UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO

Maria Claudia Figueiredo Pereira Emer

Abordagem de Teste Baseada em Defeitos para Esquemas de Dados

Campinas, SP
2007

Maria Claudia Figueiredo Pereira Emer

Abordagem de Teste Baseada em Defeitos para Esquemas de Dados

Tese de Doutorado apresentada à Faculdade de Engenharia Elétrica e de Computação como parte dos requisitos para obtenção do título de Doutor em Engenharia Elétrica. Área de concentração: Engenharia de Computação.

Orientador: Prof. Dr. Mario Jino
Co-orientadora: Profa. Dra. Silvia Regina Vergilio

Campinas, SP
2007

Maria Claudia Figueiredo Pereira Emer

Abordagem de Teste Baseada em Defeitos para Esquemas de Dados

Tese de Doutorado apresentada à Faculdade de Engenharia Elétrica e de Computação como parte dos requisitos para obtenção do título de Doutor em Engenharia Elétrica. Área de concentração: Engenharia de Computação.
Aprovação em 06/09/2007

Banca Examinadora:
Prof. Dr. Mario Jino - UNICAMP
Prof. Dr. Ivan Luiz Marques Ricarte - UNICAMP
Prof. Dr. Léo Pini Magalhães - UNICAMP
Profa. Dra. Ana Cristina Vieira de Melo - USP
Profa. Dra. Simone do Rocio Senger de Souza - USP
Prof. Dr. Marcos Lordello Chaim - USP

Campinas, SP
2007

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA E ARQUITETURA - BAE - UNICAMP

Em32a

Emer, Maria Claudia Figueiredo Pereira
Abordagem de teste baseada em defeitos para esquemas de dados / Maria Claudia Figueiredo Pereira Emer. -- Campinas, SP: [s.n.], 2007.

Orientadores: Mario Jino, Silvia Regina Vergilio
Tese (doutorado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Engenharia de software. 2. Programas de computador - testes. 3. Software – banco de dados. 4. XML (Linguagem de marcação de documento). 5. Modelagem de dados. 6. UML (Linguagem de modelagem padrão). I. Jino, Mario. II. Vergilio, Silvia Regina. III. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. IV. Título.

Título em Inglês: Fault-based testing approach for data schemas
Palavras-chave em Inglês: Data schemas, Data integrity, Fault-based testing, XML, Database
Área de concentração: Engenharia de Computação
Titulação: Doutora em Engenharia Elétrica
Banca examinadora: Ana Cristina Vieira de Melo, Ivan Luiz Marques Ricarte, Léo Pini Magalhães, Marcos Lordello Chaim, Simone do Rocio Senger de Souza
Data da defesa: 06/09/2007
Programa de Pós-Graduação: Engenharia Elétrica

COMISSÃO JULGADORA - TESE DE DOUTORADO

Candidata: Maria Claudia Figueiredo Pereira Emer

Data da Defesa: 6 de setembro de 2007

Título da Tese: "Abordagem de Teste Baseada em Defeitos para Esquemas de Dados"

Prof. Dr. Mario Jino (Presidente): _____

Profa. Dra. Ana Cristina Vieira de Melo: _____

Profa. Dra. Simone do Rocio Senger de Souza: _____

Prof. Dr. Marcos Lordello Chaim: _____

Prof. Dr. Ivan Luiz Marques Ricarte: _____

Prof. Dr. Léo Pini Magalhães: _____

Resumo

Dados são manipulados em várias aplicações de software envolvendo operações críticas. Em tais aplicações assegurar a qualidade dos dados manipulados é fundamental. Esquemas de dados definem a estrutura lógica e os relacionamentos entre os dados. O teste de esquemas por meio de abordagens, critérios e ferramentas de teste específicos é uma forma pouco explorada de assegurar a qualidade de dados definidos por esquemas. Este trabalho propõe uma abordagem de teste baseada em classes de defeitos comumente identificados em esquemas de dados. Um metamodelo de dados é definido para especificar os esquemas que podem ser testados e as restrições aos dados nos esquemas. Defeitos possíveis de serem revelados são os relacionados à definição incorreta ou ausente de restrições aos dados no esquema. A abordagem inclui a geração automática de um conjunto de teste que contém instâncias de dados e consultas a essas instâncias; as instâncias de dados e as consultas são geradas de acordo com padrões definidos em cada classe de defeito. Experimentos nos contextos de aplicações Web e de base de dados foram realizados para ilustrar a aplicação da abordagem.

Palavras-chave: Esquemas de dados, Integridade de dados, Teste baseado em defeitos, XML, Base de dados.

Abstract

Data are used in several software applications involving critical operations. In such applications to ensure the quality of the manipulated data is fundamental. Data schemas define the logical structure and the relationships among data. Testing schemas by means of specific testing approaches, criteria and tools has not been explored adequately as a way to ensure the quality of data defined by schemas. This work proposes a testing approach based on fault classes usually identified in data schemas. A data metamodel is defined to specify the schemas that can be tested and the constraints to the data in schemas. This testing approach provides means for revealing faults related to incorrect or absent definition of constraints for the data in the schema. The approach includes the automatic generation of a test set which contains data instances and queries to these instances; the data instances and queries are generated according to patterns defined in each fault class. Experiments in the contexts of Web and database applications were carried out to illustrate the testing approach application.

Keywords: Data schemas, Data integrity, Fault-based testing, XML, Database.

Aos meus pais,

Antonio e Gizelia.

Ao meu esposo,

Ivo.

Agradecimentos

A Deus, luz para o meu caminho.

Ao Ivo pelo apoio, incentivo e amor nesta jornada.

A Gizelia e Antonio, que me apoiaram desde o início de minha formação, por todo amor e compreensão.

Aos meus orientadores, Prof. Mario Jino e Profa. Silvia Regina Vergilio, pela oportunidade, orientação e amizade.

Aos amigos que fiz, especialmente a Cynthia e a Érica, que estiveram mais próximas nesta jornada.

Aos colegas dos seminários de teste de software, pela colaboração na realização deste trabalho.

Ao CNPq, pelo apoio financeiro.

"O futuro pertence àqueles que acreditam na beleza de seus sonhos."
Eleanor Roosevelt

Sumário

Lista de Figuras

Lista de Tabelas

Trabalhos Relacionados à Tese

Artigos Publicados

1. M. C. F. P. Emer, S. R. Vergilio, M. Jino. "A Testing Approach for XML Schemas". *In Proceedings of & the 29th IEEE Annual International Computer Software and Applications Conference. Workshop on Quality Assurance and Testing of Web-Based Application* (COMPSAC 2005), Vol. 2, pp. 57 - 62, July 2005.

2. M. C. F. P. Emer, I. F. Nazar, S. R. Vergilio, M. Jino. "Evaluating a Fault-Based Testing Approach for XML Schemas". *In Proceedings of the VIII IEEE Latin-American Test Workshop* (LATW 2007), March 2007.

3. M. C. F. P. Emer, S. R. Vergilio, M. Jino. "Fault-Based Testing of Data Schemas". *In Proceedings of the 19th International Conference on Software Engineering and Knowledge Engineering* (SEKE 2007), pp. 123 - 128, July 2007.

Artigos Aceitos para Publicação

1. M. C. F. P. Emer, S. R. Vergilio, M. Jino, I. F. Nazar, P. V. Caxeiro. "Uma Avaliação do Teste Baseado em Defeitos em Esquemas de Banco de Dados Relacional". *In Proceedings of the 1st Brazilian Workshop on Systematic and Automated Software Testing* (SAST 2007) em conjunto com o *XXI Simpósio Brasileiro de Engenharia de Software* (SBES 2007), outubro 2007.

Capítulo de Livro

1. M. C. F. P. Emer, S. R. Vergilio, M. Jino. "Teste de Aplicações Web". In: M. E. Delamaro, J. C. Maldonado, M. Jino (Org.). *Introdução ao Teste de Software*. Rio de Janeiro: Elsevier, 2007.

Capítulo 1

Introdução

1.1 Contexto

O teste é uma fase importante do desenvolvimento de um produto de software que contribui para a obtenção de produtos confiáveis e para avaliação do nível de qualidade desses produtos [60]. O teste é realizado com o objetivo de revelar defeitos no software. Nesse sentido, um teste é dito bem sucedido se foi capaz de revelar defeitos ainda desconhecidos [53].

Técnicas e critérios de teste têm sido propostos para nortear o processo de teste, auxiliando a seleção e avaliação de conjuntos de casos de teste, visando a determinar aqueles que aumentam as chances de revelar defeitos.

Entre as técnicas de teste existentes podem ser citadas: a técnica estrutural, que deriva casos de teste a partir da implementação do programa; a técnica funcional, que usa a especificação funcional do programa para derivar casos de teste; e a técnica baseada em defeitos, que deriva casos de teste para mostrar a presença ou ausência de defeitos típicos no programa.

Um critério de teste é um predicado que, em geral, fornece uma medida de cobertura para quantificar a atividade de teste e quando satisfeito, o teste pode ser encerrado [47]. Alguns exemplos de critérios de teste são: a) critérios estruturais: Critério Todos os Nós, Critério Todos os Arcos [62], Critério Todos os Caminhos Linearmente Independentes de McCabe [51], Critérios Potenciais-usos [47], etc; b) critérios funcionais: Particionamento em Classes de Equivalência e Grafo de Causa-Efeito [60, 53]; c) critérios baseados em defeitos: Critério Análise de Mutantes [18]. Esses diversos critérios de teste são vistos como complementares, porque revelam diferentes tipos de defeitos. Em estudos empíricos, os critérios associados à técnica baseada em defeitos têm se mostrado mais eficazes em relação ao número de defeitos detectados; porém, são mais custosos quanto ao número de casos de teste e de execuções do programa [50].

Outra questão relacionada ao processo de teste é a necessidade de ferramentas que apóiem a

aplicação dos critérios de teste em programas simples e complexos. No entanto, existem limitações para a completa automatização da atividade de teste, como por exemplo, o problema da geração de dados de teste para satisfazer um dado critério, que envolve questões indecidíveis, tais como: mutantes equivalentes [5] e caminhos não executáveis [31].

O teste envolve muitos aspectos em aplicações de software; por exemplo, código, ambiente operacional e dados. O foco deste trabalho está em testar a definição de dados armazenados e manipulados por uma aplicação, considerando-se que estes dados precisam ser confiáveis para que falhas sejam evitadas.

Em geral, os dados em aplicações são especificados por meio de esquemas. Um esquema define a estrutura lógica dos dados, ou seja, no esquema, os dados e seus relacionamentos são descritos. O esquema é projetado de acordo com a especificação dos dados da aplicação. Destacam-se nesse contexto: esquemas de dados para XML e para aplicações de bases de dados relacionais. A linguagem XML vem sendo muito utilizada em aplicações baseadas na Web para o intercâmbio de informações e o modelo de dados relacional tem sido o mais comumente empregado em aplicações de base de dados.

No contexto de XML, esquemas podem ser escritos, por exemplo, por meio de XML *Schema Language* (linguagem de esquema XML [77]); e no contexto de base de dados relacional, esquemas são representados por meio de linguagens de modelagem de dados textuais ou gráficas, um exemplo de modelagem conceitual de dados mais difundida é o MER (Modelo Entidade-Relacionamento [15]). Entretanto outros tipos de esquemas podem e devem também ser considerados: esquemas XML escritos em outras linguagens (por exemplo, DTD - Definição de Tipo de Documento [76]).

Um exemplo de defeito em um esquema de dados poderia ser: a especificação dos dados da aplicação define que um dado denominado “senha” deve conter exatamente 8 caracteres alfanuméricos; no esquema, o dado “senha” está definido com tamanho 6 (tamanho em relação à quantidade de caracteres alfanuméricos permitidos). Portanto, existe um defeito na definição do dado “senha” no esquema, que pode refletir-se em falhas na aplicação que manipula esse dado.

Um meio de avaliar se dados definidos por esquemas estão corretos, isto é, se refletem a especificação dos dados, é aplicar abordagens, critérios e ferramentas de teste específicas para esquemas de dados. Dessa forma, a integridade dos dados definidos no esquema pode ser assegurada e, conseqüentemente, um nível mais alto de qualidade e confiança no software que utiliza esses dados pode ser alcançado. Entretanto, o teste de esquemas é uma forma pouco explorada de assegurar a qualidade dos dados de uma aplicação. A maioria dos trabalhos testa a aplicação, podendo ou não utilizar informações do esquema de dados.

Quando consideram-se esquemas XML, abordagens de teste são geralmente aplicadas para testar a aplicação Web [24, 25, 41, 45, 46, 63]. Outros trabalhos abordam o teste de aplicações Web

manipulando os dados ou o esquema de dados [42, 55, 83]. Alguns trabalhos abordam o teste de esquema. Dentre esses destacam-se [32, 44]. Li e Miller [44] propõem operadores de mutação para esquemas XML; no entanto, eles não apresentam o processo de teste e não esclarecem como os casos de teste são gerados para detectar defeitos no esquema. Franzotte e Vergilio [32] introduzem novos operadores de mutação e aplicam o critério Análise de Mutantes em esquemas XML; entretanto, os dados de teste no processo de teste devem ser gerados manualmente pelo testador, os operadores de mutação podem ser aplicados somente em XML *Schemas*, mutantes equivalentes podem ser gerados e devem ser determinados pelo testador.

No contexto de base de dados, existem vários trabalhos que investigam o teste das aplicações. Porém, o objetivo de muitos desses trabalhos é o teste de uma aplicação de base de dados envolvendo a geração de dados, a aplicação e o projeto [10, 12, 13, 22, 39, 43, 70, 72, 84]. Alguns trabalhos abordam o teste da aplicação de base de dados usando informações do esquema de dados [11, 64], sem explorar o teste desses esquemas. Aranha et al. [1] testam esquemas de base de dados executando operações (sentenças SQL) na base de dados para revelar defeitos na definição de atributos e relacionamentos especificados no esquema, empregando critérios de teste baseados na definição e uso de atributos de uma base de dados relacional. Nesse trabalho, as sentenças SQL e os dados de teste são gerados manualmente.

Os trabalhos mencionados acima, tanto no contexto de XML quanto no contexto de base de dados relacional, possuem algumas limitações, tais como: não podem ser aplicados em diferentes contextos associados a esquemas de dados e os dados de teste são gerados manualmente.

1.2 Motivação

Com base no contexto anteriormente apresentado, as principais motivações para a realização deste trabalho são apresentadas:

- A importância de esquemas de documentos XML, dado que existe uma demanda crescente de aplicações Web e do uso de XML; além disso, a importância de esquemas de dados em aplicações de base de dados, largamente utilizadas;

- A necessidade de identificar defeitos inseridos em esquemas de dados para assegurar a consistência de dados definidos por esses esquemas;

- A necessidade de desenvolver abordagens e ferramentas de teste específicas que permitam revelar defeitos em esquemas de dados;

- A existência de poucos trabalhos que exploram o teste de esquemas de dados como modo de auxiliar na garantia de qualidade de software; em geral esses trabalhos visam o teste de esquemas específicos, não permitindo a aplicação em diferentes contextos e, na maioria das vezes não oferecendo suporte automatizado;

- Critérios baseados em defeitos têm se mostrado mais eficazes quanto ao número de defeitos revelados em programas tradicionais. Explorar esse tipo de critério no contexto de teste de esquemas pode ser bastante promissor.

1.3 Objetivos

Com base na importância de esquemas de dados no contexto de XML e de base de dados, e da existência de poucos trabalhos que investigam o teste de esquema, o principal objetivo deste trabalho é propor uma abordagem genérica baseada em defeitos para o teste de esquemas de dados, ou seja, uma abordagem de teste baseada em defeitos típicos introduzidos em esquemas, que possa ser empregada em esquemas de dados de diferentes contextos. Essa abordagem de teste é denominada Análise de Instâncias de Dados Alternativas. Os objetivos secundários deste trabalho podem ser resumidos em:

- Investigar o teste de aplicações de software que utilizem esquemas de dados ou que testem esses esquemas;

- Introduzir um metamodelo de dados para representar aspectos dos esquemas mais utilizados;

- Identificar os defeitos comumente inseridos durante o desenvolvimento de um esquema de dados e propor uma classificação para os mesmos;

- Definir o processo de teste que deve ser executado para a realização do teste de esquema com o emprego da abordagem de teste;

- Estudar aspectos de implementação da abordagem e processo de teste propostos para permitir sua utilização prática;

- Avaliar a aplicabilidade e eficácia da abordagem de teste.

1.4 Organização

Este trabalho é organizado como segue. No Capítulo 2 são apresentados conceitos básicos sobre esquemas de dados, modelagem de dados e teste de software. No Capítulo 3 são descritos trabalhos

relacionados. No Capítulo 4 é introduzida a abordagem de teste de esquemas de dados que está sendo proposta neste trabalho. No Capítulo 5, contextos de uso da abordagem são apresentados, juntamente com exemplos de sua aplicação. No Capítulo 6, a avaliação da abordagem de teste proposta é descrita e os resultados obtidos são discutidos. Esse capítulo contém também uma descrição da ferramenta, denominada XTool, que apóia a aplicação da abordagem. No Capítulo 7 são discutidas as conclusões e os trabalhos futuros. O trabalho contém 5 apêndices, nos quais são disponibilizadas informações sobre a sintaxe de OCL, empregada na definição do metamodelo, e dados adicionais em relação aos estudos de caso de avaliação da abordagem de teste.

Capítulo 2

Conceitos Básicos

Este capítulo apresenta uma visão geral sobre esquemas de dados, para os quais a abordagem de teste proposta neste trabalho é aplicada. Também são apresentados conceitos básicos sobre metamodelagem e teste de software. Esses temas são abordados por estarem relacionados aos conceitos empregados para a definição da abordagem de teste.

