Ledyvânia Franzotte

UTILIZANDO ANÁLISE DE MUTANTES PARA REALIZAR O TESTE DE DOCUMENTOS *XML SCHEMA*

CURITIBA

2006

Ledyvânia Franzotte

UTILIZANDO ANÁLISE DE MUTANTES PARA REALIZAR O TESTE DE DOCUMENTOS *XML SCHEMA*

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: Prof[a]. Dra. Silvia Regina Vergílio

Universidade Federal do Paraná

CURITIBA

2006

"Começar é a parte mais importante de qualquer trabalho."

Platão

Dedico este trabalho aos meus pais, Ledy Braun Franzotte e Vardenir Franzotte, e ao meu irmão, Wagner Luizio Franzotte, pelo apoio e compreensão durante toda esta caminhada.

Agradeço primeiramente a *Deus* a oportunidade de realizar o Mestrado e, principalmente, a conquista deste título.

Agradeço a minha *família* o apoio que me foi dado desde o começo.

Agradeço a minha orientadora *Silvia Regina Vergílio* a paciência e o aprendizado adquirido.

Agradeço a todos os meus *colegas* do mestrado que, com alegria, em diversos momentos ajudaram a amenizar os momentos difíceis e complicados pelos quais passamos.

E agradeço, em especial, as minhas amigas os momentos de alegrias e brincadeiras. À *Regina de Cássia Nandi*, sempre presente, desde o início, em todos os momentos alegres e tristes pelos quais passamos, e à *Francielle Biguelini*, sempre disposta a ajudar e apoiar nos momentos que precisei. Também agradeço ao meu namorado, *Rodrigo Cordeiro dos Santos*, a paciência e o carinho demonstrado durante o Mestrado.

SUMÁRIO

RESUMO

Diversas aplicações *Web* utilizam documentos XML para troca de informações, tanto entre componentes de aplicações distintas quanto entre componentes da própria aplicação. Estes documentos, na maioria dos casos, obedecem a uma gramática previamente descrita por um esquema.

O tipo de esquema que está se tornando mais utilizado é o documento *XML Schema*. Este fato deve-se principalmente a algumas de suas características, tais como: possuir uma gramática rica na representação dos dados e poder ser escrito utilizando-se a linguagem XML que é a mesma usada para escrever documentos.

Estes esquemas são “traduzidos” de especificações textuais escritas em linguagem natural, e consequentemente, é usual que neste processo de tradução aconteçam enganos que acabam permitindo que defeitos semânticos estejam presentes nos documentos XML utilizados pela aplicação.

Este trabalho apresenta um processo de teste para revelar defeitos em documentos *XML Schema* baseado na técnica Análise de Mutantes. Operadores de mutação são propostos tendo-se como base os erros mais comuns cometidos ao se projetar um documento *XML Schema*. Para dar suporte aos operadores propostos, foi implementada uma ferramenta denominada XTM. Com o auxílio dessa ferramenta alguns experimentos puderam ser realizados. Resultados desses experimentos mostram a aplicabilidade dos operadores propostos bem como sua eficácia em revelar defeitos.

ABSTRACT

XML language is largely used by *Web*-based applications to exchange data among different components. XML documents, in most cases, follow a previously grammar or schema that describes which elements and data types are expected by the application.

XML Schema has become very popular, due to its characteristics, such as: rich grammar to represent data; expressed in XML, which is the same format used in documents.

These schema are “translated” from specifications written in natural language, and consequently, in this process some mistakes are usually made in this process, resulting in semantic faults in the XML documents.

This work introduces a testing process to reveal *XML Schema* faults based on Mutation Analysis technique. Mutation operators are proposed considering the most common mistakes made in the project of *XML Schemas*. A tool, named XTM, to support the proposed operators was implemented. By using this tool some experiments were accomplished. Results from these experiments show the applicability of the operators, as well as, their efficacy to reveal faults.

LISTA DE TABELAS

LISTA DE FIGURAS

GLOSSÁRIO

Sigla	***Definição***
AOC	Attribute Occurence Change
API	Applications Programming Interface
CCR	Complex Type Compositors Replacement
COC	Complex Type Order Change
CSP	Change Singular Plural
DOM	Document Object Model
DT	Data Type
DTC	Derived Type Control Replacement
DTD	Document Type Definition
EBNF	Extended Backus Naur Form
EEC	Enumeration Element Change
ENM	Element Namespace Removal
ENR	Element Namespace Replacement
EOC	Element Occurence Change
GO	Group Order
HTML	HyperText Markup Language
IT	Insert_Tree
JDOM	Java DOM
NCC	Number Constraint Change
QGR	Qualification Globally Replacement
QIR	Qualification Individually Replacement
RAR	Restriction Arguments Replacement
REQ	Required
RPC	Remote Procedure Call
RT	Remove_Tree
RTG	Regular Tree Grammar
SAX	Simple API for XML

SLC	String Lenght Change
SNC	*XML Schema* Namespace Changes
SO	Size Occurence
SPC	SimpleType Pattern Change
SQC	Schema Quality Checker
SQL	Structured Query Language
STE	SubTree_Exchange
TDE	Target and Default Exchange
UCR	Use of Controller Replacement
W3C	World Wide *Web* Consortium
XML	eXtensive Markup Language
XQUERY	XML Query Language
XSV	*XML Schema* Validador
XTM	Ferramenta para Teste de Documentos ***XML Schema*** Baseado em **T**este de **M**utação

1. INTRODUÇÃO

1.1. Contexto

XML (*eXtensible Markup Language*) [XML05] é uma linguagem de marcação para conteúdo. É um padrão utilizado pelos documentos usados na troca de informações entre diversos tipos de aplicações, como por exemplo, aplicações *Web*.

Estes documentos, na maioria dos casos, obedecem a uma sintaxe previamente definida por algum esquema. Os esquemas referem-se a metadados para validação e especificação de elementos que os documentos XML podem conter. As linguagens de definições de esquemas mais utilizadas são: DTD (*Data Type Definition*) [DTD05] e *XML Schema* [XMLS05].

Atualmente a linguagem *XML Schema* está sendo mais utilizado que a DTD, pois ela é expressa em XML, o mesmo formato dos documentos usados pelas aplicações. Como o desenvolvedor já está familiarizado com a linguagem XML, o documento *XML Schema* acaba tornando-se mais fácil de ser escrito e compreendido por não necessitar de aprendizado de uma outra linguagem. Este é um dos fatores que influenciam no crescente uso do *XML Schema*.

Os esquemas são utilizados apenas para validar a sintaxe do documento XML, ou seja, qualquer elemento existente no documento XML tem que estar especificado no esquema, caso contrário o documento não será considerado válido.

Os documentos devem ser bem-formados e válidos. A verificação da formatação é um passo executado sem o conhecimento de um esquema. A ferramenta mais comum é o disponibilizado no próprio site da W3C [XSV05], mas podem também ser usadas outras ferramentas tais como *XMLSpy* [SPY05]*, Stylus Studio* [STY05] dentre outras disponíveis no mercado.

A validação, contudo, só é obtida usando-se esquemas, pois são eles que definem a gramática a qual o documento XML deve obedecer. É de

responsabilidade do desenvolvedor definir o que está sendo especificado. Devido a isso, diversos tipos de erros de sintaxe podem acontecer, como por exemplo, o nome de uma *tag* estar escrito de maneira incorreta, o tipo de um conteúdo mal definido, dentre outros. Estes erros refletem em problemas semânticos nos documentos XML que são utilizados na troca de mensagens. Se os esquemas contiverem erros, a validação dos documentos XML fica prejudicada. Ou seja, se for utilizado um documento XML incorreto, criado baseado em um esquema com defeitos, diversas falhas nas aplicações envolvidas poderão ocorrer. Por isso, para assegurar a qualidade das aplicações *Web,* o teste dos esquemas de acordo com a sua especificação é fundamental.

A atividade de teste visa revelar defeitos nas aplicações. Para isto, é necessário planejar, projetar, executar e avaliar os resultados obtidos na etapa de testes. Diversas técnicas podem ser utilizadas para tentar identificar o maior número possível de erros. A cada técnica estão associados diversos critérios a serem satisfeitos.

As principais técnicas utilizadas são [PRE05]: funcional, que deriva dados de teste considerando aspectos funcionais do *software*; estrutural, que deriva dados de teste baseados em fluxo de controle e de dados; e a técnica baseada em defeitos, que tem como objetivo mostrar que certos tipos de defeitos estão presentes no programa em teste. O critério Análise de Mutantes, que é um critério baseado em defeitos [DEM78], tem se mostrado um dos mais eficazes para revelar defeitos [WON94].

Este critério é baseado em dois pressupostos [DEM78]: na hipótese do programador competente, que diz que os programadores desenvolvem programas próximos do correto e, no efeito do acoplamento, ou seja, defeitos complexos são revelados revelando-se defeitos simples.

A idéia deste critério é inserir pequenas modificações no programa em teste. Cada programa modificado é denominado mutante. Dados de teste são executados com o propósito de que todos os mutantes criados sejam mortos, ou seja, deve-se executar o programa mutante com um dado de teste de tal forma que sua saída seja diferente da saída do programa original para aquele dado. Os que não forem mortos são considerados vivos e caso não exista um caso de teste

que diferencie o comportamento do programa mutante do comportamento do programa original este é dito mutante equivalente.

Diversos trabalhos de pesquisa têm focalizado o teste de aplicações *Web* [KUN00, LIU00a, LIU00b, OFF02, WU04, LEE01, OFF04, XU05, LI05, EME05]. Dentre esses, os cinco últimos focalizam técnicas baseadas em defeito.

Em [LEE01], Lee e Offutt introduzem a idéia de mutação de interação introduzindo operadores para modificar documentos XML. Os documentos XML modificados são usados como casos de teste para testar interações entre componentes *Web*. Extensões desse trabalho resultaram em uma abordagem baseada em perturbação de dados para o teste de *Web Services* [OFF04]. Mensagens XML e SOAP são modificadas por operadores e utilizados como casos de teste. Com esse mesmo objetivo Xu et al [XU05] introduziram operadores de mutação para os esquemas. Os esquemas modificados são utilizados para gerar documentos XML mutantes válidos e inválidos no teste de *Web Services*.

São poucos os trabalhos que têm o objetivo de validar documentos *XML Schema* [LI05, EME05]. Em [LI05], Li e Miller introduzem alguns operadores de mutação para documentos *XML Schema*. Estes operadores descrevem alguns defeitos cometidos ao se projetar esquemas. Entretanto, muitos deles geram esquemas que não são considerados válidos, facilmente descobertos na validação dos esquemas. Os autores também não introduziram um processo que realmente aplique o critério análise de mutantes, ou seja, não abordam algumas questões tais como diferenciar o comportamento do programa original e do mutante.

Emer et al [EME05] descrevem uma abordagem cujo objetivo é também testar esquemas. Utilizando classes de defeito em esquemas, documentos XML mutantes são gerados e consultas a esses documentos são executadas. Os resultados dessas consultas são então analisados para se identificar defeitos nos esquemas.

As idéias apresentadas nos trabalhos descritos anteriormente são bastante promissoras, entretanto, os trabalhos não aplicam efetivamente o critério Análise de Mutantes. Xu et al [XU05] não têm o objetivo de testar os esquemas, Emer et al [EME05] não gerou esquemas mutantes, Li e Miller [LI05], apesar de gerarem

mutantes, não definiram um processo que permita a utilização dos mesmos no teste de esquemas. Também não abordam algumas questões, tais como: Como diferenciar o comportamento dos esquemas mutantes e original? Que casos de teste utilizar?

1.2. Motivação

Dado o contexto acima, têm-se as seguintes questões como motivação para o presente trabalho:

1. teste de software é uma atividade fundamental para assegurar a qualidade do software e a aplicação de técnicas sistemáticas pode diminuir o custo dessa atividade;
2. O critério Análise de Mutantes tem se mostrado um dos mais eficazes em revelar defeitos;
3. Importância de realizar o teste de esquemas no contexto de teste de aplicações *Web*;
4. Documentos *XML Schema* têm sido cada vez mais utilizados, entretanto a tradução de uma especificação para um documento *XML Schema* pode ser feita incorretamente;
5. Documentos *XML Schema* bem formados e válidos podem conter defeitos com relação a sua especificação. Esses defeitos não são detectados por validadores;
6. Técnicas baseadas em defeito têm sido estudadas e aplicadas com sucesso no teste de aplicações *Web*, *Web* Services e esquemas. Entretanto, o critério Análise de Mutantes não foi efetivamente aplicado, nem foi definido um processo adaptado ao contexto de *XML Schema*;
7. A aplicação prática e a avaliação do critério Análise de Mutantes só são possíveis se uma ferramenta estiver disponível.

1.3. Objetivos

Este trabalho tem como objetivo explorar o uso do critério Análise de Mutantes no teste de documentos *XML Schema*. Para que esse critério seja utilizado foi introduzido um conjunto de operadores de mutação baseado nos trabalhos de [LEE01, LI05, XU05, EME05] que descrevem os principais defeitos em esquemas. Foi também introduzido um processo de aplicação do critério. Para diferenciar os mutantes do esquema original em teste, documentos XML são gerados e utilizados como dados de testes.

Para dar suporte a esses operadores e a esse processo, uma ferramenta, denominada XTM (Ferramenta para Teste de Documentos ***XML Schema*** Baseado em **T**este de **M**utação) foi implementada. Com o auxílio dessa ferramenta, experimentos foram conduzidos para avaliar a proposta introduzida.

1.4. Organização do Trabalho

Este trabalho está subdividido em sete capítulos. No Capítulo 2 é apresentada uma breve introdução da linguagem XML, de esquemas e de diferentes tecnologias relacionadas. No Capítulo 3 são apresentados os principais conceitos sobre teste de software e os trabalhos relacionados.

No Capítulo 4 operadores de mutações para os documentos *XML Schema* são introduzidos. Estes operadores visam a descrever os diferentes tipos de defeitos que podem estar presentes nos esquemas. O Capítulo 5 descreve a ferramenta XTM.

No Capítulo 6 são apresentados resultados de experimentos realizados para avaliar os operadores propostos e a implementação da ferramenta. O Capítulo 7 contém as conclusões e os trabalhos futuros para a continuação deste estudo.

O trabalho contém um apêndice e dois anexos. O Apêndice A contém os principais algoritmos utilizados na implementação da XTM. O Anexo A contém a relação dos operadores propostos em [LI05] e o Anexo B contém as especificações dos esquemas utilizados nos experimentos realizados.

2. XML E TECNOLOGIAS

XML (*eXtensible Markup Language*) [XML05] é uma linguagem de marcação para conteúdo que, diferentemente do HTML [HTML99] não apresenta nenhum elemento de apresentação inerente ou embutido [TIT03]. Um documento XML é composto por uma lista de elementos de dados que possuem marcas (*tags*) de abertura e fechamento, e entre essas *tags* está contido o conteúdo o qual elas intuitivamente descrevem.

Sua estrutura lógica é definida através de uma hierarquia, assim como uma árvore, facilitando desta maneira sua manipulação. O documento é composto de partes que podem ser elementos individuais, comentários, atributos, instruções de processamento dentre outros. É a representação mais adequada para processamento e troca de dados [MAR01].

Um arquivo XML é dito bem-formado quando todas as marcações abertas foram devidamente fechadas e o aninhamento das *tags* está correto, como ilustrado pelo documento da Figura 2-1.

