CPF 1.2	48	96	34
WS CPF 1.3	48	32	23
WS CPF 2.1	16	32	17
WS CPF 2.2	16	32	17
WS CPF 2.3	16	32	17
WS CPF 2.4	16	32	0
WS CPF 2.5	16	32	17
WS CPF 3.1	16	32	17
WS CPF 3.2	16	32	17
WS CPF 3.3	16	32	17

Tabela 6.5: Comparativo entre abordagens - WS CEP - WS 4

Web Service	**Consultas XQuery** (número de consultas geradas)	**Emulação de clientes** (número de mensagens geradas)	**Ferramenta SMAT-WS** (número de mensagens pertubadas)
WS CEP	16	184	54

As Tabelas 6.6, 6.7, 6.8 e 6.9 mostram se os defeitos inseridos foram detectados em relação a cada uma das abordagens. Para cada defeito o operador e a consulta correspondente são apresentados.

Tabela 6.6: Detecção de defeitos - WS Verifica Cliente

Web Service	**Consultas XQuery**	**Emulação de clientes**	**Ferramenta SMAT-WS**
WS Verifica Cliente 1.1	Assinatura Serviços, Tipo Assinatura Serviços	Muda Tipo Elemento Mensagem	Inversão Valores
WS Verifica Cliente 1.2	Tipo Assinatura Serviços	Muda Tipo Elemento Mensagem	Inversão
WS Verifica Cliente 1.3	Assinatura Serviços, Tipo Assinatura Serviços	Muda Tipo Elemento Mensagem, Muda Input Output	Valor Limite, Tamanho, Incompleto, Nulo
WS Verifica Cliente 1.4	Assinatura Serviços, Tipo Assinatura Serviços, Input Output	Muda Tipo Elemento Mensagem, Muda Input Output	Valor Limite, Tamanho, Incompleto, Nulo
WS Verifica Cliente 2.1		Muda Tipo Elemento Mensagem	Inversão de Valores
WS Verifica Cliente 2.2		Muda Assinatura Serviço, Muda Tipo Assinatura Serviço	Inversão
WS Verifica Cliente 2.3		Muda Assinatura Serviço, Muda Tipo Assinatura Serviço	Inversão
WS Verifica Cliente 3.1	Assinatura Serviço, Tipo Assinatura Serviço	Muda Assinatura Serviço, Muda Tipo Assinatura Serviço	Inversão
WS Verifica Cliente 3.2	Tipo Assinatura Serviço	Muda Assinatura Serviço, Muda Tipo Assinatura Serviço	Inversão
WS Verifica Cliente 3.3	Tipo Assinatura Serviço	Muda Assinatura Serviço, Muda Tipo Assinatura Serviço	Inversão

Tabela 6.7: Detecção de defeitos - WS CPF

Web Service	**Consultas XQuery**	**Emulação de clientes**	**Ferramenta SMAT-WS**
WS CPF 1.1	Tipo Assinatura Serviço	Muda Tipo Elemento Mensagem	Incompleto, Valor Limite, Tamanho
WS CPF 1.2	Assinatura Serviço, Tipo Assinatura Serviço	Muda Tipo Elemento Mensagem	Incompleto, Valor Limite, Tamanho
WS CPF 1.3	Assinatura Serviço, Tipo Assinatura Serviço, Input Output	Muda Tipo Elemento Mensagem,	Incompleto, Valor Limite, Tamanho
WS CPF 2.1		Muda Tipo Elemento Mensagem, Muda Input Output	Valor Limite, Tamanho, Imcompleto, Nulo
WS CPF 2.2		Muda Tipo Elemento Mensagem	Valor Limite, Incompleto
WS CPF 2.3		Muda Tipo Elemento Mensagem, Muda Input Output	Valor Limite, Tamanho
WS CPF 2.4		Muda Tipo Elemento Mensagem, Muda Input Output	
WS CPF 2.5		Muda Tipe Elemento Mensagem, Muda Input Output	Valor Limite, Incompleto
WS CPF 3.1	Tipo Assinatura Serviço	Muda Tipo Elemento Mensagem	Valor Limite, Incompleto
WS CPF 3.2	Assinatura Serviço, Tipo Assinatura Serviço	Muda Tipo Elemento Mensagem	Valor Limite, Incompleto
WS CPF 3.3	Tipo Assinatura Serviço	Muda Tipo Elemento Mensagem, Muda Input Output	Valor Limite, Tamanho, Incompleto, Nulo

Tabela 6.8: Detecção de defeitos - WS Temperatura

Web Service	**Consultas XQuery**	**Emulação de clientes**	**Ferramenta SMAT-WS**
WS Temperatura 1.1	Tipo Assinatura Serviço		Tamanho, Incompleto, Nulo
WS Temperatura 1.2	Assinatura Serviço, Tipo Assinatura Serviço		Tamanho, Incompleto, Nulo
WS Temperatura 1.3	Assinatura Serviço,		Tamanho, Incompleto, Nulo
WS Temperatura 2.1		Muda Tipo Elemento Mensagem, Muda Input Output	Valor Limite, Tamanho, Incompleto, Nulo
WS Temperatura 2.2		Muda Tipo Elemento Mensagem, Muda Input Output	Valor Limite, Tamanho, Incompleto, Nulo
WS Temperatura 2.3		Muda Tipo Elemento Mensagem, Muda Input Output	Valor Limite, Tamanho, Incompleto, Nulo
WS Temperatura 3.1	Tipo Assinatura Serviço	Muda Tipo Elemento Mensagem, Muda Input Output	Tamanho, Incompleto, Nulo
WS Temperatura 3.2	Tipo Assinatura Serviço	Muda Tipo Elemento Mensagem, Muda Input Output	Valor Limite, Tamanho, Incompleto, Nulo

Tabela 6.9: Detecção de defeitos - WS CEP

Web Service	**Consultas XQuery**	**Emulação de clientes**	**Ferramenta SMAT-WS**
WS CEP			Valor Limite

6.2.1 Análise dos Resultados

Os resultados são analisados considerando o custo (em termos do número de casos de teste) e eficácia (em termos dos defeitos revelados). A Tabela 6.10 apresenta a quantidade de mensagens perturbadas pela SMAT-WS e a quantidade de mensagens geradas utilizando os operadores propostos. Percebe-se que o número de mensagens varia de acordo com o *Web Service* testado e que o teste de mutantes requereu um número maior de testes.

Para o teste de mutantes o operador *mudaTipoElementoMensagem* foi o que gerou o maior número de casos de testes, e foi o operador que conseguiu revelar o maior número de defeitos. Na abordagem de perturbação de dados, o operador *Boundary* foi o que gerou o maior número de mensagens perturbadas.

O número de mensagens geradas depende da quantidade de operações disponibilizadas pelo *Web Service*. Percebe-se que para o WS 1 - Verifica Cliente que possui 1 operação (*verificaCliente*) são geradas 56 mensagens perturbadas, para o WS 2 - Conversor Temperatura que possui 2 operações (*Celsius2Farenheit* e *Farenheit2Celsius*) são geradas 112 mensagens perturbadas, o WS 3 - Verifica CPF possui 3 operações (*validaCPF*, *calcDigVerif* e *geraCPF*) e são geradas 480 mensagens perturbadas. O WS 4 - Consulta CEP possui 3 operações que foram testadas, mas apresenta um número menor de mensagens perturbadas em relação ao WS 3, pois no caso específico do WS 4 foi utilizado o *Web Service* disponibilizado na internet e não foram inseridos erros conforme foi mostrado na Tabela 6.1.

O número de consultas XQuery geradas é sempre o mesmo independentemente do documento WSDL. O que faz diferir a quantidade de consultas entre os experimentos é o número de documentos WSDL modificados, criados através da inserção de defeitos. Além disso, quanto maior for o documento WSDL, no sentido de mais serviços e tipos definidos, maior será o retorno das consultas a ser analisado o que pode aumentar o esforço do teste.