2.1 Esquemas de dados

Esta seção trata de esquemas de dados, mais especificamente esquemas de XML e esquemas de base de dados relacional, isto porque a linguagem XML vem sendo muito utilizada para o intercâmbio de informações e o modelo de dados relacional tem sido o mais comumente empregado em aplicações de base de dados. Nesta seção também são abordados conceitos relacionados a XML e modelos de base de dados.

2.1.1 XML

XML (*Extensible Markup Language*) [76] é uma linguagem de marcação para conteúdo, que contém somente elementos de definição de conteúdo. Essa linguagem foi criada em 1996 pelo *World Wide Web Consortium* (W3C). A linguagem XML é flexível e adaptável para distribuição de informação na Web[1] ou entre aplicativos (software) sem as limitações de HTML (*Hypertext Markup Language*) [73]. XML permite que o significado real das informações não seja perdido, por isso XML reflete a semântica dos dados.

HTML é uma linguagem de marcação para apresentação, projetada para a disseminação de dados e, portanto, é adequada para apresentar um documento com estilo, imagens, *hiperlinks*, e similares;

[1] O termo Web é usado para indicar um conjunto de nós interconectados por meio de ligações (*links*) [17].

mas não é apropriada para acomodar uma ampla variedade de dados e de tipos de dados para a troca de informação na Web.

Algumas características importantes de XML são [73]:

- XML é uma linguagem de marcação de conteúdos.

  Uma linguagem de marcação tem a capacidade de descrever marcadores ou marcas (*tags*) inseridas em um texto, definindo o significado dado ao texto ou conteúdo associado à marca [6].

- XML permite elementos definidos pelo usuário.

  As marcas de XML são definidas pelo usuário, em dependência do domínio associado à aplicação na qual a XML está sendo empregada, ou seja, as marcas são definidas para atender às necessidades específicas de um aplicação.

- XML exige validação;

  XML requer validação para que um documento XML seja considerado válido, isto é, o documento XML deve ser analisado quanto a estar em concordância com regras definidas em um esquema de dados associado a ele. O esquema define a estrutura lógica do documento XML, determinando os dados contidos no mesmo e descrevendo esses dados.

- XML é orientada para dados.

  XML foi projetada para armazenar dados, sem elementos de apresentação, somente marcas e dados.

- XML permite troca de dados entre aplicações de software;

  XML facilita o intercâmbio de dados entre aplicações de software devido ao modo como os dados são armazenados, favorecendo a interpretação dos dados por parte das aplicações que os utilizam.

- XML é rigidamente definida e interpretada.

  XML é uma linguagem que possui regras sintáticas que devem ser seguidas nos documentos XML para que esses sejam considerados bem-formados. Algumas dessas regras são: iniciar o documento XML com uma declaração XML, especificando, por exemplo, a versão XML; possuir um elemento raiz que contenha todos os outros; aninhar as marcas corretamente; utilizar marcas de início (por exemplo, <nome>) e de fim (por exemplo, </nome>) para elementos que não estejam vazios; fechar corretamente marcas vazias (por exemplo, <autor nome=“ Ed” />); e usar aspas em todos os atributos.

Um documento XML possui uma estrutura formal e concisa, muito específica. A estrutura lógica e a estrutura física de um documento XML são definidas como segue [73]:

- Estrutura lógica: define os elementos que compõem o documento XML, estabelecendo os tipos de dados, atributos e outros;
- Estrutura física: fornece o contéudo dos elementos, tais como: texto, imagem ou outros, dependendo da estrutura.

O exemplo abaixo apresenta um documento XML simples, para exemplificar a estrutura física de um documento XML.

```
<?xml version="1.0" standalone = "yes" encoding = "UTF-8"?>
<alunos>
<aluno>
   <nome>Mariana</nome>
   <ra>12345</ra>
</aluno>
aluno>
   <nome>Rafael</nome>
   <ra>12645</ra>
</aluno>
</alunos>
```

No exemplo o documento XML armazena dados sobre alunos de determinada instituição. O elemento <alunos> é o elemento raiz; esse elemento contém o elemento <aluno>; que contém os elementos <nome> e <ra>. Os elementos <nome> e <ra>, referentes a nome e registro acadêmico do aluno respectivamente, estão associados a um conteúdo. Os documentos XML associados a um determinado esquema usam uma mesma estrutura lógica, mas variam em estrutura física, pois o seu conteúdo pode ser diferente.

Como já foi mencionado, um documento XML precisa ser validado; mas é importante ressaltar que, para ser considerado válido, ele também precisa ser bem-formado. A validação de XML é processada por analisadores (*parsers*). Um analisador é um programa que analisa a sintaxe ou estrutura de um arquivo. Em XML o analisador vai ler e interpretar o documento, primeiro testando a sua boa formação e depois verificando a sua validade com respeito a um esquema. A seguir são abordados dois tipos de esquemas XML bastante populares.

Esquema XML

Em XML a estrutura lógica do documento é separada do seu conteúdo. Essa estrutura lógica é definida por meio de um esquema, isto é, uma gramática ou vocabulário, no qual são especificadas as

regras que definem os dados dos documentos XML. Esses esquemas possibilitam que para cada aplicação sejam definidas linguagens de marcação específicas ao tipo de informação que será armazenada, capturada, manipulada e processada por meio de um documento XML. Portanto, um esquema é uma forma de definir a estrutura de documentos XML, a partir da qual instâncias XML são construídas.

DTD

A definição de tipo de documento (*Document Type Definition* - DTD) [76] é um conjunto de regras que definem a estrutura lógica de um documento XML. Em uma DTD são definidos os elementos, os atributos, as entidades e todas as regras que devem ser seguidas pelo documento XML.

A DTD pode ser declarada no documento ou como um documento externo, normalmente acessado por um Identificador de Recurso Uniforme (*Uniform Resource Identifier* - URI [2]). Uma DTD interna descreve a estrutura de um único documento; ao contrário, uma DTD externa descreve a estrutura lógica de um conjunto de documentos XML, ou seja, pode-se considerar que a DTD externa descreve um padrão de vocabulário para um conjunto de documentos XML de um mesmo domínio de aplicação.

Basicamente, uma DTD contém:

- Elementos - componentes principais de XML.
- Marcas - caracteres utilizados para delimitar os elementos.
- Atributos - informações para descrever melhor um elemento.
- Notações - referências para aplicações auxiliares ou *plug-ins*.
- Entidades - variáveis para descrever referências de texto comuns.
- PCDATA - dados tipo caractere ou texto que são analisados (caracteres que são analisados pelos *parsers* para tratar marcas no texto como marcas e traduzir entidades de acordo com o seu significado).
- CDATA - dados tipo caractere que não são analisados.

A seguir são vistos alguns símbolos usados em DTDs para indicar ocorrência, seqüência e hierarquia:

[2] URI é uma superclasse das URLs (*Uniform Resource Locator*) que identifica recursos na Internet.

- Ocorrência
  - ? - um ou nenhum (o elemento é opcional).
  - * - o elemento pode ocorrer zero ou mais vezes.
  - + - o elemento pode ocorrer uma ou mais vezes.
- Seqüência
  - , - separa itens de uma lista, indicando uma seqüência.
  - | - indica alternância.
  - `&` - indica entidade.
- Hierarquia
  - ( ) - indica um agrupamento.

As bases da DTD são os elementos e os atributos. Elementos e atributos são declarados conforme a estrutura a seguir:

```
<!ELEMENT nome_do_elemento (tipo dos dados)>
<!ATTLIST nome_do_elemento nome_do_atributo tipo valor_default>
```

Em `valor_default` no atributo, são possíveis os valores: `#REQUIRED` (obrigatório, senão retorna um erro); `#IMPLIED` (opcional); `#FIXED` valor (cada instância do elemento deve ter o valor definido); e, `EMPTY` (não contém dados, apenas uma marcação). A seguir é dado um exemplo de DTD externa para o documento XML do exemplo anterior. No exemplo, a primeira linha indica "alunos" como o elemento raiz do documento XML, que contém os outros elementos; o elemento "aluno" é composto dos elementos "nome" e "ra", sendo que "nome" e "ra" são do tipo `#PCDATA`, ou seja, dados do tipo caractere analisados.

```
<!ELEMENT alunos (aluno+)>
<!ELEMENT aluno (nome, ra)>
<!ELEMENT nome (#PCDATA)>
<!ELEMENT ra (#PCDATA)>
```

Algumas desvantagens são associadas à DTD, tais como: seguir uma sintaxe diferente de XML; ser difícil de ser lida e compreendida; e não possuir uma definição de tipos de dados mais detalhada.

XML *Schema*

A XML *Schema Language* ou simplesmente XML *Schema* [77] é uma linguagem de esquemas para XML que se tornou uma recomendação oficial do W3C. Algumas vantagens de XML *Schema* em relação à DTD são:

- usa a mesma sintaxe de XML, podendo ser analisada por um *parser* XML;
- possui suporte a tipos de dados e permite tipos de dados definidos pelo usuário;
- é extensível, ou seja, permite o reuso de um esquema em outros esquemas.

A principal função de XML *Schema* é descrever a estrutura lógica de documentos XML, definindo elementos, atributos, subelementos, a ordem dos elementos, a ocorrência dos elementos, tipos de dados de elementos e atributos, valores padrão ou fixo de elementos e atributos; enfim, todas as restrições inerentes aos dados dos elementos e atributos.

A estrutura básica de um documento XML *Schema* é:

```
<?xml version="1.0"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.xxx.com">
  <!--declaração de elementos, atributos e tipos -->
</schema>
```

Um XML *Schema* é um documento XML, por isso é necessária a declaração XML. A seguir podem ser feitas declarações dos *namespaces*, que contêm definições usadas no esquema XML (por exemplo, `xmlns="http://www.w3.org/2001/XMLSchema"`); depois pode ser declarado o *namespace* que corresponde ao esquema. O *namespace* alvo (*target namespace*) é o URI do *namespace* que deve ser usado por documentos XML validados por esse esquema (por exemplo, `targetNamespace="http://www.xxx.com"`).

Em um XML *Schema*, os elementos e atributos são declarados como segue.

Declaração de elemento: `<element name="ano" type="date"/>`
Declaração de atributo: `<attribute name="ano" type="date"/>`

Um elemento declarado como visto acima, é denominado um elemento simples porque não contém subelementos (outros elementos) ou atributos; caso contrário, o elemento é considerado um tipo complexo, ou seja, o elemento possui uma definição de tipo complexo. Já um atributo é sempre declarado como um tipo simples.

Um elemento declarado como um tipo complexo pode ser um elemento vazio, um elemento que contém subelementos, um elemento que contém somente texto ou um elemento que contém subelementos e texto. Um elemento declarado como tipo simples não contém subelementos e atributos, é baseado em tipos básicos (*string*, número, data, ...). Uma definição de tipo simples pode conter restrições ao tipo de dado para determinar limites ao seu conteúdo. A seguir são dados exemplos de definição de tipo complexo e de tipo simples com restrição de conteúdo:

Tipo complexo:

```
<complexType name="aluno">
  <sequence>
    <element name="nome" type="string"/>
    <element name="ra" type="string"/>
  </sequence>
</complexType>
```

Tipo simples:

```
<simpleType name="idade">
  <restriction base="integer">
    <minInclusive value="18"/>
    <maxInclusive value="70"/>
  </restriction>
</simpleType>
```

As restrições de contéudo controlam os valores permitidos para os elementos ou atributos de XML. Essas restrições são impostas aos elementos e atributos por meio do tipo de dado declarado, e também, com a adição de outras limitações inerentes ao conteúdo denominadas facetas (*facets*). As facetas que podem ser definidas em um XML *Schema* são:

- Restrição nos valores - especifica máximos e mínimos para valores numéricos de determinado elemento ou atributo. Essa restrição é determinada por meio das palavras-chave: *maxInclusive* e *minInclusive* (incluindo o valor indicado) ou *maxExclusive* e *minExclusive* (excluindo o valor indicado). Veja o exemplo abaixo.

```
<element name="idade">
<simpleType>
  <restriction base="integer">
    <minExclusive value="17"/>
    <maxExclusive value="71"/>
  </restriction>
</simpleType>
</element>
```

- Restrição em um conjunto de valores - determina os valores aceitáveis em um elemento ou atributo. Essa restrição é determinada por meio da palavra-chave: *enumeration*. Veja o exemplo abaixo.

```
<element name="cor">
<simpleType>
  <restriction base="string">
    <enumeration value="vermelho"/>
    <enumeration value="amarelo"/>
    <enumeration value="verde"/>
  </restriction>
</simpleType>
</element>
```

- Restrição em uma série de valores - determina uma série de números ou letras que podem ser empregadas em um elemento ou atributo. Essa restrição é determinada por meio da palavra-chave: *pattern*. Veja o exemplo abaixo.

```
<element name="serie1">
<simpleType>
  <restriction base="integer">
    <pattern value="[1-8]"/>
  </restriction>
</simpleType>
</element>
```

- Restrição em caracteres de espaço em branco - especifica como caracteres de espaço em branco (tabulação, espaço, retorno de carro e avanço de linha) devem ser processados no documento XML. Essa restrição é determinada por meio da palavra-chave: *whiteSpace* e pode receber os valores: *replace* (todos os caracteres de espaço em branco devem ser trocados por espaços), *preserve* (todos os caracteres de espaço em branco devem ser preservados) e *collapse* (todos os caracteres de espaço em branco devem ser removidos). Veja o exemplo abaixo.

```
<element name="nome">
<simpleType>
  <restriction base="string">
    <whiteSpace value="preserve"/>
  </restriction>
</simpleType>
</element>
```

- Restrição no tamanho - especifica o tamanho de um valor em quantidade de caracteres. Essa restrição é determinada por meio das palavras-chave: *length*, *maxLength* e *minLength*. Veja o exemplo abaixo.

```
<element name="nome">
<simpleType>
  <restriction base="string">
    <maxLength value="60"/>
  </restriction>
</simpleType>
</element>
```

- Restrição no dígito - especifica a quantidade de dígitos ou dígitos decimais que um valor pode ter, por meio das palavras-chave: *totalDigits* e *fractionDigits* respectivamente. Veja o exemplo abaixo.

```
<element name="numero">
<simpleType>
  <restriction base="integer">
    <totalDigits value="2"/>
  </restriction>
</simpleType>
</element>
```

Um elemento do tipo complexo também pode ser definido com base em um tipo complexo existente, com a adição de alguns elementos, por meio do *complexContent*. Veja o exemplo.

```
<xs:complexType name="pessoa">
  <xs:sequence>
    <xs:element name="nomeCompleto" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="cliente">
  <xs:complexContent>
    <xs:extension base="pessoa">
      <xs:sequence>
        <xs:element name="codigo" type="xs:string"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Além das restrições para controlar os valores de elementos e atributos em documentos XML, XML *Schema* possui indicadores que controlam aspectos como ocorrência, ordem e grupo.

Indicadores de ocorrência são usados para especificar quantas repetições de um determinado elemento podem ocorrer em um documento XML. Esses indicadores são: *maxOccurs* (quantidade máxima de ocorrência de um elemento) e *minOccurs* (quantidade mínima de ocorrência de um elemento). O valor padrão (*default*) para *maxOccurs* e *minOccurs* é 1 (uma ocorrência do elemento). O valor de *maxOccurs* igual a *unbounded* significa que a quantidade de repetições para o elemento é ilimitada. Veja o exemplo a seguir.

```
<element name="aluno" type="string" minOccurs="0" maxOccurs="unbounded"/>
```

Os indicadores de ordem para elementos de um documento XML são estabelecidos por *all*, *choice* e *sequence*. *all* indica que os subelementos de um elemento podem aparecer em qualquer ordem e que cada subelemento deve ocorrer no máximo uma vez. *choice* especifica que somente um dos subelementos de um elemento pode aparecer, deve ser feita uma escolha. *sequence* indica que os subelementos de um elemento devem aparecer na seqüência em que foram definidos. Veja o exemplo.

```
<element name="aluno">
<complexType>
<all>
    <element name="nome" type="string"/>
    <element name="ra" type="string"/>
</all>
</complexType>
</element>
```

Os indicadores de grupo são *group* e *attributeGroup*. Esses indicadores definem respectivamente conjuntos de elementos e atributos relacionados, que podem ser referenciados em uma definição de outro grupo ou de um tipo complexo, caracterizando o reuso de uma definição de elementos ou atributos. A referência a um grupo é feita por meio da palavra-chave *ref*. Veja exemplos abaixo.

Grupo para elemento

```
<group name="grupoaluno">
<sequence>
    <element name="nome" type="string"/>
    <element name="ra" type="string"/>
</sequence>
</group>
<!--referencia ao grupo definido-->
```

```
<element name="aluno">
<complexType>
<sequence>
    <group ref="grupoaluno"/>
    <element name="endereco" type="string"/>
</sequence>
</complexType>
</element>
```

Grupo para atributo

```
<attributeGroup name="atgraluno">
  <attribute name="nome" type="string"/>
  <attribute name="sobrenome" type="string"/>
</attributeGroup>
```

Em XML *Schema* o conteúdo de um documento XML associado a um esquema pode ser estendido com elementos ou atributos não especificados nesse esquema, por meio dos elementos *<any>* e *<anyAttribute>*. Veja um exemplo do uso do elemento *<any>* a seguir.

Definição do conteúdo de aluno em um esquema.

```
<element name="aluno">
<complexType>
<sequence>
    <element name="nome" type="string"/>
    <element name="ra" type="string"/>
    <any minOccurs="0"/>
</sequence>
</complexType>
</element>
```

Definição do conteúdo de endereço em outro esquema que será estendido em aluno.

```
<element name="endereco">
<complexType>
<sequence>
    <element name="rua" type="string"/>
</sequence>
</complexType>
</element>
```

Fragmento do XML associado aos dois esquemas acima porque o esquema aluno possui o elemento *<any>*.

