Igor Fabiano Nazar

X-Tool: Uma Ferramenta de Teste de Esquemas para Estrutura de Dados

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 Dr.ª Silvia Regina Vergílio

CURITIBA

2007

AGRADECIMENTOS

Agradeço à professora Silvia Regina Vergílio, minha orientadora, por seus ensinamentos, dedicação e apoio. A todos os professores que me proporcionaram novos conhecimentos. Em Especial a Maria Cláudia Figueiredo Pereira Emer por me mostrar a direção a seguir.

Agradeço à Coordenação do Curso de Pós-Graduação em Informática da Universidade Federal do Paraná pela educação e confiança depositadas em mim.

Por fim agradeço aos meus pais, amigos e familiares que sempre me apoiaram e incentivaram nos momentos em que senti dificuldades em prosseguir.

Sumário

RESUMO

O uso de sistemas baseados em computação exige o desenvolvimento de software com alto padrão de qualidade. Um dos pontos em que pode ser analisada a qualidade de um sistema é com relação à integridade das informações que a aplicação manipula, que podem ser definidas por um esquema que fornece regras referentes à estrutura lógica dos dados e das informações permitidas. Uma forma de garantir a confiabilidade nos dados e conseqüentemente nas aplicações, é testar o esquema. Este trabalho descreve a ferramenta X-Tool que testa esquemas de dados, utilizando classes de defeito previamente definidas. A ferramenta X-Tool contribui para facilitar a atividade de teste e reduzir esforços e tempo gasto nesta tarefa.

ABSTRACT

The use of computer-based systems demands the development of high-level quality software. One of the fundamental points related to the quality of a system is associated to the integrity of the information that the application manipulates. Information of an application is usually defined by a schema, which provides rules about the logical data structure and allowed information. Testing schema is very important to ensure the data reliability of the applications. This work describes X-Tool, a tool that tests schema of data, using fault classes previously defined. X-Tool contributes to the test activity, by reducing efforts and time spent in this phase.

Lista de Figuras

Lista de Tabelas

1 Introdução

1.1 Contexto

O uso de sistemas baseados em computação exige o desenvolvimento de software com alto padrão de qualidade. Nesse sentido, novos métodos e técnicas têm sido propostos pela disciplina de Engenharia de Software visando à produção de software de alta qualidade e baixo custo.

Um dos pontos em que pode ser analisada a qualidade de um sistema é com relação à integridade das informações que a aplicação manipula, usadas tanto para a integração entre sistemas quanto para armazenamento de dados. Devido à grande variedade de dados que podem ser representados, há a necessidade de se definir uma estrutura para os mesmos, para isso utilizam-se esquemas, os quais fornecem regras referentes aos elementos permitidos e à estrutura lógica dos dados.

No contexto de integração de sistemas são geralmente utilizados os *Web Services,* os quais fazem trocas e distribuições de dados através de mensagens que utilizam a linguagem XML (*eXtensible Markup Language*) [W3C06a] para padronizar essas informações. Cada documento XML pode estar referenciando um esquema (*DTD* [W3S06], *XML Schema* [W3C06b], entre outros) que define sua estrutura lógica através de restrições.

No armazenamento de dados, o foco é para bases de dados relacionais que exigem um esquema para descrever suas entidades, tipo de dados, relacionamentos e restrições. Um esquema ou modelo conceitual é escrito utilizando a DDL (*Data Definition Language*). Existem alguns modelos que representam visualmente um esquema, entre os quais se destacam o DER (Diagrama Entidade Relacionamento)[WIN06c] e o IDEF1x (*Integration DEFinition*) [WIN06d].

No entanto, os esquemas podem conter defeitos referentes à interpretação da estrutura dos dados e validar incorretamente algumas informações, causando falhas em um sistema. A atividade de teste se propõe a auxiliar na descoberta

destes defeitos. O teste de esquemas contribui para que seja assegurada a integridade dos dados e a confiabilidade nas aplicações, buscando assim garantir a qualidade do sistema.

A maioria dos trabalhos encontrados na literatura visam o teste das aplicações que manipulam os dados. Poucos são os trabalhos que focalizam o teste de esquemas. Li e Miller [LI05] e Franzotte e Vergilio [FRA06] introduzem um conjunto de operadores de mutação para o teste de esquema no contexto de XML. Emer [EME07b] introduz uma abordagem baseada em defeitos. A abordagem de Emer é genérica e pode ser aplicada a diversos tipos de esquemas, além disso, os dados de teste são formados por dados e consultas a esses dados que podem ser geradas automaticamente. Entretanto para que essa abordagem possa ser utilizada é indispensável que uma ferramenta de suporte esteja disponível. O teste é uma atividade custosa e propensa a erros se executada manualmente. A utilização de ferramentas é fundamental para reduzir esforços e custos dessa atividade.

1.2 Objetivo do trabalho

O objetivo desta dissertação é apresentar a ferramenta X-Tool (*XML and Relational Database Schema Testing Tool*), que visa a revelar defeitos em esquemas de estrutura de dados, referentes à definição incorreta ou ausência de restrições, as quais são responsáveis por definir regras que validam os dados associados ao esquema em teste. Os defeitos são detectados por meio de consultas aos dados considerados válidos pelo esquema, explorando as classes de defeito previamente definidas pela abordagem de Emer [EME07b], que descrevem os principais erros que podem acontecer ao se projetar esquemas.

1.3 Organização do trabalho

O restante desta dissertação está organizado da seguinte forma. No Capítulo 2 apresentam-se os esquemas e linguagens de consulta utilizadas pela ferramenta. O Capítulo 3 traz uma visão geral sobre teste de software e teste de

esquemas. O Capítulo 4 descreve a ferramenta proposta. O Capítulo 5 traz os resultados dos experimentos realizados, nos quais foi possível realizar uma avaliação da abordagem implementada pela X-Tool. Finalmente, o Capítulo 6 apresenta as conclusões deste trabalho. O trabalho contém um apêndice (Apêndice A) que apresenta os diagramas do modelo de dados da base utilizada no experimento descrito no Capítulo 5.

2 Esquemas e linguagens de consulta

Os dados de uma aplicação podem ser de diversos tipos e formatos e existirem em grande quantidade. Além, disso, eles podem ser agrupados de diferentes maneiras. Para que esses dados possam ser organizados, esquemas são geralmente utilizados. Os esquemas são responsáveis pela organização e definição da estrutura lógica do conteúdo dos dados. Em geral, cada tipo de representação de dados adota uma maneira diferente para definir seus esquemas e manipular seus dados, mas todas seguem o mesmo princípio de estruturar as informações.

As representações de dados de interesse neste trabalho são representações para bases de dados relacionais e documentos XML.

As bases de dados relacionais são utilizadas para armazenamento de dados que contêm um grande volume de informações, seus esquemas são definidos pelo modelo conceitual. A consulta e a manipulação dos dados são, em geral, baseadas na SQL (*Structured Query Language*) [WIN06b].

Documentos XML são utilizados principalmente em trocas de mensagens e armazenamento de dados com pouco volume de informações. A definição de um esquema é opcional e pode ser representada de várias maneiras, dentre as mais utilizadas se destacam a DTD (*Document Type Definition*) [WIN06d] e o *XML Schema* [W3C06b]. A consulta aos dados de um documento pode ser realizada pela linguagem XQuery (*XML Query Language*) [W3C06d].

Este capítulo fornece uma visão geral desses esquemas para dados XML e para dados de bases relacionais.

2.1 Esquemas de Dados XML

XML (*Extensible Markup Language*) [W3C06a] é um subconjunto da SGML (*Standard Generatized Markup Language*) [W3C06e], que permite a representação de dados que são utilizados para transmissão ou armazenamento. Diversas

aplicações como *Web Services*, comércio eletrônico, cadastros, dentre outras, vêm utilizando a linguagem XML para manipular seus dados.

A idéia original da XML é ser uma linguagem de representação de dados para a Internet voltada para o contexto de recuperação de informações pela descrição de seus dados. Hoje em dia a utilização da XML não se restringe somente a um único tipo de aplicação, ela pode ser usada em qualquer contexto que necessite da recuperação de informação por sua descrição.

Sendo uma linguagem de marcação, XML utiliza delimitadores (ou *tags*) para descrever dados, assim como a linguagem HTML (*Hyper Text Markup Language*). A diferença entre elas, é que *tags* em HTML descrevem instruções para apresentação de dados em *browsers* Web, enquanto *tags* em XML denotam uma interpretação semântica para o dado delimitado por elas. Como esta interpretação é particular para cada domínio de aplicação, diz-se que XML é uma meta-linguagem, pois cada aplicação tem a liberdade de definir a sua própria linguagem de marcação de dados, descrevendo a nomenclatura e a estruturação das *tags* que julgar apropriada.

A linguagem XML apresenta as seguintes vantagens:

- é representada na forma de texto: isso permite que o documento possa ser compreensível tanto para as pessoas como pelas máquinas;
- é de fácil adaptação: suas marcações personalizadas podem ser criadas livremente para qualquer necessidade;
- é de fácil utilização: pode ser utilizada em qualquer plataforma e interpretada por qualquer tipo de sistema.

O documento XML apresentado na *Figura 2.1* contém informações referentes a uma listagem de cd e suas descrições.

```
<?xml version="1.0" ?>
<cds >
  <cd genero="instrumental" duracao="00:46:32">
    <titulo>Sucessos do Cinema</titulo>
    <artista>Richard Clayderman</artista>
    <musica>I just called to say I love you</musica>
    <musica>I will always love you</musica>
    <musica>Unchained melody</musica>
    <musica>Theme from love story</musica>
    <musica>How deep is your love</musica>
    <gravadora>Polygram</gravadora>
    <ano>1996</ano>
  </cd>
  <cd genero="instrumental">
    <titulo>Live</titulo>
    <artista>Kenny G</artista>
    <musica>Going Home</musica>
    <musica>Sade</musica>
    <musica>Silhouette</musica>
    <musica>Midnight Motion</musica>
    <musica>Home</musica>
    <musica>Don't Make Me Wait For
Love</musica>
    <musica>I've Been Missin' You</musica>
    <gravadora>BMG</gravadora>
    <ano>1993</ano>
  </cd>
<cd gênero="miscelania" duracao="">
     <titulo>Suave Veneno</titulo>
    <artista>Nana Caymmi</artista>
    <artista>Kiloucura</artista>
    <artista>Maria Bethania</artista>
    <artista>Banda Eva</artista>
    <artista>Caetano Veloso</artista>
    <musica>Suave Veneno</musica>
    <musica>Pela Vida Inteira</musica>
    <musica>E o amor</musica>
    <musica>Frisson</musica>
    <musica>Sozinho</musica>
    <gravadora>Som Livre</gravadora>
    <ano>1999</ano>
  </cd>
  <cd genero="miscelania"
duracao="00:46:32">
    <titulo>Desejos de Mulher</titulo>
    <artista>Marisa Monte</artista>
    <artista>Caetano Veloso</artista>
    <artista>Daniela Mercury</artista>
    <artista>Titas</artista>
    <artista>Rita Lee</artista>
    <musica>A sua</musica>
    <musica>Sonhos</musica>
    <musica>Mutante</musica>
    <musica>Epitafio</musica>
    <musica>Minha Vida</musica>
    <gravadora>Som Livre</gravadora>
    <ano>2002</ano>
  </cd>
 </cds>
```

Figura 2.1 Exemplo de documento XML.

2.1.1 Estrutura dos Documentos XML

Todo documento XML possui uma estrutura básica, composta de um cabeçalho, chamado de prólogo, e do restante do documento, chamado de instância. No prólogo, há informações que identificam o documento como sendo XML. A instância do documento segue o prólogo, contendo os dados propriamente ditos, organizados como uma hierarquia de elementos.

Os elementos de um documento XML são representados por marcas (*tags*), que possuem uma marca inicial delimitada por “<” marca ”>”, e um marca final delimitada por “</” marca “>”, podendo possuir conteúdo ou não entre estas marcações.

A instância do documento XML é formada por elementos simples, elementos complexos e atributos. Esses dados são geralmente especificados por um esquema que apresenta a estrutura lógica do documento, a qual define quais são os elementos, atributos e regras que restringem e modelam o conteúdo do documento.

Tipos de elementos de um documento XML:

- Elementos simples - só podem conter textos, não podendo conter nem atributos e nem elementos filhos. Os elementos do tipo simples são a base para a criação de elementos complexos e, portanto, a base para a criação de documentos XML como um todo. Na *Figura 2.2(a)* é mostrado um elemento simples com o nome "artista";
- Elementos complexos - são elementos compostos por elementos simples ou outros elementos complexos, podendo ter atributos para diferenciá-los. Na *Figura 2.2(b)* é mostrado um elemento complexo com o nome "cd" composto pelos elementos "titulo, artista, musica gravadora e ano";

(a) Elemento Simples

```
<artista>Kenny G</artista>
```

(b) Elemento Complexo

```
<cd>
   <titulo>Live</titulo>
   <artista>Kenny G</artista>
   <musica>Going Home</musica>
   <musica>Sade</musica>
   <musica>Silhouette</musica>
   <gravadora>BMG</gravadora>
   <ano>1993</ano>
</cd>
```

Figura 2.2 Elementos de um documento XML.

- Atributo - o atributo é uma descrição ou identificação para elementos complexos, é usado na marca de início logo após o nome do elemento, todos os valores de atributos devem estar entre aspas. Na *Figura 2.3* é mostrado um atributo com o nome "genero" pertencente ao elemento "cd".

```
<cd genero="instrumental">
...
</cd>
```

Figura 2.3 Atributo de um elemento complexo.

2.1.2 Validação dos Documentos XML

Para um documento ser considerado válido é necessário garantir que o mesmo esteja bem formado, ou seja, que ele obedeça a uma sintaxe de marcação para documentos XML.

Um documento bem formado é considerado válido se ele obedece às restrições impostas por um esquema (DTD [W3S06] ou *XML Schema* [W3C06b]). Se um documento for válido, ele pode ser usado para transmitir informação entre duas aplicações diferentes que também sigam as especificações.

Os esquemas são escritos seguindo restrições impostas pela descrição das informações que o documento XML receberá. A descrição apresentada na *Tabela 2.1* se refere ao conteúdo de um documento XML que armazena informações sobre cds.

Tabela 2.1 Especificações dos dados do documento "cd.xml".

Especificação para os dados que serão armazenados no documento XML "cd.xml", que contém informações sobre cds disponíveis em uma determinada loja.	
Cd	Um ou mais cds disponíveis na loja
Título	Título do cd (apenas um - necessário)
Artista	Um cd pode ser um produto de mais de um cantor (necessário)
Música	Nome da música de cada faixa do cd (uma ou mais músicas - necessário)
Gênero	Referente ao cd (apenas um - necessário)
Duração	Tempo de duração do cd (hora:minutos:segundos)
Gravadora	Nome da gravadora do cd (apenas uma)
Ano	Ano em que o cd foi lançado (apenas um – necessário)

2.1.3 Document Type Definition (DTD)

A DTD foi a primeira recomendação da W3C para a especificação de esquemas de documentos XML. A DTD é escrita por uma gramática formada basicamente por cláusulas ELEMENT e ATTLIST, com objetivo de especificar:

• os elementos e atributos que são permitidos.
• os tipos de valores permitidos para cada elemento.
• a ordem e o alinhamento dos elementos.