As Tabelas 6.11 e 6.12 apresentam a quantidade de defeitos revelados por cada operador. A Tabela 6.13 apresenta apenas as consultas que revelaram algum defeito e o número de defeitos revelados. Percebe-se que dentre os operadores propostos os que mais revelaram defeitos foram os operadores *mudaTipoElemento Mensagem* e *mudaIn-*

Tabela 6.10: Comparativo entre abordagens

Web Service	**Consultas XQuery** (número de consultas realizadas)	**Emulação de clientes** (número de mensagens geradas)	**Ferramenta SMAT-WS** (número de mensagens perturbadas)
WS Verifica Cliente - WS1	160	56	150
WS Conversor Temperatura- WS2	128	112	56
WS Verifica CPF - WS3	272	480	210
WS Consulta CEP - WS4	48	184	54

putOutput. Dentre os operadores de perturbação, destacam-se os operadores *Incomplete*, *Boundary* e *Mod_Len*. Dos defeitos inseridos, a perturbação de dados não revelou apenas um defeito, de uma versão incorreta do *Web Service* do CPF. O teste de mutação não revelou quatro defeitos, sendo três no teste do *Web Service* de temperatura e um relacionado ao *Web Service* do CEP. As consultas foram aplicadas ao *Web Service* do CEP, no entanto, como não se teve acesso à especificação do mesmo, o retorno das consultas não foi analisado, e para os outros WS as consultas conseguiram revelar todos os defeitos inseridos no documento WSDL.

Tabela 6.11: Número defeitos revelados por operador de mutação

Operador proposto	**Número de defeitos revelados**
mudaAssinaturaServiço	5
mudaTipoAssinaturaServiço	5
mudaTipoElementoMensagem	21
mudaInputOutput	12

Tabela 6.12: Número defeitos revelados por operador de perturbação

Operador SMAT-WS	**Número de defeitos revelados**
Boundary	17
Mod_Len	16
Incomplete	19
Null	12
Inversion	5
Value Inversion	3

Tabela 6.13: Número defeitos revelados por consulta

Consulta	**Número de defeitos revelados**
Consulta Assintatura Serviço	9
Consulta Tipo Assinatura Serviço	17
Consulta Input Output	2

No experimento do *Web Service* Verifica Cliente (WS-1) abserva-se um melhor desempenho da abordagem proposta devido à existência de métodos com no mínimo dois parâmetros, possibilitando que a ferramenta realizasse a combinação entre os parâmetros de um mesmo método e conseguisse aplicar todos os operadores propostos. Neste experimento tanto a abordagem proposta, quanto a ferramenta SMAT-WS detectaram os defeitos inseridos em todos os WS gerados.

Com relação ao *Web Service* da Temperatura (WS-2), a abordagem proposta neste trabalho troca o tipo dos elementos por outros tipos também existentes na declaração do WSDL. O WSDL do *Web Service* Temperatura define apenas o tipo *double* para todos os parâmetros e respostas de métodos. Com isso, apesar da ferramenta gerar a troca entre os elementos, a mesma não apresenta nenhuma eficácia, pois os tipos continuam os mesmos, e a ferramenta não revela qualquer erro. A ferramenta SMAT-WS [AV06] por outro lado, não leva em consideração apenas os tipos definidos no próprio documento WSDL. Ela realiza modificações por outros tipos e consegue, dessa forma, revelar mais defeitos. A tentativa de minimizar o número de casos de teste utilizando apenas tipos definidos no WSDL do próprio *Web Service*, neste caso se mostra ineficiente. Porém, caso exista no WSDL algum método que tenha como parâmetro ou como retorno o tipo *string*, a ferramenta também irá revelar os defeitos revelados pela ferramenta SMAT-WS.

Como mencionado anteriormente, no experimento do CPF (WS-3) é gerada uma quantidade maior de mensagens devido à presença de um número maior de métodos (serviços) no mesmo *Web Service*. Alguns dos operadores combinam tipos de um método com tipos de outro método com o objetivo de simular um erro do programador na troca de tipos entre os vários métodos (serviços) definidos. Neste experimento a ferramenta SMAT-WS apresentou um menor custo no que se refere à quantidade de mensagens geradas, no entanto não revelou um erro em um dos WS modificados.

Para o *Web Service* do CEP (WS-4), a abordagem proposta apresentou o pior desempenho, não revelando qualquer erro. A ferramenta SMAT-WS conseguiu revelar apenas um defeito, ao aplicar o operador *Unauthorized* proposto por Jeff Offut e Wuzhi Hu [OX04] que gera um tipo de SQL *Injection* e troca o valor de entrada *str* pelo valor *str' OR '1'='1*. Além disso, o operador *Unauthorized* revela um defeito que não é específico do WSDL.

É válido ressaltar que os tipos de defeitos não revelados pelas abordagens são diferentes. Por exemplo, o defeito no *Web Service* do CPF, não revelado pelo teste de perturbação, está no código e é devido a uma alteração no valor de uma variável. O defeito no *Web Service* do CEP não revelado pelo teste de mutação, está relacionado ao banco de dados, e é revelado por uma perturbação de dados do tipo SQL *Injection*. Dessa forma é verificado que as abordagens podem ser utilizadas de maneira complementar, pois o conjunto de defeitos não revelados por uma abordagem difere do conjunto não revelado pela outra.

6.3 Considerações Finais

Neste capítulo foi apresentado o estudo de caso realizado com o objetivo de verificar a eficácia e a aplicabilidade da abordagem proposta.

Para realizar o experimento foi utilizado um conjunto de quatro WS e a eles foram aplicadas as duas abordagens implementadas pela ferramenta WSeTT. A primeira consiste em aplicar consultas XQuery aos documentos WSDL que definem o *Web Service*. A segunda utiliza a emulação de clientes para se conectar ao *Web Service* e enviar mensagens SOAP modificadas pelos operadores propostos. A ferramenta SMAT-WS [AV06] também foi utilizada no mesmo conjunto de WS.

Foi realizada uma análise comparativa entre as abordagens utilizadas pelas WSeTT e SMAT-WS [AV06], e observou-se que os tipos de defeitos não revelados por uma ferramenta são diferentes dos defeitos não revelados pela outra. Dessa forma as abordagens podem ser utilizadas de maneira complementar.

CAPÍTULO 7
CONCLUSÕES E TRABALHOS FUTUROS

Este trabalho explorou o teste baseado em defeitos de WS. Duas abordagens foram propostas: o teste de mutação e a utilização da abordagem ADA, para o teste de esquemas, no contexto de documentos WSDL.

Um conjunto de operadores de mutação foi proposto para considerar a semântica do documento WSDL que define o *Web Service*. Os operadores estão associados a defeitos específicos que podem ocorrer ao escrever esses documentos. As classes de defeitos descritas por esses operadores foram adicionadas às classes de defeitos da ADA específicas para documentos XML e, para permitir a descoberta de tipos de defeitos de ambas as classes foi definido um conjunto de consultas XQuery.

As consultas possuem a vantagem de não necessitarem uma conexão com o *Web Service* para efetuar o teste, sendo dessa forma um teste mais simples de ser realizado com relação ao aspecto operacional.

Foi desenvolvida uma ferramenta para auxiliar no procedimento de teste. A ferramenta chamada WSeTT, contempla a emulação de clientes de WS (teste de mutação) e as consultas XQuery (abordagem ADA).

Foi realizado um estudo comparativo entre as abordagens implementadas e a abordagem de perturbação de dados implementada pela ferramenta SMAT-WS [AV06]. Tal estudo possibilitou comparar a eficácia e custo de cada uma das abordagens de teste. Além disso, verificou-se através do estudo de caso que a abordagem proposta neste trabalho possui um melhor desempenho para testar serviços que possuam no mínimo dois parâmetros na definição das operações disponibilizadas e que possuam mais de um tipo de dado diferente declarado. Quando o documento descreve apenas um único tipo de dado, os operadores aqui propostos fazem a troca entre esse tipo de dado, no entanto, por ser do mesmo tipo de dado os operadores podem perder eficácia.

O número de mensagens geradas pela abordagem baseada em mutação é proporcional ao número de operações disponibilizadas pelo *Web Service*. O número de consultas geradas

é sempre o mesmo para cada documento WSDL.