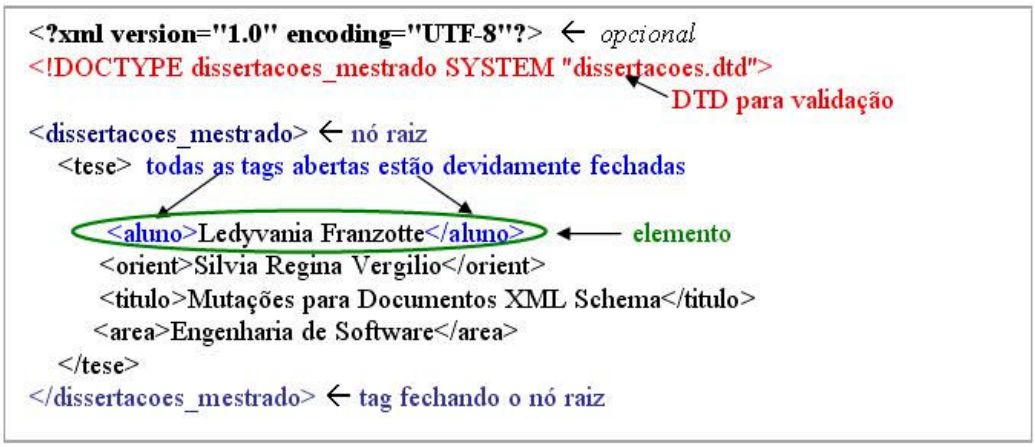

Figura 2-1: Exemplo de um documento XML

Uma vez estando o arquivo bem-formado é possível que o mesmo seja representado como uma estrutura hierárquica (árvore) conforme demonstra a Figura 2-2.

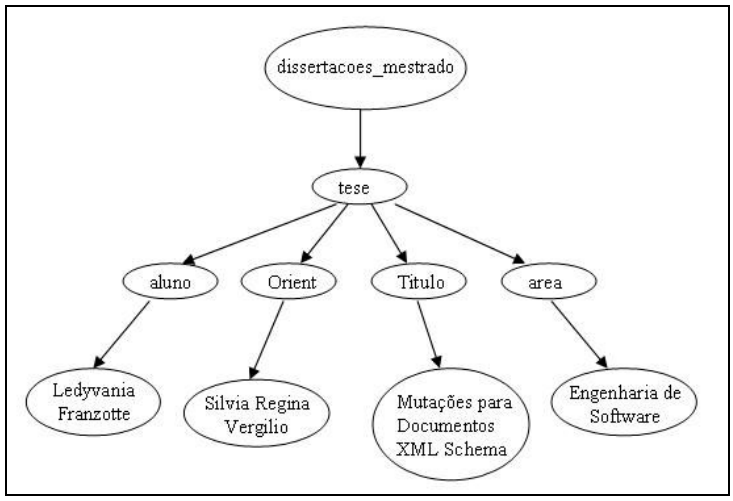

Figura 2-2: Representação Hierárquica de um Documento XML

Em um documento XML é possível codificar o conteúdo, a semântica e as esquematizações para uma grande variedade de aplicações desde simples até as mais complexas.

Como a sua estrutura é semelhante a uma árvore, este é composto de um nó chamado raiz e diversos outros nós internos denominados elementos. Para descrever o conteúdo do documento utiliza-se as tags de marcação “<” e “>”.

Um documento XML dito válido segue a estrutura lógica definida por um arquivo de validação chamado de esquema. Este pode ser por exemplo, uma DTD (*Document Type Definition*) [DTD05], ou um tipo de documento semelhante ao documento XML chamado de *XML Schema*, dentre outros.

2.1. DTD (Document Type Definition)

A DTD é utilizada para definição de tipos de elementos de um documento XML possibilitando que sua sintaxe seja válida, pois qualquer elemento que não esteja especificado na DTD ou que não esteja conforme especificado torna o documento XML inválido [MAR01].

A DTD usa uma gramática oficial, a EBNF (*Extended Backus Naur Form*) para especificar a estrutura e valores que são permitidos em um documento XML [MAR01].

A Tabela 2-1 descreve os tipos de elementos que podem estar contidos nos documentos DTD. Os dois primeiros tipos são elementos que espera-se encontrar em documentos XML, e os demais são opcionais.

A DTD define todos os elementos que estão presentes em um documento XML [TIT03]. A Figura 2-3 apresenta um exemplo de uma DTD. Este exemplo valida o documento XML mostrado Figura 2-1, ou seja, nele estão presentes todos os elementos e seus respectivos tipos permitidos no documento XML.

Tabela 2-1: Conteúdos Permitidos em um Documento DTD

Elementos	Significados
ELEMENT	Declaração de um tipo elemento XML
ATTLIST	Declarações dos atributos que podem ser designados a um tipo de elemento especifico e os valores permitidos
ENTITY	Declaração de conteúdo reutilizável
NOTATION	Declaração do formato para conteúdo que não deve ser analisado em termos de sintaxe

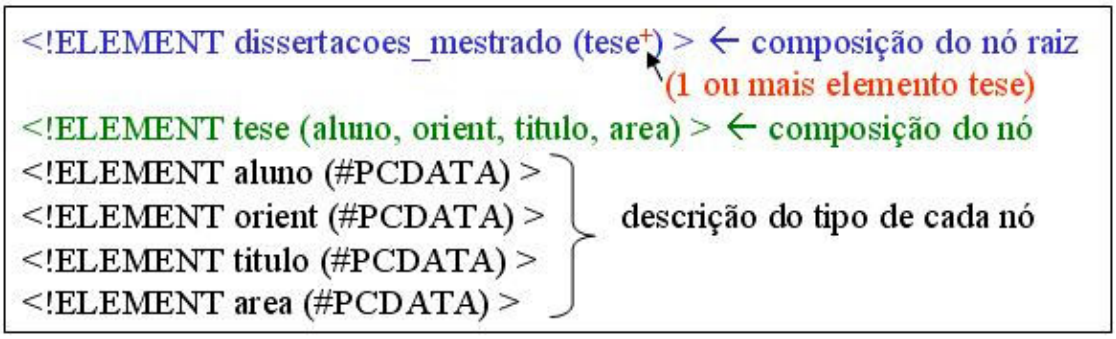

Figura 2-3: Exemplo de um Documento DTD

Os documentos DTD apresentam alguns problemas tais como [MAR01]: o entendimento e a escrita não são tão intuitivos, pois usam uma sintaxe diferente da linguagem XML (EBNF), não é possível a utilização do DOM (Seção 2.3.2) para sua leitura; não fornecem suporte para espaços identificadores (*namespaces*), e todas as definições precisam estar contidas na DTD criada não sendo possível "pegar emprestado" de outras DTD.

Devido a esta e outras características mencionadas anteriormente é que o *XML Schema*, descrito a seguir, quando comparado à DTD, mostra-se mais rico na representação das informações, pois possui uma sintaxe semelhante com a dos documentos XML.

2.2. XML SCHEMA

A origem do *XML Schema* está relacionada às limitações da DTD. Sua idéia geral é descrever (e restringir) um conjunto de instâncias de documentos XML e ser por si só um documento XML. Possui os tipos de dados primitivos como cadeias, números, booleanos e datas e, mecanismos para definir novos tipos de dados ao combinar ou restringir tipos existentes ou simplesmente ao enumerar os valores no tipo de dados [MAR01]. O exemplo da Figura 2-4 descreve um documento *XML Schema* e, assim como a DTD, também possui a descrição de todos os elementos e respectivos tipos permitidos no documento XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
  <xs:element name="dissertacoes_mestrado">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="tese" minOccurs="1">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="aluno" type="xs:string" minOccurs="1"/>
              <xs:element name="orient" type="xs:string" minOccurs="1"/>
              <xs:element name="titulo" type="xs:string" minOccurs="1"/>
              <xs:element name="area" type="xs:string" minOccurs="1"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Figura 2-4: Exemplo de um Documento *XML Schema*

Este *XML Schema* descreve a estrutura física (sintaxe) do documento XML apresentado como exemplo Figura 2-1. Nele é definido os elementos e atributos que poderão aparecer em um documento, quais elementos terão elementos filhos, qual sua ordem e o número de elementos filhos. Também define se um elemento é vazio ou possui texto, os tipos de dados para elementos e atributos e define padrões e fixa valores para elementos e atributos.

Os documentos *XML Schema* aperfeiçoam as DTD por permitirem mais precisão na expressão de alguns conceitos no vocabulário e possuírem uma estrutura mais rica na descrição da informação [MAR01]. Devido a estas características é que o uso de *XML Schema* para validação da estrutura lógica está se tornando cada vez mais freqüente nas aplicações que utilizam XML.

Em um *XML Schema* podem-se criar novos tipos de dados os quais são divididos em duas categorias: tipo de dados simples e tipo de dados complexo.

- tipo de dados simples são os elementos que possuem apenas texto, como por exemplo *integer*, *string*, *date*, entre outros. Este tipo restringe o texto que pode aparecer no valor de um atributo ou no conteúdo de um elemento textual.
- tipo de dados complexos são os elementos que podem conter outros elementos ou atributos. Este tipo restringe o conteúdo de um elemento relativamente aos atributos e elementos filho que pode ter.

Devido a todas as características apresentadas, um XML esquema possui várias vantagens em relação a um DTD:

- utiliza sintaxe XML: ou seja, um *XML Schema* é um documento XML. Isso significa que ele pode ser processado como qualquer outro XML, com as mesmas ferramentas;
- permite a definição de tipo de dado: podem-se definir elementos com tipo *integer, date, time, string*, entre outros;
- é extensível: além dos tipos de dados definidos pela especificação do *XML Schema*, é possível criar novos, inclusive derivados de tipos de dados já definidos;
- possui um poder de expressão maior; por exemplo, elementos podem ser validados através de expressões regulares, ou seja, definição de padrões para determinados campos;
- provê integração com *namespaces* XML;

- provê herança;
- o documento *XML Schema* possui facilidade de documentação por meio dos elementos *xsd:annotation* e *xsd:documentation* que são específicos para documentação;

2.3. Biblioteca de manipulação de documentos XML

Uma vez definidos os arquivos XML, suas estruturas e as aplicações que as utilizarão, é necessária uma API (*Application Programming Interface*) para manipulação destes arquivos. As API's que podem ser geralmente utilizadas são: SAX, DOM e JDOM (extensão da API DOM para aplicações JAVA) e qualquer uma delas tem a mesma finalidade: manipular as árvores do documento XML

2.3.1. SAX

A interface SAX (*Simple API for XML*) [SAX05] permite que sejam lidos dados contidos em um documento XML sem que este seja todo carregado para a memória. É uma API baseada em eventos, pois, "o analisador lê o documento e diz ao programa a respeito dos símbolos que ele encontra" [MAR01]. Ela oferece métodos que respondem a eventos produzidos durante a leitura do documento notificando quando um elemento abre, quando fecha, dentre outros.

A utilização desta API apresenta algumas vantagens, dentre elas: permite que sejam utilizados arquivos de grande tamanho (por haver pouco consumo de memória) e possibilita a criação da estrutura de dados; é simples de ser usada e a informação que se busca é rapidamente obtida. Ela é ideal para aplicações simples que não precisam manipular toda a árvore de objetos.

Assim como qualquer outra API, a SAX também possui algumas desvantagens, dentre as quais citamos [MAR01]: não há acesso aleatório ao documento, pois deve-se lidar com os dados à medida que estes são lidos, não possui uma DTD para validação, informações léxicas não estão disponíveis tais como, comentários, ordem dos atributos e caracteres de nova linha. Outra desvantagem apresentada é que ela não permite modificação no arquivo XML,

tornando este um arquivo apenas para leitura além de não servir para validar referências cruzadas.

2.3.2. DOM

A interface DOM (*Document Object Model*) [DOM05] assim como a API SAX permite que os dados de documentos XML sejam lidos. Todo o arquivo é quebrado em elementos individuais e carregado para a memória criando-se uma representação do arquivo como uma árvore.

A API DOM é uma recomendação definida pela W3C que contém a especificação para a interface que pode acessar qualquer documento estruturado XML e descreve como as cadeias de caracteres devem ser manipuladas e define os objetos, métodos e propriedades que a compõem [DOM05].

Esta API oferece um conjunto robusto de interfaces para facilitar a manipulação da árvore e dos nós, conforme mostra a Figura 2-5.

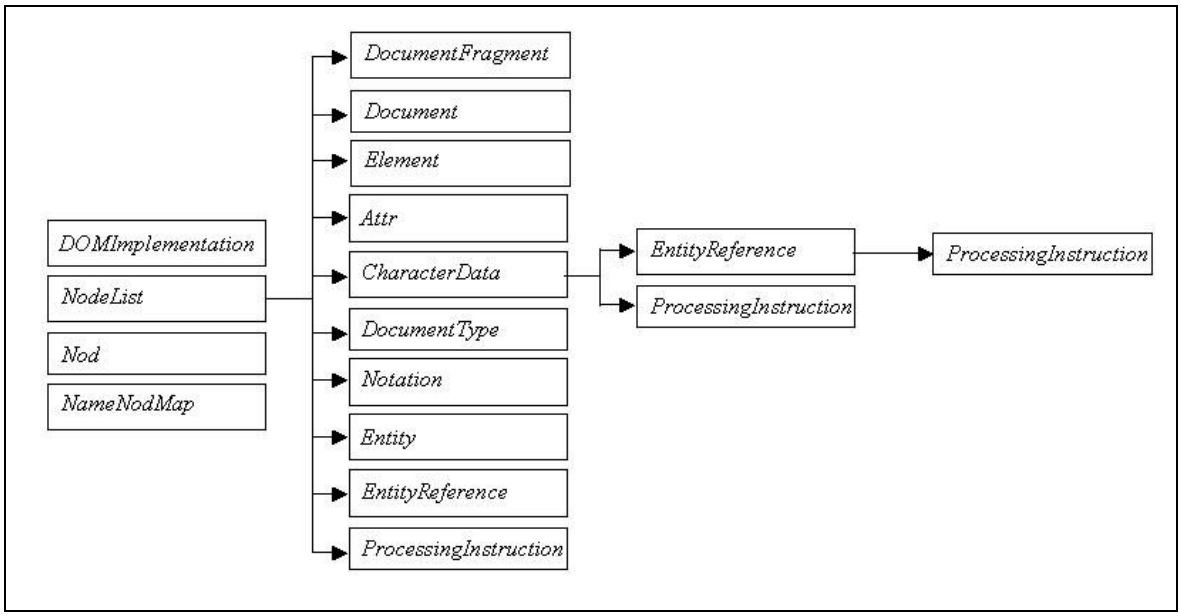

Figura 2-5: Hierarquia de Classes do DOM [DOM05]

Os métodos disponibilizados para o acesso às informações do documento servem para inserção/remoção/troca de nós, adicionar novos nós no final da árvore (inserir como sendo o último elemento), além de oferecer um método para consultar se um determinado nó possui nós filhos.

Assim como métodos, a API também disponibiliza propriedades para referenciar os nós da estrutura.

Diferente da API SAX, a DOM permite que os arquivos XML sejam alterados, permite a utilização de esquemas para validar a gramática (DTD e *XML Schema*), garante a boa formatação e simplifica a manipulação interna dos documentos, pois os elementos estão dispostos como elementos de uma árvore na memória, assim como está representado na Figura 2-2.

1.1.1.1 JDOM

A interface JDOM (Java DOM) [JDOM05] é uma extensão da API DOM. É uma API escrita em Java puro e *open source*. Foi criada para resolver alguns inconvenientes da API DOM, como, por exemplo, só possuir um único tipo de exceção, não possuir métodos típicos de classes Java como *equals*(), *hashcode*(), *clone*(), *tostring*(), além de ser uma API mais intuitiva para os desenvolvedores JAVA do que o DOM [CLA04].

Por ser uma extensão da API DOM, também está baseada na estrutura de árvores e permite o *parsing*, criação, manipulação e serialização dos documentos XML. A Tabela 2-2 apresenta as principais diferenças entre os elementos usados pelas API DOM e JDOM [CLA04].

2.4. Considerações Finais

Este capítulo apresentou a tecnologia XML, as principais linguagens de definição de esquemas utilizadas para a validação de documentos XML, bem como as principais bibliotecas de manipulação para esses documentos. Dentre os esquemas estudados, definiu-se utilizar neste trabalho a linguagem *XML Schema* pois, realizando uma comparação entre DTD e *XML Schema*, é possível diferenciá-los em vários pontos.