A *Figura 2.4* mostra uma DTD, para o documento "*cd.xml*". Neste esquema, o elemento raiz "*publicações"* é composto por um ou mais elementos do tipo *artigo*. Cada instância do elemento "*artigo"* possui um atributo "*versão"* e define uma seqüência composta por um elemento "*título"* (com conteúdo textual - #PCDATA), um ou mais elementos do tipo "*autor"* e zero ou mais elementos do tipo "*seção"*. Cada *autor* possui um "*nome"* e um "*eMail"* opcional. Cada "*seção"* possui um atributo "*nome"* e define um misto de texto e sub-elementos do tipo "*subseção"*.

```
<!ELEMENT publicacoes (artigo+)>
<!ELEMENT artigo (titulo, autor+,secao*)>
<!ATTLIST artigo versao CDATA>
<!ELEMENT titulo (#PCDATA)>
<!ELEMENT autor (nome, eMail?)>
<!ELEMENT nome (#PCDATA)>
<!ELEMENT eMail (#PCDATA)>
<!ELEMENT secao (#PCDATA | subsecao)*>
<!ATTLIST secao nome CDATA>
<!ELEMENT subsecao (#PCDATA)>
```

Figura 2.4 Definição de um esquema em DTD.

2.1.4 XML Schema (XSD)

Assim como a DTD, a linguagem *XML Schema* descreve a estrutura de um documento XML. No entanto, diferentemente da DTD, XML Schema utiliza componentes da própria linguagem XML (meta-linguagem) para restringir e documentar, além de possuir mais recursos para a especificação de esquemas.

As principais restrições para um elemento, que podem ser encontradas em um *XML Schema*, são apresentadas na *Tabela 2.2*.

Tabela 2.2 Restrições aplicadas em Schema.

Restrição	Descrição
maxOccurs, minOccurs	Indicador de ocorrência para definir com que freqüência um determinado elemento pode aparecer.
all, choice, sequence	Indicadores de ordem para definir como elementos deveriam acontecer.
type (string , decimal, integer, boolean, date, time)	Tipo de dados suportado por um elemento simples.
Enumeration	Valores enumerados (previamente definidos)
maxInclusive, maxExclusive, minInclusive, minExclusive	Valor limite (máximo e mínimo) para elementos do tipo numeral.
Pattern	Contém uma expressão regular com a qual o elemento será comparado.
whiteSpace	Tipo de espaços permitidos (espaço simples duplo, tab, quebra de linha)
length, minLength, maxLength	Números de caracteres permitidos para elementos do tipo string.
totalDigits	Número de dígitos
fractionDigits	Número máximo de dígitos na parte fracionária do elemento

O esquema apresentado na *Figura 2.5* é um documento *XML Schema*, que segue a estrutura descrita na *Tabela 2.4.* O esquema é composto pelos seguintes elementos (cd, cds, titulo, artista, musica, gravadora e ano), onde (titulo, artista, musica, gravadora e ano) são elementos simples e (cd e cds) que são elementos complexos, o elemento (cds) possui os atributos (gênero e duração).

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:element name="cds">
  <xsd:complexType>
   <xsd:sequence>
    <xsd:element name="cd" maxOccurs="unbounded">
     <xsd:complexType>
      <xsd:sequence>
       <xsd:element name="titulo" type="xsd:string"  minOccurs="1"  maxOccurs="1" />
       <xsd:element name="artista" type="xsd:string"  minOccurs="1"  maxOccurs="unbounded" />
       <xsd:element name="musica" type="xsd:string"  minOccurs="1" maxOccurs="unbounded"/>
       <xsd:element name="gravadora" type="xsd:string"  minOccurs="0" maxOccurs="1"/>
       <xsd:element name="ano"  type="xsd:integer"   minOccurs="1"  maxOccurs="1"/>
      </xsd:sequence>
      <xsd:attribute name="genero" type="xsd:string" use="required"/>
      <xsd:attribute name="duracao" type="xsd:time"   use="optional"/>
     </xsd:complexType>
    </xsd:element>
   </xsd:sequence>
  </xsd:complexType>
 </xsd:element>
</xsd:schema>
```

Figura 2.5 Definição de um esquema em XML Schema.

A descrição dos principais comandos do esquema apresentado na *Figura 2.5* é descrita na *Tabela 2.3.*

Tabela 2.3 Descrição dos elementos de um XML Schema.

Fragmento do Schema	**Descrição**
<?xml version="1.0"?>	• indica a versão utilizada.
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">	• o elemento <xsd:schema> é a raiz do documento. • xmlns:xsd indica o padrão de esquema que está sendo utilizado.
<xsd:element name="cd" maxOccurs="unbounded">	• maxOccurs - restringe o número de ocorrência máxima do elemento "cd". • unbounded - valor que indica que o número de ocorrências máximas não é definido.
<xsd:element name="ano" type="xsd:gYear" minOccurs="1" maxOccurs="1"/>	• type - especifica que o elemento "ano" tem um tipo de valor definido "gYear" que aceita somente valores do tipo ano. • minOccurs - Número mínimo de ocorrências permitidas para o elemento.
<xsd:attribute name="genero" type="xsd:string" use="required"/>	• atribute - indica que o elemento "genero" é um atributo. • use – especifica que o uso deste atributo é requerido.

Utilizar *XML Schema* ao invés da DTD encontra as seguintes vantagens. *XML Schema*:

- Permite tipos de dados primitivos (string, boolean, float, double, decimal, time date, dateTime ...);
- Permite a criação de novos tipos pelo usuário;
- É escrito em XML;
- Permite herança de tipos – sendo possível estendê-los ou restringí-los;
- Possibilita a criação de chaves primárias a partir de elementos (não permite a existência de dois elementos com o mesmo valor de conteúdo);
- Permite definir elementos diferentes com o mesmo nome, pelo uso de *namespaces*.

2.1.5 Manipulando documentos XML

Com a adoção da XML como um formato universal para troca de informações entre diversas aplicações, é natural que surja a necessidade de se manipular e se realizar consultas em documentos XML [CHA05].

Dentre as APIs (*Applications Programming Interface*) que permitem a manipulação do documento XML, destacam-se DOM [W3C06c] e SAX [WIN06a]. A primeira é definida pela W3C (*World Wide Web Consortium*), responsável por desenvolver tecnologias para a *Web*, e foi utilizada neste trabalho.

Dentre as linguagens de consulta destaca-se a linguagem XQuery [W3C06d], definida pela W3C e similar a SQL (*Structured Query Language*) [WIN06b] utilizada em base de dados relacionais.

2.1.5.1 Document Object Model (DOM)

O DOM [W3C06c] modela um documento XML como uma árvore e define um método de acesso e manipulação dos seus objetos, ele permite que programas acessem e manipulem dinamicamente a estrutura e conteúdo dos documentos.

No modelo de dados do DOM, os elementos e atributos de um documento XML são representados por nós que se interligam conforme a descrição da modelagem, na *Figura 2.6* é apresentada a árvore para o documento "*cd.xsd*" apresentado na *Figura 2.5.*

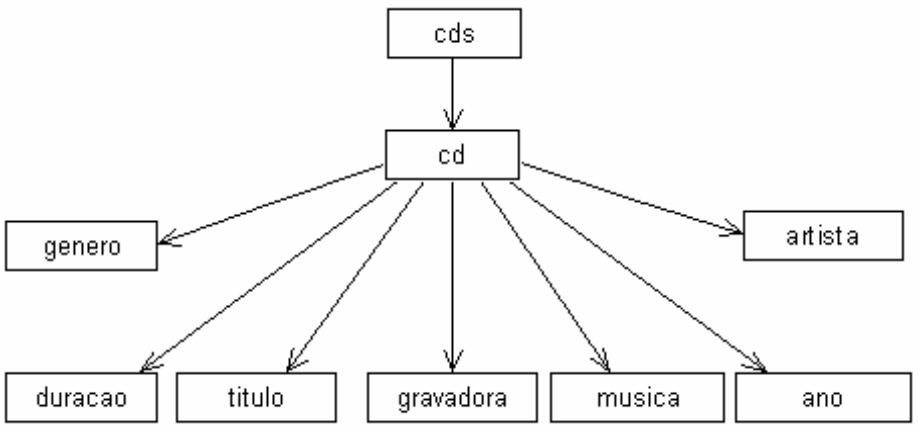


Figura 2.6 Visualização de um documento XML através do DOM.

2.1.5.2 XML Query Language (XQuery)

Visando a necessidade de consultas em documentos XML a W3C desenvolveu a linguagem XQuery [W3C06d], que é uma linguagem funcional com a sintaxe baseada em SQL para realizar consultas em informações que estão sendo representadas em XML.

Na *Figura 2.7* encontra-se um exemplo de consulta utilizando a sintaxe da XQuery. O resultado obtido é em cima do documento "*cd.xml*" apresentado na *Figura 2.1.* A consulta do exemplo retorna os valores encontrados para o elemento "*ano*" que sejam sub-elemento de "*cd*s*cd*\".

Consulta	**Resultado**
let $doc := document("cd.xml") for $i in $doc\\cds\cd\ano return <result>{$i}</result>	<result> <ano>1996</ano> <ano>1993</ano> <ano>1999</ano> <ano>2002</ano> <result>

Figura 2.7 Exemplo de Consulta em XQuery.

2.1.6 Qexo

O *Qexo* [BOT05] é uma implementação em Java que permite a utilização da linguagem *XQuery* para realizar consultas em documentos XML. Ela pode ser tanto utilizada via linha de comando onde a consulta é fornecida de forma interativa, como também pode ser utilizada como biblioteca, permitindo que outras aplicações manipulem as consultas.

Existem outras implementações que interpretam a linguagem XQuery, como: *FatDog, Microsoft XQuery, Quip, Galax* e *Tamino*, mas algumas características influenciaram na escolha do *Qexo* para este trabalho, tais como:

- Fácil utilização;
- Domínio público;
- Vasta documentação.

2.2 Dados de bases relacionais

Uma base de dados relacional pode ser vista como uma coleção de dados que estão logicamente relacionados, projetados para um propósito específico como a representação de algum aspecto do mundo real.

As interações com as bases de dados em geral são realizadas através do padrão SQL (*Structured Query Language*) [WIN06b], que é uma linguagem para definição, manutenção e consulta de informações. Os comandos da linguagem SQL dividem-se em subclasses. No contexto deste trabalho as que mais interessam são as DDL (*Data Definition Language*) e a DML (*Data Manipulation Language*).

2.2.1 Estrutura de base de dados

O esquema de uma base de dados relacional é representado pelo modelo conceitual que define os elementos, tipo de dados, restrições e relacionamentos.

Principais componentes do modelo conceitual:

- Elementos complexos ou tabelas - são objetos reais ou abstratos que representam algo do mundo real;
- Elementos simples ou atributos - são propriedades que representam informações que descrevem um elemento complexo. Cada elemento simples possui um valor que pode ser: um número, um caractere, uma data, uma imagem, um som, etc;
- Relacionamentos - são conexões entre dois ou mais elementos complexos;
- Restrições - são regras aplicadas aos elementos que impõem alguma condição de validação.

Os esquemas podem utilizar um conjunto de conceitos para descrever visualmente sua estrutura.

O MER (*Entity-Relationship Model*) [WIN06c] é um modelo conceitual que representa de uma forma direta as características de relacionamento dos dados [CHE76]. É utilizado no processo de planejamento da base de dados. Sua representação visual é dada pelo DER (Diagrama Entidade Relacionamento), que facilita a visualização da estrutura dos dados que se pretende armazenar.

O diagrama da *Figura 2.8* é um fragmento do DER apresentado no Apêndice A, onde:

- A associação "*Tem*" representa o relacionamento entre os elementos (Curso X Tipo_Curso),
- Cada instância do elemento *Tipo_Curso* pode se relacionar com "*N*" instâncias do elemento *Curso* e cada instância de *Curso* tem que se relacionar obrigatoriamente com uma instância do elemento *Tipo_Curso*;
- O elemento *Tipo_Curso* é formado pelos elementos simples (D*escrição e ID_Tipo_Curso*), onde *ID_Tipo_Curso* é a chave primária;
- O elemento *Curso* é formado pelos elementos simples (ID_Curso, ID_Tipo_Curso, Descrição e Nome) onde ID_Curso é a chave primária e ID_Tipo_Curso é uma chave estrangeira (a chave estrangeira não aparece diretamente no elemento, ela vem da associação "*Tem*").

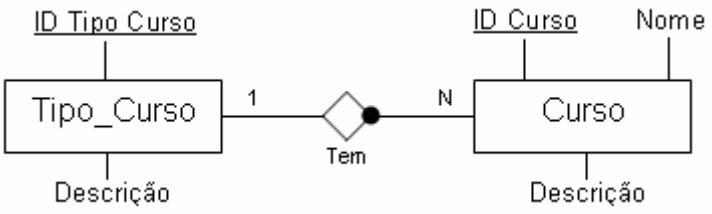


Figura 2.8 Fragmento de um Diagrama Entidade - Relacionamento.

O IDEF1X (*Integration DEFinition*) [WIN06d] é geralmente utilizado por ferramentas CASE, para desenvolver e implementar modelos de dados lógicos de um banco de dados. Está baseado no Modelo Entidade-Relacionamento e produz um modelo gráfico e lógico das informações, representando, assim, a

estrutura e a semântica da informação dentro de uma organização ou de um sistema específico.

O diagrama IDEF1X apresentado na *Figura 2.9* modela os mesmos elementos e relacionamentos vistos no diagrama da *Figura 2.8, o* IDEF1X também representa informações que não aparecem no DER, como o tipo de dado dos elementos simples (*varchar, int...*) e em respeito ao uso do elemento se ele é obrigatório ou não (*null, not null*).

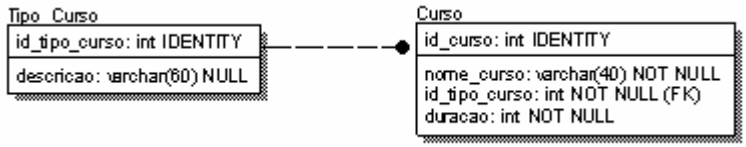


Figura 2.9 Fragmento de um diagrama IDEF1X

2.2.2 Definição de esquemas e restrições

Os esquemas para bases de dados relacionais são geralmente escritos utilizando-se a DDL (*Data Definition Language*). Essa linguagem permite a definição e alteração dos elementos de um esquema, das suas associações e restrições.

A declaração apresentada na *Figura 2.10* utiliza a DDL da base de dados PostGre [POS06] para especificar o elemento "*curso*". A descrição dos principais comandos é apresentada na *Tabela 2.4*.