Os tipos de defeitos não revelados pelas abordagens são diferentes. Em um dos experimentos o teste de pertubação não revela um defeito. Em outro experimento, é o teste de mutação que não revela um outro defeito. O estudo de caso mostrou que as abordagens podem ser utilizadas de maneira complementar, pois os defeitos não revelados por uma abordagem são diferentes dos defeitos não revelados pela outra abordagem.

Os operadores propostos neste trabalho estão baseados no documento WSDL e as abordagens para aplicá-los realizando o teste de mutação contribuem para realizar o teste de WS e para a descoberta de defeitos relacionados a sua semântica, aumentando a eficácia e confiabilidade de um *Web Service*. A ferramenta permite a automatização facilitando e diminuindo os custos da atividade de teste.

7.1 Trabalhos Futuros

Com relação a trabalhos futuros, as abordagens implementadas na ferramenta WSeTT podem ser utilizadas em conjunto com a abordagem de perturbação de dados para que sejam revelados os erros que tal abordagem detecta e não foram detectados nas abordagens implementadas. Novos operadores podem ser propostos, levando em consideração os resultados obtidos com este trabalho.

Novas extensões podem ser implementadas para a ferramenta:

- Geração de casos de teste de maneira automatizada;
- Implementação de novos operadores de mutação;
- Adicionar inteligência para auxiliar o testador na análise das mensagens de retorno;
- Neste trabalho foi realizado o teste de WS considerando sempre a interação entre dois WS (pares de WS). Uma possível extensão para esse trabalho é considerar o teste baseado em defeitos no contexto de composição de WS. Para isso poderiam ser consideradas linguagens tais como BPEL e PEWS.
- É interessante realizar experimentos com um número maior de WS, e se possível, WS que estejam em produção em algum ambiente.

BIBLIOGRAFIA

[Alm06] Lourival F. Almeida. Explorando Teste Baseado em Perturbação no Contexto de Web Services. Master's thesis, UFPR - Universidade Federal do Paraná, Curitiba - PR, 2006.

[AV06] Lourival F. Almeida and Silvia R. Vergilio. Exploring Pertubation Based Testing for Web Services. In *ICWS '06: Proceedings of the IEEE International Conference on Web Services*, pages 717–726, Washington, DC, USA, 2006. IEEE Computer Society.

[BFHS03] B. Bultan, X. Fu, R. Hull, and J. Su. Conversation specification: A new approach to design and analysis of e-service composition. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, pages 403–410, New York, NY, USA, 2003. ACM.

[Blo04] J. Bloomberg. Report: Testing Web Services, 2004. Disponível em "http://www.parasoft.com/jsp/templates/misc/soap/web_services_excerpt.pdf". Acessado em Abril de 2007.

[Boe81] B. W. Boehm. *Software Engineering Economics*. Prentice-Hall, New York, 1981.

[ByJ08] ByJG. Cepservice webservice, 2008. Disponível em: "http://www.byjg.com.br/xmlnuke-php/webservice.php/ws/cep". Acessado Maio de 2008.

[Com00] IBM Corporation Software Communications. Web Services: Taking e-business to the next level, 2000. Disponível em: "http://www-900.ibm.com/developerWorks/cn/wsdd/download/pdf/e-businessj.pdf". Acessado em Março de 2007.

[Con99] World Wide Web Consortium. HTML 4.01 Specification, 1999. Disponível em: "http://www.w3.org/TR/html401/". Acessado em Maio de 2007.

[Con01] World Wide Web Consortium. Web Services Description Language (WSDL) 1.1, 2001. Disponível em: "http://www.w3.org/TR/wsdl". Acessado em Abril de 2007.

[Con03] World Wide Web Consortium. SOAP Version 1.2 Part 1: Messaging Framework, 2003. Disponível em: "http://www.w3.org/TR/2003/REC-soap12-part1-20030624/". Acessado em Março de 2007.

[Con04a] World Wide Web Consortium. Web Services Architecture. W3C Working Group Note 11, fevereiro, 2004. Disponível em: "http://www.w3.org/TR/2004/NOTE-ws-arch-20040211". Acessado em 10/12/2006.

[Con04b] World Wide Web Consortium. World Wide Web Consortium, 2004. Disponível em: "http://www.w3.org/". Acessado em Dezembro de 2006.

[Con06a] World Wide Web Consortium. W3C Architecture Domain - Extensible Markup Language, 2006. Disponível em: "http://www.w3.org/XML/". Acessado em Março de 2007.

[Con06b] World Wide Web Consortium. XQuery 1.0: An XML Query Language, 2006. Disponível em: "http://www.w3.org/TR/xquery/". Acessado em Dezembro de 2006.

[Dav02] Neil Davidson. The red-gate software technical papers: Web Services Testing, 2002. Disponível em "http://www.red-gate.com/products/ants_load/technical_papers/web_services_testing.htm". Acessado em Abril de 2007.

[Del93] M. E. Delamaro. Proteum: Um ambiente de teste baseado na análise de mutantes. Dissertação de Mestrado. *ICMC-USP, São Carlos-SP*, 1993.

[DLS78] R. A. DeMillo, R. J. Lipton, and F. G. Sayward. Hints on test data selection: Help for the practicing programmer. *IEEE Computer*, 11(4):34–41, 1978.

[DMJ07] Márcio E. Delamaro, José C. Maldonado, and Mario Jino. *Introdução ao Teste de Software*. Ed. Campus, 2007.

[Dra04] UDDI Spec Technical Committee Draft. UDDI Version 3.0.2, 2004. Disponível em: "http://uddi.org/pubs/uddi_v3.htm". Acessado em Abril de 2007.

[Eme07] Maria Claudia F. P. Emer. *Abordagem de Teste Baseada em Defeitos para Esquemas de Dados*. PhD thesis, Faculdade de Engenharia Elétrica e de Computação, Campinas - SP, Setembro 2007.

[ERH02] W. Scott Means Elliotte Rusty Harold. *XML in a Nutshell, 2nd Edition*. O'Reilly, 2002.

[EVJ05] Maria Claudia F. P. Emer, Silvia Regina Vergilio, and Mario Jino. A Testing Approach for XML Schemas. *29th Internacional Computer Software and Applications*, 1:57–62, Julho 2005.

[EVJ07] Maria C.P.F. Emer, Silvia R. Vergilio, and Mario Jino. *Teste de Aplicações Web In: Delamaro, M.E. and Maldonado, J.C. and Jino, M. Introdução ao Teste de Software*, chapter 8. Ed. Campus, 2007.

[EVJ08] Maria Claudia F. P. Emer, Silvia Regina Vergilio, and Mario Jino. Testing Relational Database Schemas with Alternative Instance Analysis. *International Conference on Software Engineering and Knowledge Engineering, SEKE*, Junho 2008.

[Fou07a] The Apache Software Foundation. The apache velocity project, 2007. Disponível em "http://velocity.apache.org/". Acessado em Maio de 2007.

[Fou07b] The Apache Software Foundation. Axis2/Java - Apache Axis2/Java - Next Generation Web Services, 2007. Disponível em "http://ws.apache.org/axis2/". Acessado em Maio de 2007.

[FV07] Paulo N. Cruz Filho and Silvia Regina Vergilio. Teste de Software Baseado em Perturbação de Dados Dirigida por Padrões. Dissertação de Mestrado. PPGInf UFPR. 2007.

[GG75] John B. Goodenough and Susan L. Gerhart. Toward a theory of test data selection. In *Proceedings of the international conference on Reliable software*, pages 493–510, New York, NY, USA, 1975. ACM Press.

[HL06] Reiko Heckel and Marc Lohmann. Towards Contract-based Testing of Web Services. *Electronic Notes in Theorical Computer Science 82 No. 6*, 2006.

[IEE90] Institute, Electrical, and Electronics Engineers. *IEEE 90: IEEE Standard Glossary of Software Engineering Terminology*. 1990.

[KCD+03] Howard Katz, Don Chamberlin, Denise Drape, ary Fernández, Michael Bay, Jonathasn Robie, Michael Rys, Jérôme Siméon, Tim Tivy, and Philip Wadler. *XQuery from the Experts: A Guide to the W3C XML Query Language*. 2003.