Apesar da DTD ser mais simples para se trabalhar, possui a grande desvantagem de não ser XML, dificultando assim o seu uso, uma vez que o usuário precisa conhecer uma segunda ferramenta para realizar o trabalho. Por outro lado, o *XML Schema*, apesar de ser mais complexo, conhecendo-se a linguagem XML, é possível facilmente escrevê-lo para a definição de outros documentos XML.

Outra diferença, e vantagem do *XML Schema* em relação à DTD, é que permite a utilização de namespaces, ou seja, espaços identificadores que determinam o contexto dos tipos de elementos e nomes de atributos usados nos documentos XML. E por último pode-se destacar a flexibilidade do *XML Schema*, pois este além dos tipos de dados já existentes permite a criação de novos tipos. Por causa desses aspectos, e das demais características descritas na Seção 2.3, o foco desse trabalho será em documentos *XML Schema*.

Tabela 2-2: Principais Diferenças entre DOM e JDOM

DOM	**JDOM**	**Significado**
Attr	*Attribute*	Atributo
CDATASection	*CharacterData*	Tipo que descreve uma seqüência de caracteres
Comment	*Comment*	Comentários
Document	*Document*	Documento XML
DocumentType	*DocType*	Documento DTD
DocumentFragment	--	Parte do documento
DOMImplementation	--	Objeto que fornece acesso aos métodos
Element	*Element*	Elemento – Nó
Entity	*Entity*	Conteúdo reutilizável
EntityReference	--	Referência ao conteúdo
NamedNodeMap	--	Coleção de objetos Node
Node	--	Nó
NodeList	--	Lista de Nós
Notation	--	Declaração do formato para conteúdo que não deve ser analisado em termos de sintaxe
ProcessingInstruction	*ProcessingInstruction*	Instruções de Processamento
Text	*java.lang.String*	Tipo String
--	*Verifier*	
DOMException	*JDOMException, IllegalAddException, IllegalDataException, IllegalNameException., IllegalTargetException*	Tipos de exceções

Para implementar a ferramenta XTM descrita no Capítulo 5, foi utilizada a biblioteca JDOM para a manipulação dos documentos XML *e XML Schema* pois, além de ser uma extensão do DOM e possuir todas as características desta API, é uma biblioteca específica para a linguagem JAVA, a qual foi escolhida para o desenvolvimento da ferramenta. Além disso, JDOM possibilita *parsing* dos esquemas mutantes gerados o que facilitou a implementação da XTM pois, é necessário que os mutantes sejam documentos válidos e bem-formados e esta verificação é feita pela própria API através de alguns métodos de manipulação existentes.

3. TESTE DE SOFTWARE

"A atividade de teste de software é um elemento crítico da garantia de qualidade de software e representa a última revisão de especificação, projeto e codificação" [PRE05].

O principal objetivo do teste é encontrar erros. Para um programa em teste, são gerados diversos casos de testes que são compostos de dados de entrada mais saída esperada. Um caso de teste bem sucedido é aquele que consegue revelar um defeito ainda não descoberto [MYE79].

Um processo de teste envolve planejar, projetar, executar casos de teste e avaliar os resultados obtidos. A etapa de elaboração (projeto) de casos de testes é uma das mais difíceis de realizar pois dela depende a qualidade dos dados gerados. Nesta etapa escolhe-se a técnica de teste a ser utilizada.

Depois de escolhida uma técnica, devem-se considerar os critérios a serem aplicados. Um critério de teste estabelece requisitos a serem satisfeitos pelos casos de teste e pode auxiliar na seleção de dados e na avaliação de sua qualidade. Satisfazer um critério significa atender aos requisitos estabelecidos por ele [RAP85].

Para o projeto de casos de testes utilizam-se três técnicas principais, cada uma com critérios de teste associados:

- Baseada em Defeitos: os dados de teste são gerados considerando os principais defeitos mais cometidos pelos desenvolvedores. Por exemplo, o critério Análise de Mutantes [DEM78];
- Funcional: os dados de teste são gerados considerando os requisitos funcionais do sistema. Por exemplo, os critérios Análise do Valor Limite, Grafo de Causa Efeito [PRE05];
- Estrutural: Os dados de teste são gerados considerando a estrutura interna do programa em teste. Por exemplo, critérios baseados em fluxo de controle [PRE95, RAP82] e fluxo de dados [MAL91, RAP82];

A medida do porcentual dos elementos requeridos (elementos que devem ser exercitados) que foram efetivamente exercitados em relação ao número total de elementos requeridos, representa a cobertura atingida pelo critério [RAP82].

A escolha de um critério depende: a) do custo para sua execução, que avalia o número de casos de teste que será necessário criar para satisfazê-lo; b) a eficácia do critério, ou seja, a capacidade de revelar um número maior de defeitos; e, c) a dificuldade de satisfação, que indica a probabilidade de satisfazer um critério tendo satisfeito outro. Estudos da literatura têm mostrado que o critério análise de mutantes é o mais eficaz em revelar defeitos, porém o mais custoso [WON94].

O critério análise de mutantes foi introduzido por DeMillo [DEM78]. A Análise de Mutantes tem por finalidade introduzir pequenos defeitos em um programa em teste através de operadores de mutação e gerar novos programas chamados de mutantes. Ela baseia-se em dois pressupostos [DEM78]:

- Hipótese do programador competente: os programadores fazem programas bem próximo do correto;
- Efeito de acoplamento: um defeito complexo é revelado, revelando-se defeitos simples.

Pequenas modificações são introduzidas no programa através da aplicação de operadores de mutação (operadores que modificam alguma parte do código) que geram novos programas chamados mutantes. São utilizados casos de testes na execução destes mutantes para distingui-los do programa original.

Um mutante é dito “morto” quando um caso de teste conseguir fazer a distinção entre o programa mutante e o programa original gerando saídas diferentes. Se a saída do programa original for considerada correta, então este estará livre do possível defeito descrito pelo programa mutante. Caso contrário, um defeito é descoberto e o programa deverá ser corrigido. O critério Análise de Mutantes exige que todos os mutantes sejam mortos [DEM78].

Os casos de teste que matam os mutantes são classificados como eficientes. Caso após a execução de todos os casos de testes, ainda existam mutantes que gerem a mesma saída do programa original e se não for possível gerar um caso de teste cuja saída diferencie o programa original do programa mutante, os mutantes são considerados equivalentes ao programa original. A

determinação de um mutante equivalente é sempre realizada pelo usuário, pois determinar se dois programas computam a mesma função é uma questão indecidível [BUD81].

Um grau de cobertura é definido para o programa, ou seja, o *score* de mutação (S) para o conjunto de casos de testes (T) utilizados. Este *score* é definido como:

$$S(T) = Mm / (Mg - Me), \text{ onde:}$$

- Mm é o número de mutantes mortos;
- Mg é o número de mutantes gerados
- Me é o número de mutantes equivalentes

O *score* de mutação permite avaliar a adequação dos casos de testes usados e, como conseqüência, a confiança no programa testado, pois o *score* de mutação relaciona o número de mutantes mortos com o número de mutantes gerados não equivalentes. O *score* de mutação varia no intervalo entre 0 e 1 sendo que quanto maior o *score* mais adequado é o conjunto de casos de teste para o programa que está sendo testado [DEM78]

Atualmente existem ferramentas para geração de mutantes tais como a Proteum [13] que é utilizada para geração de mutantes em teste de programas desenvolvidos na linguagem C.

3.1. Trabalhos Relacionados

A utilização do teste baseado em defeitos e do teste de mutantes para a tecnologia XML tem sido alvo de pesquisas. Foram encontrados na literatura alguns artigos relacionados ao tema e que serão apresentados nas próximas seções. Estes trabalhos foram divididoss entre: testes em componentes *Web* e testes em documentos *XML Schema*.

3.1.1. Testes em Componentes Web

Lee e Offutt [LEE01] propuseram a utilização do teste de mutação para aplicações *Web*. Baseado nos documentos XML que a aplicação possui são propostos dois operadores de mutação para a geração de documentos XML mutantes. Os documentos mutantes são utilizados como casos de teste na interação entre componentes *Web* (mutação de interações).

Os operadores são:

- *LenOf*: altera o número de caracteres do conteúdo de um determinado elemento.
- *MemberOf*: Tendo-se T e T' dois documentos XML, os dados do documento T' devem existir no documento T, ou seja, estas informações devem fazer parte de T.

Os operadores criados por Lee e Offutt apenas alteram os conteúdos contidos nas *tags* dos documentos XML pois, como mencionado anteriormente, só é possível uma alteração dos conteúdos dos documentos e não na sintaxe do mesmo.

O operador *LenOf* altera o tamanho do conteúdo acrescentando ou removendo algum caractere. A Tabela 3-1 mostra alguns exemplos da funcionalidade deste operador.

Tabela 3-1: Exemplo do Operador *LenOf*

Tag Original	Tag Alterada	Operador
`<user>Ledyvânia</user>`	`<user>Ledy</user>`	Caracteres removidos
`<user>Ledyvânia</user>`	`<user>LedyvâniaXXX</user>`	Caracteres incluídos

O operador *MemberOf* valida se o conteúdo da *tag* está presente em outro documento XML que é utilizado para validações, como por exemplo, validar *login* e senha. Estes dados estariam presentes em um documento D que seria utilizado pela aplicação para consultar se os dados vindos no documento de troca de informação da aplicação estão corretos. A Tabela 3-2 mostra um exemplo deste operador.

Existe um arquivo de validação de dados (documento A) contendo todos os dados permitidos pela aplicação na troca de mensagens através de um documento XML B. Este documento tem uma de suas tags alteradas, baseando-se na ausência de valores do documento A, gerando, desta maneira, outro documento XML C. Este documento C é então utilizado para a trocar de mensagens entre as aplicações.

Como visto, estes operadores criados por Lee e Offutt apresentam uma grande limitação no testes dos documentos XML por limitar o teste ao conteúdo dos elementos.

Tabela 3-2: Exemplo do Operador *MemberOf*

Arquivo de Validação (documento A)	**Tag Original (trecho do Documento B)**	**Tag Alterada (trecho do Documento C)**

`<users>`
`<user>Ledyvânia</user>`
`</user>`
	`<users>`
	`<user>Ledyvânia</user>`
	`</user>`
		`<users>`
		`<user>Maria</user>`
		`<users>`

Em outro trabalho, Offutt e Xu [OFF04] realizam o teste de integração entre componentes de software via *Web Service* usando perturbação de dados. A mensagem de requisição é modificada e enviada uma nova mensagem, a mensagem modificada. Eles utilizaram dois tipos de perturbação: de dados e de interação.

Na perturbação de dados os valores dos dados são modificados baseando-se na idéia de teste de valores limites. Os valores utilizados são os limites especificados pelo documento *XML Schema* [OFF04].

A perturbação de interação está subdivida em outros dois tipos: RPC (*Remote Procedure Call*) e comunicação de dados. Na perturbação RPC os valores das *tags* são alterados. Quatro tipos de operadores numéricos foram definidos baseados na seguinte representação da árvore do XML esquema [OFF04]:

$T = (N, D, X, E, n_r)$, onde:

- N é o conjunto de elementos e atributos do nó;
- D é o conjunto de tipos de dados;

- X é o conjunto de restrições;
- E é o conjunto de ligações entre os nós;
- n_r é o nó raiz;

Considerando n $\in$ N, os seguintes operadores foram definidos:

- *Divide* (n): onde o valor n é alterado por 1 / n;
- *Multiply*(n): onde o valor n é alterado por n x n;
- *Negative* (n): onde o valor n é alterado por –n;
- *Absolute* (n): onde o valor n é alterado por |n|;

Mediante a possibilidade de existir comandos SQL no próprio documento XML para serem executados direto no banco de dados, Offutt definiu outros dois operadores [OFF04]:

- *Exchange* (n1, n2): quando n1 e n2 são elementos do mesmo tipo troca-se o valor n1 por n2 e vice versa;
- *Unauthorized* (str): através do uso de SQL *injection* o valor de um *string* str é alterado para str ‘ OR ‘ 1 ‘ = ‘ 1.

Estes operadores, assim como em [LEE01], alteram apenas o conteúdo das *tags*. Porém a diferença é que no caso dos operadores numéricos, consideram-se os valores especificados pelo esquema utilizado.

Outro tipo de perturbação utilizada foi perturbação na comunicação de dados. Este tem por objetivo testar os relacionamentos e restrições entre os elementos pai e filho tais como a cardinalidade entre eles [OFF04].

Xu et al [XU05] propõem alguns operadores de mutação para perturbação de esquemas, utilizando o *XML Schema.* Estes operadores têm a finalidade de perturbar os esquemas com o propósito de criar mensagens XML incorretas para testar o comportamento dos *Web Services* [XU05].

São propostos 4 (quatro) operadores primitivos e 3 (três) operadores adicionais não-primitivos para auxiliar os outros operadores primitivos criados [XU05]. Estes operadores são definidos na Tabela 3-3. Para melhor compreensão, um documento *XML Schema* foi definido com uma árvore [XU05] para a qual se utiliza a mesma notação de [OFF04].

Nenhum dos operadores definidos em [XU05] alteram o nó raiz do esquema, apenas inserem ou removem nós ou subárvores.

Com base nestes operadores, Xu et al [XU05] definiram três critérios de cobertura de testes:

- *Delete Coverage (DC)*: critério para remoção de nós
- *Insert Coverage (IC)*: critério para inserção de nós
- *Constraint Coverage (CC)*: critério para inversão de restrições;

Tabela 3-3: Descrição dos Operadores Definidos por Xu et al [XU05]

Operador	**Descrição**	**Exemplo**
insertN (e, e'$_p$, e'$_c$, n')	Insere um novo nó entre 2 (dois) nós já existentes.	Figura 3-1 A
DeleteN (n)	Remove um nó da árvore e a aresta que o liga com o nó pai	Figura 3-1 B
insertND (np, e'$_p$, n', e'$_c$, d')	Insere um novo nó *(n')* em que a definição de seu tipo será a do nó pai *(n_p)*	Figura 3-1 C
deleteND (n)	Remove da árvore um nó e a sua definição de tipo	Figura 3-1 D
insertT (n, e', T^a)	Insere uma subárvore	Figura 3-2 A
DeleteT (e)	Remove uma subárvore a partir da aresta *e*	Figura 3-2 B
changeE (e, e')	Inverte duas subárvores trocando duas arestas *(e, e')*	Figura 3-2 C

3.1.2. Teste de Mutação em documentos XML Schema

Testes em documento *XML Schema* tornam possível a validação semântica de documentos XML visto que os esquemas são usados para somente a validação sintática de documentos XML no lugar da DTD.

Para realizar o teste de esquemas, técnicas baseadas em defeito têm sido utilizadas [LI05, EME05].