```
CREATE TABLE conex.curso
(
id_curso int4 NOT NULL DEFAULT,
nome_curso varchar(40) NOT NULL,
id_tipo_curso int4 NOT NULL,
duracao int4 NOT NULL,
CONSTRAINT curso_pkey PRIMARY KEY (id_curso),
CONSTRAINT curso_id_tipo_curso_fkey FOREIGN KEY (id_tipo_curso)
REFERENCES conex.tipo_curso (id_tipo_curso) MATCH SIMPLE
ON UPDATE NO ACTION ON DELETE NO ACTION
)
```

Figura 2.10 PostGre SQL Schema

Tabela 2.4 Descrição dos comandos utilizados na Figura 2.10

Fragmento	Descrição
CREATE TABLE	CREATE TABLE: cria um novo elemento complexo.
Id_curso int4 NOT NULL	int4: tipo de dado NOT NULL: especifica que o elemento não pode ser nulo.
Nome_curso varchar(40) NOT NULL	varchar(40): tipo de dado.
CONSTRAINT curso_pkey PRIMARY KEY (id_curso)	Referencia os elementos simples que são chaves primárias do elemento complexo.
CONSTRAINT curso_id_tipo_curso_fkey FOREIGN KEY (id_tipo_curso)	Referencia os elementos simples que associam o elemento curso a outro elemento complexo.

Algumas restrições como tipo do elemento ou número de caracteres podem ser definidas juntamente com a declaração dos elementos, já restrições mais complexas que envolvem o valor de um elemento ou uma condição de ocorrência são declaradas através de uma *trigger* ou *check.*

A restrição apresentada na *Figura 2.11* é um exemplo de *check*, esse tipo de restrição envolve apenas os valores de uma única instância do elemento complexo. No exemplo apresentado é imposto que o elemento simples *"máximo"* tem que ser maior que o elemento "*mínimo*".

```
CONSTRAINT faixa_salarial_check_faixa_salarial CHECK (maximo > minimo);
```

Figura 2.11 Exemplo de restrição check

As restrições declaradas por uma *trigger* são mais abrangentes, elas podem envolver valores de outras instâncias ou de outros elementos complexos. No exemplo apresentado na *Figura 2.12 se* obriga que não podem ocorrer instâncias do elemento complexo "*aluno*" com identificadores já usados pelo elemento "*instituição*".

```
CREATE OR REPLACE FUNCTION pessoaaluno() RETURNS "trigger" AS
DECLARE   registro record;
BEGIN

  SELECT INTO registro count(*) as qtd
      FROM instituicao
      where instituicao.id_instituicao = NEW.id_aluno;

 IF registro.qtd > 0 THEN
   RAISE EXCEPTION 'já existe uma instituição com este código';
 END IF;

 RETURN NULL;

END;

LANGUAGE 'plpgsql' VOLATILE;
```

Figura 2.12 Exemplo de restrição trigger

2.2.3 Linguagem de consulta para Base de Dados

As consultas em uma base de dados são escritas utilizando o comando "*SELECT"* da DML (*Data Manipulation Language*) que permite ao usuário especificar uma consulta (*query*) que contenha as especificações do resultado desejado.

A *query* apresentada na *Figura 2.13* é referente a uma consulta para a modelagem apresentada na *Figura 2.8* O resultado retornado é a junção das instâncias do elemento *curso* com a instância relacionada em "*tipo_curso"*.