```
...
<aluno>
    <nome>Victor</nome>
    <ra>124567</ra>
    <endereco>
       <rua>Manaus</rua>
    </endereco>
</aluno>
...
```

Um elemento ou um atributo pode ter um valor padrão (*default*) ou fixo (*fixed*) definido. O valor padrão é aquele que é assumido pelo elemento ou atributo se outro valor não for especificado e o valor fixo é o único valor que o elemento ou atributo pode assumir. Para controlar a ocorrência de um atributo, esse pode ser definido como opcional (*optional*) ou obrigatório (*required*) por meio da palavra-chave *use*. Veja exemplos a seguir.

Elemento padrão ou fixo

```
<element name="clube" type="string" default="SM"/>
```

Atributo padrão ou fixo

```
<attribute name="cor" type="string" fixed="verde"/>
```

Atributo opcional ou obrigatório

```
<attribute name="idade" type="integer" use="required"/>
```

2.1.2 Base de dados

Uma base de dados compreende um conjunto de informações inter-relacionadas. Os dados de uma base de dados são armazenados e mantidos por meio de níveis de abstração, que são: o nível físico, que indica como armazenar os dados; o nível lógico, que define os dados que devem ser armazenados; e o nível visão, que permite aos usuários acessarem somente a parte que lhes interessa da base de dados [67].

Alguns conceitos importantes com relação a base de dados são:

- Modelo de dados - uma coleção de conceitos capazes de descrever os dados, seus relacionamentos, sua semântica e suas restrições.
- Instância da base de dados - um conteúdo específico da base de dados que varia com o tempo.

- Esquema da base de dados - a definição da estrutura lógica da base de dados, isto é, o projeto e a especificação dos dados que serão armazenados na base de dados.

- Sistema de Gerenciamento de Base de Dados (SGBD) - um conjunto de programas para acessar os dados de uma base de dados. Um SGBD tem por meta providenciar um ambiente conveniente e eficiente para a recuperação e armazenamento de dados de uma base de dados. Vários modelos de dados já foram propostos para SGBDs, entre eles o modelo de dados relacional.

O modelo de dados relacional [16] é baseado em uma coleção de relações que representam os dados. As relações estão associadas a entidades do mundo real. Uma relação é freqüentemente vista como uma tabela, na qual cada linha representa dados relacionados a uma determinada entidade e cada coluna corresponde a uma característica particular da entidade, ou seja, a um atributo. Os valores possíveis de cada atributo são especificados por um domínio, que é um conjunto de valores atômicos do mesmo tipo. Formalmente, um esquema de relação $R(A_1, ..., A_n)$ é um nome de relação R (nome de tabela) com uma lista de atributos A_i (nomes de colunas); cada A_i possui um domínio (tipo de dado) $dom(A_i)$. Uma relação de um esquema de relação é um conjunto de tuplas, no qual cada tupla é um elemento do produto cartesiano $dom(A_1)\ x\ ...\ x\ dom(A_n)$. Um esquema de relação descreve a estrutura dos dados e não é freqüentemente alterado; uma relação descreve o estado dos dados em um momento particular e é constantemente modificada para refletir alterações na entidade correspondente [14].

Uma base de dados relacional é baseada no modelo relacional. Desta forma, um esquema de uma base de dados relacional é um conjunto de esquemas de relações e um conjunto de restrições de integridade, que restringem os valores dos dados. Algumas dessas restrições são: restrições de domínio, especificam os possíveis valores de um atributo; restrições de unicidade, especificam determinados atributos para os quais valores duplicados não podem ocorrer; restrições de obrigatoriedade, especificam que um determinado atributo não pode ter o valor "null"; restrições de integridade referencial, especificam que valores de um determinado atributo em uma tabela R_1, devem aparecer também como valores de um atributo em uma tabela R_2; restrições de integridade semântica, restrições definidas por alguma linguagem de especificação de restrição [14].

Modelagem da base de dados

Um esquema de base de dados é obtido a partir da descrição dos dados, do relacionamento entre os dados e das restrições de consistência para os dados na base de dados, ou seja, da modelagem da base de dados. Em uma abordagem *top-down*, o projeto da base de dados é apresentado nos níveis (Figura 2.1): visão dos dados, que apresenta os requisitos da base de dados obtidos do mundo real em um nível mais alto; modelo conceitual, que é a descrição dos tipos de dados, seus relacionamentos

e regras de consistência de forma independente da implementação; modelo lógico, que é a definição lógica dos dados de acordo com o SGBD que será empregado para manipular a base de dados; e, o modelo físico, que é a especificação de como os dados serão armazenados pelo SGBD.

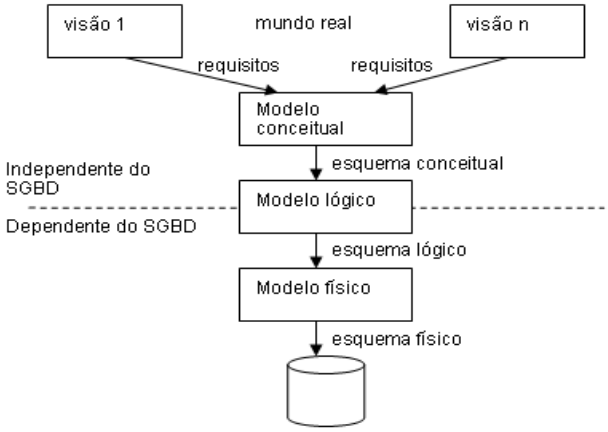

Figura 2.1: Níveis da abordagem *top-down* no projeto de uma base de dados

O modelo conceitual é independente do tipo de SGBD que será empregado para a base de dados; isto porque, este modelo registra os dados pertinentes à base de dados mas não registra como os dados devem ser armazenados no SGBD.

A modelagem conceitual de uma base de dados pode ser feita por meio do modelo entidade-relacionamento (MER) que, em geral, representa os dados e os relacionamentos da base de dados por meio de diagramas, denominados Diagrama Entidade-Relacionamento (DER). O MER expressa de forma direta as características de uma base de dados relacional; portanto, uma entidade (ou relacionamento) representada no MER corresponde a uma linha de uma tabela no modelo relacional e um conjunto de entidades (ou de relacionamentos) do mesmo tipo no MER corresponde a uma tabela do modelo relacional.

Modelo Entidade-Relacionamento

O modelo entidade-relacionamento (MER) é um modelo de dados semântico, no qual o aspecto semântico está na tentativa de representar o significado do dado [67].

Os conceitos fundamentais do MER são: entidade, atributo e relacionamento. Uma entidade é representada por um conjunto de atributos. Um atributo é uma característica de uma entidade, que possui um valor pertencente a um domínio. O domínio expressa os valores que podem ser assumidos por um atributo. Um atributo pode ser:

- simples - o atributo não pode ser subdividido;

- composto - o atributo pode ser subdividido;
- valorado - o atributo possui um único valor para determinada entidade;
- multivalorado - o atributo possui zero, um ou mais valores para uma determinada entidade;
- nulo - o atributo pode não ter valor para uma determinada entidade;
- derivado - o atributo possui um valor derivado de valores de outros atributos.

Um relacionamento é uma associação entre entidades, que pode conter atributos para descrevê-lo. Um relacionamento pode associar duas ou mais entidades e ser considerado uma ação que ocorre entre as entidades envolvidas.

Alguns conceitos do modelo ER, relacionados ao conteúdo da base de dados, são apresentados a seguir.

- **Cardinalidade** é a restrição que indica quantas ocorrências de uma entidade podem estar associadas à ocorrência de outra entidade por meio do relacionamento. A cardinalidade em relacionamentos binários pode ser classificada em: um para um (1:1), um para muitos (1:n), muitos para um (n:1) e muitos para muitos (n:n).
- **Entidade Fraca** é a existência de dependência de uma entidade em relação a outra, de modo que uma entidade A depende de uma entidade B para existir, e se B for excluída, A também deve ser excluída.
- **Atributo chave** é um conceito que permite distinguir entidades por meio de atributos que sejam suficientes para caracterizar completamente a entidade [75]. Observe que uma entidade é distinta em um conjunto de entidades e um relacionamento é distinto em um conjunto de relacionamentos na base de dados devido aos seus atributos. No entanto, com o conceito de atributo chave não é necessário que todos os atributos de uma entidade sejam listados para que a entidade seja identificada. A chave primária é formada por um ou mais atributos que identificam uma entidade obedecendo às propriedades de ser mínima (quantidade mínima de atributos capazes de distinguir a entidade de outra), de ser única (só existe uma entidade com tal valor de atributo no conjunto de entidades) e de não ser nula (o valor do atributo deve existir sempre). A chave estrangeira é um atributo de uma determinada entidade que é chave primária em outra entidade. O uso do conceito de chave estrangeira pode ocorrer no caso da existência de uma entidade fraca, dependente de uma entidade dominante, que não tem atributos suficientes para formar uma chave primária. A entidade dita entidade forte é dominante e possui chave primária.

- **Especialização/generalização** é o conceito que permite especificar atributos particulares a um subconjunto de um conjunto de entidades. Esse conceito está associado ao conceito de herança de atributos, pois os atributos de uma entidade genérica são herdados pela entidade especializada. A especialização/generalização pode ser classificada em:
  - total - se cada entidade genérica deve pertencer a uma entidade especializada;
  - parcial - se uma entidade genérica pode não estar associada a uma entidade especializada;
  - exclusiva - se uma entidade genérica somente pode estar associada a uma entidade especializada;
  - compartilhada - se uma entidade genérica pode estar associada a mais de uma entidade especializada.
- **Agregação** é uma abstração na qual um relacionamento é tratado como uma entidade. Isto ocorre porque um relacionamento não pode ser associado a outro relacionamento no MER.

Diagrama de Entidade-Relacionamento

O Diagrama de Entidade-Relacionamento (DER - [15]) é empregado na representação gráfica do MER. A seguir é apresentado na Figura 2.2 um exemplo simples de DER e descrita a notação empregada.

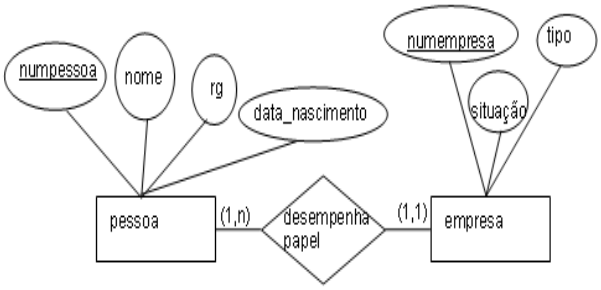

Figura 2.2: Exemplo de DER

No DER os retângulos representam as entidades. Na Figura 2.2, as entidades são *pessoa* e *empresa*. A cardinalidade (por exemplo, (1,1) e (0,n)) é anotada no diagrama junto as linhas que associam as entidades por um relacionamento. Os atributos são ligados as entidades que eles caracterizam, por exemplo, *numpessoa*, *nome*, *rg* e *data_nascimento* caracterizam a entidade *pessoa*. Os atributos sublinhados são atributos chave, por exemplo, *numpessoa*. Os atributos podem ter cardinalidade anotada ao seu lado, da mesma forma que uma entidade em um relacionamento. Considerando a notação

de cardinalidade igual a (cardinalidade mínima, cardinalidade máxima), *1* em cardinalidade mínima indica atributo obrigatório e *0* indica atributo opcional; *1* em cardinalidade máxima indica atributo com somente um valor e *n* indica atributo multivalorado.

O IDEF1X (*ICAM* [3] *DEFinition language*) [4, 38] é uma variação do DER muito adotada para gerar um modelo gráfico lógico das informações de uma base de dados, representando a estrutura e a semântica da informação em uma aplicação específica. A Figura 2.3 ilustra o exemplo da Figura 2.2 usando IDEF1X. Na Figura 2.3 as entidades *pessoa* e *empresa* são representadas por caixas, com o nome da entidade acima da caixa. Na caixa são inseridos os atributos em duas seções, na primeira seção são listados os atributos chave e na segunda seção os demais atributos. O relacionamento *desempenha papel* é mostrado por uma linha rotulada e a cardinalidade é vista por meio de pontos pretos nas extremidades dos relacionamentos indicando muitos-para-muitos e por *P* indicando um ou mais, *Z* indicando zero ou um e um número inteiro indicando um número exato.

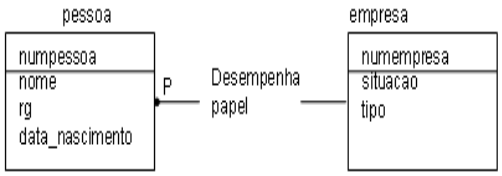

Figura 2.3: Exemplo de diagrama em IDEF1X

2.2 Metamodelagem

Esta seção aborda conceitos relacionados à metamodelagem, descrevendo a especificação MOF (*Meta Object Facility*). Essa especificação é abordada por ser utilizada para a definição do metamodelo de dados que descreve os modelos de dados que podem ser submetidos à abordagem de teste de esquema de dados proposta neste trabalho.

2.2.1 Metamodelo

Metamodelos são modelos de modelos, ou ainda, são os meios pelos quais objetos em um determinado ambiente podem entender e manipular novos objetos e dados criados a partir deles [2].

O termo *meta* é usado em qualquer linguagem ou notação capaz de fazer sentenças sobre si mesmo, usando a própria linguagem ou notação. Em se tratando de modelagem de objetos, o termo

[3] ICAM é abreviação para *Integrated Computer-Aided Manufacturing*

meta indica que os conceitos de modelagem de objetos são usados para descrever eles próprios. Dessa forma, os elementos de um modelo são instâncias de um metamodelo em um nível superior.

A maioria dos *frameworks* de modelagem adota quatro níveis em sua arquitetura [2]. Na Figura 2.4, $M0$ é o nível de dados; $M1$ é o nível de modelo tradicional; $M2$ é o nível do metamodelo; e $M3$ é o nível de meta-metamodelo.

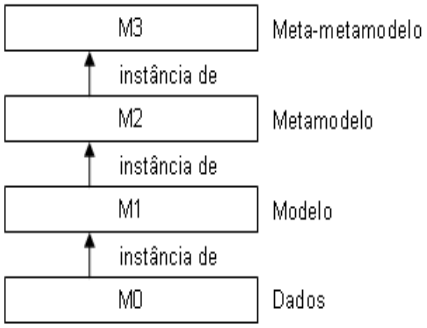

Figura 2.4: Arquitetura de Metamodelagem

2.2.2 *Meta Object Facility*

A especificação MOF (*Meta Object Facility*) [56, 57], do consórcio OMG (*Object Management Group, Inc.*), é um *framework* de integração dirigido a modelo, criado para definir, manipular e integrar metadados e dados independentemente de plataforma. MOF é um *framework* de metamodelagem usado para definir outros *frameworks* de modelagem na OMG, tais como: *Unified Modeling Language* (UML) e *Commom Warehouse Metamodel* (CWM) [33]. Para MOF, um modelo é qualquer coleção de metadados relacionados [66].

A primeira versão da especificação MOF (MOF 1.1) foi adotada pela OMG em novembro de 1997, seguida pelas versões MOF 1.3 (junho de 1999), MOF 1.4 (outubro de 2001) e, atualmente, MOF 2.0 (janeiro de 2006). Da mesma forma que em MOF 1.1, MOF 2.0 pode ser usada para definir e integrar metamodelos usando conceitos simples de modelagem de classe, por meio da notação de modelagem de classe UML. O mais importante em MOF 2.0 é a unificação dos conceitos de modelagem de MOF 2.0 e UML 2.0, de modo que seja usada uma biblioteca comum, sem qualquer adição, nas duas especificações. UML 2.0 fornece um *framework* de modelagem e notação, enquanto MOF 2.0 providencia um *framework* de gerenciamento de metadados e serviços de metadados para habilitar o desenvolvimento e interoperabilidade de sistemas dirigidos a modelos e metadados.

Os principais benefícios de MOF 2.0 são: regras simples de modelagem de metadados (subconjunto de classes UML de modelagem sem qualquer adição); tecnologias de mapeamento de MOF (por

exemplo, XMI - *XML Metadata Interchange*) também podem ser empregadas para UML; ferramentas de modelagem UML também podem ser usadas para a modelagem de metadados.

A especificação MOF usa o formato XMI para troca de metadados (dados que descrevem dados) entre repositórios [34]. XMI define como modelos MOF e instâncias de modelos MOF podem ser trocados usando XML baseado em DTDs ou XML *Schemas* gerados do modelo correspondente [33].

Arquitetura de meta-modelagem MOF

A arquitetura de MOF 1.4 é baseada na arquitetura de metamodelagem clássica em quatro camadas. A principal característica desse tipo de arquitetura é a camada de meta-metamodelo. A Figura 2.5 apresenta a arquitetura do MOF.

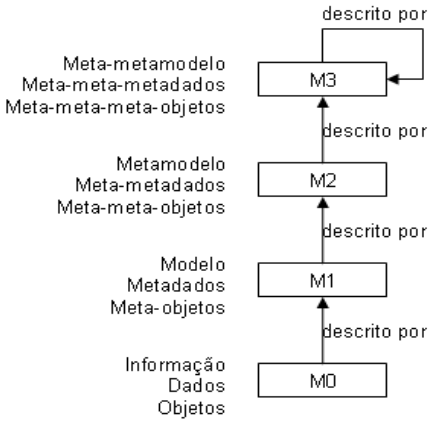

Figura 2.5: Arquitetura de MOF

O nível M3 corresponde à camada de meta-metamodelo e é denominado Modelo MOF. Esse nível descreve a estrutura e a semântica da linguagem abstrata de MOF, ou seja, a linguagem definida para a especificação de metamodelos.

O nível M2 corresponde ao metamodelo e descreve a estrutura e a semântica de um modelo particular de acordo com os requisitos específicos de um determinado domínio, usando o Modelo MOF.

O nível M1 é a camada de modelo, contendo um modelo que descreve as instâncias ou dados da camada de informação. Um modelo da camada M1 é descrito de acordo com um metamodelo do nível M2.

O nível M0 é a camada mais baixa, a camada de informação que contém as instâncias ou dados de acordo com o modelo do nível M1.

A Figura 2.6 mostra que os metamodelos UML e CWM, exemplos de metamodelos padrão definidos pelo OMG, são definidos como instâncias do Modelo MOF.

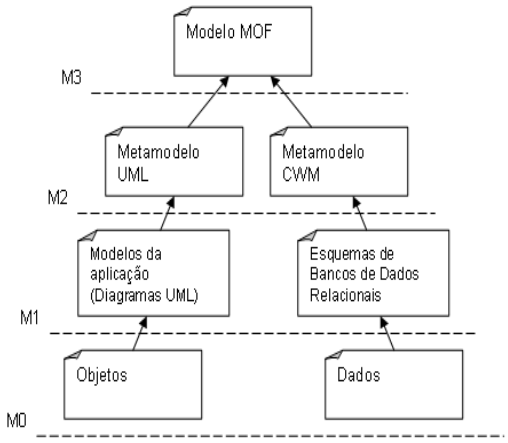

Figura 2.6: Metamodelos instâncias do Modelo MOF

Na Figura 2.6 pode ser visto que o metamodelo UML define modelos UML de uma determinada aplicação de software e é uma instância do Modelo MOF; os modelos UML são especificados como instâncias das classes correspondentes do metamodelo UML e os objetos são instâncias das classes especificadas nos modelos correspondentes. Na Figura 2.6 também pode ser observado que o metamodelo CWM é descrito como uma instância do Modelo MOF e que as tabelas, atributos e tipos de dados de um determinado esquema de banco de dados são instâncias das classes do CWM descrevendo elementos de esquemas de bancos de dados [6].

No entanto, o número de camadas na arquitetura de metamodelagem de MOF não é fixo, usualmente são quatro camadas, mas poderiam ser usadas mais ou menos camadas em dependência de como MOF está sendo empregado. MOF 2.0 permite qualquer número de camadas maior ou igual a dois; isto porque os conceitos principais da modelagem são classe e objeto (ou classificador e instância) e a capacidade de descrever um objeto por meio de sua classe.

O Modelo MOF

O Modelo MOF providencia um conjunto de construções de metamodelagem para construção de metamodelos. Esse modelo é orientado a objetos, isto é, construtores de modelagem de objetos UML são usados para representar os metamodelos. Os quatro principais construtores de MOF são: classes (modelam meta-objetos MOF), associações (modelam o relacionamento binário entre meta-objetos), tipos de dados (modelam outros dados) e pacotes (modularizam os metamodelos). Além disso, MOF é auto-descritivo, ou seja, o Modelo MOF se auto-descreve por meio das construções usadas para descrever os metamodelos.

O MOF 2.0 usa um subconjunto de UML 2.0 para providenciar conceitos e notação gráfica para o Modelo MOF. Esse modelo é formado de dois pacotes principais:

- *Essential* MOF (EMOF) - constitui um núcleo mínimo de capacidades de metamodelagem para definir metamodelos simples usando conceitos simples, mas permitindo extensões para metamodelagem mais complexa usando CMOF. EMOF se junta com pacote básico de UML 2.0 e inclui capacidades adicionais, reflexão, identificador e extensão para providenciar serviços para descobrir, manipular, identificar e estender metadados.

- *Complete* MOF (CMOF) - é o meta-metamodelo usado para especificar outros metamodelos, que é construído de EMOF e do pacote *Core* de UML. CMOF não define qualquer classe, ele une os pacotes com suas extensões para definir capacidades de metamodelagem.

Assim sendo, além de EMOF e CMOF, MOF 2.0 possui o pacote *Core* de UML e pacotes separados para tratar de capacidades adicionais, tais como: identificadores, tipos primitivos adicionais, reflexão e extensibilidade. Veja Figura 2.7.

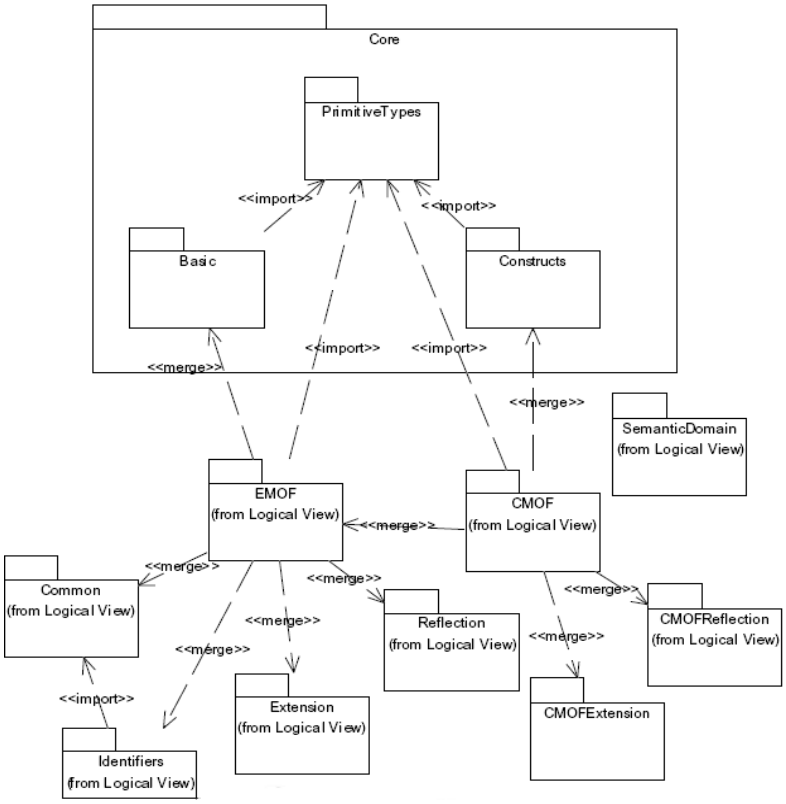

Figura 2.7: Pacotes de MOF 2.0

2.3 Teste de software

Nesta seção são discutidos conceitos básicos de teste de software, incluindo fases, critérios e ferramentas. Esses conceitos são importantes para a realização e contextualização deste trabalho no âmbito da atividade de teste de software.

2.3.1 Objetivo

O teste no processo de desenvolvimento de software é uma etapa muito importante porque possibilita um aumento no nível de confiança no desempenho das funções previstas para o software e na sua qualidade, mas não é possível afirmar ou provar por meio da realização do processo de teste que o software testado está correto [47].

O processo de teste de um software ocorre com a execução do software, na qual é observado se um dado de entrada (dado de teste) especificado gera um determinado dado de saída (resultado esperado). O dado de teste mais o resultado esperado constituem um caso de teste. No teste, pressupõe-se a existência de um oráculo (testador ou outro mecanismo) para determinar se o resultado obtido com o teste é o correto. O objetivo da execução do teste é revelar defeitos no software, de modo que um teste bem sucedido é aquele que possui casos de teste capazes de revelar defeitos ainda desconhecidos [53].

2.3.2 Fases do teste

Basicamente, quatro etapas são efetuadas durante o processo de desenvolvimento do software [53, 60]: planejamento de teste, para descrever como os testes serão conduzidos (estratégias, técnicas e critérios) e obter uma previsão de esforço, tempo e recursos; projeto de casos de teste, para selecionar casos de teste com alta probabilidade de detectar defeitos com o mínimo tempo e esforço; execução dos testes, para executar o software com os casos de teste projetados; e avaliação dos resultados dos testes, para analisar os resultados obtidos com os teste e verificar se esses resultados estão de acordo com os resultados esperados, determinando se novos casos de teste devem ser executados ou se os resultados são suficientes. Essas etapas são realizadas em quatro fases de teste:

- Teste de Unidade: testa a menor unidade do software (módulo), examinando a estrutura de dados, identificando erros de lógica e implementação.

- Teste de Integração: teste realizado na integração dos módulos do software, identificando erros de interface entre os mesmos.

- Teste de Validação: testa o software de acordo com os requisitos do usuário, identificando erros de requisitos funcionais e de desempenho.
- Teste de Sistema: testa o software com outros elementos do sistema, verificando erros de função e desempenho, tolerância a falhas e segurança.

2.3.3 Técnicas e critérios de teste

No processo de teste, existem duas questões que precisam ser abordadas: de que maneira os dados de teste devem ser selecionados para que tenham alta probabilidade de encontrar grande parte dos defeitos do software em um tempo reduzido e com o mínimo de esforço; e, de que forma saber se um software foi suficientemente testado. Critérios de teste para selecionar e avaliar conjuntos de casos de teste foram propostos para responder a essas questões e aumentar o nível de confiança na correção do software [47, 62].

Um critério de teste é um predicado que orienta o processo de teste; é usado para avaliar um conjunto de dados de teste, por meio de uma medida de cobertura, dada pelo percentual dos elementos requeridos por um critério de teste que foram exercitados pelo conjunto de dados de teste. O valor da medida de cobertura de um critério de teste pode ser usado como parâmetro para encerramento da atividade de teste [47, 62].

A seguir são apresentadas três principais técnicas de teste e os critérios correspondentes.

Técnica Funcional