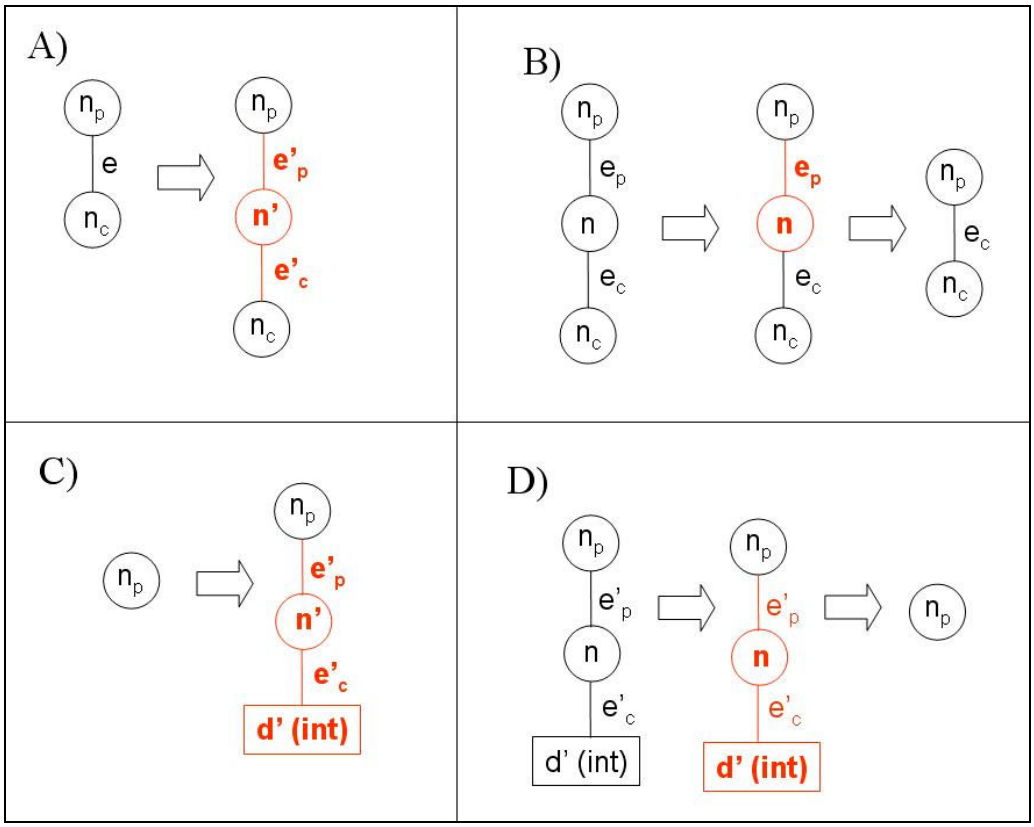

Figura 3-1: Operadores Primitivos

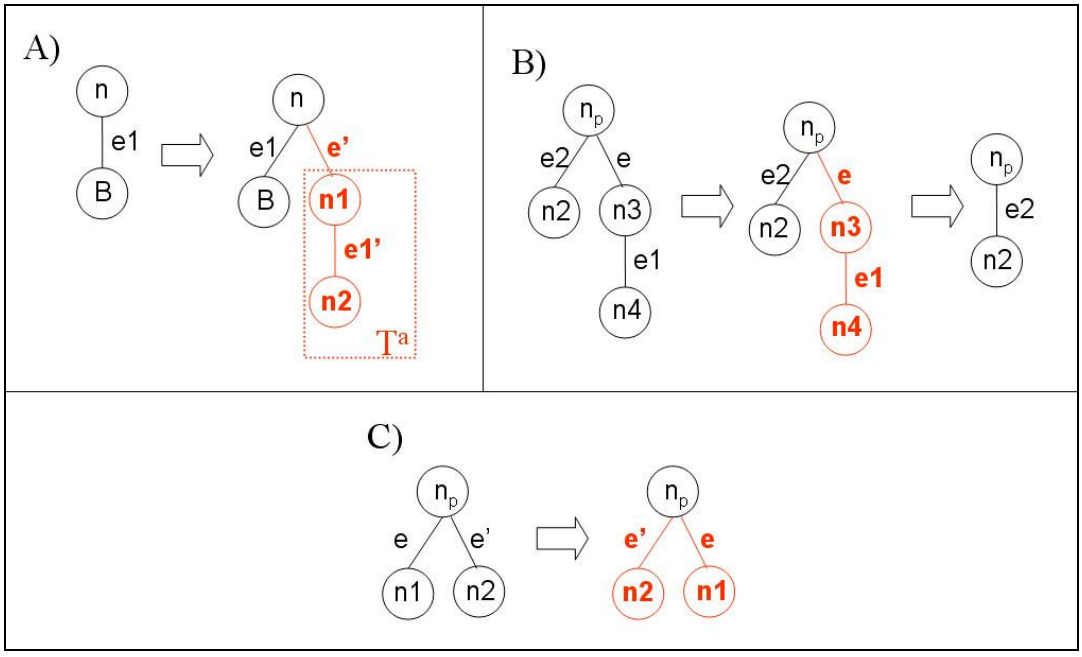

Figura 3-2: Operadores Não-Primitivos

Li e Miller [LI05] propuseram 18 (dezoito) operadores de mutação com base em características do *XML Schema* e dos elementos existentes. Cada operador altera um valor ou atributo conforme demonstra a Tabela 3-4 e a Tabela 3-5. Os exemplos de cada operador encontram-se no Anexo A.

Estes operadores foram criados com base nos erros mais comuns cometidos pelos desenvolvedores [LI05]. Após a geração dos mutantes algumas ferramentas foram utilizadas para validar os esquemas com o objetivo de mostrar que os esquemas mutantes, apesar de conterem defeitos, podiam ser validados.

Foram utilizadas as seguintes ferramentas: SQC (*Schema Quality Checker* [SQC05]), XSV (*XML Schema Validador* [XSV05]) e XMLSpy5[SPY05].

Neste trabalho Li e Miller [LI05] têm como objetivo revelar os defeitos inseridos nos esquemas usando para isso apenas alguns validadores de esquema aos quais os mutantes gerados são submetidos. Algumas questões não são abordadas no trabalho, tais como: Como aplicar o critério Análise de Mutantes? Como considerar um mutante morto? Qual caso de teste utilizar?

Outro trabalho que visa o teste de esquema é o de Emer et al [EME05]. O objetivo desta abordagem, baseada em defeitos, é revelar possíveis defeitos no *XML Schema* com relação à sua especificação.

Neste trabalho é utilizado um modelo formal para o documento *XML Schema* derivado de outros trabalhos publicados. Todo esquema é representado através de uma extensão da gramática regular de árvores (*Regular Tree Grammar* – RTG) [EME05].

Para auxiliar no processo de teste são propostas 3 (três) classes de defeitos:

- Elementos: defeitos relacionados à definição dos elementos. Subdivide-se em ordem incorreta, valores incorretos e ocorrências incorretas;
- Atributos: defeitos relacionados à definição dos atributos dos elementos. Subdivide-se em uso incorreto e valores incorretos;
- Restrições nos dados: defeitos associados às definições de restrições de dados que podem ser assumidos pelos elementos. Subdivide-se em tipo de dado incorreto, valores enumerados incorretos, valor mínimo e máximo incorreto, padrão incorreto, caracter branco incorreto, tamanho incorreto e dígitos incorretos;

Tabela 3-4: Lista de Operadores Definidos por Li e Miller

Sigla	***Nome***	***Descrição***
SNC	*XML Schema* Namespace Change	Altera o ano de referência no namespace do esquema
TDE	*Target and Default Exchange*	Inverte os valores dos atributos *namespace target* e *default*
ENR	*Element Namespace Replacement*	Altera o identificador do *namespace* para outro valor aleatório
ENM	*Element Namespace Removal*	Remove o elemento prefixado do identificador do *namespace*
QGR	*Qualification Globally Replacement*	Altera a definição da qualificação dos elementos locais e atributos das tags *elementFormDefault* e *attributeFormDefault*
QIR	*Qualification Individually Replacement*	Semelhante ao QGR, porém altera as definições de qualificação do atributo *form* nas tags *attribute*
CCR	*ComplexType Compositors Replacement*	Altera a definição da composição dos tipos complexos
COC	*ComplexType Order Change*	Altera a ordem de elementos dentro de uma composição do tipo *sequence*
CDD	*ComplexType Definition Decrease*	Remove um elemento de uma composição de qualquer tipo
EOC	*Element Occurence Change*	Altera a restrição de ocorrências mínimas e/ou máximas de um elemento

Tabela 3-5: Continuação da Lista de Operadores Definidos por Li e Miller

AOC	*Attribute Occurence Change*	Altera a restrição de tipo de ocorrência de um atributo
SPC	*SimpleType Pattern Change*	Altera as expressões regulares declaradas para uma seqüência exata de caracteres aceitáveis
RAR	*Restriction Arguments Replacement*	Altera os valores e os tipos de restrições para valores numéricos
EEC	*Enumeration Element Change*	Inclui ou remove um elemento de uma restrição para conjuntos enumerados
SLC	*String Length Change*	Altera o tamanho (máximo e/ou mínimo) permitido de uma string
NCC	*Number Constraint Change*	Altera o tamanho de um número decimal nas restrições de representação decimal
DTC	*Derived Type Control Replacement*	Altera o tipo de derivações de controle (*extension*/*restriction*)
UCR	*Use of Controller Replacement*	Altera o modificador de controle usado nas instâncias

O processo de teste proposto inicia-se na construção do modelo RTG do esquema, e baseado neste modelo as classes de defeitos são geradas. Estas classes são usadas para inserir as modificações em documentos XML válidos gerando assim documentos mutantes. Cada elemento e atributo do esquema é associado a uma das classes de defeito. Para cada associação identificada é gerada uma consulta que é então executada para cada documento XML mutante. Os resultados das consultas são confrontados com a especificação [EME05].

Nos exemplos utilizados como estudo de caso foram gerados documentos XML com modificações simples e para estes foram geradas as consultas. Os resultados obtidos revelaram defeitos nos esquemas em teste com relação às especificações.

Os defeitos revelados referem-se a ausência de restrições na definição de alguns elementos, tais como, tamanho e padrões, valores limites e número de ocorrências dos elementos.

Esta abordagem, como descrito por Emer et al [EME05], visa garantir a qualidade do dado em documentos XML ”através de um processo de testes que utiliza documentos XML mutantes e *XQuery* (linguagem de consulta para documentos XML) para revelar os defeitos mais comuns”.

3.2. Considerações Finais

A atividade de teste é uma das principais maneiras de garantir a qualidade de um software. Com base no crescente número de aplicações *Web* existentes e com a popularização das tecnologias XML é importante estender os critérios de teste para que eles possam também ser aplicados nesse contexto.

Neste capítulo foram apresentados 5 (cinco) trabalhos que aplicam técnicas baseadas em defeito. Os trabalhos [LEE01, OFF04, XU05] visam testar o comportamento dos *Web Services* com relação ao uso de documentos XML incorretos na comunicação entre as aplicações.

Em [LEE01] é proposta a alteração do próprio documento XML. Nos demais, os documentos são criados de acordo com esquemas alterados usando-se a técnica de perturbação de dados. Estes defeitos inseridos geram mutantes divergentes da especificação do esquema.

Como visto, nenhum dos trabalhos realmente aplica a análise de mutantes para testar esquemas. Em [OFF04] o objetivo é testar a interação entre componentes *Web* utilizando-se documentos XML gerados a partir de esquemas mutantes. Em [LI05] os mutantes são apenas validados com alguma ferramenta de validação, porém um processo para diferenciar os esquemas mutantes válidos do original não foi definido.

Em [EME05] embora seja apresentado um processo para validar esquemas este não gera esquemas mutantes e sim documentos XML mutantes a partir do esquema em teste.

Com base nos estudos feitos e nos trabalhos relacionados apresentados neste capitulo, propõem-se a utilização da técnica de análise de mutantes para validar documentos *XML Schema* e no próximo capítulo é introduzido um conjunto de operadores de mutação.

4. OPERADORES DE MUTAÇÃO PARA DOCUMENTOS *XML SCHEMA*

O objetivo deste trabalho é realizar teste de documentos *XML Schema* aplicando o critério Análise de Mutantes. Para a aplicação deste critério, quatro passos são necessários:

1. estudar os principais erros cometidos ao se projetar esquemas através da análise dos documentos *XML Schema*;
2. definir operadores de mutação com base nos tipos de erros previstos;
3. definir um processo de aplicação dos operadores;
4. implementar uma ferramenta de suporte;

Neste capítulo são introduzidos os operadores de mutação para documentos *XML Schema*. Estes operadores foram definidos considerando os principais enganos cometidos ao se projetar esquemas e os defeitos que podem ser introduzidos neste processo. Os trabalhos apresentados no capítulo anterior foram tomados como base para a definição do conjunto de operadores propostos.

Para a aplicação dos operadores foi definido um processo que permite a geração de dados de teste e identificação de quais esquemas mutantes se comportam diferente do esquema em teste.

4.1. Contexto do Teste

A escolha do padrão *XML Schema* para geração de mutantes para testes deve-se principalmente ao vocabulário e a estrutura rica de seu documento, possibilitando desta maneira a criação de vários tipos de operadores de mutação. O objetivo é validar a semântica de documentos XML utilizados por uma aplicação, uma vez que a validação sintática do mesmo já é feita através de um *parser* implementado, por exemplo, pela API DOM.

Os operadores implementados são utilizados na geração de mutantes para testes de documentos *XML Schema*. Cada operador insere algum tipo de defeito no esquema original.

Assim sendo, são gerados novos esquemas considerados mutantes os quais são testados através de casos de testes, que são na realidade documentos XML. A Figura 4-1 apresenta o processo proposto.

O testador fornece como entrada o esquema que será testado e o conjunto de casos de teste (documentos XML) a ser utilizado. Cada documento XML é submetido a um validador e confrontado sua validade com relação ao esquema sendo testado e com relação a cada mutante gerado.

Um esquema mutante será dito morto por um documento XML se for considerado válido de acordo com o esquema original e não o for pelo esquema mutante ou, ao contrário, se o documento for validado pelo esquema mutante e não o for pelo esquema original, ou seja, o esquema mutante se comporta diferentemente do esquema em teste.

No final, um *status* é fornecido para o mutante. O testador deverá definir esquemas equivalentes bem como avaliar os resultados obtidos, ou seja, quais casos de testes obtiveram as respostas esperadas revelando ou não a presença de defeitos no esquema.

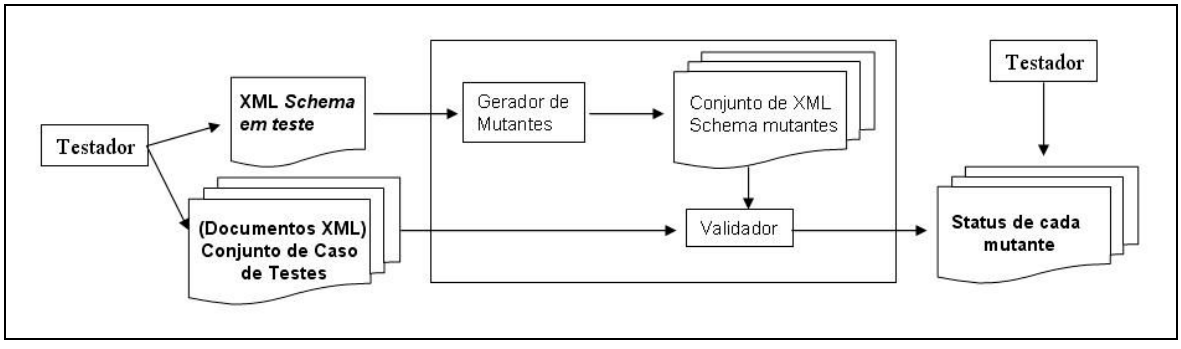

Figura 4-1: Processo Proposto

Os operadores foram divididos em dois grupos: manipulação de dados e de manipulação da estrutura. Para a representação dos operadores foi necessário definir uma representação formal para o documento *XML Schema*.

4.2. Representação dos operadores

Para uma representação mais formal será utilizada a seguinte notação para um documento *XML Schema*.