```
SELECT curso.id_curso, curso.nome, tipo_curso.descrição
FROM curso
INNER JOIN tipo_curso on curso._id_tipocurso = tipo_Curso.id_tipo_curso
```

Figura 2.13 Exemplo de consulta em uma base de dados.

Na *Tabela 2.5* é apresentada a descrição dos comandos utilizados pela consulta apresentada na *Figura 2.13.*

Tabela 2.5 Descrição dos comandos utilizados na consulta.

Comando	Descrição
SELECT	Informa quais atributos se deseja no resultado de uma consulta;
FROM	Informa em quais tabelas estão os atributos que estão sendo usados;
INNER JOIN	Faz a junção entre tabelas através de atributos que têm um relacionamento.

2.2.4 JDBC

JDBC (*Java Database Connectivity)* [SUN06a] é um conjunto de classes escritas em Java que faz a comunicação entre uma aplicação e um banco de dados relacional através da troca de instruções SQL. Para cada banco de dados há um *driver* JDBC específico.

3 Teste de Software

Teste de software é uma etapa do ciclo de desenvolvimento de software que busca alcançar padrões de qualidade no produto criado. O teste diz respeito à análise dinâmica do programa e consiste no processo de executar um sistema com o objetivo de revelar defeitos ainda não descobertos [MYE79].

Embora durante todo o processo de desenvolvimento de software sejam utilizadas técnicas, métodos e ferramentas a fim de evitar que erros sejam introduzidos no produto, a atividade de teste continua sendo de fundamental importância para a eliminação dos erros que persistem [MAL91].

Testes deveriam ser executados com todas as combinações possíveis do domínio de entrada. Entretanto, sabe-se que na maioria das vezes, o teste exaustivo é impraticável devido às restrições de tempo e custo. Desse modo é necessário determinar quais casos de teste utilizar a fim de que a maioria dos defeitos existentes possa ser encontrada, dispondo do mínimo de esforço.

Para que casos de teste com alta probabilidade de revelar defeitos sejam executados utilizam-se vários critérios com diferentes aspectos. Esses critérios fazem uma seleção dos dados de entrada para o programa, avaliando um determinado conjunto de casos de teste. As principais técnicas utilizadas são: estrutural, funcional e baseado em erros.

3.1 Teste Funcional

O teste funcional ou caixa preta tem esse nome pelo fato de tratar o software como uma caixa do qual o conteúdo é desconhecido e só é possível visualizar o lado externo [PRE00].

Os casos de teste utilizados nesta técnica são derivados a partir da especificação do sistema. Consideram-se apenas as entradas e saídas, ignorando os detalhes de implementação (código fonte do software). Assim, uma especificação correta e de acordo com os requisitos do usuário é essencial para esse tipo de teste.

Dentre os principais critérios funcionais destacam-se o Particionamento em Classes de Equivalência e a Análise do Valor Limite [PRE00].

3.2 Teste Estrutural

A técnica de teste estrutural utiliza a estrutura de controle e fluxo de dados para derivar os requisitos de teste [PRE00]. De forma geral, os critérios dessa técnica utilizam uma representação de programa conhecida como Grafo de Fluxo de Controle.

Um grafo de fluxo de controle é um grafo dirigido que possui um único nó de entrada e um único nó de saída. Para a construção de um grafo desse tipo, o programa alvo é dividido em diversos blocos distintos de comandos, que possuem duas características:

- Seus comandos são executados de forma atômica, ou seja, uma vez que o primeiro comando do bloco é executado, todos os demais são executados sequencialmente;
- Não existe desvio de execução para nenhum comando dentro do bloco. A conexão entre os blocos é feita por intermédio de arcos que indicam os possíveis fluxos de controle entre os blocos.

Os critérios estruturais baseiam-se em diferentes conceitos e componentes de programas para derivar os requisitos de teste. De forma geral, os critérios da técnica estrutural são: baseados em fluxo de controle, baseados em fluxo de dados.

Baseados em Fluxo de Controle: Esses critérios utilizam apenas características de controle da execução do programa, como comandos ou desvios, para derivar os requisitos de teste necessários. Os critérios mais conhecidos desta classe são:

- *Todos-os-nós*: exige que a execução do programa passe menos uma vez em cada nó do grafo, ou seja, que cada comando do programa seja executado pelo menos uma vez;

- *Todos-os-arcos*: requer que cada arco do grafo, cada desvio do programa, seja executado pelo menos uma vez;
- *Todos-os-caminhos*: requer que todos os caminhos do grafo de programa sejam executados.

Baseados em Fluxo de Dados: utiliza informações do fluxo de dados do programa para determinar os requisitos de teste. Esses critérios exploram as interações que envolvem definições de variáveis e referências a tais definições [RAP85]. Nessa classe destacam-se os critérios todos-os-usos [RAP85] e todos-potenciais-usos [MAL91].

3.3 Teste Baseado em Defeitos

O teste baseado em defeitos utiliza informações sobre os erros mais freqüentes cometidos no processo de desenvolvimento de software e sobre os tipos específicos de defeitos que se deseja revelar [DeM87]. Destacam-se três critérios de teste baseado em defeitos, que são:

- **Semeadura de Defeitos:** nesse critério, uma quantidade conhecida de defeitos é semeada artificialmente no programa. Após o teste, do total de defeitos encontrados verificam-se quais são naturais e quais são artificiais. Usando estimativas de probabilidade, o número de defeitos naturais ainda existentes no programa pode ser calculado;
- **Análise de Mutantes**: é um critério que utiliza um conjunto de programas ligeiramente modificados chamados de mutantes, obtidos a partir de um determinado programa P em teste. O objetivo é avaliar o quanto um conjunto de casos de teste T é adequado para o teste de P. O objetivo é encontrar um conjunto de casos de teste capaz de revelar as diferenças de comportamento existentes entre P e seus mutantes [DeM78];
- **Perturbação de dados:** é uma técnica que utiliza uma entrada de dados de um determinado sistema para gerar um conjunto de outras entradas utilizando operadores de perturbação que modificam o dado original. Ela

tem sido atualmente utilizada no teste de interação entre componentes Web, tais como *Web Services*, para modificar documentos XML e mensagens SOAP [OFF04][WUZ05].

3.4 Teste de esquemas

Como dito anteriormente, os esquemas são responsáveis por definir e organizar a estrutura dos dados de uma aplicação bem como o relacionamento entre os mesmos. Testar o esquema de uma aplicação contribui para assegurar a integridade e precisão dos dados por ela manipulados.

Na literatura, a maioria dos trabalhos encontrados visa o teste das aplicações que manipulam os dados. Poucos são os trabalhos que focalizam o teste de esquemas. Nessa seção, esses trabalhos são descritos. Primeiramente, é descrita em detalhes a abordagem de Emer [EME05] que é o foco da ferramenta X-Tool. Essa abordagem permite o teste de esquemas em geral e está baseada em classes de defeitos comuns a esses esquemas. Após isso, alguns trabalhos relacionados e ferramentas similares são discutidos.

3.4.1 Abordagem de teste de Emer [EME07b]

Emer propõe uma abordagem de teste de esquemas de estruturas de dados complexas baseada em defeitos. Para isso, foram propostas algumas classes de defeitos que descrevem os principais erros que podem acontecer ao se projetar esquemas.

Os tipos de defeitos que podem ser revelados por essa abordagem estão relacionados à ausência de restrições necessárias ou da definição incorreta de uma restrição de determinado dado. Esses defeitos refletem erros que podem ter sido cometidos durante o desenvolvimento do esquema ou durante a evolução do mesmo (atualização do esquema devido às alterações na estrutura de dados da aplicação).

As classes de defeito propostas por Emer, juntamente com suas descrições são apresentadas na *Tabela 3.1*.

Na *Figura 3.1* é apresentado o processo de teste proposto por Emer que tem como entrada os campos “esquema dos dados” e “dados originais”. O primeiro se refere ao esquema que contém as regras que validam os dados para uma determinada especificação e o segundo contém dados válidos para a especificação.

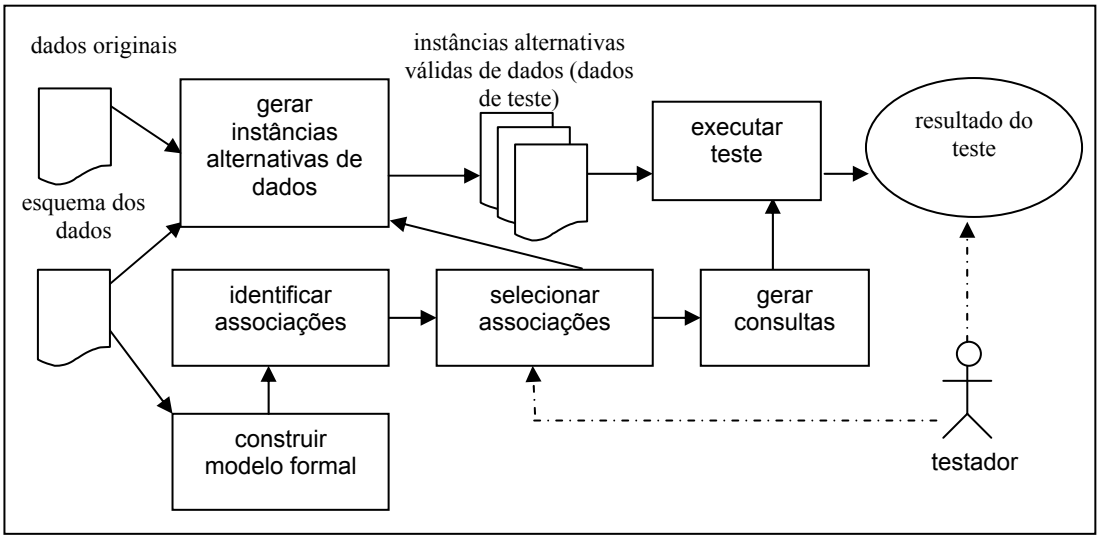


Figura 3.1 Processo de teste de esquema.

Para identificar as classes de defeitos presentes no esquema utiliza-se um modelo formal, do qual podem ser extraídos os conjuntos de elementos, atributos e restrições do esquema.

Neste modelo um esquema é representado por S = *(E,A,R,P)*, onde:

- E é um conjunto finito de elementos.
- A é um conjunto finito de atributos**.**
- R é um conjunto finito de restrições referentes a domínio, relacionamento e conteúdo associadas a elementos de E ou a atributos de A.
- P é um conjunto finito de regras de relacionamento, que podem ser representadas pelas notações:
    - $x(r_1, r_2 ..., r_n)$, onde x é um elemento em E ou x é um atributo em A, $r1$, r_2, r_n são restrições em R para x, onde n é o número de restrições.

- x($r1{:}m_1,m_2...m_n$), onde x, m_1, m_2, m_n são elementos em E e $r1$ é uma restrição em R que relaciona x a m_1, m_2, m_n, onde n é o número de elementos ou atributos.

Tabela 3.1 Classe e tipos de defeitos.

Classe de defeito		Descrição
Restrições de domínio		
TDI	Tipo de dado incorreto	Definição do tipo de dado atribuído ao elemento
VI	Valor incorreto	Valor padrão ou fixo definido para o elemento.
VEI	Valores enumerados incorretos	Conjunto de valores permitidos
VMMI	Valores máximos e mínimos incorretos	Valores limites definidos para o elemento.
LCI	Limite de caracteres (tamanho) incorreto	Número de caracteres máximos e mínimos permitidos para um elemento.
LDI	Limite de dígitos incorreto	Número de dígitos máximos e mínimos permitidos para um elemento.
MVI	Modelo de valores incorreto	Definição da seqüência de valores permitidos.
TEBI	Tratamento de espaço em branco incorreto	Tipos especiais de espaços em branco permitidos.
Restrições de referências		
UI	Uso incorreto	Definição do uso do elemento como opcional ou obrigatório.
UNI	Unicidade incorreta	Definição para o número de ocorrências permitidas para um elemento.
II	Identificador incorreto	Definição de uso e unicidade para os elementos que compõem o identificar de um elemento complexo.
Restrições de relacionamento		
OI	Ocorrência incorreta	Número máximo e mínimo de repetições do elemento incorreto
CI	Cardinalidade incorreta	Numero de ocorrências máximas e mínimas para o relacionamento entre dois elementos.
ORI	Ordem incorreta	Definição da ordem de ocorrência para os filhos de um elemento, se a ordem dos mesmos é obrigatória ou não.
EAI	Elemento associativo incorreto	Definição de restrições na ocorrência de uma associação entre dois elementos em relação à ocorrência da associação entre outros elementos.
Restrições semânticas		
COI	Condição incorreta	Definição de restrições do conteúdo de um elemento em relação ao conteúdo de outro elemento.