[Luc05] Di Lucca. Testing Web-Based Applications: the State of the Art and Future Trends. In *QATWBA'05 - Workshop of the International IEEE Computer Software and Applications Conference, COMPSAC*, page 65. IEEE Press, 2005.

[MB89] Thomas J. Mccabe and Charles W. Butler. Design complexity measurement and testing. *Communications of the ACM*, 32(12):1415–1425, Dezembro 1989.

[Mye04] G. J. Myers. *The Art of Software Testing; Revised and updated by Tom Badgett and Todd Thomas, with Corey Sandler. 2nd ed.* John Wiley & Sons, Inc., 2004.

[Naz07] I. F. Nazar. Uma Ferramenta para Teste de Esquemas para Estruturas de Dados Complexas. Dissertação de Mestrado. PPGInf UFPR. 2007.

[OX04] Jeff Offutt and Wuzhi Xu. Generating Test Cases for Web Services Using Data Perturbation. *ACM SIGSOFT*, 2004.

[PP03] Chevalley P. and Thevenod-Fosse P. A mutation analysis tool for java programs. *International journal on software tools for technology transfer*, 5(1):90–103, 2003.

[Pre92] Roger S. Pressman. *Software Engineering: a practitioner's approach.* McGraw-Hill,Inc, 1992.

[Pre02] R.S. Pressman. *Arquitetura de Software: desenvolvimento orientado para arquitetura.* McGraw-Hill, 2002.

[Qex06] Qexo. Qexo - The GNU Kawa implementation of XQuery, 2006. Disponível em: "http://www.gnu.org/software/qexo/". Acessado em Dezembro de 2006.

[SAX07] SAXON. Saxon The XSLT and XQuery Processor, 2007. Disponível em: "http://saxon.sourceforge.net/". Acessado Maio de 2007.

[SDN07a] Sun Developer Netword SDN. Downloads for Java Web Services Developer Pack 2.0, 2007. Disponível em "http://java.sun.com/webservices/downloads/previous/webservicespack.jsp". Acessado em Maio de 2007.

[SDN07b] Sun Developer Netword SDN. Java Technology, 2007. Disponível em "http://java.sun.com/". Acessado em Maio de 2007.

[SM05] Reda Siblini and Nashat Mansour. Testing Web Services. In *AICCSA '05: Proceedings of the ACS/IEEE 2005 International Conference on Computer Systems and Applications*, pages 135–vii, Washington, DC, USA, 2005. IEEE Computer Society.

[SM08a] Inc. Sun Microsystems. A brief introduction to the swing package, 2008. Disponível em "http://java.sun.com/javase/". Acessado em Maio de 2008.

[SM08b] Inc. Sun Microsystems. Java platform, standard edition, 2008. Disponível em "http://java.sun.com/javase/". Acessado em Maio de 2008.

[Sof07] Axyana Software. Qizx/open, 2007. Disponível em: "http://www.axyana.com/qizxopen/". Acessado Maio de 2007.

[TPSC02] W. T. Tsai, Ray Paul, Weiwei Song, and Zhibin Cao. Coyote: An XML-Based Framework for Web Services Testing. In *HASE '02: Proceedings of the 7th IEEE International Symposium on High Assurance Systems Engineering (HASE'02)*, page 173, Washington, DC, USA, 2002. IEEE Computer Society.

[TSCdlR06] Javier Tuya, Ma Jose Suarez-Cabal, and Claudio de la Riva. SQLMutation: A tool to generate mutants of SQL database queries. *Second Workshop on Mutation Analysis (Mutation 2006 - ISSRE Workshops)*, 0:1, 2006.

[WMM94] W. Eric Wong, Aditya P. Mathur, and José C. Maldonado. Mutation versus all-uses: An empirical evaluation of cost, strength and effectiveness, software quality and productivity: Theory, practice and training. pages 258–265, December 1994.

[XOL05] Wuzhi Xu, Jeff Offutt, and Juan Luo. Testing Web Services by XML Pertubation. In *Proceeding of the 16th IEEE International Symposium on Software Reliability Engineering (ISSRE'05)*. IEEE, 2005.

APÊNDICE A
DESCRIÇÃO DOS OPERADORES DE MUTAÇÃO

1. (a) **Nome:** mudaAssinaturaServiço

   (b) **Descrição:** troca a ordem dos parâmetros de um serviço disponibilizado no WSDL. Esse operador foi baseado no operador STCE (*SwitchTypesComplexTypeElement*) definido por Siblini [SM05]. Tal operador troca os elementos que definem a ordem dos parâmetros de um serviço do *Web Service*. O operador aqui proposto, ao invés de trocar os elementos de forma aleatória, troca somente a ordem dos elementos responsáveis por definir a assinatura de um serviço disponibilizado pelo *Web Service*.

   (c) **Base:** Operador STCE definido por Siblini e Mansour [SM05], operador *Inversion* definido por Almeida e Vergilio [AV06], operador Exchange definido por Offutt e Xu [OX04].

   (d) **Exemplo:** A operação *boolean verificaCliente(int idCliente, int idConta)*, está descrita na seção *portType* do documento WSDL como mostrado no Código A.1.

```
1  <wsdl:portType name="VerificaClienteServicePortType">
2          <wsdl:operation name="verificaCliente">
3                  <wsdl:input xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
4                          wsaw:Action="urn:verificaCliente"
5                          message="ns:verificaClienteMessage" />
6                  <wsdl:output xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
7                          message="ns:verificaClienteResponseMessage"
8                          wsaw:Action="urn:verificaCliente" />
9          </wsdl:operation>
10 </wsdl:portType>
```

Código A.1: VerificaClienteServicePortType

A operação define a mensagem *verificaClienteMessage* como *input* para operação, essa mensagem é reproduzida no Código A.2.

Por sua vez, a mensagem *verificaClienteMessage* possui o tipo complexo *verificaCliente* e é apresentada no Código A.3.

```
1 <wsdl:message name="verificaClienteMessage">
2         <wsdl:part name="part1" element="xsd:verificaCliente" />
3 </wsdl:message>
```

Código A.2: verificaClienteMessage

```
1 <xs:element name="verificaCliente">
2         <xs:complexType>
3             <xs:sequence>
4                    <xs:element name="idCliente" nillable="true" type="xs:int" />
5                    <xs:element name="idConta" nillable="true" type="xs:int" />
6             </xs:sequence>
7         </xs:complexType>
8 </xs:element>
```

Código A.3: verificaCliente

O mesmo documento WSDL que contém o elemento *verificaCliente* define outros tipos complexos, no entanto, o operador proposto será aplicado apenas no tipo complexo apresentado no Código A.3, por ser o tipo complexo responsável por definir a ordem dos parâmetros. Com essa alteração, a operação que tinha como assinatura *boolean verificaCliente(int idCliente, int idConta)* e obedecia à estrutura apresentada no Código A.3, passa a ter a assinatura *boolean verificaCliente(int idConta, int idCliente)* para obedecer o elemento após a aplicação do operador de mutação, Código A.4. As linhas 4 e 5 do Código A.4 foram modificadas pelo operador.

```
1 <xs:element name="verificaCliente">
2         <xs:complexType>
3             <xs:sequence>
4                    <xs:element name="idConta" nillable="true" type="xs:int" />
5                    <xs:element name="idCliente" nillable="true" type="xs:int" />
6             </xs:sequence>
7         </xs:complexType>
8 </xs:element>
```

Código A.4: verificaClienteMutado

2. (a) **Nome:** mudaTipoAssinaturaServiço

   (b) **Descrição:** semelhante ao operador apresentado anteriormente. A diferença é que ao invés de modificar a ordem dos parâmetros na assinatura de um serviço, esse operador modificará apenas o tipo entre dois parâmetros. Seja um método

que tenha como assinatura dois parâmetros, p1 do tipo t1 e p2 do tipo t2, nessa ordem. O operador irá modificar o tipo de p1 pelo tipo de p2 e o tipo de p2 pelo tipo de p1. No entanto, a ordem dos parâmetros na assinatura do serviço permanece inalterada.