O Teste de Caixa Preta ou Funcional é utilizado para demonstrar que as funções do software estão sendo realizadas de acordo com os requisitos da especificação. Nesse tipo de teste não há preocupação com os detalhes de implementação, sendo observado se os dados de entrada são apropriadamente aceitos e a resposta a esses dados ou saída é corretamente produzida. Para realização desse teste é necessário identificar as funções que o software deve executar, o que é feito por meio da especificação do software e, a partir disso, criar casos de teste para examinar tais funções.

O teste funcional permite que sejam revelados defeitos quanto a: funções incorretas ou ausentes, erros de interface, erros na estrutura de dados ou no acesso externo a bases de dados, erros de desempenho, erros de terminação ou inicialização. Exemplos de critérios de teste funcional, são [53, 60]:

- Particionamento em classes de equivalência: o domínio de entrada de um programa é identificado na especificação e dividido em classes de equivalência válidas e inválidas, sendo que são selecionados casos de teste a partir das classes geradas. O número de casos de teste deve ser o menor possível, porque pressupõe-se que um caso de teste de cada classe de equivalência válida e inválida é representativo de cada uma das classes.

- Análise de valores limites: considerando o particionamento em classes de equivalência, nesse caso, não são selecionados quaisquer casos de teste de cada classe, mas os casos de teste que estão nas fronteiras das classes, porque nesses pontos podem estar concentrados o maior número de defeitos.

- Grafo de causa-efeito: nesse critério são consideradas condições de entrada (causas) e possíveis ações (efeitos) para construir um grafo de causas e efeitos que é transformado em uma tabela de decisão, da qual são derivados os casos de teste.

Uma desvantagem na realização do teste funcional decorre da especificação do problema, que pode ter sido feita de modo descritivo e não formal, o que torna os requisitos de teste pouco precisos, informais e difíceis de automatizar. Uma vantagem é a necessidade de identificar apenas as entradas, a função a ser computada e o resultado esperado. Esse teste pode ser aplicado em praticamente todas as fases de teste (unidade, integração, validação e sistema).

Técnica Estrutural

O Teste de Caixa Branca ou Estrutural examina a implementação (procedimentos e dados) de um programa P. Os caminhos lógicos de P são testados por casos de teste que exercitam conjuntos específicos de condições, laços e caminhos associados a um grafo de fluxo de controle do programa P. Um grafo de fluxo de controle; $G = (N, E, s)$, onde: N é um conjunto de nós, E é um conjunto de arestas e s é o nó de entrada; é um grafo orientado com um único nó de entrada e um único nó de saída, sendo que cada vértice do grafo representa um bloco indivisível de comandos e cada aresta representa um possível desvio de um bloco para outro (Figura 2.8). Um caminho é uma seqüência finita de nós $(n_1, n_2, ..., n_k)$ com $k \geq 2$ e com um arco entre eles.

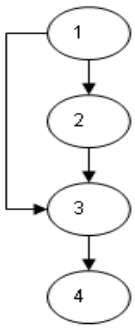

Figura 2.8: Exemplo de grafo de fluxo de controle de um programa P

Critérios de teste estrutural podem derivar casos de teste que: garantam que cada caminho de um conjunto de caminhos independentes de um módulo foi acionado pelo menos uma vez; asseguram que

todas as decisões lógicas são executadas; provocam a execução de todos os laços no seu limite e dentro do seu limite operacional; exercitam as estruturas de dados internas para assegurar sua validade; exercitam a definição e o uso de variáveis. Geralmente, os critérios de teste estrutural são classificados em:

- Critérios baseados em fluxo de controle: usam informações de fluxo de controle para determinar quais caminhos devem ser exercitados. Exemplos: critérios Todos os Nós, Todos os Arcos, e Todos os Caminhos [62];
- Critérios baseados em fluxo de dado: utilizam informações de fluxo de dados para determinar os caminhos que devem ser exercitados no teste. Exemplos: Critério Todos os Usos [61, 62] e Critérios Potenciais-Usos [47];
- Critérios baseados na complexidade: usam informações de complexidade do programa para derivar os requisitos de teste. Exemplo: Critério Todos os Caminhos Linearmente Independentes de McCabe [51].

O teste estrutural é considerado complementar em relação ao teste funcional e suas informações podem ser utilizadas em atividades de manutenção, depuração e confiabilidade de software [60].

Técnica baseada em defeitos

Teste baseado em defeitos procura demonstrar que defeitos conhecidos não estão presentes no programa produzindo um conjunto de casos de teste que diferenciem o programa original de suas alternativas [52]. As alternativas são geradas por modificações no programa original de acordo com os defeitos previamente conhecidos. Portanto, a abordagem de teste baseada em defeitos depende da definição de classes de defeitos que comumente ocorrem em programas [37]. São critérios de teste baseados em defeitos a Semeadura de Erros [5] e a Análise de Mutantes [18].

- Semeadura de Erros: insere defeitos no programa P artificialmente, com o objetivo de obter uma medida do número de defeitos artificiais revelados pelo número de defeitos naturais revelados [5]. Essa média é utilizada para estimar o número de defeitos restantes no programa P.
- Teste de mutação: explora dois pressupostos básicos: 1) hipótese do programador competente, programadores escrevem programas corretos ou muito próximos do correto, ou seja, defeitos são inseridos nos programas por meio de pequenos desvios sintáticos; e 2) efeito do acoplamento, defeitos complexos estão relacionados a defeitos simples, ou seja, se um conjunto de

casos de teste é capaz de revelar defeitos simples então esse mesmo conjunto é capaz de revelar defeitos complexos. A mutação consiste em fazer uma modificação simples no programa P, gerando os programas mutantes P_1, P_2...P_n e executando-os juntamente com P com um conjunto de casos de teste T; por fim, os resultados obtidos com a execução do programa original e dos mutantes são comparados. Se a saída do mutante é diferente da saída do original, o mutante é considerado morto; se a saída do mutante não difere da saída do programa original, o mutante é considerado vivo; se o mutante não gera saída diferente do programa original com os dados de teste, ele pode ser equivalente ao original [18]. As modificações em P são feitas por meio de operadores de mutação que são regras usadas para definir as alterações que devem ser aplicadas em P para gerar os mutantes. A análise de mutantes especifica a medida de cobertura, isto é, a medida do nível de confiança da adequação dos casos de teste analisados, dada pela razão entre o número de mutantes mortos e o número de mutantes gerados menos o número de mutantes equivalentes [19]. Essa medida varia entre 0 e 1, sendo que quanto maior o valor mais adequado é o conjunto de casos de teste T para o programa P. Outros critérios de análise de mutantes vêm sendo propostos com o propósito de reduzir os custos da aplicação dessa abordagem por meio da seleção de um subconjunto de mutantes gerados, porém sem reduzir a eficácia do critério [49]. Exemplos desses critérios são: Mutação Aleatória, Mutação Restrita e Mutação Seletiva.

2.3.4 Principais limitações da atividade de teste

A atividade de teste possui algumas limitações que não permitem sua completa automatização. Um exemplo de uma etapa da atividade de teste que é difícil de ser automatizada é a geração de dados de teste. As principais limitações são:

- Correção coincidente - o programa contém um defeito que é executado, sendo produzido um estado incorreto, porém coincidentemente um resultado correto é obtido.

- Caminhos não executáveis - não existe combinação possível de valores de entrada, parâmetros e variáveis globais do programa que causem a execução de um caminho dito não executável.

- Mutantes equivalentes - é uma questão indecidível se dois programas computam a mesma função.

- Caminhos ausentes - corresponde a uma funcionalidade que deveria estar implementada no programa, mas não foi implementada. O caminho no programa referente a essa função é dito ausente.

2.3.5 Comparação entre critérios

Estudos teóricos e empíricos comparam os critérios de teste para esclarecer como obter o melhor resultado com o menor custo no processo de teste. Esses estudos são guiados pelos seguintes fatores [82]:

- custo - esforço necessário para a aplicação de um critério, dado pelo número de casos de teste ou métricas inerentes ao critério;
- eficácia - capacidade de um critério em detectar um maior número de defeitos em relação ao outro;
- dificuldade de satisfação - probabilidade de satisfazer um critério tendo satisfeito outro.

Os critérios de teste funcionais, estruturais e baseados em defeitos são complementares porque revelam diferentes tipos de defeitos [60]. Em estudos empíricos [48, 74, 81], que consideram os fatores citados acima, o teste baseado em defeitos tem se mostrado mais eficaz porque tem maior probabilidade de revelar defeitos, porém mais custoso em relação ao número de casos de teste, quantidade de execuções e determinação de programas equivalentes.

Os critérios estruturais têm como desvantagem [31, 74] o fato de não ser muito fácil aprender seus conceitos. Além disso, apresentam outras limitações inerentes à própria atividade de teste, tais como: caminhos não executáveis e caminhos ausentes.

2.3.6 Ferramentas de teste

A aplicação de técnicas, métodos e critérios de teste é inviável na prática sem o suporte de ferramentas que os automatizem, mesmo para sistemas e programas pouco complexos. A seguir são mencionadas algumas ferramentas de interesse no meio acadêmico.

A ferramenta PokeTool (*Potential Uses Criteria Tool for Program Testing*) [9] apóia a aplicação dos critérios Potenciais-Usos em programas escritos em C, Pascal, Fortran, Cobol. A ferramenta Mothra [20] foi implementada para apoiar a utilização do critério Análise de Mutantes em programas Fortran. A ferramenta PROTEUM (*Program Testing Using Mutants*) [21] foi desenvolvida para auxiliar a utilização do critério Análise de Mutantes para o teste de programas em linguagem procedimental (exemplo, a linguagem C). Existem ferramentas que estendem a aplicação do critério Análise de Mutantes para o teste de especificação em termos de MEFs (Máquinas de Estados Finitos), Redes de Petri e Statecharts; são elas, respectivamente: PROTEUM-RS/MEF [28], PROTEUM-RS/PN [69] e PROTEUM-RS/ST [29]. As ferramentas ASSET [31], PROTESTE [68] e ATAC [35] apóiam os critérios de Rapps e Weyuker [62]; as duas primeiras para o teste de programas escritos em Pascal e a última para o teste de programas escritos em C.

2.4 Considerações finais

Em XML, o esquema pode ser visto como uma especificação para validar a correção de documentos XML empregados no intercâmbio de dados. Porém, muitas vezes os documentos XML são usados em aplicações sem um esquema associado a ele, existindo somente uma especificação dos dados escrita em documentos do projeto da aplicação, e conseqüentemente o nível de confiabilidade nos dados armazenados nesses documentos XML é menor.

Uma base de dados pode ser baseada em diferentes modelos de dados. No entanto, o modelo de dados mais comumente utilizado é o modelo relacional. O projeto da base de dados passa pelas fases de modelo conceitual, modelo lógico e modelo físico. O modelo conceitual descreve os dados, seus relacionamentos e restrições inerentes aos dados e é o foco desse trabalho. Portanto, nesse capítulo foi abordado o Modelo Entidade-Relacionamento, empregado para modelar bases de dados idealizadas para o modelo relacional. Também foi apresentado o uso do DER para representar graficamente o modelo ER.

Em relação a conceitos de metamodelagem, foi abordada principalmente a especificação MOF, por ser de interesse do trabalho. MOF é um *framework* capaz de definir, manipular e integrar metamodelos em diferentes domínios. Esse *framework* foi sendo aprimorado e atualmente sua especificação está na versão 2.0. Nessa versão, MOF é composto de um núcleo, do EMOF e do CMOF. A quantidade de camadas permitidas por MOF 2.0 é definida em no mínimo duas.

Este capítulo também abordou conceitos importantes a respeito de teste de software, tais como: as principais fases, técnicas, critérios e ferramentas de teste.

Quanto à completa automatização da atividade de teste, existem limitações inerentes a essa atividade que não permitem que isso ocorra. Algumas limitações são: caminhos ausentes, caminhos não executáveis, mutantes equivalentes e correção coincidente. Uma das etapas do teste que é afetada por essas limitações é a geração de dados de teste, que na maioria das vezes é feita manualmente. Outra questão inerente à atividade de teste é a impossibilidade da completa automatização do oráculo.

É importante ressaltar que a partir de estudos teóricos e empíricos, as técnicas de teste são vistas como complementares porque revelam diferentes tipos de defeitos. E ainda, que o teste baseado em defeitos tem se mostrado mais eficaz em relação ao número de defeitos revelados. No entanto, é mais custoso devido ao grande número de casos de teste, à grande quantidade de execuções do programa e à dificuldade de determinação de mutantes equivalentes.

Os conceitos abordados neste capítulo são relevantes para o desenvolvimento deste trabalho de pesquisa e para o entendimento dos trabalhos relacionados apresentados no próximo capítulo. Alguns desses trabalhos tratam do teste de esquema de dados, que visa a descoberta de defeitos na definição dos dados, por exemplo, quanto à definição de domínio. Outros usam informações do esquema no teste da aplicação ou da comunicação entre componentes de software.

Capítulo 3

Trabalhos Relacionados

Neste capítulo são apresentados trabalhos que envolvem o teste de aplicações Web e de base de dados. Esses trabalhos são importantes porque usam informações de esquemas de dados para testar a aplicação ou testam esses esquemas.

Muitos trabalhos na literatura focalizam o teste de aplicações Web ou de bases de dados; porém, o teste de esquemas de dados associados às mensagens XML e às bases de dados pertencentes a essas aplicações é pouco explorado. O teste de esquemas é importante porque pode aumentar a confiança na integridade e exatidão dos dados manipulados por essas aplicações de software.

3.1 Aplicações Web

Uma aplicação Web é dinâmica e heterogênea. A palavra heterogênea é comumente utilizada para indicar as diferentes formas de comunicação entre os componentes de software e as diversas tecnologias envolvidas [41, 45, 46]. Exemplos de tais tecnologias são: HTML, XML, *scripts* [1] e serviços Web (Web *services* [2]) [17]. Essas aplicações são empregadas em muitas atividades e precisam de garantia de qualidade e confiabilidade, o que pode ser obtido por meio de metodologias e ferramentas de teste para essas aplicações. Existem alguns trabalhos que têm adaptado ou modificado abordagens de teste de software tradicional para o teste de aplicações Web [23, 24, 25, 41, 45, 46, 63]; e existem outros trabalhos que testam somente alguns aspectos da aplicação Web; por exemplo, a interação entre componentes Web por meio de mensagens XML [42] e serviços Web [55, 83]. Outros trabalhos testam esquemas de dados em formato XML [32, 44]. A seguir, alguns desses trabalhos são abordados.

[1] os *scripts* permitem a execução de comandos dentro de um documento HTML, por exemplo, *JavaScript*.

[2] serviços Web são uma coleção de funções ou serviços disponíveis na rede que podem ser utilizados pelas aplicações Web.

3.1.1 Teste de aplicações que utilizam esquemas

Lee e Offutt [42] aplicam análise de mutantes para gerar casos de teste e testar a interação entre componentes da Web realizada por meio de mensagens XML. Um caso de teste é um documento XML distinto, gerado de uma DTD para avaliar o comportamento funcional da interação entre componentes. Componentes Web são um conjunto de processos de software ou combinação de processos que são executados no ambiente da *World Wide Web*, tais como: *Java server pages*, *Java servlets*, *JavaScripts*, *Active server pages*, *Java beans* e *non-bean classes*, bancos de dados e outros. Essa abordagem de teste é usada para verificar a correção semântica da interação entre componentes Web analisando as mensagens XML usadas para troca de dados. Mensagens XML mutantes são geradas a partir da original, por meio de uma alteração simples, e executadas na interação entre os componentes Web, gerando resultados que são analisados pelo testador. Um Modelo de Especificação de Interação (*Interaction Specification Model* - ISM) formalmente definido, composto por uma DTD, interfaces de mensagem e restrições, é empregado para conduzir as alterações nas mensagens XML. Os resultados da execução dos casos de teste apontam o comportamento dos mutantes de interação, de modo que esses mutantes são classificados como morto, vivo ou equivalente, da mesma forma que na análise de mutantes para software tradicional [18]. O testador examina os mutantes vivos para decidir se são equivalentes ou se novos casos de teste devem ser gerados, ou ainda, se os resultados obtidos com os casos de teste executados são satisfatórios.

Offutt e Xu [55] exploram o teste de interação entre serviços Web gerando casos de teste por perturbação de dados. O teste é realizado com a alteração de uma mensagem de requisição a um serviço Web e executando-a para analisar a resposta e o comportamento do outro serviço Web. A perturbação de dados ocorre de duas formas: perturbação de valores de dados e perturbação de interação. Na perturbação de valores de dados, valores de dados nas mensagens SOAP (*Simple Object Access Protocol*), usadas para enviar mensagens de requisição e resposta entre serviços Web, são modificados em relação ao tipo de dado. As mensagens SOAP são documentos em formato XML. Na perturbação de interação são alteradas mensagens de comunicação RPC (*Remote Procedure Calls*) usando SOAP, que são mensagens com valores para os argumentos de funções de procedimento remoto, e mensagens de comunicação de dados, que são mensagens para a transferência de dados. Um modelo formal para documentos XML, denominado RTG (*Regular Tree Grammar*), é introduzido para aplicar a perturbação de dados. Esse modelo representa um XML *Schema* e deriva documentos XML baseados nesse esquema. As definições do esquema, representadas no modelo, são adotadas para realizar as alterações nas mensagens XML.

Xu et al. [83] testam a comunicação baseada em XML entre serviços Web, modificando o esquema XML para produzir e transmitir dados inválidos e avaliar se os serviços Web estão executando efetivamente suas funções. Um modelo formal para XML é apresentado para representar a estrutura

de um esquema XML e para que os dados de teste de XML sejam produzidos. Esse modelo define três elementos: nós (elementos e atributos), tipos de dados e ramos (relação entre os nós). As alterações no esquema XML são realizadas por meio de quatro operadores de perturbação de esquema primitivos: *insertN* (insere um nó entre dois nós conectados por um ramo), *deleteN* (deleta um nó), *insertND* (insere um nó e seu tipo de dado), *deleteND* (deleta um nó e seu tipo de dado); e três operadores de perturbação de esquema não-primitivos: *insertT* (insere uma subárvore), *deleteT* (deleta uma subárvore), *changeE* (troca dois ramos da árvore). Os critérios de cobertura de teste: cobertura de exclusão (*Delete Coverage* - DC), cobertura de inserção (*Insert Coverage* - IC) e cobertura de restrição (*Constraint Coverage* - CC) são introduzidos baseados nos operadores de perturbação que modificam a estrutura do esquema XML ou que alteram restrições de valores no esquema. O caso de teste é uma mensagem XML inválida gerada com base nos esquemas modificados com os operadores de perturbação. O caso de teste inválido é usado na comunicação entre os serviços Web para verificar o comportamento desses serviços e revelar defeitos.

3.1.2 Teste em esquemas XML

Li e Miller [44] propõem operadores de mutação para esquemas XML escritos em XML *Schema*. Esses operadores fazem alterações simples no esquema, alterando um valor ou um atributo, e são classificados em cinco grupos. Os operadores estão descritos na Tabela 3.1. Os autores mostram que muitos defeitos descritos pelos operadores de mutação não são detectados pelos analisadores (*parsers*) comumente usados para validar documentos XML *Schema*. Entretanto, eles não apresentam um experimento no qual os operadores de mutação são aplicados no processo de teste de um esquema XML e não mencionam como os operadores devem ser usados para gerar casos de teste para a execução do teste de esquema.

Tabela 3.1: Operadores de mutação [44]

Operador	**Descrição**
XML *Namespace*	
SNC - XML *Schema Namespace Changes*	altera o *namespace* da linguagem XML *Schema*
TDE - *Target and Default Exchange*	troca o *target namespace* de XML *Schema* e o *default namespace*
ENR - *Element Namespace Replacement*	modifica o *namespace* identificador do elemento
continua na próxima página	

Tabela 3.1 – continuação da página anterior

Operador	Descrição
ENM - *Element Namespace Removal*	remove o *namespace* identificador do elemento
QGR - *Qualification Globally Replacement*	modifica a definição de qualificação de elementos e atributos locais
QIR - *Qualification Individually Replacement*	modifica a definição de qualificação de elementos e atributos individualmente
Definição de Tipo Complexo	
CCR - *ComplexType Compositors Replacement*	troca o tipo de compositor (ou de ordem) indicada para um elemento complexo
COC - *ComplexType Order Change*	troca a ordem dos elementos de um elemento complexo
CDD - *ComplexType Definition Decrease*	reduz o número de elementos de um elemento complexo
EOC - *Element Occurrence Change*	altera o valor da restrição de ocorrência de um elemento
AOC - *Attribute Occurrence Change*	altera o valor da ocorrência de um atributo
Declaração de Tipo Simples	
SPC - *SimpleType Pattern Change*	altera a expressão regular usada na restrição *pattern*
RAR - *Restriction Arguments Replacement*	altera o valor de restrições de limites
EEC - *Enumeration Element Change*	remove ou adiciona valores enumerados
Facetas de Tipo	
SLC - *String Length Change*	altera o valor de restrições de limites para número de caracteres de uma string
NCC - *Number Constraint Change*	altera o valor de restrições de quantidade de dígitos para números
Herança	

continua na próxima página

Tabela 3.1 – continuação da página anterior

Operador	Descrição
DTC - *Derived Type Control Replacement*	altera os modificadores de tipo para tipos complexos, que descrevem como derivar novos tipos pela troca de conteúdo de tipos existentes
UCR - *Use of Controller Replacement*	altera os modificadores de controle, que controlam a derivação e o uso de um tipo particular

Um exemplo de alteração descrita pelo operador de mutação EEC - *Enumeration Element Change* (Tabela 3.1), extraído de Li e Miller [44], pode ser visto a seguir, juntamente com o esquema original.

Fragmento do esquema original