- T → árvore
- E → conjunto de Elementos
- D → conjunto de Dados
- N → conjunto de Nós
- A → conjunto de Atributos
- O → conjunto de Operadores
    - GO → Group Order
    - REQ → Required
    - DT → DataType
    - SO → SizeOccurs
    - CT → ChangeTag
    - CSP → ChangeSingPlural
    - LO → LengthOf
    - STE → SubTree_Exchange
    - IT → Insert_Tree
    - RT → Remove_Tree

4.3. Operadores de Manipulação de Dados

Estes operadores fazem as alterações referentes aos valores dos atributos e o nome das *tags* dos elementos na árvore. Neste grupo encontram-se os seguintes operadores:

- *Group Order* - GO
- *Required* - REQ
- *DataType* - DT
- *SizeOccurs* - SO: semelhante ao operador EOC [LI05])
- *ChangeTag* - CT: semelhante aos operadores CCR, DTC E RAR [LI05]

- *ChangeSingPlural* - CSP
- *LengthOf* - LO: semelhante ao operador SLC [LI05] e ao lengthOf definido por [LEE01]

E os seguintes operadores de Li e Miller [LI05] também fazem parte deste grupo: *SNC, QGR, AOC, QIR, CCR, UCR, RAR, NCC, DTC, SPC* e *TDE.*

A seguir são apresentados a definição e um exemplo prático dos operadores propostos neste trabalho para auxiliar na compreensão. São eles:

- **Group_Order (GO):** Altera a ordem na qual os elementos devem estar no grupo. O domínio do operador é *{sequence, one, many, choice}*
  Exemplo:
  <group order=”**ONE**”>
  Mutações possíveis:
  <group order = “**SEQ**”>;<group order= “**CHOICE**”>; <group order= “**MANY**”>

- **Required (REQ):** Altera o tipo da obrigatoriedade dos atributos. O domínio do operador é *{yes, no}*
  Exemplo:
  <attributetype name="bar" dt:type="int" required="**YES**"/>
  Mutações possível:
  <attributetype name="bar" dt:type="int" required="**NO**"/>

- **DataTypes (DT):** Altera o tipo dos elementos/atributos. O domínio do operador são todos os tipos definidos, por exemplo, {*integer, string, float, decimal, int, enumaration*}.
  Exemplo:
  <attributetype name="bar" dt:type="**INT**" required="yes"/>
  Mutações possíveis*:*
  <attributetype name = "bar" dt:type="**STRING**" required="yes"/>;
  <attributetype name="bar" dt:type="**FLOAT**" required="yes"/>

- **LenghtOf (LO):** Altera o tamanho do nome dos elementos.

Exemplo:
<element type="**A**"/>
Mutações possíveis:
<element type="**AXXX**"/>

- **ChangeSingPlural (CSP)**: Altera o tamanho do elemento adicionando ou removendo o caracter ‘s’ no final da string.
  Exemplo 1:
  <element name="**A**"/>
  Mutações possível:
  <element name="**AS**"/>
  Exemplo 1:
  <element type="**AS**"/>
  Mutações possíveis:
  <element type="**A**"/>

- **ChangeTag (CTP):** Altera as *tags* dos nós.
  Exemplo:
  <**attributetype** name="bar" dt:type="int" required="yes"/>
  Mutações possível:
  <**ELEMENTTYPE** name="bar" dt:type="int" required="yes"/>

- **SizeOccurs (SO):** Altera-se os tamanhos das ocorrências Máximas e/ou Mínimas tanto dos tipos fixos (*strings*) quanto dos tipos criados pelo usuários.
  Exemplo:
  <element type="B" minOccurs="**1**"/>
  Mutações possível:
  <element type="B" minOccurs="**0**"/>

O operador SO tem como objetivo também testar os limites superior e inferior dos valores dos atributos *minOccurs* e *maxOccurs* para testar o número de ocorrências permitidas para dado elemento do esquema.

4.4. Operadores de Manipulação de Estrutura

Foram criados 3 (três) operadores para geração de mutantes alterando-se a estrutura da árvore do esquema. Os exemplos citados para os operadores são baseados no esquema mostrado na Figura 4-2

```
<?xml version = "1.0"?>
<schema targetNS = "myschema.xsd"
   version = "1.0"
   xmlns = "http://www.w3.org/1999/XMLSchema">
<elementtype name="foo" content="eltonly" order="one">
   <attributetype name="bar" dt:type="int" required="yes"/>
   <group order="one">
     <element type="X">
     <element type="Y">
   </group>
   <group order="seq">
     <element type="A">
     <element type="B" minOccurs="1">
     <element type="C">
   </group>
</elementtype>
</schema>
```

Figura 4-2: Exemplo de um *XML Schema*

Neste segundo grupo estão os seguintes operadores os quais englobam alguns operadores definidos por [LI05] e [XU05]:

- *SubTree_Exchange* - STE: semelhante ao operador *changeE* [XU05] e COC [LI05].
- *Insert_Tree* - IT: semelhante aos operadores *insertN*, *insertND* [XU05] e EEC [LI05]
- *Remove_Tree* - DT*:* semelhante aos operadores *deleteN*, *deleteND* [XU05], CDD e EEC [LI05]

Para cada operador proposto para este grupo são apresentados sua definição e um exemplo prático para auxiliar na compreensão. São eles:

- *SubTree_Exchange*: Invertem as subárvores desde que os elementos que estão sendo invertidos sejam do mesmo tipo. A Figura 4-3 mostra dois exemplos práticos desde operador.
- *Insert_Tree*: Acrescenta um nó na estrutura da subárvore. A Figura 4-4 mostra um exemplo deste operador.

Exemplo 1: Elementos invertidos entre 2 nós distintos	Exemplo 2: Elementos invertidos dentro do próprio nó cuja order é SEQ
<group order="one"> <ELEMENT TYPE="A"> <element type="Y"> </group> <group order="seq"> <ELEMENT TYPE="X"> <element type="B" minOccurs="1"> <element type="C"> </group>	<group order="one"> <element type="X"> <element type="Y"> </group> <group order="seq"> <ELEMENT TYPE="B" MINOCCURS="1"> <ELEMENT TYPE="C"> <ELEMENT TYPE="A"> </group>

Figura 4-3: Exemplo do Operador SubTree_Exchange

```
<group order="one">
          <element type="X">
          <element type="Y">
          <ELEMENT TYPE="TESTE"> INSERIDO
</group>
<group order="seq">
          <ELEMENT TYPE="B" MINOCCURS="1">
          <ELEMENT TYPE="C">
          <ELEMENT TYPE="A">
</group>
```

Figura 4-4: Exemplo do Operador Insert_Tree

- *Delete_Tree*: Um dos nós ou subárvores é removido da estrutura da árvore. A Figura 4-5 mostra dois exemplos deste operador.

Exemplo 1: Remoção de um nó filho	Exemplo 2: Remoção de uma subarvore.
<group order="one"> <#ELEMENT TYPE="A"#> REMOVIDO <element type="Y"> </group> <group order="seq"> <ELEMENT TYPE="X"> <element type="B" minOccurs="1"> <element type="C"> </group>	<group order="one"> <element type="X"> <element type="Y"> </group> <#group order="seq"> REMOVIDO <ELEMENT TYPE="B" MINOCCURS="1"> <ELEMENT TYPE="C"> <ELEMENT TYPE="A"> </group#>

Figura 4-5: Exemplo do Operador Delete_Tree

4.5. Considerações Finais

Esse capítulo introduziu um conjunto de operadores de mutação e um processo de teste para permitir a aplicação do critério Análise de Mutantes em documentos *XML Schema*.

Foram introduzidos dois grupos de operadores. O primeiro grupo gera modificações em tipos de dados e atributos dos documentos. Esses operadores foram propostos considerando os trabalhos de Li e Miller [LI05], Offutt e Xu [OFF04] e Emer et al [EME05]. O segundo grupo de operadores modifica a estrutura da árvore do documento *XML Schema* são baseados no trabalho de Xu et al [XU05].

O processo de aplicação dos operadores considera documentos XML como casos de teste para diferenciar os esquemas mutantes gerados e obter cobertura.

Para avaliar a proposta desse capítulo, a ferramenta XTM foi implementada e é descrita no capítulo a seguir.

5. A FERRAMENTA XTM

XTM (Ferramenta para Teste de Documentos ***XML Schema*** Baseado em **T**este de **M**utação) é uma ferramenta que automatiza a geração de mutantes para um dado documento *XML Schema* em teste, implementando os operadores descritos no capítulo anterior. Foi completamente desenvolvida na linguagem JAVA utilizando-se a API JDOM para a leitura e manipulação dos esquemas. No decorrer da leitura da árvore os operadores são aplicados e uma cópia do documento é salva criando-se assim diversos esquemas mutantes que são usados para validar um conjunto de documentos XML usados como dados de teste.

Com o auxílio de uma ferramenta externa, *Stylus Studio* [STY05], alguns dados de teste, ou seja, documentos XML, foram criados para avaliar o conjunto de mutantes a fim de revelar defeitos no esquema em teste.

A Figura 5-1 mostra quais os passos do processo foram implementados pela XTM.

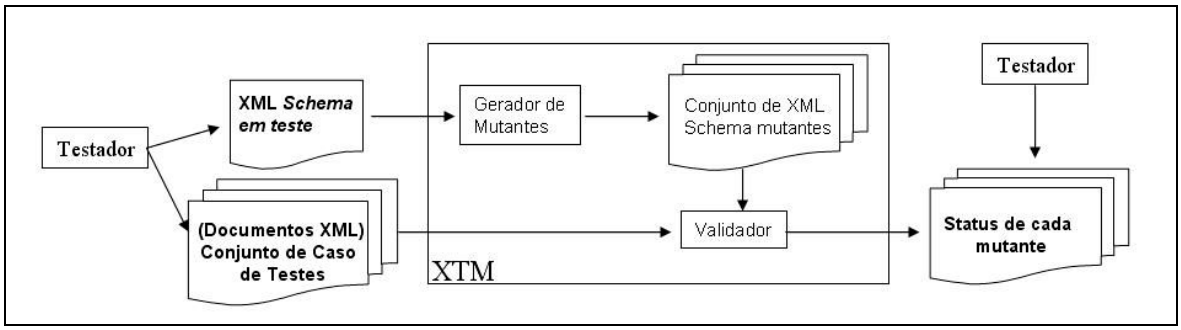

Figura 5-1: Ferramenta *XTM*

A ferramenta tem como entrada um *XML Schema*, o qual deve estar sintaticamente correto, ou seja, deve ser um documento bem formado, e um conjunto de dados de teste. A partir do esquema em teste são gerados diversos mutantes aplicando-se o conjunto de operadores de mutação previamente definidos no Capítulo 4: GO, REQ, DT, SO, CTP, CSP, LO, STE, IT, RT. Para

poder realizar comparações também foram implementados os seguintes operadores de Li e Miller [LI05]: SNC, QCR, AOC, QIR, CCR, UCR, SLC, RAR, NCC, DTC. Devido ao escopo mais abrangente dos operadores CCR, RAR e DTC, decidiu-se implementá-los no lugar do operador proposto CTP pois, pode-se constatar que estes operadores abrangiam o operador CTP em um escopo mais amplo.

Após a geração dos esquemas mutantes, estes são utilizados na tentativa de validação dos documentos XML informados como dados de teste. Conforme as tentativas de validação são executadas, e o *status* de cada mutante vai sendo informado. Este, conforme a saída gerada, pode ser considerado morto ou vivo. Dentre os mutantes que permanecem vivos, segundo análise do testador, alguns podem ser considerados esquemas equivalentes.

Neste processo, o testador avalia os dados de testes que geraram uma saída diferente do *XML Schema* original. Caso o mutante tenha gerado a saída correta, um defeito é identificado no esquema em teste.

Os principais algoritmos utilizados na implementação da XTM estão no Apêndice A. Um exemplo de utilização é apresentado na próxima seção.

5.1. Exemplo de utilização da ferramenta

A Figura 5-2 mostra um exemplo de execução da ferramenta para o grupo de operadores de manipulação de dados.

Além de gerar os mutantes, os dados de teste (documentos XML, neste caso o único documento aluno.xml), são validados com o esquema em teste (o esquema aluno.xsd), e é informado se este é valido ou não. No caso do exemplo o documento é valido. Após a validação desse caso de teste (aluno.xml) pelos esquemas mutantes gerados, os mutantes vivos são indicados. Ao final é informado o número total de mutantes criados por esse operador e o número total de mutantes vivos. Analogamente esse passo é conduzido para os outros operadores e dados de teste restantes para se obter o *score* de mutação.

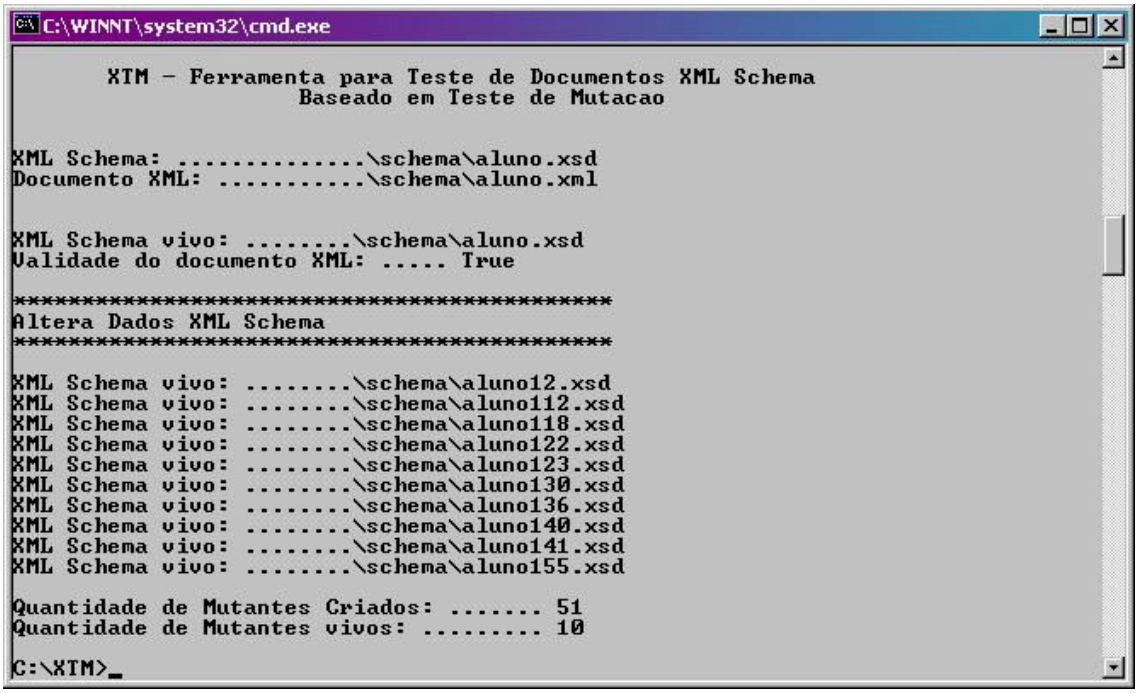

Figura 5-2: Exemplo do Funcionamento da Ferramenta XTM – Operadores de Dados

5.2. Considerações Finais

Neste capítulo foi descrita a ferramenta XTM, suas principais funcionalidades e um exemplo de uso. A XTM implementa os operadores de mutação e o processo descritos no Capítulo 4. Ela gera documentos *XML Schema* mutantes para um dado esquema em teste e permite que um dado de teste (documento XML) seja ou não validado pelos esquemas mutantes gerados. A validação ou não de um documento XML é utilizada para descobrir defeitos no esquema.

A implementação da ferramenta XTM permite a aplicação do teste de mutação em esquemas e a condução de experimentos para a avaliação dos operadores propostos. O próximo capítulo descreve tais experimentos.

6. EXPERIMENTOS

Com o objetivo de avaliar os operadores propostos e validar a implementação da XTM, bem como comparar os operadores e a abordagem deste trabalho com outros trabalhos, foram realizados dois experimentos. Neste capítulo, os experimentos são descritos. São apresentados seus objetivos, metodologia adotada, resultados e análise dos mesmos. Para realizar os experimentos foram utilizados os seguintes esquemas: aluno.xsd, disciplina.xsd, obras.xsd, usuário.xsd, sih.xsd e sia.xsd

Estes esquemas referem-se a 3 (três) diferentes sistemas: sistema de matricula (aluno.xsd e disciplina.xsd), sistema de biblioteca (obras.xsd e usuarios.xsd) e sistema hospitalar (sia.xsd, sih.xsd). As especificações desses esquemas estão no Anexo B.

Os dois primeiros sistemas são os mesmos utilizados por Emer et al [EME05] e foram escritos por alunos da disciplina de Engenharia de Software da Universidade Federal do Paraná - UFPR. O último pertence a uma aplicação real desenvolvida para um hospital. Todos foram utilizados para avaliar a eficácia dos operadores. Os dois primeiros sistemas, utilizados por Emer et al [EME05], permitiu uma comparação com a abordagem desses autores. O último sistema não possuía defeitos conhecidos em seus esquemas e, para que esses esquemas pudessem ser também utilizados na análise da eficácia, alguns defeitos foram aleatoriamente semeados pelos seus desenvolvedores.