(c) **Base:** Operador STCE definido por Siblini e Mansour [SM05], operador *Inversion* definido por Almeida e Vergilio [AV06], operador Exchange definido por Offutt e Xu [OX04].

(d) **Exemplo:** tomando como base o exemplo apresentado no operador *mudaAssinaturaServiço*, suponha que o elemento complexo *verificaCliente*, apresentado no Código A.3 e que possui os elementos *idCliente* do tipo inteiro e *idConta* também do tipo inteiro, passe a possuir o elemento *idConta* como do tipo *float* (apenas para fins de exemplo, suponha também que *idConta* agora representa o valor da conta), conforme apresentado no Código A.5

```
1 <xs:element name="verificaCliente">
2         <xs:complexType>
3             <xs:sequence>
4                    <xs:element name="idCliente" nillable="true" type="xs:int" />
5                    <xs:element name="idConta" nillable="true" type="xs:float" />
6             </xs:sequence>
7         </xs:complexType>
8 </xs:element>
```

Código A.5: verificaCliente alterado o tipo de idConta para float

O tipo complexo apresentado no Código A.5 seria transformado no Código A.6 após ser aplicado o operador mudaTipoAssinaturaServiço. As linhas 4 e 5 do Código A.6 foram modificadas pelo operador de mutação.

```
1 <xs:element name="verificaCliente">
2         <xs:complexType>
3             <xs:sequence>
4                    <xs:element name="idCliente" nillable="true" type="xs:float" />
5                    <xs:element name="idConta" nillable="true" type="xs:int" />
6             </xs:sequence>
7         </xs:complexType>
8 </xs:element>
```

Código A.6: verificaCliente alterando o tipo de idConta para float após mutação

3. (a) **Nome:** mudaTipoElementoMensagem

   (b) **Descrição:** modifica o tipo de um elemento definido em uma mensagem por um elemento de outro tipo definido em outra mensagem também presente no mesmo documento WSDL.

   (c) **Base:** Operador SMP definido por Siblini e Mansour [SM05].

   (d) **Exemplo:** o Código A.7 mostra o trecho do documento WSDL que descreve a mensagem utilizada em uma operação para o cálculo do quadrado de um determinado número (*double quadrado (int x)*). O operador irá modificar o tipo de uma mensagem pelo tipo de uma outra mensagem também definida no mesmo documento WSDL. O resultado do operador é mostrado no Código A.8, que teve a linha 6 modificada.

```
1 <wsdl:message name="quadradoMessage">
2         <wsdl:part name="part1" element="xsd:quadrado" />
3 </wsdl:message>
4
5 <wsdl:message name="quadradoResponseMessage">
6         <wsdl:part name="part1" element="xsd:quadradoResponse" />
7 </wsdl:message>
```

Código A.7: quadradoMessage

```
1 <wsdl:message name="quadradoMessage">
2         <wsdl:part name="part1" element="xsd:quadrado" />
3 </wsdl:message>
4
5 <wsdl:message name="quadradoResponseMessage">
6         <wsdl:part name="part1" element="xsd:quadrado" />
7 </wsdl:message>
```

Código A.8: quadradoMessage após mutação

4. (a) **Nome:** mudaInputOutput

   (b) **Descrição:** modifica o tipo da mensagem de *input* pelo tipo da mensagem de *output* de uma mesma operação definida no documento WSDL.

   (c) **Base:** Operador SMP definido por Siblini e Mansour [SM05].

   (d) **Exemplo:** esse exemplo utiliza a mesma operação apresentada no operador mudaTipoElementoMensagem. O operador será aplicado ao trecho do docu-

mento WSDL apresentado no Código A.9. Após aplicar o operador, a operação quadrado terá a ordem dos elementos *input* e *output* alteradas conforme mostrado no Código A.10, que teve as linhas 5 e 8 modificadas.

```
1  <wsdl:operation name="quadrado">
2
3          <wsdl:input xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
4                  wsaw:Action="urn:quadrado"
5                  message="ns:quadradoMessage" />
6
7          <wsdl:output xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
8                  message="ns:quadradoResponseMessage"
9                  wsaw:Action="urn:quadrado" />
10
11 </wsdl:operation>
```

Código A.9: operationQuadrado

```
1  <wsdl:operation name="quadrado">
2
3          <wsdl:input xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
4                  wsaw:Action="urn:quadrado"
5                  message="ns:quadradoResponseMessage"  />
6
7          <wsdl:output xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
8                  message="ns:quadradoMessage"
9                  wsaw:Action="urn:quadrado" />
10
11 </wsdl:operation>
```

Código A.10: operationQuadrado após mutação

APÊNDICE B
DESCRIÇÃO DAS CONSULTAS XQUERY

1. (a) **Nome:** consultaAssinaturaServico

   (b) **Descrição:** realiza consulta XQuery aos elementos do documento WSDL modificados pelo operador de mutação mudaAssinaturaServiço.

   (c) **Operador no qual se baseia:** Operador mudaAssinaturaServiço

   (d) **Exemplo:** utilizando o mesmo exemplo apresentado no operador mudaAssinaturaServiço. As partes do documento WSDL relevantes para a consulta são mostradas nos Códigos B.1, B.2, B.3, B.4, B.5.

   O trecho apresentado no Código B.1 define o serviço *VerificaClienteService*. Pela definição apresentada, este serviço possui dois *ports* cujos nomes são *VerificaClienteServiceSOAP11port* e *VerificaClienteServiceSOAP12port*, respectivamente. Neste exemplo, para demonstrar a aplicação da consulta, é utilizado o segundo *port*, *VerificaClienteServiceSOAP12port*. Neste *port* está definido o *binding* (Código B.2) de nome *ns:VerificaClienteServiceSOAP12Binding* que possui como tipo o valor *ns:VerificaClienteServicePortType*. O Código B.3 apresenta o portType *VerificaClienteServicePortType* que define a operacao de nome verificaCliente. Esta operação possui como entrada a mensagem *ns:verificaClienteMessage* e como mensagem de retorno *ns:verificaClienteResponseMessage*. Estas mensagens são mostradas no Código B.4. A mensagem *ns:verificaClienteMessage* possui o elemento *xsd:verificaCliente* como *part*. O Código B.5 apresenta a seção types do documento WSDL e nele está descrito o elemento *xsd:verificaCliente* ao qual a mensagem *ns:verificaClienteMessage* fazia referência. Este elemento é do tipo complexo formado por uma sequencia de dois elementos cujos nomes são idCliente e idConta, respectivamente.

   Descrita as partes do documento WSDL, a consulta irá percorrer todo o documento WSDL e irá retornar o nome do método e os parâmetros do mesmo.

Para isso é realizada uma consulta XQuery que inicia analisando a seção *service* (Código B.1) onde obtém a informação do *binding* (Código B.2) correspondente. Através do *binding* é possível chegar a seção *portType* (Código B.3) que define a mensagem utilizada (Código B.4) que é do tipo *verificaCliente* (Código B.5). Após essa análise a consulta XQuery retorna as informações contidas no Código B.6.

```
<wsdl:service name="VerificaClienteService">
  <wsdl:port name="VerificaClienteServiceSOAP11port"
    binding="ns:VerificaClienteServiceSOAP11Binding">
      <soap:address
          location="http://localhost:8080/axis2/services/VerificaClienteService" />
  </wsdl:port>
  <wsdl:port name="VerificaClienteServiceSOAP12port"
    binding="ns:VerificaClienteServiceSOAP12Binding">
      <soap12:address
        location="http://localhost:8080/axis2/services/VerificaClienteService" />
  </wsdl:port>
</wsdl:service>
```

Código B.1: seção service

```
<wsdl:binding name="VerificaClienteServiceSOAP12Binding"
  type="ns:VerificaClienteServicePortType">
    <soap12:binding transport="http://schemas.xmlsoap.org/soap/http"
      style="document" />
          <wsdl:operation name="verificaCliente">
            <soap12:operation soapAction="urn:verificaCliente" style="document" />
                <wsdl:input>
                  <soap12:body use="literal" />
                </wsdl:input>
                <wsdl:output>
                  <soap12:body use="literal" />
                </wsdl:output>
          </wsdl:operation>
</wsdl:binding>
```