```
<xsd:simpleType name="USState">
   <xsd:restriction base="xsd:string">
      <xsd:enumeration value="AK">
      <xsd:enumeration value="AL">
      <xsd:enumeration value="AR">
      <!-- e outros ... -->
   </xsd:restriction>
</xsd:simpleType>
```

Fragmento do esquema mutante

```
<xsd:simpleType name="USState">
   <xsd:restriction base="xsd:string">
      <xsd:enumeration value="AK">
  /*DELETE  <xsd:enumeration value="AL">*/
      <xsd:enumeration value="AR">
      <!-- e outros ... -->
   </xsd:restriction>
</xsd:simpleType>
```

Franzotte e Vergilio [32] introduzem um novo conjunto de operadores de mutação para documento XML *Schema*. Um esquema XML é representado como uma árvore, na qual os elementos são chamados de nós. Os operadores de mutação introduzidos por Franzotte e Vergilio [32] produzem uma modificação simples no esquema XML em teste e são divididos em mutação elementar e mutação estrutural. Os operadores de mutação elementar alteram valores de atributos e elementos trocando a ordem, o uso, o tipo de dado, o tamanho do nome (marca), o tamanho do conteúdo, a marca (*tag*)

e a ocorrência. Os operadores de mutação estrutural modificam a estrutura da árvore do esquema, invertendo, adicionando e removendo nós da árvore. Esses operadores são apresentados na Tabela 3.2.

Tabela 3.2: Operadores de mutação da ferramenta XTM [32]

Operador	**Descrição**
Mutação Elementar	
GO - *GroupOrder*	troca a ordem dos elementos em um grupo
REQ - *Required*	altera o uso do atributo (obrigatório ou opcional)
DT - *DataType*	modifica o tipo de dado de elementos e atributos
LO - *LenghtOF*	altera o tamanho do nome do elemento
CSP - *ChangeSing-Plural*	modifica o tamanho do elemento adicionando ou removendo caracteres
CTP - *ChangeTag*	troca as *tags* mais comuns dos nós
Mutação Estrutural	
STE - *SubTreeExchange*	inverte sub-árvores abaixo de algum nó
IT - *InsertTree*	adiciona um nó na estrutura de uma sub-árvore
RT - *RemoveTree*	remove um nó (ou sub-árvore) da estrutura da árvore

Os autores apresentam o processo de teste usado para a aplicação do teste de mutação e obtenção de métricas de cobertura. Os operadores alteram o esquema em teste, gerando os esquemas mutantes. Um exemplo de esquema em teste e esquema mutante gerado pelo operador de mutação REQ - *Required* (Tabela 3.2), extraído de Franzotte e Vergilio [32], é dado a seguir.

Fragmento do esquema em teste

```
<attributetype name="bar" dt:type="int" required="YES">
```

Fragmento do esquema mutante