Na *Figura 3.2* é apresentado o Modelo Formal para o esquema da *Figura 2.5.*

E : (cds, cd, titulo, artista, musica,gravadora, ano)
A : (genero, duracao)
R : (complexType, maxOccurs, minOccurs, type, use)
P :(
cds(n1 complexType)
cd(n2 n3 n4 n5 n6 n7 n8 complexType, maxOccurs)
titulo(minOccurs, maxOccurs, type)
artista(minOccurs, maxOccurs, type)
musica(minOccurs, maxOccurs, type)
gravadora(minOccurs, maxOccurs, type)
ano(minOccurs, maxOccurs, type)
genero(use, type)
duracao(use, type)
)

Figura 3.2 Modelo Formal.

Para cada associação entre um elemento e uma restrição selecionada baseada nas classes de defeitos identificadas pelo modelo formal ou selecionada pelo testador, são geradas as associações de defeito, as quais aplicam alterações simples nos dados de entrada. Essas alterações estão baseadas nas classes de defeito e geram instâncias alternativas de dados, as quais são validadas, ou seja, são submetidas a um teste de validação para verificar se as mesmas seguem as regras estabelecidas pelo esquema. As instâncias que foram consideradas válidas pelo esquema são utilizadas como entrada em consultas específicas, geradas para todas as classes de defeito. Com a análise dos resultados obtidos nas consultas são revelados os possíveis defeitos no esquema, com relação a sua especificação.

No processo de teste, o esquema e os dados de entrada são fornecidos pelo testador. A representação *S* refere-se ao esquema em teste. Baseando-se em *S*, os elementos, os atributos e os relacionamentos entre restrições e elementos ou atributos são associados às classes de defeitos, previamente definidas. Em seguida, estas associações são selecionadas. A seleção das associações pode ser feita automaticamente ou pelo testador que também pode identificar outras associações entre os componentes do esquema (elementos, atributos, relacionamentos entre elementos restrições relacionadas aos elementos ou atributos) e classes de defeito. Isso possibilita a detecção de defeitos que poderiam não ser detectados através das associações, identificadas na

representação *S*, isto é, revelar os defeitos relacionados à ausência da definição de restrições.

Os dados alternativos são gerados através da modificação dos dados originais. Estas modificações simples são feitas através de inserções ou modificações nos dados originais de acordo com os padrões definidos para as classes de defeito. As associações selecionadas são usadas para guiar a geração dos dados alternativos, indicando o elemento do esquema, o dado original que deve ser modificado e a classe de defeito que define as modificações que devem ser aplicadas. Assim, as alternativas representam possíveis defeitos no esquema.

Os dados alternativos gerados são separados em válidos e inválidos com relação ao esquema em teste; isto é, uma alternativa gerada não pode estar fora das especificações exigidas pelo esquema em teste. As alternativas inválidas não são consultadas.

As associações selecionadas também guiam a geração de consultas, que são geradas automaticamente de acordo com os testes associados a cada classe de defeito. As consultas para cada associação selecionada são geradas e executadas apenas para os dados alternativos válidos.

Os dados de teste são formados pelos dados alternativos válidos e pela consulta a estas alternativas. A especificação do resultado esperado para as alternativas é obtida através da especificação dos dados. O testador compara os resultados obtidos pelo teste com a especificação para descobrir defeitos no esquema de dados.

3.4.2 Trabalhos Relacionados

A abordagem de teste de Emer, descrita na seção anterior, permite o teste de esquemas de uma forma genérica. Ela foi instanciada no contexto de documentos XML [EME05] e de bases de dados relacionais [EME07a]. Na literatura, encontram-se trabalhos que possuem objetivos similares, mas que, entretanto não abordam ambos os tipos de contexto.

Aranha et al [ARA00] implementaram uma ferramenta chamada *RDBTool* com o objetivo de realizar o teste estrutural de esquemas de bases de dados relacionais. A idéia é gerar e executar consultas em uma base de dados para exercitar definições e usos de atributos de elementos presentes no esquema. No contexto de XML, destacam-se os trabalhos de Li e Muller [LI05] e Franzotte e Vergilio [FRA06]. Ambos os trabalhos têm como objetivo o teste baseado em defeitos de esquemas escritos em *XML Schema*. A idéia é aplicar o critério Análise de Mutantes ao esquema. Franzotte e Vergilio [FRA06] introduziram um conjunto de operadores de mutação que inclui os operadores de Li e Muller. Esses operadores estão descritos na *Tabela 3.2*.

Tabela 3.2 Operadores de Franzotte e Vergílio [FRA06].

Operador		**Descrição**
GO	Group Order	Altera a ordem na quais os elementos devem estar no grupo.
REQ	Required	Altera o tipo da obrigatoriedade dos atributos.
DT	Data Type	Altera o tipo dos elementos/atributos. O domínio do operador são todos os tipos definidos, por exemplo, {integer, string, float, decimal, int, enumeration}
LO	Length Of	Altera o tamanho do nome dos elementos.
CSP	Change Sing Plural	Altera o tamanho do elemento adicionando ou removendo os caracteres.
CTP	Change Tag	Altera as tags dos nós
SO	Size Occurs	Alteram-se os tamanhos das ocorrências máximas e/ou mínimas tanto dos tipos fixos {strings} quanto dos tipos criados pelo usuário.
STE	Sub Tree Exchange	Inverte as sub-arvores abaixo de algum nó
IT	Insert Tree	Insere um nó (ou sub-árvore) na estrutura da árvore.
DT	Remove Tree	Remove o nó (ou sub-árvore) da estrutura da árvore.

Para permitir a aplicação desses operadores, uma ferramenta, chamada XTM [FRA06] foi implementada. Para um esquema em teste, XTM gera diversos esquemas mutantes. O testador deve manualmente gerar um documento XML, que será o caso de teste, que faça com que o esquema mutante se comporte diferentemente do esquema original. O comportamento do esquema é validar ou não o documento XML. Similarmente ao teste tradicional, poderão existir esquemas equivalentes que devem ser identificados manualmente.

Embora a abordagem de Emer e de Franzotte e Vergilio sejam ambas baseadas em defeitos e algumas de suas classes de defeitos descrevam o mesmo tipo de defeito, existem algumas diferenças entre elas. A abordagem de Franzotte e Vergilio permite somente o teste de esquemas *XML* não pode ser aplicada no contexto de bases de dados relacionais. Para Franzotte e Vergílio, o dado de teste é um documento XML a ser validado ou não pelo esquema. Esse caso de teste é gerado manualmente pelo testador. A XTM apenas valida ou não o documento e produz a cobertura. Automatizar a geração desses dados de teste apresenta inúmeras restrições e não pode ser completamente realizada.

Na abordagem de Emer um dado de teste é formado por um conjunto de documentos XML e de consultas a esses documentos. A vantagem é que a geração dos dados de teste pode ser completamente automatizada. A automatização dessa abordagem é o objetivo desse trabalho e será tratado no próximo capítulo que descreve a ferramenta X-Tool para permitir o teste de esquema em ambos os contextos. Aspectos de avaliação da abordagem com relação ao teste de mutação e operadores de Franzotte e Vergílio são apresentados no Capítulo 5.

4 A Ferramenta X-Tool

Neste capítulo é descrita a ferramenta denominada X-Tool (*XML and Relational Database Schema Testing Tool*), que provê suporte automático à abordagem proposta por Emer [EME07b], visando revelar os defeitos cometidos com maior freqüência na definição incorreta de um esquema.

A X-Tool possibilita o teste de esquemas XML que sigam o padrão *XML Schema* e também esquemas para base de dados relacionais escritas para o SGBD *PostGre.*

A ferramenta foi implementada utilizando a linguagem Java com o emprego da API (*Applications Programming Interface*) DOM (*Document Object Model*) [W3C06c] usada para manipular e validar o conteúdo de documentos XML, do *Qexo* [BOT05] que permite consultar documentos XML por meio da linguagem XQuery (*XML Query*) [W3C06d] e do *JDBC (Java Database Connectivity)* [SUN06a] usado para manipular e consultar informações em bases de dados PostGre através das linguagens DQL (Linguagem de Consulta de Dados) e DML (Linguagem de Manipulação de Dados) que são subconjuntos da SQL (*Structured Query Language*) [WIN06b].

4.1 Arquitetura da ferramenta

A arquitetura da ferramenta e a comunicação entre os seus principais módulos são apresentadas na *Figura 4.1*.

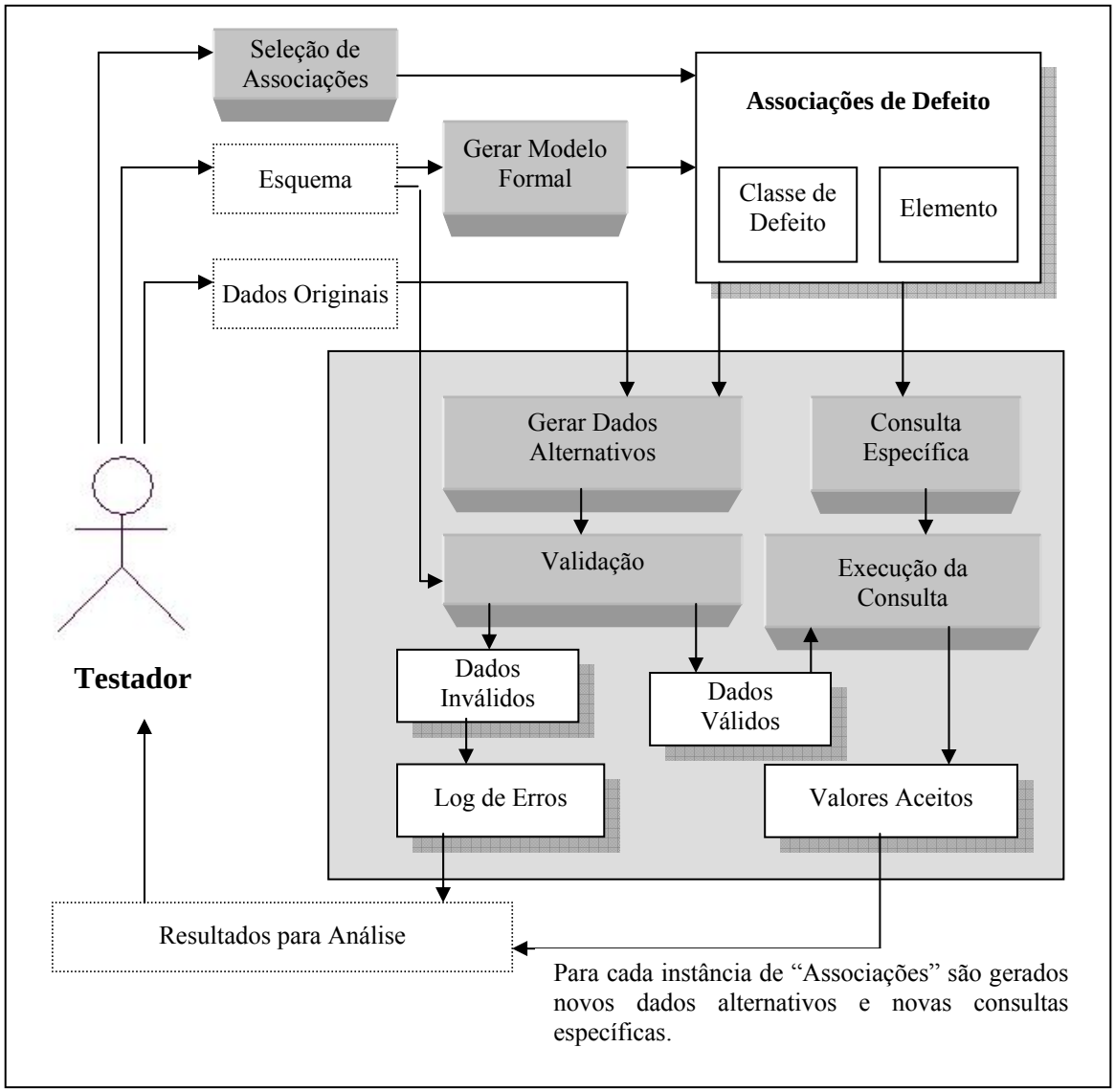


Figura 4.1 Arquitetura da ferramenta.

Descrição dos itens da Figura 4.1

- *Esquema* - regras que validam os dados, segundo uma especificação e que será testado pela X-Tool.
- *Dados Originais* (ou *Dados de Entrada)* - dados válidos para o esquema.
- *Modelo Formal* - associação dos elementos e das restrições encontrados no esquema.
- *Gerar Modelo Formal* - procedimento do sistema que extrai os elementos e as restrições de um esquema e gera o modelo formal.

- *Seleção de associações* - procedimento manual realizado pelo testador, que associa elementos e classes de defeito.
- *Associações de Defeito* ou *Associações* - conjunto de associações entre elementos e classes de defeito.
- *Consulta Específica* - consulta na linguagem XQuery ou SQL que tem como objetivo extrair informações específicas, relacionadas ao elemento e à classe de defeito.
- *Execução da consulta* - Procedimento do sistema, que aplica a consulta especifica em cima dos dados válidos.
- *Gerar Dados Alternativos* - Procedimento do sistema, que gera novos dados, baseados nos dados originais com pequenas alterações definidas pela classe de defeito.
- *Validação* - Procedimento do sistema que valida os dados alternativos segundo o esquema, os dados que seguem as regras do esquema são considerados válidos e o que não obedecem são ditos como inválidos.
- *Valores Aceitos* - É o resultado da consulta específica em cima dos dados válidos.
- *Dados Inválidos* - São os dados alternativos que não obedeceram a alguma regra do esquema.
- *Dados Válidos* - São os dados alternativos que obedecem às regras do esquema.
- *Log de Erros* - Relatório contendo o motivo pelo quais os esquemas inválidos foram rejeitados pelo esquema.
- *Resultados para Análise* - É formado pelos valores válidos e log de erros gerados para todas as associações.

A X-Tool é iniciada pelo testador usando como parâmetro de entrada o esquema a ser testado e um conjunto de dados válidos para o esquema.

O esquema é então transformado para o seu modelo formal correspondente. A partir desse modelo a ferramenta associa os elementos com as respectivas classes de defeitos, gerando as associações de defeito. O testador também poderá indicar outras associações, que não foram selecionadas automaticamente pela ferramenta.

As associações de defeito são usadas para conduzir a geração dos dados alternativos. Esses dados alternativos são gerados produzindo pequenas variações nos dados de entrada originais que procuram expor defeitos descritos pelas classes de defeito.