6.1. Experimento 1

Objetivo

Este experimento teve como objetivo mostrar que as ferramentas de validação de esquemas, geralmente utilizadas, tais como a ferramenta da W3C por exemplo, não são capazes de identificar os defeitos introduzidos pelos

operadores de mutação, neste caso, os operadores propostos neste trabalho. Esse experimento também foi realizado por Li e Miller [LI05] para avaliar seus operadores. A mesma metodologia foi seguida e permitiu comparações entre os conjuntos de operadores definidos.

Ferramentas utilizadas

Para validar os esquemas foram utilizadas duas das mesmas ferramentas utilizadas no trabalho de Li e Miller [LI05]: Validador da W3C [XSV05] e *XMLSpy* [SPY05]. Além dessas, também foi utilizada outra ferramenta, o *Stylus Studio* 6 [STY05].

Para este experimento algumas observações referentes às validações efetuadas pelas ferramentas são feitas, pois isto influencia diretamente a quantidade de mutantes válidos gerados pelos operadores de mutação propostos tanto por Li e Miller [LI05] quanto os propostos neste trabalho:

- O atributo *minOccurs* não pode ter ocorrências iguais a 0 (zero) quando o *group order* for do tipo *All*;
- Os tipos de dados não podem ser declarados diferente de, por exemplo, ***xs**:string;*
- Quando existem referências a outros nós, estas referências são validadas;
- O atributo *minOccurs* não pode ter valores negativos (válido somente para o validador da W3C);

Descrição do Experimento

Para cada esquema em teste foram realizados os seguintes passos:

1. Gerar mutantes utilizando todos os operadores implementados pela XTM.
2. Validar sintaticamente os esquemas mutantes utilizando as ferramentas da seção anterior

Resultados

A Tabela 6-1 mostra o número total de mutantes gerados por esquemas para todos os operadores de mutação implementados. Este total não considera a validade dos documentos. Nesta tabela é detalhado o número de mutantes gerados para o grupo de operadores de dado, sendo que, os 11 (onze) primeiros

operadores são os propostos por Li e Miller [LI05] e os demais foram propostos neste trabalho porém, os operadores de Li e Miller que não foram implementados são similares a estes operadores. O total de mutantes gerados também é apresentado na tabela.

Tabela 6-1: Mutantes Gerados por Operador de Mutação

Oper.	***XML Schema***						**Total**	**Total**
	Aluno	**Disciplina**	**Usuario**	**Obras**	**Sih**	**Sia**	**Válidos**	**Gerado**
1. SNC	1	1	1	1	1	1	6	0
2. QCR	1	1	1	1	1	1	6	6
3. AOC	0	0	0	0	0	0	0	0
4. QIR	0	0	0	0	0	0	0	0
5. CCR	4	4	6	4	16	16	50	50
6. UCR	0	0	0	0	0	0	0	0
7. SLC	0	0	0	0	0	0	0	0
8. RAR	0	0	0	0	3	3	6	6
9. NCC	0	0	0	0	0	0	0	0
10. DTC	0	0	0	0	1	1	2	2
11. SPC	0	0	0	0	0	0	0	0
12. GO	0	0	0	0	0	0	0	0
13. CSP	10	10	11	8	36	34	109	107
14. REQ	0	0	0	0	0	0	0	0
15. DT	12	12	10	8	52	48	142	139
16. SO	23	20	32	25	80	75	255	218
17. LO	10	10	11	8	35	36	110	110
18. IT	22	28	28	20	80	83	261	62
19. STE	28	28	28	15	77	96	271	268
20. RT	4	15	15	12	54	57	157	153
Total	115	129	143	102	436	451	1376	1021

A Tabela 6-2, a Tabela 6-3 e a Tabela 6-4 apresentam o número de mutantes gerados não validados pelo número de mutantes validados. Foram utilizadas respectivamente as ferramentas W3C [XSV05], XMLSpy [SPY05] e *Stylus Studio* [STY05].

Nessas tabelas os operadores estão agrupados da seguinte forma para facilitar a compreensão e análise dos resultados: Dados (operadores do grupo de Dados), STE (SubTree_Exchange), IT (InsertTree) e RT (RemoveTree). Também são apresentados o número e a porcentagem de mutantes válidos gerados para cada esquema e para cada grupo de operadores.

Análise dos resultados

Considerando o número total de mutantes gerados, conforme a Tabela 6-1, pode-se considerar que, embora alguns mutantes inválidos tenham sido gerados, a grande maioria pode ser validada conforme mostra resumidamente a Tabela 6-5 feita com base nas demais tabelas apresentadas.

Tabela 6-2: Esquemas Mutantes Não Válidos pelo Número de Mutantes Válidos – W3C

XML Schema	**W3C**				**Total de Válidos**	**% mutantes válidos**
	Dados	**STE**	**IT**	**RT**		
Aluno.xsd	26/35	0/28	19/3	0/4	70	66,66
Disciplina.xsd	16/42	0/28	24/4	0/15	89	74,79
Obras.xsd	13/59	0/28	24/4	0/15	106	80,30
Usuário.xsd	11/48	0/11	17/3	0/12	74	78,72
Sia.xsd	68/146	0/77	70/10	2/52	285	68,67
Sih.xsd	85/139	0/96	71/12	2/55	304	75,80
Total de Válidos	471	268	36	153	928	-
% Válidos	81,62	100	13,79	97,45	-	74,15

Conforme demonstra a Tabela 6-5, com exceção dos operadores de inserção de nós (*InsertTree)*, todos os demais geraram pelo menos mais de 60% de mutantes válidos. Portanto, nem todos os defeitos inseridos representados por

esses operadores são revelados apenas utilizando-se *parsers* na validação dos esquemas mutantes, muitos destes possíveis defeitos geraram mutantes válidos segundo o experimento feito.

Tabela 6-3: Esquemas Mutantes Não Válidos pelo Número de Mutantes Válidos – XMLSpy

XML Schema	**XMLSpy**				**Total de Válidos**	**% mutantes válidos**
	Dados	**STE**	**IT**	**RT**		
Aluno.xsd	21/40	0/28	19/3	0/4	75	71,42
Disciplina.xsd	16/42	0/28	24/4	0/15	89	74,79
Obras.xsd	29/40	0/28	24/4	0/15	90	68,18
Usuário.xsd	23/36	0/11	17/3	0/12	62	65,96
Sia.xsd	64/150	0/77	70/10	2/52	289	69,63
Sih.xsd	89/137	0/96	71/12	2/55	300	74,81
Total de Válidos	448	268	36	153	905	-
% Válidos	77,64	100	13,79	97,45	-	70,80

Tabela 6-4: Esquemas Mutantes Não Válidos pelo Número de Mutantes Válidos – Stylus Studio

XML Schema	**Stylus Studio**				**Total de Válidos**	**% mutantes válidos**
	Dados	**STE**	**IT**	**RT**		
Aluno.xsd	26/35	0/28	19/3	0/4	70	66,63
Disciplina.xsd	23/35	0/28	24/4	0/15	82	68,90
Obras.xsd	30/42	0/28	24/4	0/15	89	67,42
Usuário.xsd	24/35	0/11	17/3	0/12	59	62,76
Sia.xsd	89/125	0/77	70/10	2/52	264	70,55
Sih.xsd	99/127	0/96	71/12	2/55	290	72,32
Total de Válidos	399	268	36	153	856	-
% Válidos	81,58	100	13,79	97,45	-	68,10

Apenas o operador de inserção de novos nós apresentou uma grande quantidade de mutantes inválidos, porém, isto se deve ao fato do tipo de nó

inserido não levar em consideração o contexto no qual ele está sendo aplicado, gerando, desta maneira, diversos mutantes inválidos. Para se evitar isso novos experimentos deverão ser conduzidos e uma possível redefinição desse operador será realizada.

Tabela 6-5: Porcentagem Média Total de Mutantes Validados

XML Schema	**% Média Mutantes válidos**				
	Dados	**SBT**	**IT**	**RT**	**Total**
Aluno.xsd	60.10	100	32,14	100	68,24
Disciplina.xsd	68.39	100	14,29	100	72,83
Obras.xsd	68.12	100	14,29	100	71,97
Usuarios.xsd	67.23	100	15	100	69,15
Sia.xsd	65.57	100	12,50	96.3	69,62
Sih.xsd	59.43	100	14,46	96,49	74,62
Total	64.80	100	17,11	98,8	71,02

Como visto anteriormente na Tabela 6-1 a maioria dos mutantes foram gerados com os operadores de mutação propostos neste trabalho. Os operadores de Li e Miller [LI05] geraram poucos mutantes. Não foi possível aplicar alguns operadores em nenhum dos esquemas utilizados, pois estes não atendiam as especificações para aplicação dos operadores visto que se tratavam de exemplos simples.

Quanto mais complexo for o esquema a ser utilizado mais abrangente será a aplicação dos operadores. Muitos operadores propostos por Li e Miller [LI05] requerem estruturas complexas tais como herança. Estas não estavam presentes nos esquemas desse experimento. Uma melhor análise deve ser efetuada para verificar o impacto desses operadores e avaliar se essas estruturas são pouco utilizadas pelos projetistas. Esses operadores podem ser desabilitados pelo testador em tais casos, se desejado.

6.2. Experimento 2

Objetivo

O objetivo dos operadores propostos neste trabalho é descrever defeitos relacionados à semântica e não apenas a defeitos sintáticos. Como apontado pelos resultados do Experimento 1, os defeitos não são revelados apenas com validadores. Para revelar os defeitos descritos pelos operadores é necessário aplicar o teste de mutantes.

O Experimento 2 teve como objetivo a aplicação dos operadores propostos e verificar quantos dados de teste são necessários por operador para identificar todos os mutantes gerados, qual o número de esquemas equivalentes gerados, e o número de defeitos revelados. Este experimento permitiu que uma comparação com o trabalho de Emer et al [EME05] fosse realizada. Para a comparação, apenas os esquemas utilizados no referido trabalho foram considerados: aluno.xsd, disciplina.xsd, obras.xsd e usuario.xsd.

Passos

Para cada esquema foram considerados todos os esquemas mutantes válidos gerados pela XTM no passo anterior. Foi utilizado na ferramenta XTM um mecanismo de validação utilizando-se a API SAX o qual descarta mutantes não válidos para validar de documentos XML. Este mecanismo é semelhante às validações feitas pelo validador da W3C.

Os seguintes passos foram realizados:

1. Para a realização deste experimento foi utilizado um conjunto inicial de dados de teste, isto é, documentos XML que estavam disponíveis para os esquemas descritos por seus desenvolvedores. Cada documento do conjunto de dados de teste foi submetido à validação de acordo com o esquema em teste e com cada esquema mutante gerado. O esquema mutante é considerado morto se ele produziu uma saída diferente da saída do esquema sendo testado.
2. Para os mutantes que permaneceram vivos após o Passo1, isto é, produziram as mesmas saídas que o esquema original para todos os documentos no conjunto de teste inicial. Nesse passo, documentos XML adicionais foram

gerados com o objetivo de mata-los. Nenhum esquema mutante equivalente ao original foi identificado, consequentemente, todos puderam ser mortos.

3. O *score* de mutação foi então calculado.

No Passo 2 é necessário avaliar o resultado de um mutante para considerá-lo morto. Ele será considerado morto se validou um documento XML e o esquema original não o fez, ou, caso contrário, um documento XML não pôde ser validado pelo mutante, mas o foi pelo esquema sendo testado. No caso de o resultado do mutante ser o correto, um defeito no esquema em teste foi revelado. Esse resultado foi utilizado para a análise da eficácia.

Resultados

Os resultados obtidos com este experimento são apresentados na Tabela 6-6. Está identificado para cada esquema o número de mutantes válidos usados no experimento, quantidade de mutantes equivalentes gerados, número de dados de testes distintos usados e o *score* de mutação obtido. A quantidade de defeitos revelados pela quantidade de defeitos conhecidos por esquema também é informada.

Tabela 6-6: Resultados Obtidos no Experimento 2

XML Schema	**Qtd mutantes válidos**	**Mutantes Equivalentes**	***Score* de Mutação**	**Qtd. Dados de Testes**	**Qtd. Defeitos Revelados / existentes**
Aluno.xsd	60	0	1	3	10/10
Disciplina.xsd	79	0	1	4	8/8
Obras.xsd	95	0	1	3	4/4
Usuário.xsd	66	0	1	3	9/9
Sia.xsd	250	0	1	4	0/0
Sih.xsd	268	0	1	4	1/1

No caso do sistema hospitalar a quantidade de defeitos conhecidos é a o número de defeitos semeados no esquema pelo desenvolvedor, e para os demais esquemas foi utilizada como base a quantidade de defeitos encontrados no trabalho de Emer et al [EME05].

Os tipos de defeitos revelados nos 4 (quatro) primeiros esquemas foram: tamanho, padrão e tipo de dado de alguns campo, como *login* e senha por exemplo, número de ocorrências e valores limites permitidos. O esquema Sia.xsd não possuía nenhum defeito com base em sua especificação. O esquema Sih.xsd teve o defeito referente ao valor limite permitido para um tipo de dado criado, inserido manualmente, revelado.

Análise dos Resultados

Para cada um dos sistemas utilizados os dados de testes foram gerados utilizando-se a ferramenta *Stulys Studio*.

Neste experimento, com a utilização de alguns dados de teste, foi possível detectar todos os defeitos (naturais e semeados) nos esquemas. Uma característica destes dados de teste utilizados foi o alto grau de acoplamento apresentado por todos. Ao tentar revelar um determinado erro outros erros também eram revelados, ou seja, o efeito de acoplamento pode ser constatado neste experimento.

Os operadores propostos mostraram-se mais eficazes no auxílio das detecções de defeitos do que apenas a utilização de ferramentas para validar os mutantes gerados podendo, desta maneira, identificar os defeitos nos esquemas usados no experimento.

Realizando uma comparação com a abordagem feita por Emer et al [EME05], neste experimento também foi possível revelar os mesmos defeitos revelados em sua abordagem para os esquemas em testes dos sistemas de matrícula e de biblioteca. Em seu experimento, para o documento Aluno.xsd foram gerados 22 consultas a partir de 22 documentos XML mutantes gerados. Para o mesmo documento, o experimento deste trabalho necessitou de 3 documentos XML para identificar todos os mutantes gerados.

Não houve a identificação de nenhum mutante equivalente para nenhum dos esquemas, todos os mutantes foram mortos. O número de documentos XML usados para revelar os defeitos foi pequeno, resultando em um baixo custo para a aplicação do critério Análise de Mutantes.

Pode-se constatar no segundo experimento realizado que os operadores propostos demonstraram mais eficácia na identificação de defeitos em esquemas

do que o primeiro experimento. Quase todos os defeitos conhecidos nos documentos *XML Schema* foram revelados usando os operadores propostos. Apenas 1 dos defeitos conhecidos que estava presente no esquema do sistema hospitalar (SIH.xsd) foi revelado usando um dos operadores propostos por Li e Miller [LI05], neste caso o operador RAR. Este operador não pertence ao escopo dos operadores definidos neste trabalho e, consequentemente não teria sido revelado pelos operadores propostos neste trabalho.

6.3. Considerações Finais

Os experimentos realizados demonstraram que para revelar defeitos em esquemas, não é suficiente apenas usar alguma ferramenta de validação. Muitos mutantes gerados através da aplicação dos operadores de mutação são válidos, necessitando assim, outra maneira de detectar os defeitos.

Com a utilização do processo definido, bem como dos operadores propostos, foi possível relevar todos os defeitos conhecidos nos esquemas em testes. Foram revelados os mesmos defeitos detectados pela abordagem proposta por Emer et al [EME05]. Os operadores propostos neste trabalho não revelaram apenas 1 defeito, revelado exclusivamente pelos operadores propostos por Li e Miller [LI05].