```
<attributetype name="bar" dt:type="int" required="NO">
```

O esquema em teste e os mutantes são usados para validar um conjunto de dados de teste providenciado pelo testador. Os mutantes são considerados mortos, se um resultado diferente é obtido da validação do esquema mutante em relação ao esquema em teste para o mesmo dado de teste. Um *score* de mutação é obtido depois da validação de todo conjunto de teste. A ferramenta XTM (*Tool for XML Schema Testing Based on Mutation*) foi implementada na linguagem Java e apóia o teste de esquemas escritos em XML *Schema*. A XTM aplica o critério análise de mutantes empregando os operadores de mutação apresentados na Tabela 3.2. Mutantes equivalentes podem ser gerados e precisam ser identificados pelo testador; os casos de teste também são gerados manualmente.

3.2 Aplicações de base de dados

O teste em aplicações de base de dados envolve aspectos relacionados à correção, tais como: comportamento da aplicação em relação à especificação; esquema da base de dados em relação aos dados do mundo real modelados; acurácia na base de dados, segurança e privacidade; e execução correta de operações de inserção, atualização e exclusão de dados [12]. Nesse contexto, existem vários estudos sendo conduzidos para o teste de aplicações de base de dados. Esses estudos envolvem geração de dados, teste do projeto e da aplicação de base de dados [10, 12, 13, 22, 39, 43, 70, 72, 84]; alguns estudos exploram o uso de informações do esquema para o teste da aplicação de base de dados [11, 64]; e, outros investigam o teste do esquema associado à base de dados [1]. A maioria desses estudos considera aplicações de base de dados relacional. Alguns desses trabalhos são apresentados a seguir.

3.2.1 Teste em aplicações de base de dados utilizando informações do esquema

Robbert e Maryanski [64] elaboram um plano de teste de base de dados a partir do modelo conceitual, ou seja, do projeto do esquema da base de dados, para testar uma aplicação de base de dados. O Modelo Entidade-Relacionamento (MER) é usado para extrair entidades, atributos, relacionamentos e restrições da base de dados. Um gerador de plano de teste aponta os testes que devem ser executados pelo testador e quando parar. Casos de teste específicos são definidos para tratar de condições excepcionais, extremas e de valores de fronteira. Uma equipe de testadores independentes deve executar os testes e por fim, os resultados do teste devem ser analisados. Na execução do teste todas as classes de valores válidos devem ser aceitas, todas as classes de valores inválidos devem ser identificadas e todas as entidades e relacionamentos devem ser exercitados. O gerador de plano de teste permite a atualização do plano de teste quando for necessário realizar o teste novamente devido à manutenção da estrutura da base de dados. Os autores não esclarecem como é realizada a seleção de casos de teste e não definem claramente o critério de teste que está sendo utilizado na abordagem de teste.

Chan et al. [11] introduzem uma abordagem de teste baseada em defeitos para verificar a correção de sentenças SQL que manipulam registros em instâncias de base de dados. Os operadores de mutação são baseados em informações obtidas do modelo conceitual da aplicação da base de dados (MER). Esses operadores de mutação alteram sentenças SQL embutidas em uma aplicação de base de dados em teste. As sentenças SQL mutantes são executadas juntamente com a original e os resultados são comparados. O fato de um mutante ser equivalente ao original deve ser avaliado pelo testador. Nessa abordagem é usada a idéia do teste de mutação fraca, na qual é analisado o estado intermediário da instância da base de dados no ponto em que a sentença SQL é acionada da unidade do programa. Um

caso de teste é formado por uma instância de base de dados com os valores de entrada da aplicação. Porém, não é esclarecido como os casos de teste são selecionados.

3.2.2 Teste em esquemas de dados associados à base de dados relacional

Aranha et al. [1] propõem a realização de teste em base de dados para focalizar a estrutura lógica dos dados e os próprios dados. O processo de teste executa operações (sentenças SQL) na base de dados para revelar defeitos na definição de atributos e relacionamentos especificados no esquema, empregando critérios de teste baseados na definição e uso de atributos de uma base de dados relacional. Essa abordagem de teste compreende: o teste de relação, que exercita uma relação da base de dados para detectar defeitos na estrutura dos atributos e de suas definições; o teste de relacionamento, que exercita as chaves para encontrar defeitos no relacionamento entre as relações. O caso de teste é uma sentença SQL associada a um resultado esperado. A ferramenta RDBTool (*Relational Data-Base Testing Tool*) foi implementada para apoiar essa abordagem de teste. Essa ferramenta executa a análise estática do esquema da base de dados em teste, extraindo dados sobre a definição de atributos e relacionamentos e determinando os elementos requeridos com o auxílio do testador, que deve escolher os elementos que serão testados e os critérios de teste que serão empregados. Os critérios de teste, citados em [1], são: todos os tipos de definição, todos os tipos de uso predicativo, associação de tipos de definição, pelo menos um tipo de definição e pelo menos um tipo de uso predicativo. Com base nessas informações são geradas as sentenças SQL para testar cada elemento requerido e executar o teste. Os resultados do teste permitem que o testador avalie a cobertura dos critérios e possíveis defeitos no esquema.

Um exemplo de aplicação do critério "pelo menos um tipo de uso predicativo", extraído de Aranha et al. [1], é dado a seguir. Suponha que esse critério esteja sendo aplicado a um atributo "Idade"; é requerida a execução de pelo menos um grupo das operações *Select*, *Update* e *Delete*, considerando "Idade" $<$ 18 e "Idade" $>=$ 18, para que o critério seja satisfeito. O grupo relacionado a operação SQL *Update* que poderia ser executado é especificado a seguir:

Grupo *Update*

```
Operação: Update <cláusula> where Idade < 18
Relação: Aluno
Atributo: Idade
Resultado Esperado: Não satisfaz a restrição - 0 tuplas atualizadas
-----
Operação: Update <cláusula> where Idade >= 18
Relação: Aluno
Atributo: Idade
```