Código B.2: seção binding

```
<wsdl:portType name="VerificaClienteServicePortType">
        <wsdl:operation name="verificaCliente">
                <wsdl:input xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
                        wsaw:Action="urn:verificaCliente"
                        message="ns:verificaClienteMessage" />
                <wsdl:output xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
                        message="ns:verificaClienteResponseMessage"
                        wsaw:Action="urn:verificaCliente" />
        </wsdl:operation>
</wsdl:portType>
```

Código B.3: seção portType

```
<wsdl:message name="verificaClienteMessage">
        <wsdl:part name="part1" element="xsd:verificaCliente" />
</wsdl:message>

<wsdl:message name="verificaClienteResponseMessage">
        <wsdl:part name="part1" element="xsd:verificaClienteResponse" />
</wsdl:message>
```

Código B.4: seção message

```
<wsdl:types>
  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    attributeFormDefault="qualified"
    elementFormDefault="qualified"
    targetNamespace="http://service.verificacliente.solino.br/xsd">
      <xs:element name="verificaCliente">
        <xs:complexType>
            <xs:sequence>
              <xs:element name="idCliente" nillable="true" type="xs:int" />
              <xs:element name="idConta" nillable="true" type="xs:int" />
            </xs:sequence>
          </xs:complexType>
      </xs:element>
        <xs:element name="verificaClienteResponse">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="return" nillable="true" type="xs:boolean" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
  </xs:schema>
</wsdl:types>
```

Código B.5: seção types

```
ConsultaAssinaturaServico

        Método: verificaCliente
                Paramêtro: name="idCliente"
                Paramêtro: name="idConta"
```

Código B.6: ConsultaAssinaturaServico

2. (a) **Nome:** consultaTipoAssinaturaServiço

   (b) **Descrição:** realiza a consulta Xquery aos elementos do documento WSDL modificados pelo operador de mutação mudaTipoAssinaturaServiço. Dessa forma são retornados os tipos dos parâmetros definidos na assinatura dos serviços.

   (c) **Operador no qual se baseia:** Operador mudaTipoAssinaturaServiço

   (d) **Exemplo:** para localizar a parte do documento WSDL modificada pelo operador de mutação é realizado um procedimento semelhante ao descrito para o operador consultaAssinaturaServiço. Dessa forma, as demais consultas apresentarão apenas os trechos relacionados à consulta que está sendo exemplificada. A consulta consultaTipoAssinaturaServiço após percorrer o documento WSDL chega ao trecho do documento WSDL mostrado no Código B.8. Esse trecho de código é o responsável por definir os tipos de paramêtros do serviço verificaCliente. A consulta irá retornar as informações contidas no Código B.8, obtidas a partir do Código B.7.

```
<xs:element name="verificaCliente">
        <xs:complexType>
            <xs:sequence>
                  <xs:element name="idCliente" nillable="true" type="xs:int" />
                  <xs:element name="idConta" nillable="true" type="xs:int" />
            </xs:sequence>
        </xs:complexType>
</xs:element>
```

Código B.7: verificaCliente

```
ConsultaTipoAssinaturaServico

        Método: verificaCliente
                Paramêtro: name="idCliente" type="xs:int"
                Paramêtro: name="idConta"   type="xs:int"
```

Código B.8: ConsultaTipoAssinaturaServico

3. (a) **Nome:** consultaTipoElementoMensagem

   (b) **Descrição:** realiza a consulta XQuery aos elementos do documento WSDL modificados pelo operador de mutação mudaTipoElementoMensagem. Em ou-

tras palavras, a consulta retorna os tipos das mensagens definidas pelo *Web Service*.

(c) **Operador no qual se baseia:** Operador mudaTipoElementoMensagem

(d) **Exemplo:** a consulta percorre o documento WSDL e chega ao trecho do documento WSDL mostrado no Código B.9 que define as mensagens *quadradoMessage* e *quadradoResponseMessage*. A primeira mensagem possui como *port* o elemento *xsd:quadrado* e a segunda mensagem possui o elemento *xsd:quadradoResponse*. Essas informações são obtidas pela consulta e são apresentadas no Código B.10.

```
<wsdl:message name="quadradoMessage">
        <wsdl:part name="part1" element="xsd:quadrado" />
</wsdl:message>

<wsdl:message name="quadradoResponseMessage">
        <wsdl:part name="part1" element="xsd:quadradoResponse" />
</wsdl:message>
```

Código B.9: quadradoMessage

```
ConsultaTipoElementoMensagem

        Mensagem: quadradoMessage
                Part: name="part1" element="xsd:quadrado"

        Mensagem: quadradoResponseMessage
                Part: name="part1" element="xsd:quadradoResponse"
```

Código B.10: ConsultaTipoElementoMensagem

4. (a) **Nome:** consultaInputOutput

(b) **Descrição:** realiza a consulta XQuery aos elementos do documento WSDL modificados pelo operador de mutação mudaInputOutput. A consulta busca a mensagem de *input* e *output* de uma dada operação definida pelo *Web Service*.

(c) **Operador no qual se baseia:** Operador mudaInputOutput

(d) **Exemplo:** a consulta percorre o documento WSDL e chega ao trecho do documento WSDL mostrado no Código B.11. Este trecho descreve a operação *quadrado* que possui como entrada a mensagem *ns:quadradoMessage* e como

retorno a mensagem *ns:quadradoResponseMessage*. A consulta retorna essas informações, que são mostradas no Código B.12.

```
<wsdl:operation name="quadrado">

        <wsdl:input xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
                wsaw:Action="urn:quadrado"
                message="ns:quadradoMessage" />

        <wsdl:output xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
                message="ns:quadradoResponseMessage"
                wsaw:Action="urn:quadrado" />

</wsdl:operation>
```

Código B.11: operationQuadradoConsulta

```
ConsultaInputOutput

        Operation: quadrado
                Input:   message ="ns:quadradoMessage"
                Output:  message ="ns:quadradoResponseMessage"
```

Código B.12: ConsultaInputOutput

5. (a) **Nome:** Consulta *Incorrect Order*

   (b) **Descrição:** gera uma consulta XQuery ao documento WSDL que retorne a ordem definida nos elementos *complexType*. A ordem retornada pela consulta será comparada à ordem descrita na especificação.

   (c) **Operador no qual se baseia:** Classe de defeito *Incorrect Order* proposta por Emer et al. [EVJ05]

   (d) **Exemplo:** a consulta percorre o documento WSDL até encontrar pela definição de tipos complexos. A consulta irá retornar a ordem definida pelos elementos do tipo complexo encontrados. Para o tipo complexo apresentado no Código B.13 a consulta irá retornar as informações contidas no Código B.14.

```
<xs:complexType>
  <xs:sequence>
    <xs:element name="idCliente" nillable="true" type="xs:int" />
    <xs:element name="idConta" nillable="true" type="xs:int" />
   </xs:sequence>
</xs:complexType>
```

Código B.13: Exemplo ComplexType

```
complexType
  sequence
    element: idCliente
    element: status
```

Código B.14: Retorno Consulta Ordem Incorreta

6. (a) **Nome:** consulta *Incorrect Value*

   (b) **Descrição:** consulta por valores *default* e *fixed* de um elemento.