O experimento contribuiu para comprovar a aplicabilidade e eficácia dos operadores propostos, inclusive mostrando que esses consideram outros aspectos não considerados pelos operadores encontrados na literatura, como por exemplo os operadores propostos por Li e Miller.

7. CONCLUSÃO E TRABALHOS FUTUROS

Neste trabalho foi definido um processo de teste utilizado para testar documentos *XML Schema*. Este processo aplica o critério Análise de Mutantes. Documentos XML são utilizados para diferenciar o comportamento dos esquemas mutantes do comportamento do esquema original sendo testado.

Foi proposto um conjunto de operadores baseado em outros trabalhos e considerando os defeitos mais comuns presentes em documentos *XML Schema* e que influenciam a semântica de documentos XML.

Para auxiliar na geração automática dos mutantes foi desenvolvida uma ferramenta que implementa os operadores e o processo proposto chamada XTM - Ferramenta para Teste de Documentos ***XML Schema*** Baseado em **T**este de **M**utação

Nos experimentos realizados, para validar os operadores propostos e a implementação da XTM, foi possível detectar que a ferramenta gera uma grande quantidade de mutantes válidos. Destes, uma parte validou a sintaxe dos documentos XML usados como dados de teste.

Analisando-se os dados de testes distintos utilizados para matar os mutantes gerados, foi possível identificar que haviam defeitos nos esquemas usados, pois embora sintaticamente os documentos XML estivessem corretos sua semântica não correspondia ao resultado esperado, ou seja, o conteúdo contido nos documentos XML não possuía os valores esperados para algumas *tags*. Através dessa análise foi possível identificar que havia algum defeito no documento *XML Schema*.

Com a implementação da ferramenta XTM foi possível aplicar o critério Análise de Mutantes no teste de documentos XML esquemas para validar a semântica dos documentos XML e também a própria especificação do esquema.

A geração de mutantes é um processo custoso, pois é diretamente proporcional ao tamanho e a complexidade do esquema em teste. Quanto mais elementos e atributos, bem como especificações de tipos houver maior será o

número de mutantes que podem ser gerados e, consequentemente maior o número de dados de testes a serem utilizados para matá-los.

Embora a XTM auxilie na geração dos mutantes e detecção de defeitos tanto no esquema quanto na própria especificação dos mesmos, existe um problema com relação ao operador Insert. Dentre todos os mutantes gerados para este operador quase 100% deles são inválidos. Este operador considera apenas o nó do tipo <*element*></*element*> na inserção sem analisar o contexto no qual o nó está sendo acrescido. Assim, pode gerar inúmeros mutantes inválidos como demonstrou os experimentos realizados.

Apesar da desvantagem apresentada com esses operadores, pode-se concluir que, os operadores propostos e a XTM contribui no aumento de qualidade dos software que utilizam documentos XML e *XML Schema* em suas aplicações pois, é possível detectar defeitos associados à semântica dos documentos XML revelando-se os defeitos existentes quando comparados com sua especificação.

7.1. Trabalhos Futuros

Como trabalho futuro, é sugerido um aperfeiçoamento da implementação da ferramenta considerando-se 2 (dois) aspectos: torná-la gráfica e acrescentar a opção de escolher a porcentagem de mutantes para geração de mutantes para cada operador. Com essas melhorias a ferramenta se tornará mais fácil de ser utilizada pelos desenvolvedores.

É importante também que alguns operadores sejam redefinidos. O operador que realiza inserções de nós na estrutura do *XML Schema* deve ser melhorado considerando o contexto no qual está sendo realizada uma inserção. Este operador deve considerar apenas os elementos possíveis de serem aplicados na subárvore do *XML Schema* em que se encontra.

Além da melhora dos operadores existentes, outros operadores podem ser definidos principalmente para os grupos de operadores de dados para aumentar a abrangência do teste neste escopo.

Novos experimentos deverão ser conduzidos com outros conjuntos de esquemas para avaliar a XTM e os operadores propostos.

Esses operadores e os esquemas gerados também poderão ser utilizados em outros contextos tais como testar a comunicação entre componentes e *Web Services* analogamente aos trabalhos definidos no Capítulo 3.

REFERÊNCIAS

[BUD81] BUDD, T. A. **Mutation analysis: ideas, examples, problems and prospects, computer program testing**. USA: North-Holand Publishing, 1981.

[CLA04] CLABEN, Michel. **JDOM, The Java Dom**. Dezembro, 2000. www.link.com. Acesso em: Setembro, 2004.

[DEL93] DELAMARO, M. E. **Proteum – Um ambiente de teste baseado na análise de mutantes.** Dissertação de mestrado, ICMC/USP, São Carlos, SP. 1993.

[DEM78] DEMILLO, R. A; LIPTON, R. J. **Hints on test data selection: help for practicing programmer**. IEEE Computer, v. 11, n. 4, p. 34-41, Abril, 1978.

[DOM05] W3C. **Document Object Model (DOM).** Disponível em: http://www.w3.org/DOM/. Acesso em: Fevereiro, 2005.

[DTD05] W3C. **Document Type Definition**. Disponível em: http://www.w3schools.com/dtd/default.asp. Acesso em: Fevereiro, 2005.

[EME05] EMER, M. C. F. P. and VERGILIO, S. R. and JINO, M. **A Testing Approach for *XML Schemas***. In: The 29th Annual International Computer Software and Applications Conference, COMPSAC 2005 - QATWBA 2005, Julho, 2005.

[HTML99] W3C. **HyperText Markup Language - HTML 4.01 Specification W3C Recommendation**. Dezembro, 1999.

[JDOM05] **JDOM, JAVA DOM.** Disponível em: http://www.jdom.org/. Acesso em: Agosto, 2005.

[KUN00] KUNG, D.C. and LIU, C.H. and HSIA, P. **An Object-Oriented *Web* Test Model for Testing *Web* Applications.** In: The 24th Annual International Computer Software and Applications Conference, COMPSAC 2000, p: 537-542, 2005.

[LEE01] LEE, Suet Chun. OFFUTT, Jeff, **Generating Test Cases for XML-based *Web* Component Interaction Using Mutation Analysis**. In: Proceedings of the 12th International Symposium o Software Reliabilty Engineering, p. 200-209, Hong Kong, China, Novembro, 2001.

[LI05] LI, Jian Bing, MILLER, James. **Testing the Semantics of W3C *XML Schema***. In: he 29th Annual International Computer Software and Applications Conference, COMPSAC 2005 - QATWBA 2005, Julho, 2005.

[LIU00a] LIU, C.H. and KUNG, D.C. and HSIA, P. and HSU, C.T. **Structural Testing of *Web* Applications.** In: 11th International Symposium on Software Reliability Engineering, p: 84-96, 2000.

[LIU00b] LIU, C.H. and KUNG, D.C. and HSIA, P. **Object-based data flow testing of *Web* applications**. In: First Asia-Pacific Conference on Quality Software, p: 7-16, 2000.

[MAL91] MALDONADO, J. C. **Critérios potenciais usos: uma contribuição ao teste estrutural de software**. 1991. Tese (Doutorado) - DCA/FEE/UNICAMP, Campinas, jul. 1991.

[MAR01] MARTIN, Didier et al. **Professional XML**. Rio de Janeiro: Ciência Moderna Ltda, 2001.

[MYE79] MYERS, G. **The art of software testing**. New York: J. Wiley, 1979.

[OFF02] OFFUTT, J. **Quality attributes of *Web* Software Applications.** IEEE Software Journal, vol 2, p: 25-32, 2002.

[OFF04] OFFUTT, J. and XU, W. **Generating Test Cases for *Web* Services Using Data Perturbation**. In: TAV-*WEB* Proceedings, Setembro, 2004.

[PRE05] PRESSMAN, Roger S. **Engenharia de Software**. 5a. ed. McGraw-Hill, 2005.

[RAP82] RAPPS, S.; WEYUKER, E. J. **Data flow analysis techniques for test data selection.** In: INTERNATIONAL CONFERENCE ON SOFTWARE, p: 272-278, Tokio, 1982 .

[RAP85] RAPPS,S., WEYUKER, E. J. **Selecting software test data using data flow information**. IEEE Trans. Softw. Eng. SE-11, 4 (Apr.), 367-375, 1985.

[SAX05] **SAX Project.** Disponível em: http://www.saxproject.org/. Acesso: Fevereiro, 2005.

[SPY05] **XMLSpy2005**. Disponivel em: http://www.altova.com/products_ide.html. Acesso: Novembro, 2005.

[SQC05] **IBM *XML Schema* Quality Checker**, version 2.2. http://www.alphaworks.ibm.com/tech/xmlsqc. Acesso em: Novembro, 2005.

[STY05] **Stylus Studio 2006**. Disponível em: http://www.stylusstudio.com/xml_download.html. Acesso em: Novembro, 2005.

[TIT03] TITTEL, Ed. **XML**. São Paulo: Bookman, 2003.

[WON94] WONG, W.E. and MATHUR, A.P. and MALDONADO, J.C. **Mutation Versus All-uses: An Empirical Evaluation of Cost, Strength and Effectiveness.** In: Software Quality and Productivity – Theory, Practice, Education and Training, Hong Kong, Dezembro, 1994.

[WU04] WU, Y. and OFFUTT, J. **Modeling and testing *Web*-based Applications.** Disponível em: citeseer.ist.psu.edu/551504.html. Julho, 2004.

[XML05] W3C. **Extensible Markup Language (XML) 1.0 (second edition) – W3C recommendation**, Outubro 2000. Disponível em: http://www.w3.org/XML. Acesso em: Janeiro, 2005.

[XMLS05] W3C. ***XML Schema* recommendation**, Maio, 2001. http://www.w3.org/tr/. Acesso em: Janeiro, 2005.

[XSV05] **W3C Validator for *XML Schema***. Disponível em: http://www.w3.org/2001/03/*Web*data/xsv . Acesso: Novembro, 2005.

[XU05] XU, Wuzhi, OFFUTT, Jeff, LUO, Juan. **Testing *Web* Services by XML Pertubation**. In: Proceedings of the 16th IEEE International Symposium on Software Reliability Engineering, Novembro, 2005.

APÊNDICE A ALGORITMOS DA XTM

No corpo principal é executado um algoritmo para a chamada dos algoritmos de geração de mutantes com base nos operadores de dados e de estrutura da árvore.

Os próximos algoritmos, utilizados para implementar os operadores, foram todos definidos como algoritmos recursivos, portanto será descrito apenas seu corpo principal.

Principal

```
1. Informa o nome do esquema e o nome do documento XML
2. Lê o esquema;
3. Percorre a árvore para alterar os dados do esquema;
4. Percorre a árvore para alterar a ordem de nós irmãos no
   esquema;
5. Percorre a árvore para inserir nós;
6. Percorre a árvore para remover um nó da árvore;
7. Para todos os mutantes gerados, tenta validar o documento
   XML usando os mutantes criados;
```

Algoritmo para alteração de dados:

```
1. Se nó = nulo // fim do esquema
   1.1.   encerra execução;
2. Se no = raiz
   2.1.   Aplica os operadores SNC e QGR;
3. Se tag = 'group order'
   3.1.   Aplica operador GO;
4. Se tag = 'element'
   4.1.   Aplica operadores CSP, REQ, DT e SO;
5. Se tag = 'attribute'
   5.1.   Aplica operadores CSP, AOC, QIR e SO;
6. Se tag = 'sequence' ou 'choice' ou 'all'
   6.1.   Aplica operadores SO e CCR;
7. Se tag = 'complexType'
   7.1.   Aplica operadores CSP e UCR;
8. Se tag = 'lenght'
   8.1.   Aplica operador SLC (lenghtOf);
9. Se   tag   =   'minInclusive'   ou   'minExclusive'   ou
   'maxIncluisive' ou 'maxExclusive'
   9.1.   Aplica operador RAR;
```

```
10.  Se tag = 'totalDigits' ou 'fractionDigits'
  10.1.  Aplica operador NCC;
11.  Se tag = 'restriction' ou 'extension'
  11.1.  Aplica operador DTC;
12.  Salva mutante gerado;
```

Algoritmo para alterar a ordem dos nós irmãos

```
1. Se nó = nulo // fim do esquema
  1.1.   encerra execução;
2. nro_filhos = Lê número de filhos do nó
3. Se número maior que 1
  3.1.   De i= 1 até nro_filhos – 1
  3.2.   De j = i até nro_filhos
  3.3.   Troca nó na posição i com o nó na posição j
```

Algoritmo para inserir nós na árvore

```
1. Se nó = nulo // fim do esquema
  1.1.   encerra execução;
2. Insere um nó como primeiro filho;
3. Salva mutante gerado;
4. Insere um nó como último filho;
5. Salva mutantes gerado;
```

Algoritmo para remover nós da árvore

```
1. Se nó = nulo // fim do esquema
  1.1.   encerra execução;
2. Se no <> raiz // nó raiz não pode ser removido
  2.1.   Remove nó;
3. Salva mutante gerado;
```

ANEXO A OPERADORES DEFINIDOS POR LI E MILLER

Neste anexo são mostrados alguns exemplos dos operadores definidos por Li e Miller [LI05] em seu trabalho descrito no Capítulo 3.

- ***XML Schema Namespace Changes (SNC)***

Original:

```
<schema xmlns="http://www.w3.org/2001/XML Schema">
```

Mutante:

```
<schema xmlns="http://www.w3.org/1999/XML Schema">
```

- ***Target and Default Exchange (TDE)***

Original:

```
<schema
xmlns=http://www.w3.org/2001/XML Schema
xmlns:po=http://www.example.com.PO1
targetNamespace:"http://www.example.com/PO1">
```

Mutante:

```
<schema
xmlns= http://www.example.com.PO1
xmlns:po=http://www.w3.org/2001/XML Schema
targetNamespace:"http://www.example.com/PO1">
```

- ***Element Namespace Replacement (ENR)***

Original:

```
<element name="purchaseOrder" type="po:PurchaseOrderType">
```

Mutante:

```
<element name="purchaseOrder" type="newpo:PurchaseOrderType">
```

- ***Element Namespace Removal (ENM)***

Original:

```
<element name="purchaseOrder" type="po:PurchaseOrderType">
```

Mutante:

```
<element name="purchaseOrder" type=" PurchaseOrderType">
```

- ***Qualification Globally Replacement (QGR)***

Original:

```
<schema ... elementFormDefault="unqualified">
```

Mutante:

```
<schema ... elementFormDefault="qualified">
```

- ***Qualification Individually Replacement (QIR)***

Original:

```
<attribute name="publicKey" type="base64Binary" form="qualified">
```

Mutante:

```
<attribute name="publicKey" type="base64Binary" form="unqualified">
```

- ***ComplexType Compositors Replacement (CCR)***

Original:

```
<complexType>
      <sequence>
            <! – elementos -->
      </sequence>
</complexType>
```

Mutante:

```
<complexType>
      <choice> /* ou all */
      <! – elementos -->
      </choice> /* ou all */
</complexType>
```

- ***ComplexType Order Change (COC)***

Original:

```
<complexType>
      <sequence>
      <element name="name" type="string"/>
      <element name="street" type="string"/>
      <element name="city" type="string"/>
      </sequence>
</complexType>
```

Mutante:

```
<complexType>
      <sequence>
      <element name="name" type="string"/>
      <element name="city" type="string"/>
      <element name="street" type="string"/>
```

```
      </sequence>
</complexType>
```

- ***ComplexType Definitin Decrease (CDD)***

Original:

```
<complexType>
      <sequence>
      <element name="name" type="string"/>
      <element name="city" type="string"/>
      <element name="street" type="string"/>
      </sequence>
</complexType>
```

Mutante:

```
<complexType>
      <sequence>
      <element name="city" type="string"/>
      <element name="street" type="string"/>
</sequence>
</complexType>
```

- ***Element Occurence Change (EOC)***

Original:

```
<xsd:element name="item" minOccurs="0" maxOccurs="unbounded">
```

Mutante 1:

```
<xsd:element name="item" minOccurs="1" maxOccurs="unbounded">
```