```
Resultado Esperado: Satisfaz a restrição - pelo menos 1 tupla atualizada
```

Essa abordagem de teste é baseada em um critério de teste estrutural e aplicada apenas para base de dados relacional. A geração das sentenças SQL e dos dados de teste é manual.

3.3 Considerações finais

Este capítulo discutiu trabalhos relacionados ao teste de aplicações Web e teste de aplicações de base de dados, enfatizando o teste de esquema XML e de esquema associado à base de dados relacional.

Os trabalhos de Lee e Offutt [42], Offutt e Xu [55] e Xu et al. [83] exploram a idéia de operadores de mutação e perturbação de dados para o teste da interação entre componentes Web usando mensagens XML. No entanto, esses trabalhos não tratam do teste de esquema de dados para XML.

O trabalho de Li e Miller [44] introduz operadores de mutação para XML *Schema*, porém não define o processo de teste que deve ser executado para que os defeitos descritos pelos operadores sejam revelados e não menciona como casos de teste devem ser gerados para detectar esses defeitos. No trabalho de Franzotte e Vergilio [32], um novo conjunto de operadores de mutação é introduzido, o critério análise de mutantes é aplicado para XML *Schema* e uma ferramenta para apóiar a execução do teste de mutação é implementada. Entretanto, durante o processo de teste o testador deve gerar os casos de teste manualmente; esquemas mutantes equivalentes podem ser gerados e devem ser identificados pelo testador. Além disso, esses trabalhos [32, 44] introduzem operadores de mutação para esquemas XML escritos apenas em XML *Schema*.

O trabalho de Robbert e Maryanski [64] usa informações do esquema da base de dados para gerar um plano de teste para a aplicação de base de dados e o trabalho de Chan et al. [11] propõe uma abordagem de teste baseada em defeitos para testar sentenças SQL de uma aplicação de base de dados, usando informações obtidas do modelo de dados conceitual da base de dados. Portanto, esses trabalhos usam informações obtidas do esquema de dados da aplicação de base de dados, mas não testam o esquema de dados associado à essas aplicações. No trabalho de Aranha et al. [1], esquemas associados à base de dados são testados por meio de critérios de teste baseados na definição e uso de atributos, usados para gerar sentenças SQL. Entretanto, esse trabalho focaliza apenas aplicações de base de dados relacional, as sentenças SQL e os dados de teste são gerados manualmente.

Portanto, alguns trabalhos exploram o teste de aplicações utilizando informações do esquema de dados associado a essas aplicações e poucos trabalhos investigam o teste de esquema de dados. Os trabalhos que testam esquemas de dados são específicos para um tipo de esquema, ou seja, esquema XML escrito em uma determinada linguagem ou esquemas associados a aplicações de base de dados

relacional. Alguns trabalhos não mencionam como devem ser gerados os casos de teste e outros não providenciam a geração automática do conjunto de teste.

A abordagem de teste proposta no próximo capítulo difere dos trabalhos que visam o teste de esquemas de dados apresentados nesse capítulo porque: é baseada em defeitos; pode ser aplicada em esquemas de diferentes contextos; e permite a geração automática do conjunto de teste.

Capítulo 4

Análise de Instâncias de Dados Alternativas

Neste capítulo é introduzida a abordagem de teste de esquema de dados denominada Análise de Instâncias de Dados Alternativas. Essa abordagem é baseada em defeitos, envolvendo a geração de instâncias de dados e de consultas. Inicialmente, a abordagem de teste é definida e no decorrer do capítulo cada parte componente da abordagem de teste é descrita.

4.1 Definição

A abordagem de teste proposta neste trabalho, denominada Análise de Instâncias de Dados Alternativas (AIDA - *Alternative Data Instance Analysis*), tem por objetivo detectar defeitos em esquemas de dados. O esquema de dados de uma aplicação de software baseada na Web ou de base de dados contém a definição dos dados manipulados por essas aplicações. Se o esquema estiver incorreto, ou seja, se alguma definição referente aos dados contiver um defeito em relação à especificação dos dados, dados inválidos podem ser aceitos para o processamento da aplicação de software, podendo causar falhas nessa aplicação, ou ainda, dados válidos podem não ser aceitos, também gerando resultados inesperados no processamento da aplicação de software. Dessa forma, a idéia central desta abordagem de teste é testar o esquema de dados para garantir a integridade dos dados definidos por ele e manipulados por uma aplicação de software.

A AIDA é genérica, ou seja, pode ser aplicada em esquemas de dados de diferentes contextos. Para isso, um metamodelo baseado em MOF que permite que sejam instanciados e interpretados os componentes do esquema é definido na abordagem. A AIDA é uma abordagem de teste de esquema baseada em defeitos. Portanto, defeitos comuns que podem ser introduzidos no esquema durante o seu desenvolvimento são identificados e organizados em classes de defeitos.

A AIDA é capaz de revelar os defeitos cobertos pelas classes de defeitos e que estejam relacionados à definição incorreta ou ausente de restrições aos dados no esquema.

No processo de teste da AIDA, uma instância de dados associada ao esquema em teste sofre alterações simples gerando instâncias de dados alternativas. Essas instâncias de dados são geradas com base em padrões especificados nas classes de defeitos previamente identificadas. Consultas às instâncias de dados alternativas também são geradas a partir de padrões definidos nas classes de defeitos. As instâncias de dados representam os possíveis defeitos no esquema e as consultas são capazes de revelar esses defeitos. Portanto, a abordagem é denominada Análise de Instâncias de Dados Alternativas devido ao uso de instâncias de dados alternativas para detectar defeitos em esquemas de dados.

A Figura 4.1 ilustra o processo de teste definido na AIDA. No processo de teste, o testador providencia o esquema de dados que será testado e a instância de dados correspondente. Uma representação formal para o esquema é construída. A partir dela associações de defeitos são identificadas automaticamente. Uma associação de defeito é um elemento ou atributo e restrições aos dados do esquema, previstas no metamodelo, associados a uma classe de defeito. As associações de defeitos também podem ser identificadas pelo testador manualmente (elas se referem à definição ausente de restrições aos dados). Com base nas associações de defeitos são geradas as instâncias de dados alternativas. Essas instâncias são classificadas em válidas e inválidas de acordo com o esquema sob teste. As consultas também são geradas de acordo com as associações de defeitos. Os dados de teste são formados por alternativas válidas e consultas correspondentes. Esses dados de teste são executados e os resultados de teste, inclusive as alternativas inválidas, são analisados pelo testador.

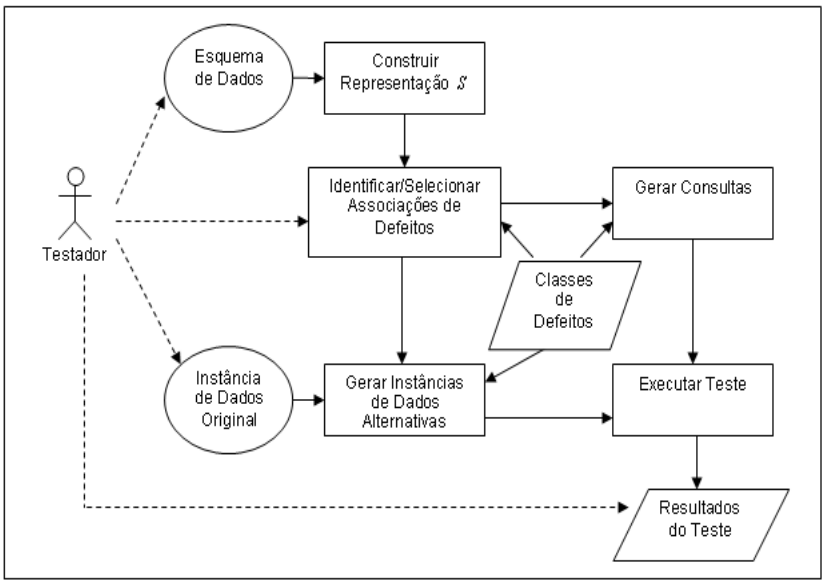

Figura 4.1: Processo de teste definido na AIDA

4.2 Modelo de dados

Um modelo de dados é uma descrição formal de informações que serão armazenadas e manipuladas por aplicações de software. Em um modelo de dados são definidos dados relacionados entre si, descritos por meio de elementos, atributos, restrições e relacionamentos. Os componentes do modelo de dados podem ser especificados como segue:

- Elementos: representam entidades ou objetos do mundo real;
- Atributos: caracterizam os elementos;
- Restrições: definem condições limitantes para elementos e atributos;
- Relacionamentos: estabelecem uma associação entre elementos.

Neste trabalho, o modelo de dados é definido por um metamodelo MM. Esse metamodelo é apresentado em notação UML e segue a especificação MOF, composta de quatro níveis. O metamodelo MM apresentado na Figura 4.3 está na camada M2, conforme ilustrado na Figura 4.2.

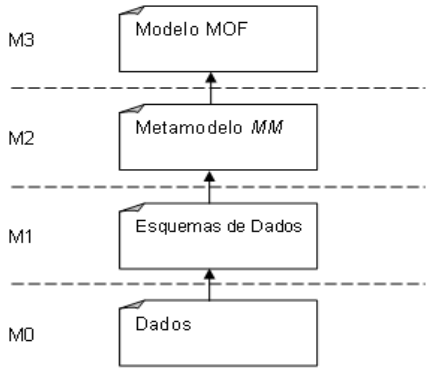

Figura 4.2: Metamodelo MM no modelo MOF

A Figura 4.3 apresenta o metamodelo MM, que consiste das classes: Elemento (entidades ou objetos), Atributo (características dos elementos), Restrição (restrições associadas aos elementos e atributos) e, da classe associativa: Associação (propriedades da associação reflexiva da classe Elemento). Um elemento possui atributos e está associado a outros elementos; um elemento ou um atributo possui zero ou mais restrições; uma associação entre elementos é definida pelo seu tipo.

As restrições identificadas no metamodelo MM para esquemas de dados foram obtidas com base em inspeções realizadas em esquemas de dados e a partir da investigação de trabalhos da literatura, tais como os trabalhos de Lee e Offut [42] e Offutt e Xu [55]. Essas restrições são definidas usando

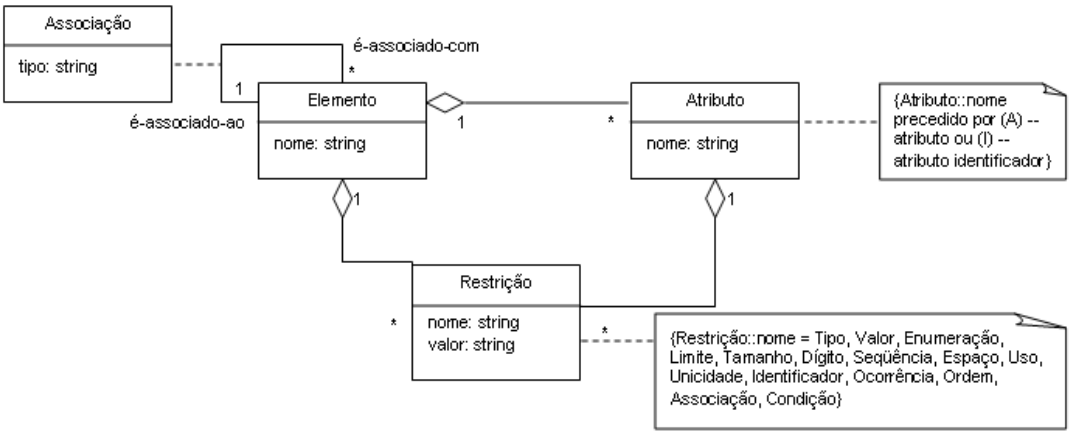

Figura 4.3: Metamodelo MM

OCL (*Object Constraint Language*) e são apresentadas a seguir. Cada restrição é apresentada por meio de: denominação, dada pela notação $R_{TipodeRestricao}$; definição intuitiva; e definição formal usando OCL. Os conceitos de OCL empregados para definir as restrições no metamodelo MM são apresentados no Apêndice A.

Restrição: Tipo (R_{type}) - A restrição Tipo refere-se aos tipos de dados (*string*, inteiro, real, booleano, data, hora) que podem ser atribuídos ao conteúdo de elementos e atributos.
Definição Formal:
$context\ r.Restricao$
$inv\ :\ r.nome = 'tipo'\ implies\ (r.valor = 'string'\ or\ r.valor = 'inteiro'\ or\ r.valor = 'real'\ or\ r.valor = 'booleano'\ or\ r.valor = 'data'\ or\ r.valor = 'hora')$

Restrição: Valor (R_{value}) - A restrição Valor refere-se a um valor padrão ou fixo definido pelo usuário para o conteúdo de um elemento ou atributo.
Definição Formal:
$context\ r.Restricao$
$inv:\ r.nome = 'valor'\ implies\ (r.valor = 'fixo'\ or\ r.valor = 'padrao')$

Restrição: Enumeração ($R_{enumeration}$) - A restrição Enumeração refere-se a uma lista de valores enumerados que podem ser atribuídos ao conteúdo de elementos ou atributos.
Definição Formal:
$context\ r.Restricao$
$inv:\ r.nome = 'enumeracao'\ implies\ r.valor = 'enumeracao'$

Restrição: Limite (R_{bound}) - A restrição Limite refere-se aos limites superior e inferior atribuídos aos valores numéricos de elementos ou atributos.
Definição Formal:
$context\ r.Restricao$
$inv:\ (r.nome = \ 'limite\ inferior'\ or\ r.nome = \ 'limite\ superior')\ implies\ r.valor = \ 'inteiro'$

Restrição: Tamanho (R_{length}) - A restrição Tamanho refere-se à quantidade de caracteres permitida para o conteúdo do tipo *string* de um elemento ou atributo.
Definição Formal:
$context\ r.Restricao$
$inv:\ (r.nome = \ 'tamanho'\ or\ r.nome = \ 'tamanho\ maximo'\ or\ r.nome = \ 'tamanho\ minimo')$
$implies\ r.valor\ = \ 'inteiro'$

Restrição: Dígito (R_{digit}) - A restrição Dígito refere-se à quantidade de dígitos ou dígitos decimais permitida para um valor numérico de um elemento ou atributo.
Definição Formal:
$context\ r.Restricao$
$inv:\ (r.nome = \ 'digito'\ or\ r.nome = \ 'digito\ decimal')\ implies\ r.valor = \ 'inteiro'$

Restrição: Seqüência ($R_{pattern}$) - A restrição Seqüência refere-se à seqüência de caracteres ou números permitidos para o conteúdo de um elemento ou atributo.
Definição Formal:
$context\ r.Restricao$
$inv:\ r.nome = \ 'sequencia'\ implies\ r.valor = \ 'string'$

Restrição: Espaço (R_{space}) - A restrição Espaço refere-se ao tratamento dado aos caracteres de espaço no conteúdo de um elemento ou atributo.
Definição Formal:
$context\ r.Restricao$
$inv:\ r.nome = \ 'espaco'\ implies\ (r.valor = \ 'preserva'\ or\ r.valor = \ 'remova'\ or\ r.valor = \ 'troca')$

Restrição: Uso (R_{use}) - A restrição Uso refere-se à definição de um atributo como opcional ou obrigatório.
Definição Formal:
$context\ r.Restricao$
$inv:\ r.nome = \ 'uso'\ implies\ (r.valor = \ 'opcional'\ or\ r.valor = \ 'obrigatorio')$

Restrição: Unicidade (R_{unique}) - A restrição Unicidade refere-se à definição do conteúdo de um atributo como único ou não.
Definição Formal:
$context\ r.Restricao$
$inv:\ r.nome = \ 'unicidade'\ implies\ (r.valor = \ 'unico'\ or\ r.valor = \ 'nao\ unico')$

Restrição: Identificador ($R_{identifier}$) - A restrição Identificador refere-se à definição de um atributo como identificador.
Definição Formal:
$context\ r.Restricao$
$inv:\ r.nome = \ 'identificador'\ implies\ (r.valor = \ 'chave\ primaria'\ or$
$r.valor = \ 'chave\ estrangeira'\ or\ r.valor = \ '')$

Restrição: Ocorrência (R_{occur}) - A restrição Ocorrência refere-se ao número de vezes, máximo ou mínimo, que um determinado elemento pode ocorrer.
Definição Formal:
$context\ r.Restricao$
$inv:\ (r.nome = \ 'ocorrencia\ maxima'\ or\ r.nome = \ 'ocorrencia\ minima')\ implies\ r.valor = \ 'inteiro'$

Restrição: Ordem (R_{order}) - A restrição Ordem refere-se a ordem que elementos-filho de um determinado elemento devem seguir.
Definição Formal:
$context\ r.Restricao$
$inv:\ r.nome = \ 'ordem'\ implies\ (r.valor = \ 'sequencia'\ or\ r.valor = \ 'qualquer'\ or\ r.valor = \ 'escolha')$

Restrição: Associação ($R_{association}$) - A restrição Associação refere-se ao tipo de associação que pode relacionar dois ou mais elementos (cardinalidade, generalização/especialização, agregação e elemento associativo).
Definição Formal:
$context\ r.Restricao$
$inv:\ r.nome = \ 'associacao'\ implies\ ((Elemento.associacao[é-associado-ao].tipo = \ 'cardinalidade'\ and\ r.valor = \ 'cardinalidade')\ or\ (Elemento.associacao[é-associado-ao].tipo = \ 'generalizacao/especializacao'\ and\ r.valor = \ 'generalizacao/especializacao')\ or\ (Elemento.associacao[é-associado-ao].tipo = \ 'agregacao'\ and\ r.valor = \ 'agregacao')\ or\ (Elemento.associacao[é-associado-ao].tipo = \ 'elemento\ associativo'\ and\ r.valor = \ 'elemento\ associativo'))$

Restrição: Condição ($R_{condition}$) - A restrição Condição refere-se a uma condição semântica, ou seja, um predicado que deve ser satisfeito pelo conteúdo de determinado atributo ou elemento.
Definição Formal:
$context\ r.Restricao$
$inv:\ r.nome = \ 'condicao'\ implies\ r.valor = \ 'oclExpression'$

Com base no metamodelo MM da Figura 4.3, modelos de dados representando domínios específicos são definidos, de modo que o metamodelo MM permite que sejam instanciados e interpretados os componentes do modelo de dados. A Figura 4.4 ilustra o metamodelo MM e o modelo de dados M descrito por esse metamodelo. O modelo de dados M representa dados de clientes e funcionários da base de dados de uma determinada empresa. Para exemplificar que os componentes do modelo M são instâncias do metamodelo MM, observe na Figura 4.4 que: as classes "Cliente" e "Funcionário" são especializações da classe "Pessoa", a associação generalização/especialização é uma instância da associação entre elementos prevista no MM; a classe "Pessoa" é uma instância da classe Elemento; o tipo de dado *string* do atributo "nome" da classe "Pessoa" é uma instância da classe Restrição e o atributo "ncarteira" da classe "Funcionário" é uma instância da classe Atributo no MM.

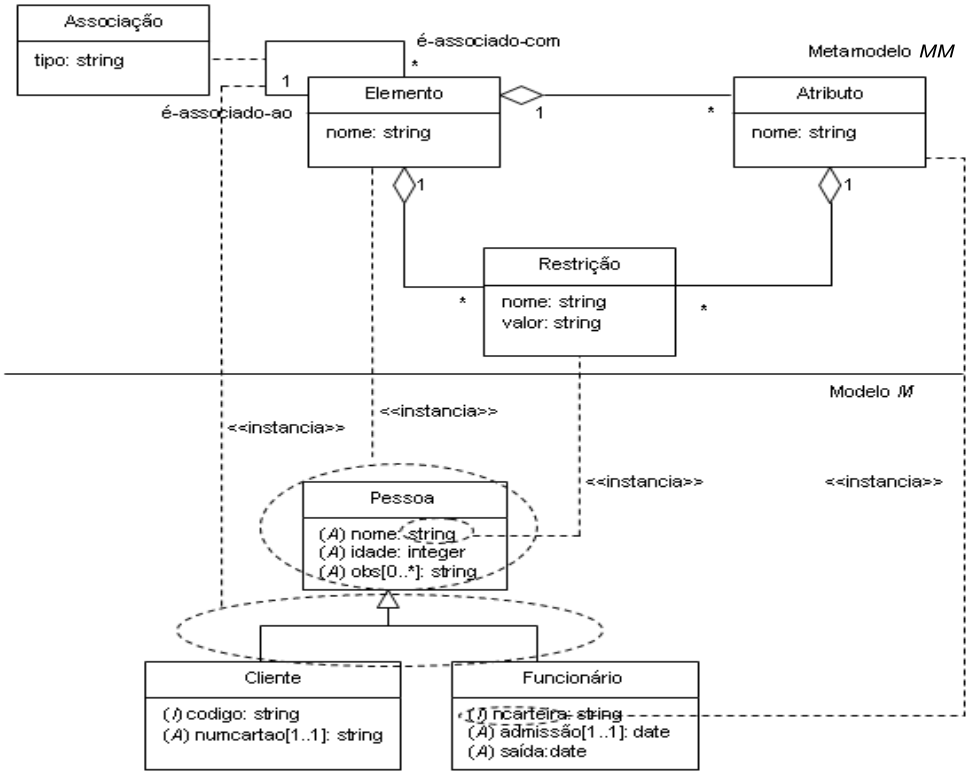

Figura 4.4: Modelo de dados M descrito pelo Metamodelo MM

4.3 Representação formal

A representação formal do modelo de dados de um esquema de dados provê a identificação dos elementos, atributos, restrições e relacionamentos entre eles. Na abordagem de teste, essa representação é necessária para que sejam geradas as instâncias de dados e as consultas a essas instâncias automaticamente.

Neste trabalho um esquema de dados S representado por um modelo de dados M é denotado 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 ao domínio, definição, relacionamento e conteúdo de elementos ou de atributos.
- P é um conjunto de regras de associação que relacionam elementos, atributos e restrições. Uma regra de P pode ter um dos seguintes formatos. Considere $U = E \cup A$.
  - $p(x,\ y) |\ x,\ y \in U,\ x \neq y$;
  - $p(x,\ r) |\ x \in U \ \wedge\ r \in R$;
  - $p(x,\ r,\ SU) |\ x \in U \wedge\ r \in R \ \wedge\ SU = \{u_1,\ u_2,\ ...,\ u_m\} \subset U$,
    $\forall\, u_i \neq x,\ 1 \leq i \leq m,\ m \geq 1$, onde m é o número de elementos e atributos de SU.

A seguir é apresentado um exemplo de uso da representação formal empregada nesta abordagem de teste. Seja o modelo de dados M visto na Seção 4.2 (Figura 4.4), $S = (E,\ A,\ R,\ P)$ é dada por:

$E = \{Pessoa,\ Cliente,\ Funcionario\}$
$A = \{nome,\ idade,\ obs,\ codigo,\ numcartao,\ ncarteira,\ admissao,\ saida\}$
$R = \{tipo,\ associacao,\ ocorrência,\ identificador,\ uso\}$
$P = \{p_1(Pessoa,\ nome),\ p_2(Pessoa,\ idade),\ p_3(Pessoa,\ obs),$
$p_4(Pessoa,\ associacao,\ Cliente,\ Funcionario),\ p_5(nome,\ tipo),$
$p_6(idade,\ tipo),\ p_7(obs,\ tipo),\ p_8(obs,\ ocorrência),$
$p_9(Cliente, codigo),\ p_{10}(Cliente,\ numcartao),\ p_{11}(Cliente,\ associacao,\ Pessoa),$
$p_{12}(codigo,\ tipo),\ p_{13}(codigo,\ identificador),\ p_{14}(numcartao,\ tipo),$
$p_{15}(numcartao,\ uso),\ p_{16}(Funcionario,\ ncarteira),\ p_{17}(Funcionario,\ admissao),$
$p_{18}(Funcionario,\ saida),\ p_{19}(Funcionario,\ associacao,\ Pessoa),$
$p_{20}(ncarteira,\ tipo),\ p_{21}(ncarteira,\ identificador),\ p_{22}(admissao,\ tipo),$
$p_{23}(admissao,\ uso),\ p_{24}(saida, tipo)\}$

4.4 Classes de defeitos

A abordagem de teste proposta é baseada em defeitos. Portanto, possíveis defeitos em esquemas de dados agrupados em classes de defeitos são empregados para nortear a execução do teste. As classes de defeitos foram definidas de acordo com as restrições para esquemas de dados identificadas no metamodelo MM. Essas classes estão associadas a defeitos típicos, relacionados à definição de restrições aos dados, que podem ser introduzidos durante o projeto de um esquema ou devido a sua atualização, em conseqüência de alterações sofridas nos dados definidos pelo esquema.

É importante observar que esses defeitos são identificados no metamodelo MM e são instanciados para um modelo de dados M, que representa um esquema de dados S específico para uma aplicação de software. Portanto, as classes de defeitos são instanciadas para um esquema de dados S, de acordo com as restrições aos dados previstas para o esquema de dados S e para a aplicação correspondente.

Os defeitos estão classificados em quatro grupos de restrições: domínio; definição; relacionamento; e semântica. As classes de defeitos são definidas como segue, a partir das restrições definidas na Seção 4.2.

Suponha um esquema $S = (E,\ A,\ R,\ P)$, onde:
$E = \{e_1,\ e_2,\ ...,\ e_n\}$;
$A = \{a_1,\ a_2,\ ...,\ a_w\}$;
$R = \{r_1,\ r_2,\ ...,\ r_k\}$;
$P = \{p_1(e_1,\ e_2),\ p_2(e_1,\ a_1),\ p_3(e_1,\ r_2,\ u_1,\ ...,\ u_i,\ ...,\ u_m),$
$p_4(e_2,\ r_1),\ p_5(a_1,\ r_1),\ ...\}$;
$n\ \geq\ 1$, onde n é o número de elementos de E;
$w\ \geq\ 0$, onde w é o número de atributos de A;
$k\ \geq\ 1$, onde k é o número de restrições de R;
$1\ \leq\ i\ \leq\ m,\ m\ \geq\ 1$, onde m é o número de elementos e atributos de SU, $SU\ =\ \{u_1,\ u_2,\ ...,\ u_i,\ ...,\ u_m\}\ |\ U\ =\ E\ \cup\ A\ \wedge\ SU\ \subset\ U$.

Considere: $R\ \subseteq\ R_{dom}\ \cup\ R_{def}\ \cup\ R_{rel}\ \cup\ R_{sem}$, tal que:
$R_{dom}\ = R_{type}\ \cup\ R_{value}\ \cup\ R_{enumeration}\ \cup\ R_{bound}\ \cup\ R_{length}\ \cup\ R_{digit}\ \cup\ R_{pattern}\ \cup\ R_{space}$;
$R_{def}\ = R_{use}\ \cup\ R_{unique}\ \cup\ R_{identifier}$;
$R_{rel}\ = R_{occur}\ \cup\ R_{order}\ \cup\ R_{association}$;
$R_{sem}\ = R_{condition}$.

- *Grupo 1 (G1) - Restrições de Domínio*: defeitos relacionados à definição de domínio dos conteúdos de elementos ou atributos.