   (c) **Operador no qual se baseia:** Classe de defeito *Incorrect Value* proposta por Emer et al. [EVJ05]

   (d) **Exemplo:** a consulta percorre o documento WSDL até encontrar pela definição de tipos complexos que contenham a definição de valores *default* e/ou *fixed* para um elemento. O Código B.15 possui dois elementos que possuem valores default definidos. O primeiro elemento possui o valor 1 como default e o segundo elemento possui o valor 0 como default. Para o tipo complexo apresentado no código B.15 a consulta irá retornar as informações contidas no Código B.16.

```
<xs:complexType>
  <xs:sequence>
    <xs:element name="idCliente" nillable="true" type="xs:int" default="1" />
    <xs:element name="idConta" nillable="true" type="xs:int"  default="0" />
   </xs:sequence>
</xs:complexType>
```

Código B.15: Exemplo ComplexType

```
complexType
  sequence
    element: idCliente default="0"
    element: idConta   default="1"
```

Código B.16: Retorno da Consulta *Incorrect Value*

7. (a) **Nome:** consulta *Incorrect Occurrence*

   (b) **Descrição:** consulta pelo número mínimo e máximo de ocorrências de um elemento

   (c) **Operador no qual se baseia:** classe de defeito *Incorrect Occurrence* proposta por Emer et al. [EVJ05]

   (d) **Exemplo:** a consulta percorre o documento WSDL até encontrar pela definição de tipos complexos que contenham a definição de número mínimo (*minOccurs*) e/ou máximo *maxOccurs* de ocorrência para um elemento. O Código B.17 possui o primeiro elemento que possui o valor mínimo de ocorrências igual a 1 e o segundo elemento que possui como valor mínimo de ocorrências o valor 0 e como número máximo de ocorrências o valor *unbounded*. Para o tipo complexo apresentado no código B.17 a consulta irá retornar as informações contidas no Código B.18.

```
<xs:complexType>
  <xs:sequence>
    <xs:element name="idCliente" nillable="true" type="xs:int"
          minOccurs="1" />
    <xs:element name="idConta" nillable="true" type="xs:int"
          minOccurs="0" maxOccurs="unbounded" />
   </xs:sequence>
</xs:complexType>
```

Código B.17: Exemplo ComplexType

```
complexType
  sequence
    element: idCliente minOccurs: "1"
    element: idConta   minOccurs: "0"   maxOccurs: "unbounded"
```

Código B.18: Retorno da Consulta *Incorrect Occurence*

8. (a) **Nome:** consulta *Incorrect Use*

(b) **Descrição:** consulta pelo tipo de uso, opcional ou requerido, de um atributo.

(c) **Operador no qual se baseia:** Classe de defeito *Incorrect Use* proposta por Emer et al. [EVJ05]

(d) **Exemplo:** a consulta percorre o documento WSDL até encontrar pela definição de tipos complexos que contenham a definição de tipo de uso (requerido ou opcional) para um elemento. O Código B.19 possui o elemento *lang* que tem o seu uso requerido. Para o tipo complexo apresentado no código B.19 a consulta irá retornar as informações contidas no Código B.20.

```
1 <xs:attribute name="lang" type="xs:string" use="required"/>
```

Código B.19: Exemplo atributo

```
1 Attribute
2     name: lang
3     use: required
```

Código B.20: Retorno da Consulta *Incorrect Use*

9. (a) **Nome:** consulta *Incorrect Value* relacionado à classe de atributos

(b) **Descrição:** consulta se um atributo possui um valor *default* ou *fixed*

(c) **Operador no qual se baseia:** Classe de defeito *Incorrect Value* proposta por Emer et al. [EVJ05]

(d) **Exemplo:** semelhante a consulta IncorrectValue apresentada no item 6, porém a consulta descrita aqui está relacionado à classe de atributos. A consulta percorre o documento WSDL até encontrar pela definição de atributos que contenham a definição de valores *default* e/ou *fixed*. O Código B.21 possui o atributo *lang* que tem o valor *BR* como fixo. Para o tipo atributo apresentado no código B.21 a consulta irá retornar as informações contidas no Código B.22.

```
1  <xs:attribute name="lang" type="xs:string" fixed="BR"/>
```

Código B.21: Exemplo atributo

```
1  Attribute
2      name: lang
3      fixed: BR
```

Código B.22: Retorno da Consulta *Incorrect Value*

10. (a) **Nome:** consulta *Incorrect Data Types*

    (b) **Descrição:** consulta pelos tipos de dados mais comuns no XML esquema: string, decimal, integer, boolean, date e time.

    (c) **Operador no qual se baseia:** Classe de defeito *Incorrect Data Types* proposta por Emer et al. [EVJ05]

    (d) **Exemplo:** a consulta percorre o documento WSDL até encontrar pela definição de tipos complexos. A consulta irá retornar o tipo dos elementos definidos no tipo complexo. O Código B.23 possui um tipo complexo que define uma sequência dos elementos *idCliente* do tipo *xs:int* e *idConta* também do tipo *xs:int*. Para o tipo complexo apresentado no código B.23 a consulta irá retornar as informações contidas no Código B.24.

```
1  <xs:complexType>
2    <xs:sequence>
3      <xs:element name="idCliente" nillable="true" type="xs:int" />
4      <xs:element name="idConta" nillable="true" type="xs:int" />
5     </xs:sequence>
6  </xs:complexType>
```

Código B.23: Exemplo ComplexType

```
complexType
  sequence
    element: idCliente  type="xs:int"
    element: idConta    type="xs:int"
```

Código B.24: Retorno da Consulta *Incorrect Data Types*

11. (a) **Nome:** consulta *Incorrect Enumerated Values*

    (b) **Descrição:** consulta os valores aceitáveis para um elemento.

    (c) **Operador no qual se baseia:** Classe de defeito *Incorrect Enumerated Values* proposta por Emer et al. [EVJ05]

    (d) **Exemplo:** a consulta percorre o documento WSDL até encontrar pela definição de tipos enumerados. A consulta irá retornar os valores aceitáveis para o tipo enumerado. O Código B.25 apresenta os valores aceitáveis Palio, Celta e Fiesta para o elemento *car*. Para o trecho apresentado no código B.25 a consulta irá retornar as informações contidas no Código B.26.

```
<xs:element name="car">

<xs:simpleType>
  <xs:restriction base="xs:string">
    <xs:enumeration value="Palio"/>
    <xs:enumeration value="Celta"/>
    <xs:enumeration value="Fiesta"/>
  </xs:restriction>
</xs:simpleType>

</xs:element>
```

Código B.25: Exemplo de enumeração

```
element name="car"
  restriction base: "xs:string"
    enumeration value: "Palio"
    enumeration value: "Celta"
    enumeration value: "Fiesta"
```

Código B.26: Retorno da Consulta *Incorrect Enumerated Values*

12. (a) **Nome:** consulta *Incorrect Maximum and Minimum Values*

    (b) **Descrição:** consulta os valores máximo e mínimo de valores numéricos.

(c) **Operador no qual se baseia:** Classe de defeito *Incorrect Value* proposta por Emer et al. [EVJ05].

(d) **Exemplo:** a consulta percorre o documento WSDL até encontrar pela definição de elementos numéricos que contenham restrições de valores mínimo e máximo. O Código B.27 possui o elemento *age* que é um valor númerico que tem 0 (zero) como valor mínimo e 120 como valor máximo. Para o código apresentado em B.27 a consulta irá retornar as informações contidas no Código B.28.

```
1  <xs:element name="age">
2
3  <xs:simpleType>
4    <xs:restriction base="xs:integer">
5      <xs:minInclusive value="0"/>
6      <xs:maxInclusive value="120"/>
7    </xs:restriction>
8  </xs:simpleType>
9
10 </xs:element>
```

Código B.27: Exemplo de restrição MaxInclusive MinInclusive

```
1  element name="age"
2    restriction base: "xs:integer"
3      minInclusive value: "0"
4      maxInclusive value: "120"
```

Código B.28: Retorno da Consulta *Incorrect Maximum and Minimum Values*

13. (a) **Nome:** consulta *Incorrect Pattern.*

(b) **Descrição:** consulta o padrão definido para um elemento.

(c) **Operador no qual se baseia:** Classe de defeito *Incorrect Pattern* proposta por Emer et al. [EVJ05].