As alterações realizadas para cada classe de defeito em cima dos dados de entrada são apresentadas na *Tabela* 4.1. A descrição para essas alterações são apresentadas na *Tabela 4.2.*

A *Figura 4.2* apresenta um exemplo contendo dados alternativos gerados para uma associação de defeito entre o elemento "*ano"* e a classe de defeito "*Valores Máximos e Mínimos Incorretos",* vista na *Tabela 4.1.* Os dados alternativos gerados contêm as alterações descritas por "*VMMI_ESN"* apresentadas na *Tabela 4.2*

Tabela 4.1 Alterações realizadas para cada classe de defeito

Restrições de definição			
Tipo de Defeito	**Classe de defeito**	**Tipo elemento**	**Descrição**
UI	Uso incorreto	ES	UI_ES
		EC	UI_EC
UNI	Unicidade incorreta	ES	UNI_ES
		EC	UNI_EC
II	Identificador incorreto	EC	UI_ES UNI_EC
Restrições de domínio			
Tipo de Defeito	**Classe de defeito**	**Tipo elemento**	**Descrição**
TDI	Tipo de dado incorreto	ES	TDI_ES
VI	Valor incorreto	ES	TDI_ES LCI_ES UI_ES
VEI	Valores enumerados incorretos	ES	TDI_ES LCI_ES
VMMI	Valores máximos e mínimos incorretos	ESN	LCI_ES VMMI_ESN
		ESD	VMMI_ESD
LCI	Limite de caracteres (tamanho) incorreto	ES	LCI_ES
LDI	Limite de dígitos incorreto	ES	LCI_ES
MVI	Modelo de valores incorreto	ES	TDI_ES LCI_ES
TEBI	Tratamento de espaço em branco incorreto	ES	TEBI_ES
Restrições de relacionamento			
Tipo de Defeito	**Classe de defeito**	**Tipo elemento**	**Descrição**
OI	Ocorrência incorreta	ES	OI_ES
		EC	OI_EC
ORI	Ordem incorreta	EC	ORI_EC
CI	Cardinalidade incorreta	EC	CI_EC
EAI	Elemento associativo incorreto	EC	EAI_EC
Restrições semânticas			
Tipo de Defeito	**Classe de defeito**	**Tipo elemento**	**Descrição**
COI	Condição incorreta	ESN	COI_ESN
		ESD	COI_ESD

EC – Elemento complexo ou composto.
ES – Elemento simples.
ESN – Elemento simples numérico.
ESD – Elemento simples data.

Tabela 4.2 Descrição das alterações associadas a cada classe de defeito.

Descrição	Alterações realizadas
TDI_ES	Substitui o valor original de cada instância do elemento pelos valores (“string”, “12345”, “-1234”, “1234.56”, “abc1234”, “2005-10-22”, “12:52:32”).
VMMI_ESN	Substitui o valor original de cada instância do elemento pelos valores (0, 2147483647, -2147483648, metade do valor, dobro do valor, converte todos os dígitos para o dígito 9).
VMMI_ESD	Substitui o valor original de cada instância de E1 pelos valores (data + 1 dia, data + 1 mês, data + 1 ano, data + 3 dias, data + 3 meses, data + 3 anos, data – 1 dia, data – 1 mês, data – 1 ano, data – 3 dias, data – 3 meses, data – 3 anos)
LCI_ES	Substitui o valor original de cada instância do elemento pelos valores com: • o dobro do número de caracteres ou dígitos; • metade do número de caracteres ou dígitos.
TEBI_ES	Substitui o valor original de cada instância do elemento pelos valores com: • insere espaço; • duplica a quantidade de espaços; • substitui espaço por TAB; • substitui espaço por quebra de linha; • somente espaço.
UI_ES	Substitui o valor original de cada instância do elemento pelos valores com: • valor nulo.
UI_EC	Exclui todos os elementos.
UNI_ES	Duplica elemento pai, alterando a chave primária se o elemento não pertencer à mesma.
UNI_EC	Duplica os elementos.
ORI_EC	• Inverte ordem dos elementos; • Deixa somente um elemento.
CI_EC	• Sendo (E1 e E2) elementos complexos que possuem uma restrição de associação, associa-se todas as instâncias de E2 a uma instância de E1. • Remove todos os elementos associados.
EAI_EC	• Sendo (E1 e E2, E3) elementos complexos, onde e1 se relaciona com E2 e E2 se relaciona com E3, inserem-se ocorrências do relacionamento (E2 e E3) no relacionamento de (E1 e E2). •Remove todos os elementos associados. De (E2 e E3).
COI_ESN	Substitui o valor original de cada instância de E1 pelos valores do teste de VMMI_ESN em cima de E2.
COI_ESD	Substitui o valor original de cada instância de E1 pelos valores do teste de VMMI_ESD em cima de E2.

<table>
<tr><th colspan="2">Associação de Defeito</th></tr>
<tr><td colspan="2">Raiz: \\cds\cd
Elemento: ano
Classe de defeito: Valores máximos e mínimos incorretos</td></tr>
<tr><th>Elemento Original</th><th>Dados Alternativos</th></tr>
<tr><td><cds>
<cd>
<titulo>Live</titulo>
<artista>Kenny G</artista>
<musica>Going Home</musica>
<musica>Sade</musica>
<musica>Silhouette</musica>
<gravadora>BMG</gravadora>
<ano>1993</ano>
</cd>
</cds></td><td><dados_alternativos>
<cds>
<cd>
..
<ano>0</ano>
..
</cd>
</cds>
..<ano>2147483647</ano>..
..<ano>-2147483648</ano>..
..<ano>996</ano>..
..<ano>3986</ano>..
..<ano>9999<ano>..
</dados_alternativos></td></tr>
</table>

Figura 4.2 Exemplo de Associação de Defeito e Dados Alternativos.

Em cima dos dados alternativos gerados são realizadas consultas específicas, os resultados obtidos fornecem informações que são analisadas pelo testador com o intuito de revelar defeitos referentes à classe de defeito proposta pelo teste. As descrições dos valores obtidos pelas consultas para cada classe de defeito são apresentadas na *Tabela 4.3.*

Tabela 4.3 Resultados obtidos pelas classes de defeito.

Classe de Defeito	Valores Retornados
Tipo de Dado Incorreto	Tipos de dados que são aceitos pelo elemento.
Valor Incorreto	Valores que são aceitos pelo elemento.
Valores Enumerados Incorreto	Lista de valores aceitáveis pelo elemento.
Valores Máximos e Mínimos Incorretos	Valores que estão dentro do limite inferior e superior para o elemento.
Limite de Caracteres Incorreto	Número de caracteres ou dígitos permitidos para o elemento.
Limite de Dígitos Incorreto	Número de caracteres ou dígitos permitidos para o elemento.
Modelo de Valores Incorreto	Seqüência de valores permitidos para o elemento.
Tratamento de Espaço em Branco Incorreto	Mostra a representação de espaços em branco permitidos para o elemento.
Uso Incorreto	ES – Valores que são aceitos pelo elemento.
	EC – Quantidade de instâncias encontradas para o elemento.
Unicidade Incorreta	Valores e quantidade de ocorrências encontradas para o elemento.
Identificador Incorreto	Valores aceitos para o conjunto de identificadores do elemento.
Ocorrência Incorreta	ES – Valores e quantidade de ocorrências encontradas para o elemento.
	EC – Numero de instâncias encontradas para o elemento.
Ordem Incorreta	Retorna os elementos filhos na ordem que se encontram.
Cardinalidade Incorreta	Número de instâncias encontradas do elemento que estão relacionadas a uma instância de outro elemento, e os valores que fazem o relacionamento.
Elemento Associativo Incorreto	Número de instâncias encontradas do elemento que estão relacionadas a um relacionamento entre outros dois elementos, e os valores que fazem o relacionamento.
Condição Incorreta	Valores de um elemento que se encontram relacionados a valores de outro elemento.

EC – Elemento Complexo (composto por outros elementos).
ES – Elemento Simples.

A *Tabela 4.4* apresenta um exemplo de consulta realizada pela linguagem *XQuery*. Como dados de entrada foram utilizados os dados alternativos apresentados na *Figura 4.2*.

Tabela 4.4 Consulta para os dados alternativos.

Consulta	Resultado
let $doc := document("dados_alternativos.xml") for $i in $doc\\ dados_alternativos \cds\cd\ano Return <result>{$i}</result>	<result> <ano>0</ano> <ano>2147483647</ano> <ano>-2147483648</ano> <ano>996</ano> <ano>3986</ano> <ano>9999<ano> <result>

Os resultados obtidos pelas consultas aos dados alternativos formam um relatório para o testador que os compara com os resultados esperados segundo a especificação do esquema, a fim de levantar as definições incorretas ou a ausência de restrições no esquema.

4.2 Diagrama da Ferramenta

O teste de esquema usando a X-Tool é feito a partir de um projeto, o qual armazena todos os dados necessários durante sua execução. *A* representação desses dados é feita pelo diagrama de classe apresentado na *Figura 4.3*.

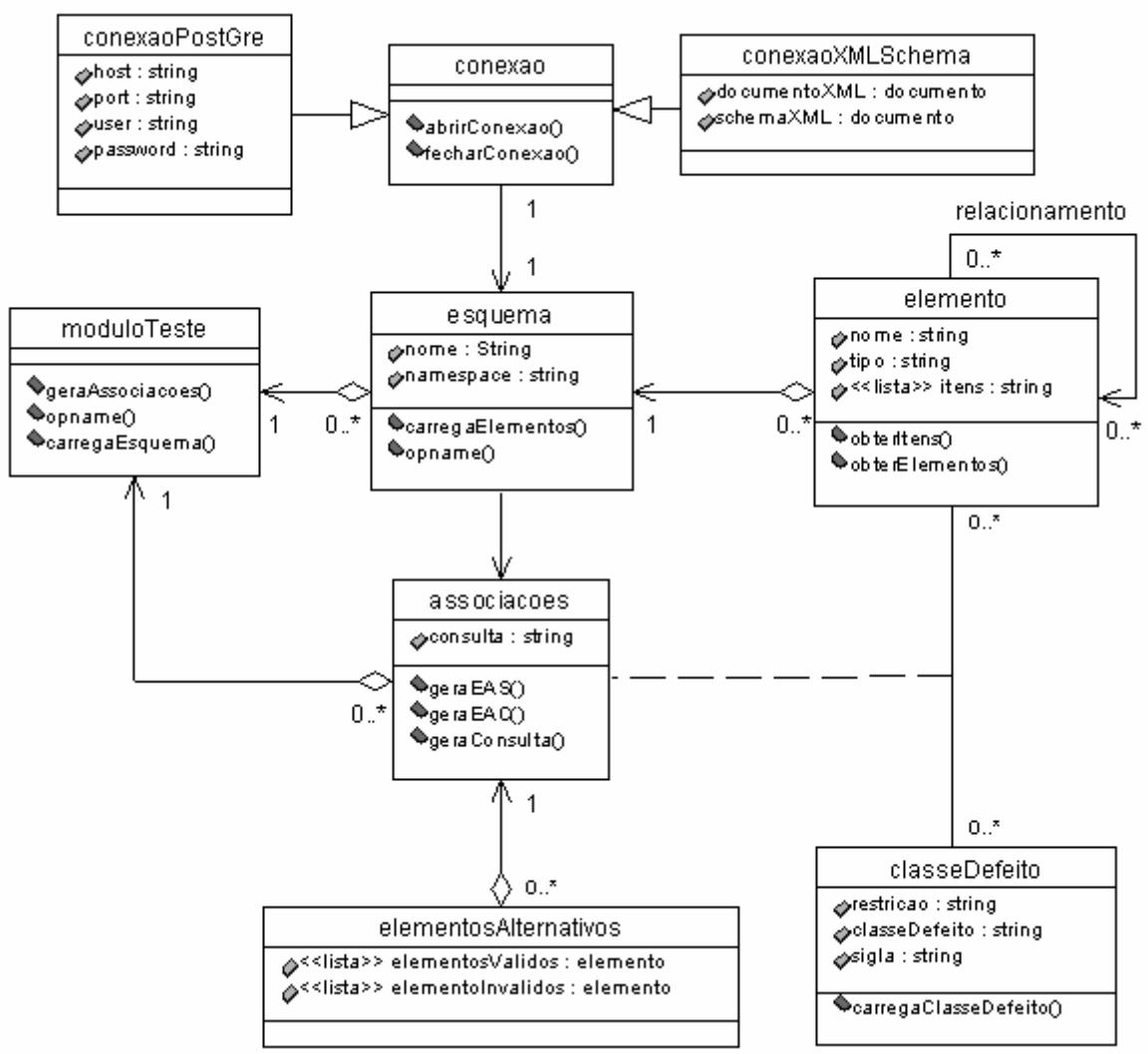


Figura 4.3 Diagrama de classe da ferramenta X-Tool.

Descrição das Classes:

- *moduloTeste* – é a classe principal da ferramenta responsável pela interação entre a interface do usuário com as demais classes.
- *esquema* – classe responsável por armazenar o esquema em teste e algumas informações sobre o mesmo como o nome e *nameSpace*.
- *elemento* – representa os elementos de um esquema, o seu tipo (simples ou complexo) e os valores originais que foram passados como entrada para o elemento.
- *classeDefeito* – possíveis restrições que podem ocasionar algum defeito.
- *associações* – classe que associa os elementos obtidos pela classe "*elemento*" com as restrições obtidas pela classe "*classeDefeito*".
- *elementosAlternativos* – classe responsável por armazenar os elementos alternativos que foram gerados.
- *conexão* – classe responsável por gerenciar a conexão com o esquema.

A classe conexão possui duas subclasses:

- c*onexaoPostGre* - classe responsável pela conexão com uma base de dados PostGre.
- *conexaoXMLSchema* - classe responsável pelo acesso a documentos XML.

Cada estrutura de dados possui uma metodologia própria para sua definição e manipulação. A X-Tool disponibiliza o teste para dois tipos de estruturas de esquema (*XML Schema* e *PostGre SQL Schema*), cada um deles utiliza um módulo de conexão diferente, contendo os métodos de manipulação específicos para a estrutura.

No caso do *XML Schema*, a classe de conexão usada é a "*conexaoXMLSchema*" que usa a API DOM para manipular os dados e a linguagem XQuery para consultar os dados válidos. Para o tipo *PostGre SQL*

Schema a classe utilizada é a “conexaoPostGre” que usa a DML (Linguagem de Manipulação de Dados) para manipular os dados e a DQL (Linguagem de Consulta de Dados) para consultar os dados válidos.

4.3 Utilização da Ferramenta

A X-Tool disponibiliza uma interface gráfica baseada na arquitetura apresentada na *Figura 4.1*, por meio da qual o testador tem acesso às funcionalidades da ferramenta como seleção do esquema e da base de dados, edição e seleção de casos de teste e análise dos resultados.

A seleção do esquema a ser testado é realizada pela interface apresentada na *Figura 4.4*, a qual possibilita a escolha entre os tipos de esquema (*XML Schema* ou *Schema PostGre SQL*).

Para base de dados *PostGre* o testador deve entrar com as informações de onde o esquema se localiza (*Host* e *Port)*, informações de autenticação (*User* e *Password)* e qual base deseja-se testar (*Base* e *NameSpace)*.

Para documentos *XML,* o testador deve informar onde se encontram o documento que contém o esquema *XML Schema* e o documento que traz os dados de entrada *Documento XML*.

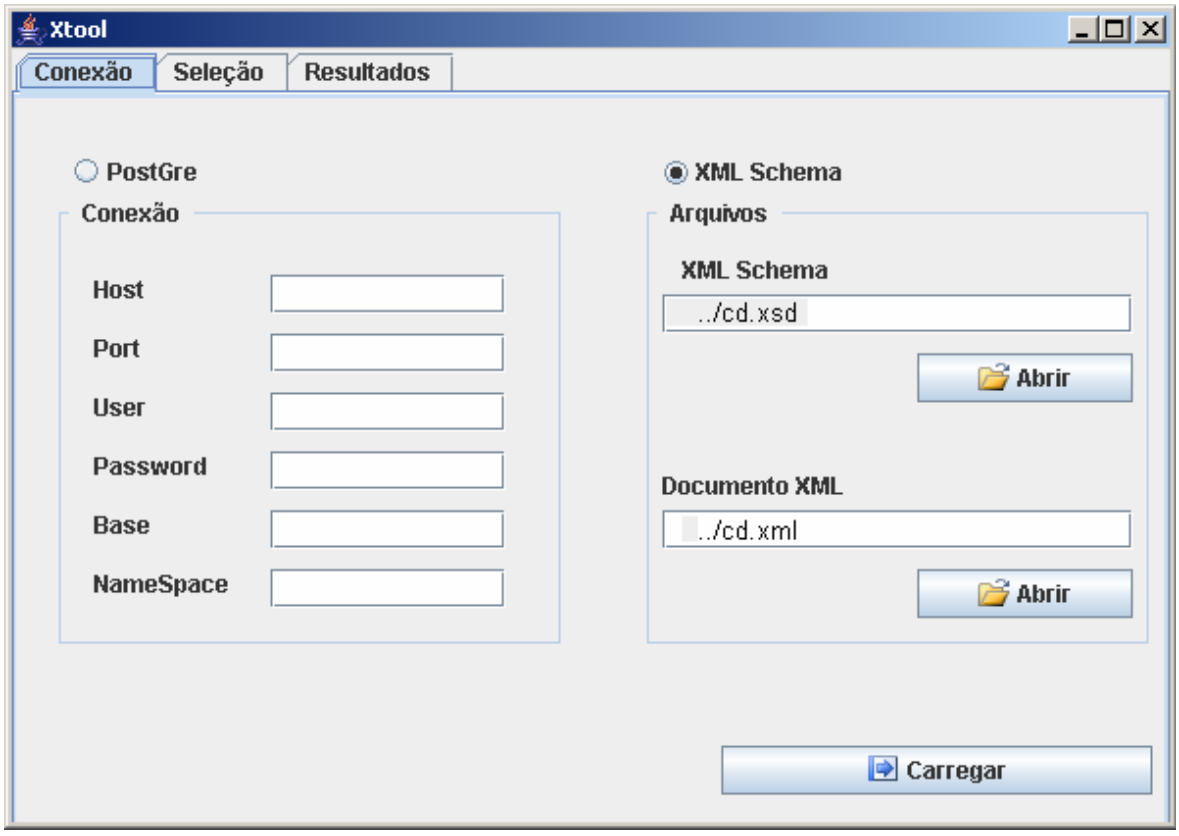


Figura 4.4 Interface para a conexão com o esquema.

Para exemplificar o uso da ferramenta do ponto de vista do usuário, será utilizado o documento "*cd.xml*" da *Figura 4.5*, juntamente com o esquema apresentado na *Figura 4.6* com o nome de "*cd.xsd*". Ambos os arquivos serão passados como parâmetro de entrada para a aplicação.