Mutante 2:

```
<xsd:element name="item" minOccurs="0" maxOccurs="1">
```

- ***Attribute Occurence Change (AOC)***

Original:

```
<xsd:attribute name="partNum" use="required"/>
```

Mutante 1:

```
<xsd:attribute name="partNum" use="optional"/>
```

Mutante 2:

```
<xsd:attribute name="partNum" use="prohibited"/>
```

- ***SimpleType Pattern Change (SPC)***

Original:

```
<xsd:simpleType name="SKU">
      <xsd:restriction base="xsd:string">
      <xsd:pattern value="\d{3}-{A-Z}{2}">
      </xsd:restriction>
```

```
</xsd:simpleType>
```

Mutante 1:

```
<xsd:pattern value="\d{4}-{A-Z}{2}">
```

Mutante 2:

```
<xsd:pattern value="\d{3}-{A-Y}{2}">
```

- ***Restriction Arguments Replacement (RAR)***

Original:

```
<xsd:simpleType name="myInteger">
      <xsd:restriction base="xsd:integer">
      <xsd:minInclusive value="10000"/>
      <xsd:maxInclusive value="99999"/>
      </xsd:restriction>
</xsd:simpleType>
```

Mutante 1:

```
<xsd:minExclusive value="10000"/>
<xsd:maxInclusive value="99999"/>
```

Mutante 2:

```
<xsd:minInclusive value="10000"/>
<xsd:maxExclusive value="99999"/>
```

Mutante 3:

```
<xsd:minInclusive value="9999"/>  /* ou 10001 */
<xsd:maxInclusive value="99999"/>
```

Mutante 4:

```
<xsd:minInclusive value="10000"/>
<xsd:maxInclusive value="99998"/> /* ou 100000*/
```

- ***Enumeration Element Change (EEC)***

Original:

```
<xsd:simpleType name="USState">
      <xsd:restriction base="xsd:string">
      <xsd:enumeration value="AK"/>
      <xsd:enumeration value="AL"/>
      <xsd:enumeration value="AR"/>
      </xsd:restriction>
</xsd:simpleType>
```

Mutante 1:

```
/* REMOVE  <xsd:enumeration value="AK"/>  */
<xsd:enumeration value="AL"/>
<xsd:enumeration value="AR"/>
```

Mutante 2:

```
<xsd:enumeration value="AK"/>
<xsd:enumeration value="AL"/>
<xsd:enumeration value="AR"/>
<xsd:enumeration value="AW"/>     /* ADICIONA */
```

- ***String Length Change (SLC)***

Original:

```
<xsd:restriction base="USStateList">
      <xsd:length value="6"/>
</xsd:restriction>
```

Mutante:

```
<xsd:restriction base="USStateList">
      <xsd:length value="5"/>   /* Ou value="7"  */
</xsd:restriction>
```

- ***Derived Type Control Replacement (DTC)***

Original:

```
<complexContent>
      <restriction base="ipo:Itens">
      <! Outras definições -->
      </restriction>
</complexContent>
```

Mutante:

```
<complexContent>
      <extension base="ipo:Itens">
      <! Outras definições -->
      </extension>
</complexContent>
```

- ***Use of Controller Replacement (UCR)***

Original:

```
<complexType name="Adress" final="restriction">
<complexType name="Adress" block="restriction">
<simpleType name="Postcode">
      <restriction base="string">
      <length value="7" fixed="true"/>
      </restriction>
</simpleType>
```

Mutante:

```
<complexType name="Adress" final="#all">
<complexType name="Adress" block="extension">
/* REMOVE   <length value="7" fixed="true"/>  */
```

ANEXO B ESPECIFICAÇÕES DOS ESQUEMAS UTILIZADOS NOS EXPERIMENTOS

Os esquemas utilizados nos experimentos foram extraídos de 3 sistemas sendo um deles um sistema real desenvolvido para um hospital. Os demais sistemas tratam-se de um sistema de biblioteca e de matriculas. No total foram testados 6 (seis) esquemas sendo, dois esquemas por sistema.

1 Sistema de Biblioteca

Descrição do Sistema

O usuário cadastrado poderá acessar o sistema mediante *login* e senha para consultar obras disponíveis no acervo da biblioteca. O usuário que estiver com a mensalidade em dia poderá acessar no máximo o conteúdo de três obras por mês. Portanto, o acesso do usuário a uma obra deve ser armazenado.

Descrição da estrutura da informação

- Obras --> código, título, descrição, conteúdo, autor, editora, local, ano, ISBN.
- Usuário --> código, nome, login, senha, pagamento de mensalidade (mês e ano), quantidade de acesso ao conteúdo de obras.

Observações

os códigos devem possuir uma quantidade de dígitos limitada (obras e usuário: seis), o login deve conter até oito caracteres, a senha deve conter 4 caracteres e 2 números em qualquer ordem, a obra pode ter um ou mais autores.

2 Sistema de Matriculas

Descrição do Sistema

O aluno poderá acessar o sistema com logina e senha para realizar matrícula. O aluno somente poderá se matricular em disciplinas do seu período ou pendentes e cujos pré-requisitos tenham sido cumpridos.

Descrição da estrutura da informação

- Aluno --> código, nome, login, senha, curso, período, disciplinas cursadas, disciplinas matriculadas;
- Disciplina --> código, nome, carga horária, créditos, período, ementa, código do professor, código de pré-requisitos (código de disciplinas cursadas necessárias).

Observações

os códigos devem possuir uma quantidade de dígitos limitada(aluno seis e disciplina três), o login deve conter até oito caracteres, a senha deve conter 4 caracteres e 2 números em qualquer ordem, as disciplinas podem ter 1 ou 2 professores e zero ou mais pré-requisitos.

3 Sistema Hospitalar

3.1 Arquivo - Ficha de Internamento Hospitalar

Objetivo

Coletar informações dos gastos com os internamentos Hospitalares pagos pelo SUS. Esse arquivo deverá ser enviado por todas instituições que internam pacientes SUS.

Dicionário de Dados

CGC: Especificar o CGC/CNPJ da prestadora de serviço. Deve possuir quatorze (14) dígitos numéricos (sem pontos, traços ou barras).

DATA: Especificar a data em que o arquivo foi gerado. Deve possuir o seguinte formato: DD/MM/AAAA (D = Dia, M = Mês, A = Ano).

NR_AIH: Especificar o número da AIH em caso de reapresentação. Deve possuir onze (11) dígitos incluindo a barra do digito verificador. Exemplo: 241762094-2.

DATA_INTERNAMENTO: Especificar a data em que o paciente foi internado. Deve possuir o seguinte formato: DD/MM/AAAA (D = Dia, M = Mês, A = Ano).

DATA_ALTA: Especificar a data em que o paciente recebeu alta. Deve possuir o seguinte formato: DD/MM/AAAA (D = Dia, M = Mês, A = Ano).

VALOR_TOTAL: Especificar o custo total do internamento. Deve possuir números reais positivos com no máximo duas(2) casas decimais separadas por ponto(.). Exemplo: 1275.50 (mil duzentos e setenta e cinco reais e cinqüenta centavos).

CPF_MED_SOLICITANTE: Especificar o CPF do médico solicitante do internamento. Deve possuir onze (11) dígitos numéricos (sem pontos, traços ou barras).

CPF_MED_AUDITOR: Especificar o CPF do médico que realizou a auditoria do laudo médico. Deve possuir onze (11) dígitos numéricos (sem pontos, traços ou barras).

CPF_MED_RESPONSAVEL: Especificar o CPF do médico responsável pelo internamento do paciente no hospital. Deve possuir onze (11) dígitos numéricos (sem pontos ou traços).

CID: Especificar o código internacional de doenças. Deve possuir no máximo dez (10) dígitos alpha numéricos.

NR_PRONTUARIO: Especificar o identificador do paciente no hospital. Deve possuir no máximo quinze (15) dígitos alpha numéricos.

NOME: Especificar o nome do paciente internado. Deve possuir no máximo cento e cinqüenta dígitos (150) dígitos alpha numéricos.

DATA_NASC: Especificar a data de nascimento do paciente. Deve possuir o seguinte formato: DD/MM/AAAA (D = Dia, M = Mês, A = Ano).

RG: Especificar o registro geral do paciente. Deve possuir de (8) a nove (9) dígitos apenas numéricos (sem pontos, traços ou barras).

CPF: Especificar o cadastro de pessoa física do paciente internado. Deve possuir onze (11) dígitos numéricos (sem pontos, traços ou barras).

LOGRADOURO: Especificar o nome do logradouro (rua, avenida) que reside o paciente. Deve possuir no máximo cem (100) dígitos alpha numéricos.

CEP: Especificar o CEP do logradouro que reside o paciente. Deve possuir no máximo vinte (20) dígitos alpha numéricos.
NR_PREDIAL: Especificar o número predial que reside o paciente. Deve possuir no máximo dez (10) dígitos alpha numéricos.
BAIRRO: Especificar o nome do bairro que reside o paciente. Deve possuir no máximo cem (100) dígitos alpha numéricos.
COD_IBGE_CIDADE: Especificar o código da cidade com base no IBGE que reside o paciente. Deve possuir no máximo 10 dígitos alpha numéricos.
CIDADE: Especificar o nome da cidade que reside o paciente. Deve possuir no máximo cem (100) dígitos alpha numéricos.
COMPLEMENTO: Especificar informações complementares sobre o local em que o paciente reside. Deve possuir no máximo cinqüenta (50) dígitos alpha numéricos.
COD_SIHSUS: Especificar o código SIH SUS do procedimento. Deve possuir oito (8) dígitos numéricos (sem pontos, traços ou barras).
DESCRICAO_PROC: Especificar a descrição do procedimento. Deve possuir no máximo dois mil e quinhentos (2500) dígitos alpha numérico.
VALOR_UNIT: Especificar o valor unitário do procedimento. Deve possuir números reais positivos com no máximo duas(2) casas decimais separadas por ponto(.). Exemplo: 1275.50 (mil duzentos e setenta e cinco reais e cinqüenta centavos).
QTD: Especificar a quantidade de vezes que o procedimento foi executado. Deve possuir somente números inteiros positivos. Exemplo: 10.

Observações

No campo procedimentos executados <proc_executados> </proc_executados> do arquivo deverão ser colocados todos os procedimentos executados no paciente (procedimentos especiais, serviços profissionais).

3.2 Arquivo - Ficha de Realização de Procedimentos Ambulatoriais

Objetivo

Coletar informações de gastos Ambulatórias SUS. Esse arquivo deverá ser enviado por todas instituições que atendem pacientes SUS (ficha de atendimento ambulatorial) bem como executam as requisições de exames/procedimentos de média e alta complexidade.

Dicionário de Dados

CGC: Especificar o CGC/CNPJ da prestadora de serviço. Deve possuir quatorze (14) dígitos numéricos (sem pontos, traços ou barras).

DATA: Especificar a data em que o arquivo foi gerado. Deve possuir o seguinte formato: DD/MM/AAAA (D = Dia, M = Mês, A = Ano).

DATA_ATENDIMENTO: Especificar a data em que o paciente foi atendido. Deve possuir o seguinte formato: DD/MM/AAAA (D = Dia, M = Mês, A = Ano).

DATA_SOLICITACAO: Especificar a data em que os procedimentos foram solicitados pelo médico. Deve possuir o seguinte formato: DD/MM/AAAA (D = Dia, M = Mês, A = Ano).

CPF_MED_SOLICITANTE: Especificar o CPF do médico solicitante dos procedimentos. Deve possuir onze (11) dígitos numéricos (sem pontos, traços ou barras).

CPF_MED_AUDITOR: Especificar o CPF do médico que realizou a auditoria do laudo médico. Deve possuir onze (11) dígitos numéricos (sem pontos, traços ou barras).

VALOR_TOTAL: Especificar o custo total dos procedimentos realizados. Deve possuir números reais positivos com no máximo duas (2) casas decimais separadas por ponto(.). Exemplo: 1275.50 (mil duzentos e setenta e cindo reais e cinqüenta centavos).

CID: Especificar o código internacional de doenças. Deve possuir no máximo dez (10) dígitos alpha numéricos.

NR_PRONTUARIO: Especificar o identificador do paciente na prestadora de serviço. Deve possuir no máximo quinze (15) dígitos alpha numéricos.

NOME: Especificar o nome do paciente. Deve possuir no máximo cento e cinqüenta dígitos (150) dígitos alpha numéricos.

DATA_NASC: Especificar a data de nascimento do paciente. Deve possuir o seguinte formato: DD/MM/AAAA (D = Dia, M = Mês, A = Ano).

RG: Especificar o registro geral do paciente. Deve possuir de (8) a nove (9) dígitos apenas numéricos (sem pontos, traços ou barras).
CPF: Especificar o cadastro de pessoa física do paciente internado. Deve possuir onze (11) dígitos numéricos (sem pontos, traços ou barras).
LOGRADOURO: Especificar o nome do logradouro (rua, avenida,) que reside o paciente. Deve possuir no máximo cem (100) dígitos alpha numéricos.
CEP: Especificar o CEP do logradouro que reside o paciente. Deve possuir no máximo vinte (20) dígitos alpha numéricos.
NR_PREDIAL: Especificar o número predial que reside o paciente. Deve possuir no máximo dez (10) dígitos alpha numéricos.
BAIRRO: Especificar o nome do bairro que reside o paciente. Deve possuir no máximo cem (100) dígitos alpha numéricos.
COD_IBGE_CIDADE: Especificar o código da cidade com base no IBGE que reside o paciente. Deve possuir no máximo 10 dígitos alpha numéricos.
CIDADE: Especificar o nome da cidade que reside o paciente. Deve possuir no máximo cem (100) dígitos alpha numéricos.
COMPLEMENTO: Especificar informações complementares sobre o local em que o paciente reside. Deve possuir no máximo cinqüenta (50) dígitos alpha numéricos.
COD_SIASUS: Especificar o código SIA SUS do procedimento. Deve possuir oito (8) dígitos numéricos (sem pontos, traços ou barras).
DESCRICAO_PROC: Especificar a descrição do procedimento. Deve possuir no máximo dois mil e quinhentos (2500) dígitos alpha numéricos.
VALOR_UNIT: Especificar o valor unitário do procedimento. Deve possuir números reais positivos com no máximo duas(2) casas decimais separadas por ponto(.). Exemplo: 1275.50 (mil duzentos e setenta e cindo reais e cinqüenta centavos).
QTD: Especificar a quantidade de vezes que o procedimento foi executado. Deve possuir somente números inteiros positivos. Exemplo: 10.

Observações

- elemento data de atendimento <DATA_ATENDIMENTO> </DATA_ATENDIMENTO> deverá ser especificado na estrutura do arquivo e

deverá ser preenchido somente quando os procedimentos realizados forem provenientes de uma ficha de atendimento ambulatorial;

- O elemento data de solicitação <DATA_SOLICITACAO> </DATA_SOLICITACAO> deverá ser especificado na estrutura do arquivo e deverá ser preenchido somente quando os procedimentos executados forem provenientes de uma requisição de exames/procedimentos;
- O elemento CPF do médico solicitante <CPF_MED_SOLICITANTE> </CPF_MED_SOLICITANTE> deverá ser especificado na estrutura do arquivo e deverá preenchido somente quando os procedimentos executados forem provenientes de uma requisição de exames/procedimentos;
- O elemento CPF do médico auditor <CPF_MED_AUDITOR> </CPF_MED_AUDITOR> deverá ser especificado na estrutura do arquivo e deverá ser preenchido somente quando os procedimentos executados forem provenientes de uma requisição de exames/procedimentos de alta complexidade;
- O elemento procedimento principal <PROC_PRINCIPAL> </PROC_PRINCIPAL> deverá ser especificado na estrutura do arquivo e deverá ser preenchido somente quando os procedimentos executados forem provenientes de uma requisição de exames/procedimentos de alta complexidade.
- O elemento CID (código internacional de doenças) deverá ser especificado na estrutura do arquivo e deverá ser preenchido somente quando os procedimentos executados forem provenientes de uma requisição de exames/procedimentos de alta complexidade.