(d) **Exemplo:** a consulta percorre o documento WSDL até encontrar pela definição de elementos que contenham restrições de padrões aceitáveis. O Código B.29 possui o elemento *letter* que é um valor do tipo string que aceita qualquer padrão formado pelos caracters de a até z. Para o código apresentado em B.29 a consulta irá retornar as informações contidas no Código B.30.

```
<xs:element name="letter">

<xs:simpleType>
  <xs:restriction base="xs:string">
    <xs:pattern value="[a-z]"/>
  </xs:restriction>
</xs:simpleType>

</xs:element>
```

Código B.29: Exemplo de restrição em padrões

```
element name="letter"
  restriction base: "xs:string"
    pattern value: "[a-z]"
```

Código B.30: Retorno da Consulta *Incorrect Pattern*

14. (a) **Nome:** consulta *Incorrect White Spaces Characters*

    (b) **Descrição:** consulta o modo como o processador XML deve processar os caracteres de espaço em branco.

    (c) **Operador no qual se baseia:** Classe de defeito *Incorrect White Space Characters* proposta por Emer et al. [EVJ05].

    (d) **Exemplo:** a consulta percorre o documento WSDL até encontrar pela definição de como o processador XML deve tratar os caracteres em branco. Pode ser definido como *default* que permite que aplicação controle os espaço em branco quando necessário e *preserve* onde os caracteres em branco devem ser mantidos. O Código B.31 informa ao processador XML que os caracteres em branco devem ser mantidos (valor *preserve*). Para o código apresentado em B.31 a consulta irá retornar as informações contidas no Código B.32.

```
<xs:element name="address">

<xs:simpleType>
  <xs:restriction base="xs:string">
    <xs:whiteSpace value="preserve"/>
  </xs:restriction>
</xs:simpleType>

</xs:element>
```

Código B.31: Exemplo de restrição em padrões

```
element name="letter"
  restriction base: "xs:string"
    whitespace value: "preserve"
```

Código B.32: Retorno da Consulta *Incorrect White Spaces characters*

15. (a) **Nome:** consulta *Incorrect Length*.

    (b) **Descrição:** consulta a restrição de tamanho de um elemento.

    (c) **Operador no qual se baseia:** Classe de defeito *IncorrectLength* proposta por Emer et al. [EVJ05].

    (d) **Exemplo:** a consulta percorre o documento WSDL até encontrar pela restrição do tamanho de um elemento. O Código B.33 possui o elemento *password* que é um valor do tipo string que deve ter o tamanho igual a 8. Para o código apresentado em B.33 a consulta irá retornar as informações contidas no Código B.34.

```
<xs:element name="password">

<xs:simpleType>
  <xs:restriction base="xs:string">
    <xs:length value="8"/>
  </xs:restriction>
</xs:simpleType>

</xs:element>
```

Código B.33: Exemplo de restrição no tamanho

```
element name="letter"
  restriction base: "xs:string"
    length value: "8"
```

Código B.34: Retorno da Consulta *Incorrect Length*

16. (a) **Nome:** consulta *Incorrect Digits*.

    (b) **Descrição:** consulta a quantidade total de dígitos que um tipo númerico pode conter.

    (c) **Operador no qual se baseia:** Classe de defeito *Incorrect Digits* proposta por Emer et al. [EVJ05].

(d) **Exemplo:** a consulta percorre o documento WSDL até encontrar por restrições na quantidade total de dígitos que um tipo numérico pode conter. O Código B.35 possui o elemento *amount* que é um valor do tipo numérico que tem um número total de dígitos igual a 8 e número de dígitos fracionais igual a 2. Para o código apresentado em B.35 a consulta irá retornar as informações contidas no Código B.36.

```
<simpleType name="amount">
  <restriction base="decimal">
    <totalDigits value="8"/>
    <fractionDigits value="2" fixed="true"/>
  </restriction>
</simpleType>
```

Código B.35: Exemplo de restrição de dígitos incorretos

```
element name="amount"
  restriction base: "decimal"
    totalDigits     value ="8"
    fractionDigits  value ="2"
```

Código B.36: Retorno da Consulta *Incorrect Digits*

APÊNDICE C
DESCRIÇÃO DOS DEFEITOS INSERIDOS

Tabela C.1: Defeitos inseridos - WS Temperatura

Web Service	**Resumo defeito inserido**	**Local onde foi inserido o defeito**
WS Temperatura 1.1	Modificao o tipo do part da mensagem name = "Celsius2FarenheitMessage"de double para integer	WSDL
WS Temperatura 1.2	Trocado o part da mensagem Celsius2FarenheitMessage pelo part da mensagem Celsius2Farenheit ResponseMessage	WSDL
WS Temperatura 1.3	modificao o tipo do part da mensagem name = "Celsius2Farenheit ResponseMessage"de double para integer	WSDL
WS Temperatura 2.1	Mmodificou o metodo Celsius2Farenheit alterando o sinal * para + e o de + para -	Código
WS Temperatura 2.2	modificou o metodo Farenheit2Celsius substituoi o sinal - por /	Código
WS Temperatura 2.3	Metodo Celsius2Farenheit trocou o parametro de double para int	Código
WS Temperatura 3.1	Modificao o tipo do part da mensagem name = "Celsius2FarenheitMessage"de double para integer	WSDL e Código
WS Temperatura 3.2	Modificado o tipo de retorno do metodo Celsius2Farenheit de double para int	WSDL e Código

Tabela C.2: Defeitos inseridos - WS Verifica Cliente

Web Service	**Resumo defeito inserido**	**Local onde foi inserido o defeito**
WS Verifica Cliente 1.1	Na mensagem verificaClienteMessage foi altera a ordem entre idCliente e idConta	WSDL
WS Verifica Cliente 1.2	Alterado o atributo idConta da mensagem verificaClienteMessage. O tipo foi alterado de xs:int para xs:boolean	WSDL
WS Verifica Cliente 1.3	Trocado o atributo part da mensagem verificaClienteMessage pelo atributo part da mensagem verificaClienteResponseMessage	WSDL
WS Verifica Cliente 1.4	Na seção operation, trocada a mensagem de input pela mensagem de output para a operação verificaCliente	WSDL
WS Verifica Cliente 2.1	No método verificaCliente foi invertida a ordem dos parâmetros	Código
WS Verifica Cliente 2.2	Trocado o tipo do parâmetro idCliente de int para String	Código
WS Verifica Cliente 2.3	Alterado o retorno do método verificaCliente de boolean para String	Código
WS Verifica Cliente 3.1	Na mensagem verificaClienteMessage foi altera a ordem entre idCliente e idConta. Alterado a implementação para refletir a modificação	WSDL e Código
WS Verifica Cliente 3.2	Trocado o tipo do parâmetro idCliente de int para String no documento WSDL e na implementação	WSDL e Código
WS Verifica Cliente 3.3	Modificado o tipo do retorno do metodo verificaCliente de boolean para String	WSDL e Código

Tabela C.3: Defeitos inseridos - WS CPF

Web Service	**Resumo defeito inserido**	**Local onde foi inserido o defeito**
WS CPF 1.1	Trocado atributo boolean para string na mensagem ValidaCPFResponseMessage	WSDL
WS CPF 1.2	Na mensagem validaCPFResponseMessage trocado part name = "return"type = "xsd:boolean"por part name = "cpf"type = "xsd:string".	WSDL
WS CPF 1.3	Na operacao validaCPF, trocada a mensagem de input para ns:validaCPFResponse Message e a de output para ns:validaCPFMessage	WSDL
WS CPF 2.1	Modificado o parâmetro do metodo calcDigVerif. De String foi substituido para Integer	Código
WS CPF 2.2	Modificado o retorno do metodo validaCPF. De boolean para String	Código
WS CPF 2.3	Inserido defeito no método calcDigVerif(String num) onde exisita peso– foi substituido por peso++	Código
WS CFP 2.4	Inserida a modificacao no metodo geraCPF(). Variável peso ficou com índice 8, quando deveria ser 9	Código
WS CPF 2.5	inserido defeito no metodo validaCPF(String cpf) em uma substring o indice 9 ao 11 foi alterado para 8 ao 10	Código
WS CPF 3.1	Modificado o retorno de boolean para String	WSDL e Código
WS CPF 3.2	Na mensagem validaCPFResponseMessage trocado part name="return"type = "xsd:boolean"por part name="cpf"type = "xsd:string"	WSDL e Código
WS CPF 3.3	Modificado o parametro do metodo calcDigVerif. De String foi substituido para Integer	WSDL e Código