```
<?xml version="1.0" ?>
<cds >
  <cd genero="instrumental" duracao="00:46:32">
    <titulo>Sucessos do Cinema</titulo>
    <artista>Richard Clayderman</artista>
    <musica>I just called to say I love you</musica>
    <musica>I will always love you</musica>
    <musica>Unchained melody</musica>
    <musica>Theme from love story</musica>
    <musica>How deep is your love</musica>
    <gravadora>Polygram</gravadora>
    <ano>1996</ano>
  </cd>
  <cd genero="instrumental">
    <titulo>Live</titulo>
    <artista>Kenny G</artista>
    <musica>Going Home</musica>
    <musica>Sade</musica>
    <musica>Silhouette</musica>
    <musica>Midnight Motion</musica>
    <musica>Home</musica>
    <musica>Don't Make Me Wait For
Love</musica>
    <musica>I've Been Missin' You</musica>
    <gravadora>BMG</gravadora>
    <ano>1993</ano>
  </cd>
  <cd genero="miscelania" duração="">
    <titulo>Suave Veneno</titulo>
    <artista>Nana Caymmi</artista>
    <artista>Kiloucura</artista>
    <artista>Maria Bethania</artista>
    <artista>Banda Eva</artista>
    <artista>Caetano Veloso</artista>
    <musica>Suave Veneno</musica>
    <musica>Pela Vida Inteira</musica>
    <musica>E o amor</musica>
    <musica>Frisson</musica>
    <musica>Sozinho</musica>
    <gravadora>Som Livre</gravadora>
    <ano>1999</ano>
  </cd>
  <cd genero="miscelania"
duracao="00:46:32">
    <titulo>Desejos de Mulher</titulo>
    <artista>Marisa Monte</artista>
    <artista>Caetano Veloso</artista>
    <artista>Daniela Mercury</artista>
    <artista>Titas</artista>
    <artista>Rita Lee</artista>
    <musica>A sua</musica>
    <musica>Sonhos</musica>
    <musica>Mutante</musica>
    <musica>Epitafio</musica>
    <musica>Minha Vida</musica>
    <gravadora>Som Livre</gravadora>
    <ano>2002</ano>
  </cd>
 </cds>
```

Figura 4.5 Documento cd.xml

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="cds">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="cd" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="titulo" type="xsd:string"  minOccurs="1"  maxOccurs="1" />
              <xsd:element name="artista" type="xsd:string"  minOccurs="1"  maxOccurs="unbounded" />
              <xsd:element name="musica" type="xsd:string"  minOccurs="1" maxOccurs="unbounded"/>
              <xsd:element name="gravadora" type="xsd:string"  minOccurs="0” maxOccurs="1"/>
              <xsd:element name="ano"  type="xsd:integer"   minOccurs="1"  maxOccurs="1"/>
            </xsd:sequence>
            <xsd:attribute name="genero" type="xsd:string" use="required"/>
            <xsd:attribute name="duracao" type="xsd:time"   use="optional"/>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Figura 4.6 Esquema cd.xsd

Após o esquema e os dados (base de dados ou documento XML) serem carregados, o usuário deve especificar quais testes deverão ser realizados. A *Figura 4.7* apresenta a interface que permite ao testador selecionar os testes que deverão ser realizados em cima dos elementos do esquema, alguns deles são selecionados automaticamente baseados nas informações contidas no esquema. Os testes se referem às classes de defeito que estão separadas pelo tipo de restrição. A interface também permite a edição dos testes que dependem de outros elementos e a exclusão dos mesmos.

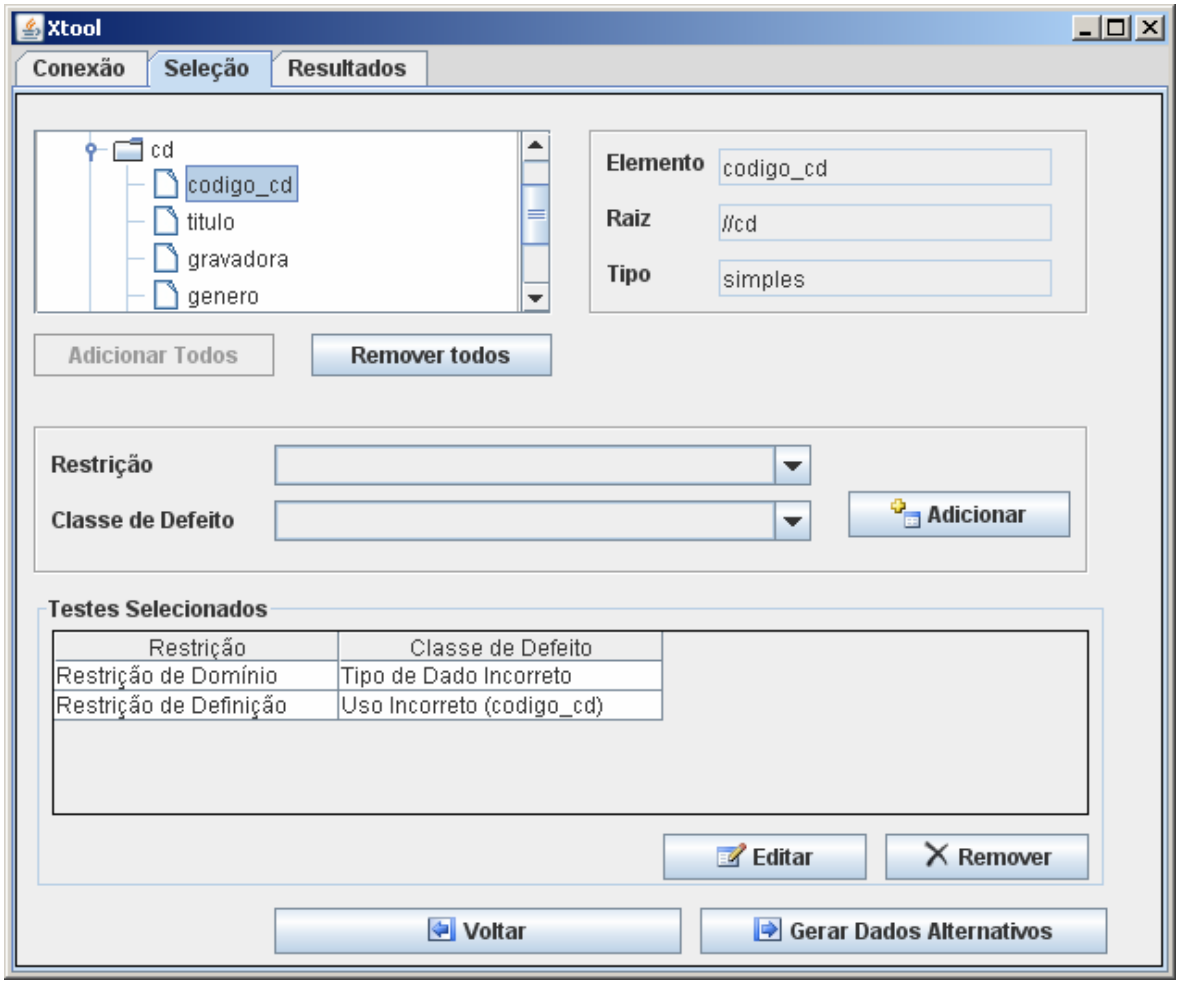


Figura 4.7 Interface para a seleção das classes de defeito.

Os dados alternativos gerados e as informações obtidas são visualizados pela seleção de uma associação *"elemento – classe defeito"* e separados em válidos e inválidos segundo a definição de validade do dado pelo esquema. Além dos dados gerados, o usuário poderá analisar informações sobre a complexidade dos dados de entrada no campo *"Informações"*, apresentado na *Figura 4.8.*

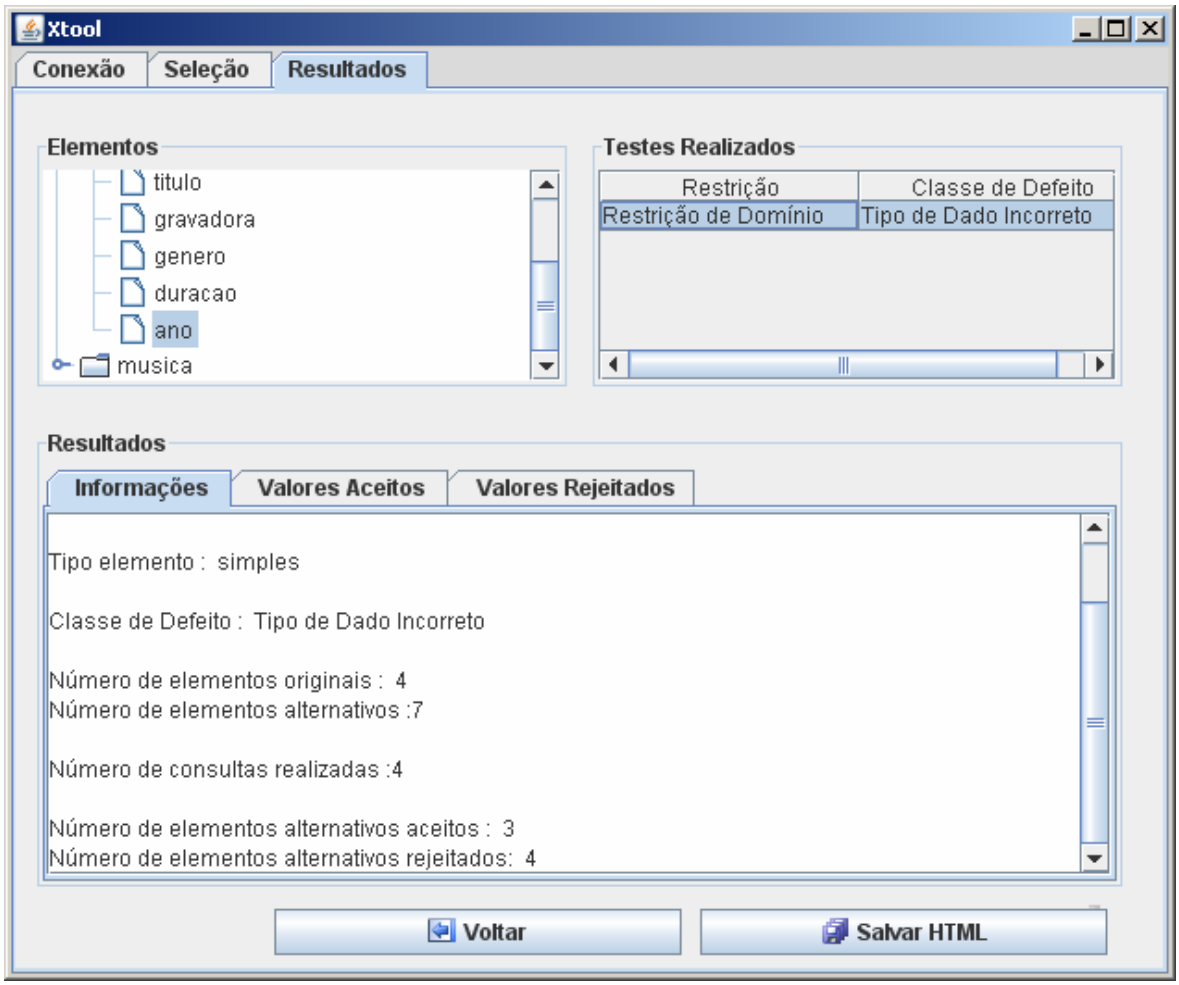


Figura 4.8 Interface contendo o resultado dos testes.

O resultado apresentado na *Figura 4.9* refere-se a um fragmento do resultado para a consulta realizada sobre os dados alternativos válidos gerados para a associação "*ano - tipo dado".*

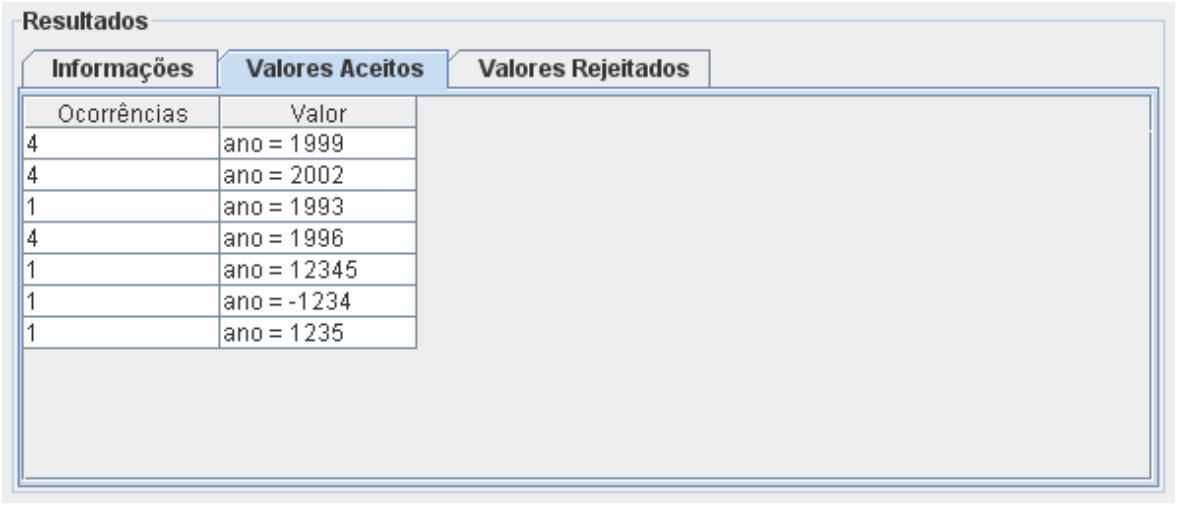


Figura 4.9 Fragmento do relatório para documentos válidos.

A *Figura 4.10* apresenta um fragmento do resultado da consulta sobre os dados alternativos inválidos para a associação "*ano - tipo dado*".

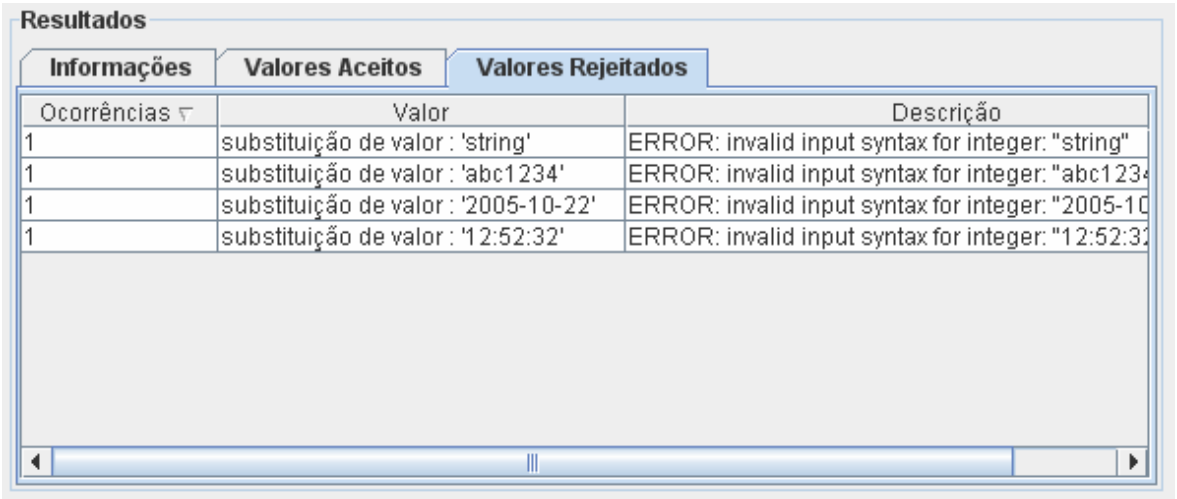


Figura 4.10 Fragmento do relatório para documentos inválidos.

Com os relatórios em mãos, cabe ao testador ou usuário da ferramenta compará-los com as especificações dos dados esperados, a fim de levantar as definições incorretas ou a ausência de restrições no esquema. Para este exemplo é utilizada a especificação apresentada na *Tabela 2.1* que se refere ao esquema "*cds*".

Na *Figura 4.9* são apresentados os resultados para a consulta da associação *"ano - tipo de dado"*, e entre as variações válidas pode ser visto o

conteúdo *“-1234”*, que é um valor numérico inválido para o conteúdo de um elemento do tipo *“ano”*.

Neste exemplo, o esquema em teste permitiu que um documento XML contendo valores inválidos para o elemento “ano” fosse aceito, tal defeito foi identificado através da comparação dos resultados esperados pela especificação e os valores obtidos pela ferramenta para o teste da classe de defeito “Tipo de Dado Incorreto”.

5 Experimentos de Validação

Para validar a implementação da X-Tool, assim como a abordagem proposta por Emer [EME07b] foram realizados três experimentos.

O primeiro e segundo experimentos tiveram respectivamente o objetivo de utilizar a X-Tool no teste de documentos XML e base de dados relacionais. O terceiro experimento teve como objetivo avaliar a eficácia da abordagem de Emer no contexto de *XML Schema* comparando com o teste de mutação e operadores propostos por Franzotte e Vergílio [FRA06]. A seguir são descritos os experimentos.

5.1 Experimento 1

Neste experimento foram utilizados seis esquemas XML: *aluno, disciplina, obras, usuários, cd e pedido* projetados para: um sistema de biblioteca, um de matrícula e outro de uma loja de cds. As principais características dos esquemas são apresentadas na *Tabela 5.1.* Essas características tais como número de elementos, número de atributos, etc. implicam diretamente no número de elementos alternativos gerados pela ferramenta.

Tabela 5.1 Dados sobre a complexidade do esquema XML.

Esquema	Número de elementos	Número de atributos	Número de restrições	Profundidade do esquema	Número de registros
Aluno	10	0	4	3	5
Disciplina	10	0	4	3	6
Obras	11	0	4	4	7
Usuário	8	0	4	3	6
Cd	7	2	5	3	4
Pedido	8	1	6	4	1

Os esquemas *cd.xsd* e *pedido.xsd*, são relacionados às bases de dados da loja e foram testados independentemente da aplicação que manipula seus dados. Os sistemas, biblioteca, registro de cursos e matrículas, foram desenvolvidos por dois grupos de estudantes do curso de graduação, onde cada grupo recebeu as especificações dos sistemas e dos esquemas.

É importante observar que os defeitos revelados não foram semeados, são erros cometidos durante a implementação dos esquemas.

O sistema de biblioteca descreve uma base de dados XML que contém os dados dos usuários registrados e dos títulos disponíveis na coleção, onde os usuários registrados podem acessar o sistema de biblioteca através de um usuário e senha, para consultar títulos disponíveis na coleção. Cada usuário pode acessar três títulos por mês se sua mensalidade estiver em dia. A base de dados é baseada em dois documentos *XML Schema*: *user.xds* e *title.xsd.*

O sistema de matrículas descreve uma base de dados XML que contém os dados dos estudantes e as disciplinas disponíveis no programa, onde um estudante pode acessar o sistema de matrícula com seu *login* (usuário e senha). Ele pode registrar-se somente nas disciplinas para as quais possui os pré-requisitos para o seu curso. A base de dados é baseada em dois documentos *XML Schema* : *student.xsd* e *course.xsd*.

Usando as restrições encontradas nos esquemas, a ferramenta gerou um modelo formal para cada um dos esquemas em teste, que serviu como base para a seleção das classes de defeito que deveriam ser testadas para cada um dos elementos.

O número de associações obtidas entre os elementos e as classes de defeito são apresentados na *Tabela 5.2*, que também fornece o número total de documentos XML alternativos válidos e inválidos que foram gerados e número de consultas realizadas.

Tabela 5.2 Número de associações e dados alternativos gerados.

Esquema	Associações	Consultas	Documentos XML	
			Válidos	Inválidos
Aluno	16	16	241	42
Disciplina	16	16	283	43
Obras	16	16	252	138
Usuário	12	12	187	73
Cd	16	16	176	44
Pedido	14	14	84	77
Total	**90**	**90**	**1223**	**417**

O número de defeitos encontrados é apresentado na *Tabela 5.3,* que descreve as 18 associações de defeito (*elemento x classe de defeito*) que revelam falhas na definição dos elementos segundo as especificações dos esquemas.

Tabela 5.3 Defeitos revelados.

Esquema	Elemento / Atributo	Classe de defeito	Total de defeito(s)
Aluno	codigo	Tipo de dado	1
	senha	Tipo de dado	1
	disciplina_cursada	Tipo de dado	1
	disciplina_matriculada	Tipo de dado	1
Disciplina	codigo	Tipo de dado	1
	carga-horária	Tipo de dado	1
	créditos	Tipo de dado	1
	código_professor	Ocorrência	1
Obras	codigo	Tipo de dado	1
	Ano	Tipo de dado	1
Usuário	Código	Tipo de dado	1
	pag_mensalidade	Tipo de dado	1
	Senha	Tipo de dado	1
Cd	Ano	Tipo de dado	1
	Gênero	Uso	1
	Duração	Tipo de dado	1
Pedido	Preço	Tipo de dado	1
	pedidoID	Uso	1
Total geral de defeitos revelados			18

Observa-se que a abordagem foi eficaz, revelando erros cometidos na definição dos esquemas das aplicações, os quais na maioria das vezes se referem à definição incorreta do tipo de dado.

5.2 Experimento 2

Para a realização do segundo experimento foi utilizado um esquema (*conex*), também modelado por alunos de um curso de graduação, o esquema descreve uma base de dados, responsável pelo armazenamento de informações sobre alunos egressos de uma universidade, as informações se referem a dados: pessoais, acadêmicos e profissionais dos ex-alunos.

A base original foi desenvolvida para o SGBD MSSQL Server 2000, como está é uma aplicação proprietária, a base foi reescrita para o PostGre. Os defeitos encontrados neste experimento referem-se aos erros cometidos durante a transcrição da DDL (*Data Definition Language*) e na interpretação incorreta dos diagramas que especificam o esquema.

O esquema em teste contém 73 elementos simples e 19 elementos complexos, a distribuição dos elementos é apresentada na *Tabela 5.4*, que também informa o número de registros encontrados para cada elemento.

Esse experimento foi realizado em duas etapas, na primeira etapa os dados analisados foram gerados através das associações selecionadas automaticamente pela ferramenta, as quais são feitas seguindo o modelo formal do esquema. A segunda etapa constituiu na análise dos resultados obtidos para as associações selecionadas pelo testador.

Na *Tabela 5.5* são apresentados os números de associações, consultas, dados alternativos e defeitos revelados para cada etapa do experimento.

Os resultados obtidos para cada uma das etapas foram comparados com o DER e o IDEF1X do esquema, que podem ser encontrados no *Apêndice A*. Através da análise dessas informações foram revelados os defeitos apresentados pela *Tabela 5.6.*

Tabela 5.4 Dados sobre a complexidade dos elementos complexos.

Elemento Complexo	Número de elementos simples	Número de registros
Curso	4	6
Tipo_curso	2	4
Instituição	4	4
Tipo_intituicao	2	5
Ramo_atividade	2	5
Uf	2	7
Cidade	3	16
Telefone	5	4
Tipo_telefone	2	5
Aluno	7	3
Faixa_salarial	3	7
Vinculo_empregatício	2	2
Tipo_observacao	2	10
Cargo	4	5
Nível	2	5
Pessoa	8	8
Emprego	8	6
Observação	5	20
Histórico_curso	6	4

Tabela 5.5 Número de associações e dados alternativos gerados

Teste selecionado pelo(a)	Associações	Consultas	Dados Alternativos		Defeitos Revelados
			Válidas	Inválidas	
Ferramenta	274	990	716	1527	12
Testador	23	250	226	193	15
TOTAL	**297**	**1240**	**942**	**1720**	**27**

Observa-se que a maioria dos defeitos encontrados estão relacionados ao limite incorreto de dígitos e caracteres, bem como ao uso incorreto de um elemento simples. Mais de 50% dos defeitos foram revelados utilizando-se associações estabelecidas pelo testador.

Tabela 5.6 Defeitos revelados para o Experimento 2.

Elemento complexo	Elemento simples	Defeito revelado	Etapa
Tipo_curso	Descrição	Uso Incorreto	2
Instituição	Id_ramo_atividade	Uso Incorreto	2
Tipo_intituicao	Descrição	Uso Incorreto	2
Ramo_atividade	Descrição	Uso Incorreto	2
Uf	Sigla	Limite de Caracteres Incorreto	1
Telefone	Ddd	Limite de Dígitos Incorreto	1
	Ddd	Tido de Dado Incorreto	1
	Numero	Limite de Dígitos Incorreto	1
	Numero	Tido de Dado Incorreto	1
	Id_pessoa	Uso Incorreto	2
Aluno	Sexo	Limite de Caracteres Incorreto	1
	Rg	Limite de Caracteres Incorreto	1
	Data_nascimento	Uso Incorreto	2
	Sexo	Valores Enumerados Incorreto	2
Faixa_salarial	Mínimo	Tipo de Dado Incorreto	1
Vinculo_empregatício	Descrição	Uso Incorreto	2
Tipo_observacao	Descrição	Uso Incorreto	2
Cargo	Cargo (Id_nivel)	Cardinalidade	2
	Descrição	Uso Incorreto	2
Pessoa	Cep	Tipo de Dado Incorreto	1
	Cep	Limite de dígitos Incorreto	1
	Id_cidade	Uso Incorreto	2
	Nome	Uso Incorreto	2
	Email	Uso Incorreto	2
Emprego	Data_entrada	Uso Incorreto	2
Observação	Nivel_satisfacao	Limite de Dígitos	1
	Nivel_satisfacao	Tipo de Dado Incorreto	1

5.3 Experimento 3

O terceiro experimento utilizou cinco esquemas produzidos pela ferramenta *HyperModel* [HIP06], com os nomes *catalog.xsd, seller.xsd, transaction.xsd, cart.xsd e customer.xsd, os quais* contêm informações sobre um serviço de comércio eletrônico, usado por um exemplo retirado do site da ferramenta.

O objetivo deste experimento foi verificar se a X-Tool e a abordagem de teste por ela implementada poderiam revelar os defeitos descritos pelos operadores propostos por Franzotte e Vergílio [FRA06].

Na *Tabela 5.7* são apresentadas informações referentes à complexidade da estrutura dos esquemas e dos dados de entrada.

Tabela 5.7 Dados sobre a complexidade do esquema XML.

Esquema	Número de elementos	Número de atributos	Número de restrições	Profundidade do esquema	Número de registros
Catalog	20	0	5	4	6
Seller	20	1	1	4	1
Transaction	51	0	5	5	9
Cart	36	0	5	4	4
Customer	21	0	7	4	1

O primeiro passo do experimento constituiu em obter a especificação dos esquemas, para isso foi executado o teste de esquema com a ferramenta para cada esquema, o resultado obtido foi assumido como os resultados esperados pelo testador.

Para cada esquema em teste, a ferramenta gerou um resultado contendo o modelo formal RTG, as associações entre os elementos e as classes de defeito, os dados alternativos válidos e inválidos obtidos para cada associação e os resultados das consultas para os dados válidos. Na *Tabela 5.8* são apresentados os números obtidos através dos resultados para os esquemas.

Tabela 5.8 Número de associações e dados alternativos gerados.

Esquema	Associações	Consultas	Documentos XML	
			Válidos	Inválidos
Catalog	28	28	345	87
Seller	45	45	174	60
Transaction	86	86	723	189
Cart	71	71	235	110
Customer	46	46	139	63
Total	**276**	**276**	**1616**	**509**

O segundo passo do experimento foi obter resultados do teste com a ferramenta X-Tool para esquemas com defeitos. Para obter esses esquemas foram gerados novos esquemas com base no original contendo uma mutação na definição das restrições, para isso se utilizou a ferramenta XTM que aplica os operadores propostos por *Franzotte e Vergilio* [FRA06].

A *Tabela 5.9* apresenta os operadores aplicados pela ferramenta XTM e qual análise deve ser exercida sobre os resultados da X-Tool para se revelar o defeito inserido no esquema.

Tabela 5.9 Operadores da XTM e análises requeridas para detectar do defeito gerado.

XTM		X-Tool
Operador	**Descrição**	
DT – DataTypes	Altera o tipo dos elementos.	- Operador (Tipo de Dado Incorreto)
LO – LenghtOf	Altera o tamanho dos nomes dos elementos	- Operador (Tipo de Dado Incorreto) - Modelo Formal
CSP -ChangeSingPlural	Modifica o tamanho do elemento pela adição ou remoção do caractere 's' no final da string	- Operador (Tipo de Dado Incorreto) - Modelo Formal
STE -SubTreeExchange	Inverte as sub-arvores abaixo de algum nó	- Operador (Ordem Incorreta)
RT – DeleteTree	Remove o nó (ou sub-árvore) da estrutura da árvore	- Modelo Formal

O último passo do experimento foi comparar os resultados obtidos com os resultados esperados pelo testador e verificar se o defeito inserido no esquema pode ser detectado.

A *Tabela 5.10* apresenta os resultados obtidos pelas ferramentas durante o experimento. As informações da XTM descrevem os operadores utilizados e o número de esquemas gerados para cada esquema em teste, descartando os mutantes equivalentes. Os resultados obtidos pela X-Tool apresentam a quantidade total de associações e o número de documentos XML alternativos válidos e inválidos gerados para os esquemas mutantes. Nessa tabela a última coluna apresenta o número de esquemas mutantes cujo defeito inserido pelo operador da XTM foi revelado através das consultas da X-Tool.

Tabela 5.10 Mutações reveladas.

Esquema	XTM		X-Tool			Número de defeitos revelados
	Operador	Esquemas mutantes	Associações	Alternativas		
				válidas	inválidas	
Catalog	STE	52	1456	17948	4516	52
	CSP	20	560	6564	1740	20
	DT	56	1400	16832	4768	56
	RT	39	904	7466	2138	39
	LO	20	560	6564	1740	20
Seller	STE	56	2464	9688	3360	56
	CSP	20	880	3462	1359	20
	DT	55	2422	9256	3561	55
	RT	27	1082	4143	1423	27
	LO	21	924	3633	1260	21
Transaction	CSP	51	4386	5567	1181	51
	DT	35	3008	3889	881	35
	LO	51	4386	5561	1175	51
	RT	46	3772	4765	996	46
	STE	136	11696	15112	3416	136
Cart	CSP	36	2556	2989	433	36
	DT	27	1915	2285	370	27
	LO	36	2556	2984	428	36
	RT	32	2144	2425	419	32
	STE	160	11360	13440	2080	160
Customer	CSP	21	966	1295	329	21
	DT	16	731	944	211	16
	LO	21	966	1295	329	21
	RT	19	782	1006	224	19
	STE	65	2990	3835	845	65

Pode-se observar que o número de defeitos revelados é sempre igual ao número de esquemas mutantes. Como cada esquema mutante contém uma única alteração na sua definição conclui-se que a X-Tool conseguiu revelar todos os defeitos, correspondentes às alterações que foram feitas nos esquemas pelos operadores de mutação da ferramenta XTM.

6 Conclusões e Trabalhos Futuros

O objetivo desta dissertação foi apresentar uma ferramenta que apóia o teste de esquemas, visando a revelar defeitos cometidos na definição de um esquema. Os defeitos são detectados por testes baseados em classes de defeito previamente estudadas e propostas na literatura.

Os testes realizados pela ferramenta consistem em gerar dados alternativos e realizar consultas específicas sobre os dados, o resultado obtido é usado para analisar se o esquema está obedecendo a sua especificação.

Atualmente a ferramenta possibilita o teste de dois tipos específicos de esquemas: *XML Schema* que é utilizado em documentos XML e o *PostGre SQL Schema* que define bases de dados relacionais.

Analisando os resultados obtidos pelos experimentos pode-se concluir que a ferramenta contribui para a qualidade do *software* que utiliza esquemas de dados. Durante o processo de validação da ferramenta foram realizados três experimentos, dois deles testaram esquemas para documentos XML, e o terceiro utilizou um esquema para bases de dados, permitindo demonstrar que a ferramenta revela os defeitos referentes à definição incorreta dos esquemas.

A X-Tool é executada independentemente da aplicação que utiliza os esquemas, porém, é necessário que a especificação dos dados da aplicação esteja disponível para o testador analisar os resultados obtidos com a execução da ferramenta.

É importante ressaltar que, diferentemente da XTM [FRA06], para a qual se necessita identificar os casos de teste e os mutantes equivalentes, na X-Tool o testador apenas tem que checar as saídas. Outra diferença é que a XTM, não apóia o teste de esquemas de bases de dados relacionais.

A X-Tool deriva os dados de teste automaticamente, que são formados pelos dados alternativos (sejam eles documentos XML ou base relacionais) e pelas consultas. O testador apenas precisa checar a saída com os dados esperados.

Além disso, no experimento realizado foi possível utilizar a X-Tool para revelar todos os defeitos introduzidos pelos operadores de mutação da XTM.

6.1 Trabalhos Futuros

Como trabalho futuro sugere-se:

- Refinamento da ferramenta:
    - aplicação de novas classes de defeito;
    - permitir o teste de outros tipos de esquemas, tais como esquemas de bases de dados orientadas a objetos;
    - comparação automática das saídas com informações fornecidas pelo usuário, para diminuir custos.
- Realização de novos experimentos:
    - que explorem outras circunstâncias de uso de esquemas como os utilizados em *Web-Services*;
    - experimentos para medir a eficiência e custos da ferramenta;
    - avaliação da eficácia e utilização da ferramenta em esquemas de aplicações reais.

Referências Bibliográficas

[ARA00] Aranha, M.C.L.F.M.; Mendes, N.C.; Jino, M.; Toledo, C.M.T.. **RDBTool: A Support Tool for Testing Relational Database**. XI ICST: International Conference of Software Techology. Agosto de 2000.

[BOT05] Bothner, Per. **Qexo: The GNU Kawa implementation of Xquery**, disponível em: http://www.gnu.org/software/qexo/. Acessado em 2006.

[CHA05] Chamberlin, Don. **Book excerpt explores origin of the XML Query language**, 2003, Disponível em: http://www-106.ibm.com/developerworks/xml/library/xxqbook.html. Acessado em 2005.

[CHE76] Chen, P. P.. **The Entity-Relationship Model – Toward a Unified View of Data.** ACM Transactions on Database System (TODS), 1976.

[DeM78] DeMillo, R. A.; Lipton, R. J.; Sayward, F. G. **Hints on Test Data Selection: Help for the Practicing Programmer**. IEEE Computer, Abril de 1978.

[DeM87] DeMillo, R.A. **Software Testing and Evaluation. The Benjamin/Cummings Publishing Company**, 1987.

[EME05] EMER, M.C.F.P.; VERGILIO, S.R.; JINO, M.. **A Testing Approach for XML Schemas**. Proceedings of the 29th Annual International Computer Software and Applications Conference. Workshop on Quality Assurance and Testing of Web-Based Applications. COMPSAC 2005.

[EME07a] EMER, M.C.F.P.; NAZAR, I.F.; JINO, M.; VERGILIO, S.R.. **A Fault-Based Testing Approach for Database Schemas.** Submetido ao SBES 2007.

[EME07b] Emer, M.C.F.P.. **Abordagem de Teste Baseada em Defeitos para Esquemas de Dados.** Tese de Doutorado, UNICAMP, Campinas, SP, 2007.

[FRA06] Franzotte, L; Vergilio, S.R. **Applying Mutation Testing to XML Schemas**, Proceedings of the 18th International Conference on Software Enginerring and Knowledge Engineering (SEKE'06), Julho de 2006.

[HIP06] Carson, D. **hiperModel Application**. http://www.xmlmodeling.com/models. Acessado em 2006.

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

[LI05] Li, J. B.; Miller, J. **Testing the Semantics of W3C XML Schema**. Proceedings of the 29th Annual International Computer Software and Applications Conference COMPSAC 2005, Julho de 2005.

[MYE79] Myers, G.J. **The Art of Software Testing**. J Wiley & Sons, New York, NY 1979.

[POS06] **PostGre SQL** disponível em: http://www.postgresql.org/docs/. Acessado em 2006.

[PRE00] Pressman, R.S. **Engenharia de Software**. Rio de Janeiro: McGraw-Hill, 2002.

[RAP85] Rapps, S; Weyuker, E.J. **Selecting software test data using data flow information**. IEEE Transations on Software Engineering, 1985.

[SUN06a] **Java Database Connectivity** (JDBC), disponível em: http://java.sun.com/javase/technologies/database/, Acessado em 2006..

[OFF04] Offut, J.; XU, W. **Generating Test Cases for Web Services Using Data Perturbation**. In TAV-WEB Proceedings/ACM SIGSOFT, Setembro 2004.

[W3C06a] **Extensible Markup Language** (XML), disponível em: http://www.w3c.org/XML/, Acessado em 2006.

[W3C06b] **XML Schema**, disponível em: http://www.w3c.org/XML/Schema/, Acessado em 2006.

[W3C06c] **Document Object Model** (DOM), disponível em: http://www.w3c.org/DOM/, Acessado em 2006.

[W3C06d] **XML Query Language**. (XQuery), disponível em: http://www.w3c.org/XML/Query/. Acessado em 2006.

[W3C06e] **Standard Generatized Markup Language** (SGML), disponível em: http://www.w3c.org/TR/html4/intro/sgmltut.html , Acessado em 2006.

[W3S06] **Document Type Definition** (DTD), disponível em: http://www.w3schools.com/dtd/dtd_intro.asp, Acessado em 2006.

[WIN06a] **Simple API for XML** (SAX), disponível em: http://en.wikipedia.org/wiki/Simple_API_for_XML, Acessado em 2006.

[WIN06b] **Structured Query Language** (SQL), disponível em: http://en.wikipedia.org/wiki/SQL, Acessado em 2006.

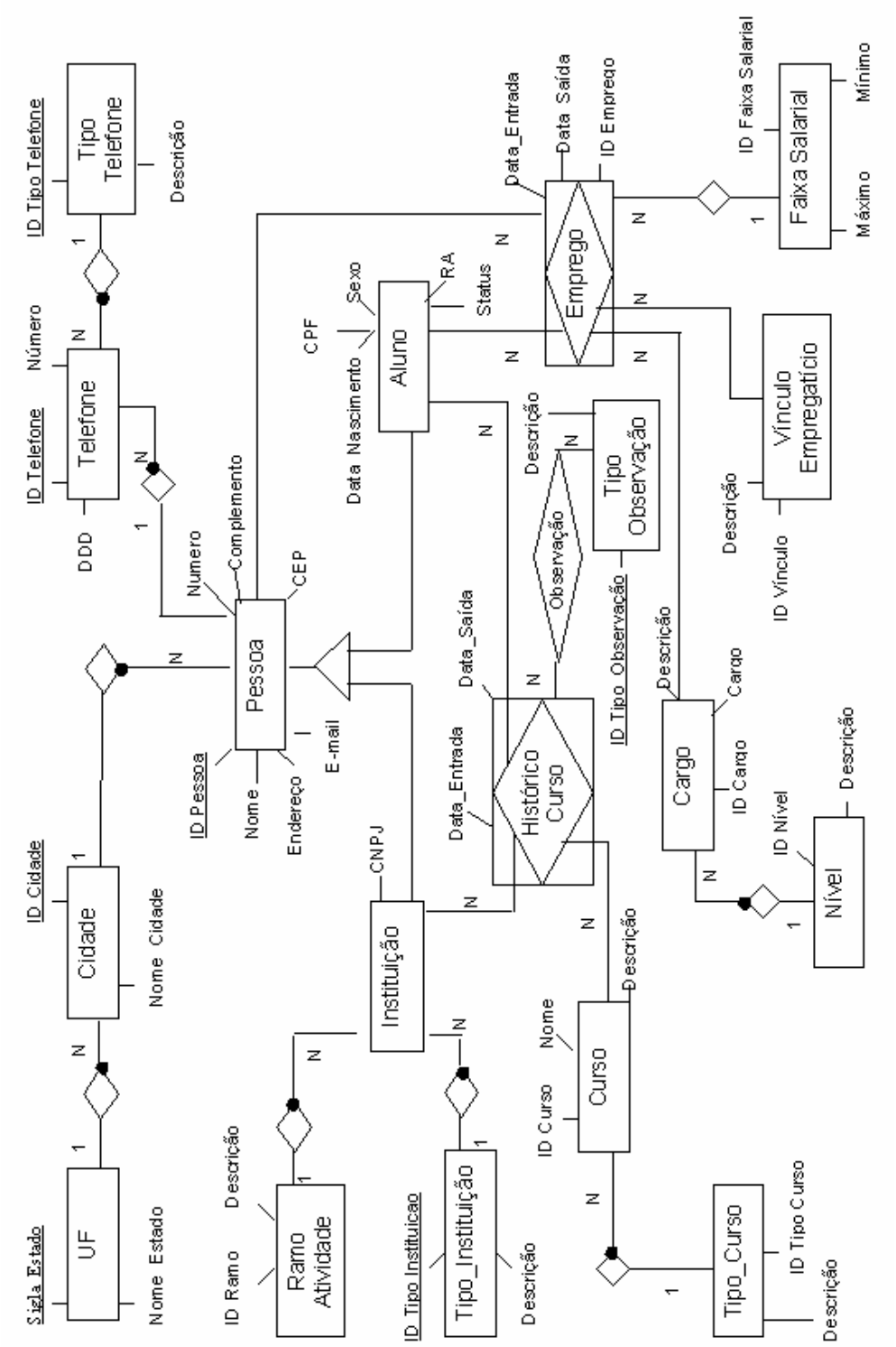
[WIN06c] **Model Entity-Relationship** (MER), disponível em: http://en.wikipedia.org/wiki/Entity-Relationship_Model, Acessado em 2006.

[WIN06d] **Integration DEFinition** (IDEF1x), disponível em: http://en.wikipedia.org/wiki/IDEF1X, Acessado em 2006.

[WUZ05] Wuzhi Xu; Offutt J.; Luo J. **Testing Web Services by XML Perturbation**, IEEE International Symposium on Software Reliability Engineering, Novembro 2005.

Apêndice A – Diagramas para os Esquemas do Experimento 2

A.1 Diagrama Entidade Relacionamento



A.2 Diagrama IDEF1X