Definição: $\exists\ p(u_i,\ r_1) \mid u_i\ \in\ U,\ r_1\ \in\ R_{dom}$. Considere $p(u_i,\ r_1^{'})$ correta segundo a especificação dos dados de S, $(r_1^{'})\ \in\ R_{dom}$. Se $p(u_i,\ r_1)\ \neq\ p(u_i,\ r_1^{'})\ \rightarrow\ p(u_i,\ r_1)$ está incorreta.

Suponha que para as classes de defeito de domínio identificadas a seguir, $\exists$ uma definição incorreta da restrição de domínio $r_1\ \in\ R_{dom}$ para um elemento ou atributo u_i, se $r_1^{'}\ \in\ R_{dom}$ está correta de acordo com a especificação dos dados e $r_1\ \neq\ r_1^{'}$ para u_i.

- *Tipo de Dado Incorreto (TDI)*: Considere $r_1,\ r_1^{'}\ \in\ R_{type}$.

  Exemplo: Suponha a_1 referente ao preço de um produto e r_1 o tipo de dado permitido para a_1, r_1 definido como "inteiro". Considere que $r_1^{'}$ é a restrição de tipo de dado correta para a_1, $r_1^{'}$ foi especificado como "real". r_1 é diferente de $r_1^{'}$, então r_1 é uma definição de restrição de tipo de dado incorreta para a_1.

- *Valor Incorreto (VI)*: Considere $r_1,\ r_1^{'}\ \in\ R_{value}$.

  Exemplo: Suponha a_1 referente ao preço de um produto e r_1 a restrição de valor especificada para a_1, definida como "fixo". Considere que $r_1^{'}$ é a restrição de valor correta para a_1, $r_1^{'}$ é "padrão". r_1 é diferente de $r_1^{'}$, então r_1 é uma definição de restrição de valor incorreta para a_1. A definição incorreta da restrição de valor r_1 também pode estar relacionada ao valor incorreto especificado para um atributo a_1 para o qual r_1 é definido como "fixo" ou "padrão".

- *Valores Enumerados Incorretos (VEI)*: Considere $r_1,\ r_1^{'}\ \in\ R_{enumeration}$.

  Exemplo: Suponha a_1 referente às cores de um semáforo e r_1 os valores possíveis para a_1, r_1 definida como "vermelho, laranja, verde". Considere que $r_1^{'}$ é a restrição de valores enumerados correta para a_1, $r_1^{'}$ é "vermelho, amarelo, verde". r_1 é diferente de $r_1^{'}$, então r_1 é uma definição de restrição de valores enumerados incorreta para a_1.

- *Valores Máximos e Mínimos Incorretos (VMMI)*: Considere $r_1,\ r_1^{'}\ \in\ R_{bound}$.

  Exemplo: Suponha a_1 referente à duração de um determinado curso superior, r_1 o valor mínimo possível para a conclusão de a_1, r_1 definida como "0". Considere que $r_1^{'}$ é a restrição de valor mínimo correta para a_1, $r_1^{'}$ é "5". r_1 é diferente de $r_1^{'}$, então r_1 é uma definição de restrição de valor mínimo incorreta para a_1.

- *Tamanho Incorreto (TI)*: Considere $r_1,\ r_1^{'}\ \in\ R_{length}$.

  Exemplo: Suponha a_1 referente à descrição de determinado curso superior, r_1 a quantidade máxima de caracteres para o conteúdo de a_1, r_1 definida como "10". Considere que $r_1^{'}$ é a restrição de quantidade máxima correta para a_1, $r_1^{'}$ é "60". r_1 é diferente de $r_1^{'}$, então r_1 é uma definição de restrição de quantidade máxima incorreta para a_1.

- *Dígito Incorreto (DI)*: Considere $r_1,\ r_1^{'}\ \in\ R_{digit}$.

  Exemplo: Suponha a_1 referente ao preço de um produto e r_1 a quantidade de dígitos decimais permitidos para a_1, r_1 definida como “3”. Considere que $r_1^{'}$ é a restrição de limite de dígitos correta para a_1, $r_1^{'}$ é “2”. r_1 é diferente de $r_1^{'}$, então r_1 é uma definição de restrição de limite de dígitos incorreta para a_1.

- *Seqüência de Valores Incorreta (SVI)*: Considere $r_1,\ r_1^{'}\ \in\ R_{pattern}$.

  Exemplo: Suponha a_1 referente à senha de um usuário de um determinado sistema e r_1 a seqüência de valores permitidos para a_1, r_1 definida como "[a-z0-9]". Considere que $r_1^{'}$ é a restrição de seqüência de valores correta para a_1, $r_1^{'}$ é “[a-zA-Z0-9]”. r_1 é diferente de $r_1^{'}$, então r_1 é uma definição de seqüência de valores incorreta para a_1.

- *Caracteres de Espaço em Branco Incorretos (CEBI)*: Considere $r_1,\ r_1^{'}\ \in\ R_{space}$.

  Exemplo: Suponha a_1 referente à descrição de determinado curso superior, r_1 a especificação de caracteres de espaço em branco para o conteúdo de a_1, r_1 definida como “troca”. Considere que $r_1^{'}$ é a restrição de tratamento de caracteres de espaço em branco correta para a_1, $r_1^{'}$ é “preserva”. r_1 é diferente de $r_1^{'}$, então r_1 é uma definição de tratamento de caracteres de espaço em branco incorreta para a_1.

- *Grupo 2 (G2) - Restrições de Definição*: defeitos relacionados à definição de elementos ou atributos, relativos a integridade dos dados.

  Definição: $\exists\ p(u_i,\ r_1)\ |\ u_i\ \in\ U,\ r_1\ \in\ R_{def}$. Considere $p(u_i,\ r_1^{'})$ correta segundo a especificação dos dados de S, $(r_1^{'})\ \in\ R_{def}$. Se $p(u_i,\ r_1)\ \neq\ p(u_i,\ r_1^{'})\ \rightarrow\ p(u_i,\ r_1)$ está incorreta.

  Suponha uma das classes de defeito de restrição de definição identificadas a seguir, $\exists$ uma definição incorreta da restrição de definição $r_1\ \in\ R_{def}$ para um elemento ou atributo u_i se $r_1^{'}\ \in\ R_{def}$ está correta de acordo com a especificação dos dados e $r_1\ \neq\ r_1^{'}$ para u_i.

  - *Uso Incorreto (UI)*: Considere $r_1,\ r_1^{'}\ \in\ R_{use}$.

    Exemplo: Suponha a_1 referente à descrição de determinado curso superior, r_1 a especificação de uso para a_1, r_1 definida como “opcional”. Considere que $r_1^{'}$ é a restrição de uso correta para a_1, $r_1^{'}$ é “obrigatorio”. r_1 é diferente de $r_1^{'}$, então r_1 é uma definição de uso incorreta para a_1.

  - *Unicidade Incorreta (UNI)*: Considere $r_1,\ r_1^{'}\ \in\ R_{unique}$.

    Exemplo: Suponha a_1 referente à senha de um usuário de um determinado sistema, r_1 a especificação de unicidade para o conteúdo de a_1, r_1 definida como “unico”. Considere

que $r_1^{'}$ é a restrição de unicidade correta para a_1, $r_1^{'}$ é "nao unico". r_1 é diferente de $r_1^{'}$, então r_1 é uma definição de unicidade incorreta para a_1.

– *Identificador Incorreto (II)*: Considere $r_1,\ r_1^{'} \in R_{identifier}$, e ainda, que um atributo definido como identificador está incorreto se $\exists\, p(a_1,\ r_2) \wedge p(a_1,\ r_3) \mid r_2 \neq obrigatorio \vee r_3 \neq unico,\ r_2 \in R_{use}, r_3 \in R_{unique}$. Isto é, para que um atributo seja identificador, o atributo também deve ser obrigatório e único.

Exemplo: Suponha a_1 referente ao nível de acesso de um usuário em um determinado sistema é obrigatório e único, r_1 a especificação de a_1 como identificador, r_1 é "chave primaria". No entanto, a_1 não foi definido com chave na especificação dos dados. Portanto, $r_1^{'}$ é a restrição de identificador correta para a_1, $r_1^{'}$ é " ". r_1 é diferente de $r_1^{'}$, então r_1 é uma definição de identificador incorreta para a_1.

- *Grupo 3 (G3) - Restrições de Relacionamento*: defeitos relacionados à definição de tipos de relacionamentos entre elementos ou atributos.

  – *Ocorrência Incorreta (OI)*:

  Definição: $\exists\, p(u_i,\ r_1) \mid u_i \in U,\ r_1 \in R_{occur}$. Considere $p(u_i,\ r_1^{'})$ correta segundo a especificação dos dados de S, $(r_1^{'}) \in R_{occur}$. Se $p(u_i,\ r_1) \neq p(u_i,\ r_1^{'}) \rightarrow p(u_i,\ r_1)$ está incorreta.

  Suponha que $\exists$ uma definição incorreta da restrição de relacionamento $r_1 \in R_{occur}$ para um elemento ou atributo u_i, se $r_1^{'} \in R_{occur}$ está correta segundo a especificação dos dados e $r_1 \neq r_1^{'}$, ou seja, *OI* é uma definição incorreta da quantidade de vezes mínima ou máxima que um mesmo elemento ou atributo pode ocorrer.

  Exemplo: Considere e_2 referente ao nome de um usuário e_1 em um determinado sistema, r_1 a especificação de ocorrência máxima para e_2, r_1 definida como "n", ou seja, um usuário pode ter mais de um nome. Considere que $r_1^{'}$ é a restrição de ocorrência máxima correta para e_2, $r_1^{'}$ é "1" (um usuário pode ter somente um nome). r_1 é diferente de $r_1^{'}$, então r_1 é uma definição de ocorrência máxima incorreta para e_2.

  – *Ordem Incorreta (ORI)*:

  Definição: $\exists\, p(e_1,\ r_2,\ u_1,\ ...,\ u_i,\ ...,\ u_m) \mid e_1 \in E,\ r_2 \in R_{order},\ u_1,\ ...,\ u_i,\ ...,\ u_m \in U$. Considere $p(e_1,\ r_2^{'},\ u_1,\ ...,\ u_i,\ ...,\ u_m)$ correta segundo a especificação dos dados de S, $(r_2^{'}) \in R_{order}$. Se $p(e_1,\ r_2,\ u_1,\ ...,\ u_i,\ ...,\ u_m) \neq p(e_1,\ r_2^{'},\ u_1,\ ...,\ u_i,\ ...,\ u_m) \rightarrow p(e_1,\ r_2,\ u_1,\ ...,\ u_i,\ ...,\ u_m)$ está incorreta.

  Suponha que $\exists$ uma definição incorreta da restrição de relacionamento $r_2 \in R_{order}$ para um elemento e_1 em relação aos elementos ou atributos $u_1,\ ...,\ u_i,\ ...,\ u_m$, se $r_2^{'} \in R_{order}$

está correta segundo a especificação dos dados e $r_2 \neq r_2^{'}$ para e_1 no relacionamento com $u_1, ..., u_i, ..., u_m$, ou seja, *ORI* é uma definição incorreta da ordem em que os elementos ou atributos de um elemento podem aparecer.

Exemplo: Considere e_1 referente a um usuário, e_2 referente ao nome de um usuário, e_3 referente ao endereço do usuário e r_2 a especificação de ordem para e_2 e e_3, elementos de e_1, r_2 definida como "sequencia", ou seja, e_2 e e_3 devem aparecer na ordem em que foram definidos para e_1. Considere que $r_2^{'}$ é a restrição de ordem correta para e_1, $r_2^{'}$ é "qualquer", ou seja, qualquer ordem. r_2 é diferente de $r_2^{'}$, então r_2 é uma definição de ordem incorreta para e_1.

– *Associação Incorreta (AI)*:

Definição: $\exists\, p(e_1,\ r_2,\ e_2,\ ...,\ e_n) \mid e_1,\ e_2,\ ...,\ e_n \in E,\ r_2 \in R_{association}$. Considere $p(e_1,\ r_2^{'},\ e_2,\ ...,\ e_n)$ correta segundo a especificação dos dados de S, $r_2^{'} \in R_{association}$. Se $p(e_1,\ r_2,\ e_2,\ ...,\ e_n) \neq p(e_1,\ r_2^{'},\ e_2,\ ...,\ e_n) \rightarrow p(e_1,\ r_2,\ e_2,\ ...,\ e_n)$ está incorreta.

Suponha que $\exists$ uma definição incorreta da restrição de relacionamento $r_2 \in R_{association}$ de um elemento e_1 em relação a um ou mais elementos $e_2, ..., e_n$, se $r_2^{'} \in R_{association}$ está correta segundo a especificação dos dados e $r_2 \neq r_2^{'}$ para o relacionamento entre e_1 e $e_2, ..., e_n$, ou seja, *AI* é uma definição incorreta de cardinalidade (número de ocorrências de um elemento em relação a outro elemento em um relacionamento), generalização/especialização (um ou mais elementos herdam propriedades de um outro elemento), agregação (um elemento é parte ou está contido em outro elemento) elemento associativo (relacionamento que possui atributos e sua existência depende da existência do relacionamento).

Exemplo: Considere e_1 referente ao elemento venda e e_2 referente a produto, r_2 a especificação de cardinalidade de e_1 em relação a e_2, r_2 definida como "1". Considere que $r_2^{'}$ é a restrição de cardinalidade correta para e_1 em relação a e_2, $r_2^{'}$ é "n". r_2 é diferente de $r_2^{'}$, então r_2 é uma definição incorreta do relacionamento de cardinalidade para e_1.

- *Grupo 4 (G4) - Restrições semânticas*: defeitos relacionados à definição de restrições em relação ao conteúdo dos dados, escritas por regras de negócio.

  – *Condição Incorreta (COI)*:

  Definição: $\exists\, p(u_i,\ r_1) \mid u_i \in U,\ r_1 \in R_{condition}$. Considere $p(u_i,\ r_1^{'})$ correta segundo a especificação dos dados de S, $r_1^{'} \in R_{condition}$. Se $p(u_i,\ r_1) \neq p(u_i,\ r_1^{'}) \rightarrow p(u_i,\ r_1)$ está incorreta.

  Suponha que $\exists$ uma definição incorreta de uma restrição semântica de condição $r_1 \in R_{condition}$ para um elemento ou atributo u_i, se $r_1^{'} \in R_{condition}$ é correta de acordo com a

especificação dos dados e $r_1 \neq r_1^{'}$ para u_i, ou seja, *COI* é a definição incorreta de um predicado que expressa uma condição, que deve ser satisfeita pelo conteúdo de determinado elemento ou atributo.

Exemplo: Suponha a_1 referente à data de admissão de um empregado, a_2 referente à data de demissão desse mesmo empregado, r_1 a especificação de uma condição que deve ser satisfeita por a_1 em relação a a_2, r_1 definida como "$a_1 > a_2$". Considere que $r_1^{'}$ é a restrição de condição correta para a_1, $r_1^{'}$ é "$a_1 < a_2$". r_1 é diferente de $r_1^{'}$, então r_1 é uma definição de condição incorreta para a_1 em relação a a_2.

4.5 Associações de defeitos

Uma associação de defeito é formada por elementos ou atributos de S associados a uma classe de defeito definida na seção anterior. As associações de defeitos de um esquema de dados S formam um conjunto Σ, dado por:

$\Sigma = \{(x,\ Classe\ de\ defeito),\ (y : t,\ Classe\ de\ defeito),\ (v : w\ ...\ z,\ Classe\ de\ defeito),\ ...\}$;

onde:

$x,\ y,\ t,\ v,\ w,\ ...,\ z\ \in\ U | U = E\ \cup\ A$.

Um exemplo de associação de defeito, de acordo com o exemplo de representação formal S visto na Seção 4.3 para o modelo M (Figura 4.4), é dada por:

- $(numcartao, G2 - UI)$

Nesse exemplo, o atributo "numcartao" é associado à classe de defeito Restrição de Definição - Uso Incorreto porque na representação formal esse atributo está relacionado à restrição uso definida no metamodelo MM (Seção 4.2).

4.6 Instâncias de dados alternativas

As instâncias de dados alternativas $(I_1,\ I_2,\ ...,\ I_i,\ ...,\ I_n)$, onde n é o número de instâncias alternativas, são geradas por meio de uma alteração simples na instância de dados original I_0, válida para o esquema em teste S. Essas alterações são feitas pela inclusão, modificação e exclusão de elementos ou atributos de I_0, de acordo com os padrões definidos para cada classe de defeito (Tabela 4.1).

Cada alteração gera uma instância de dados diferente. As alterações são guiadas pelo conjunto Σ de associações de defeitos identificadas em S. Desse modo, $I_1,\ I_2,\ ...,\ I_i,\ ...,\ I_n$ representam os possíveis defeitos que podem ser revelados em S. $I_1,\ I_2,\ ...,\ I_i,\ ...,\ I_n$ são validadas com relação a S e separadas em válidas e inválidas.

Por exemplo, as instâncias alternativas geradas para uma instância de dados original segundo a associação de defeito $(numcartao, G2 - UI)$, seriam:

- Instância de dados original (I_0): numcartao = 3245671209
- Alternativa 1 (I_1): numcartao = 3245671209 (mantém o conteúdo original)
- Alternativa 2 (I_2): numcartao = *null* (altera o conteúdo para valor nulo)
- Alternativa 3 (I_3): numcartao = 8215611180 (insere o atributo)
- Alternativa 4 (I_4): (exclui o atributo)

Tabela 4.1: Tipos de alterações na instância de dados original para gerar as instâncias de dados alternativas

Classe de Defeito	**Descrição da alteração**
Grupo 1 - Restrições de Domínio	
TDI - Tipo de Dado Incorreto	- mantém o conteúdo - conteúdo somente com caracteres - conteúdo somente com dígitos - conteúdo com caracteres e dígitos - troca o conteúdo por data (*date*) - troca o conteúdo por hora (*time*)
VI - Valor Incorreto	- duplica o último caractere ou dígito do conteúdo - exclui conteúdo - mantém o conteúdo
VEI - Valores Enumerados Incorretos	- mantém o conteúdo - duplica último caractere/dígito
VMMI - Valores Máximos e Mínimos Incorretos	- mantém o valor (mínimo e máximo) - duplica um dígito, acrescenta um número x de zeros ou multiplica por um número x (máximo) - substitui todos os dígitos por 9 (máximo) - substitui por um número positivo (máximo permitido)
continua na próxima página	

Tabela 4.1 – continuação da página anterior

Classe de Defeito	**Descrição da alteração**
	- exclui um dígito, exclui um número x de dígitos ou divide por um número x (mínimo) - substitui por zero (mínimo) - substitui por um número negativo (mínimo permitido)
TI - Tamanho Incorreto	- mantém o conteúdo - exclui um número x de caracteres - duplica um número x de caracteres
DI - Dígito Incorreto	- mantém o conteúdo - exclui um número x de dígitos - duplica um número x de dígitos
SVI - Seqüência de Valores Incorreta	- mantém o conteúdo - exclui um número x de caracteres - duplica um número x de caracteres - conteúdo somente com caracteres - conteúdo somente com dígitos - conteúdo com caracteres e dígitos - troca o conteúdo por data (*date*) - troca o conteúdo por hora (*time*)
CEBI - Caracteres de Espaço em Branco Incorretos	- mantém o conteúdo - acrescenta espaços - exclui espaços - substitui espaço em branco por tabulação e vice-versa - substitui espaço em branco por quebra de linha e vice-versa - somente espaços em branco
Grupo 2 - Restrições de Definição	
UI - Uso Incorreto	- mantém o conteúdo - altera valor para valor nulo - insere o atributo

continua na próxima página

Tabela 4.1 – continuação da página anterior

Classe de Defeito	**Descrição da alteração**
	- exclui o atributo
UNI - Unicidade Incorreta	- mantém o conteúdo - duplica o atributo
II - Identificador Incorreto	- mantém o conteúdo - duplica o mesmo identificador - deixa sem valor - altera um caractere ou um dígito do identificador aleatoriamente
Grupo 3 - Restrições de Relacionamento	
OI - Ocorrência Incorreta	- mantém a ocorrência - exclui ocorrência de elemento um número x de vezes - repete ocorrência de elemento um número x de vezes - insere ocorrência de um elemento x, associando-o ao elemento z ao invés do y
ORI - Ordem Incorreta	- mantém a ordem dos elementos - inverte ordem dos elementos - deixa somente um elemento
AI - Associação Incorreta	- mantém a associação - insere e exclui a ocorrência de outras associações entre os elementos do relacionamento
Grupo 4 - Restrições Semânticas	
COI - Condição Incorreta	- insere dados com valores diferentes dos permitidos pela condição

4.7 Consultas

O conjunto de consultas Q é gerado em uma linguagem de consulta apropriada para o modelo de dados do esquema S que está sendo testado. Cada classe de defeito possui um padrão de consulta, no

qual é definido o resultado que deve ser retornado pela consulta (Tabela 4.2) para que o defeito seja revelado. As consultas padrão são instanciadas para cada instância de dados alternativa válida I_i, de acordo com as associações de defeitos identificadas em S.

Por exemplo, a consulta gerada por meio da associação de defeito $(numcartao, G2{-}UI)$, deveria:

- retornar o número de atributos e de elementos associados a esses atributos para verificar se o atributo é opcional ou obrigatório.

Suponha que o esquema de dados que gerou a associação de defeito $(numcartao, G2 - UI)$ foi escrito em XML *Schema*. A linguagem de consulta *XQuery* seria usada para gerar a consulta de acordo com o padrão estabelecido para a classe de defeito G2-UI. Essa consulta (Q_1) pode ser ilustrada da seguinte forma:

```
<resultado_G2UI>{
for $l in document("alternativa1.xml")/pessoa
let $cont1 := count($l//cliente)
return
   <resultado_elemento>
      {$cont1}
   </resultado_elemento>,
   for $i in document("alternativa1.xml")/pessoa
   let $cont2 := count($i//cliente/@numcartao)
   return
     <resultado_atributo>
        {$cont2}
     </resultado_atributo>
}</resultado_G2UI>
```

Tabela 4.2: Tipos de consultas padrão de acordo com as classes de defeito

Classe de Defeito	**Tipo de consulta**
Grupo 1 - Restrições de Domínio	
TDI - Tipo de Dado Incorreto	A consulta retorna os valores de um atributo para verificar o tipo de dado aceito
continua na próxima página	

Tabela 4.2 – continuação da página anterior

Classe de Defeito	**Tipo de consulta**
VI - Valor Incorreto	A consulta retorna a quantidade do atributo que está especificada, quantos entre esses atributos são iguais e quais os valores especificados para esses atributos, sem repetição de valores, indicando quantas vezes cada um deles foi assumido pelo atributo, para verificar se existe um valor padrão ou fixo.
VEI - Valores Enumerados Incorretos	A consulta retorna somente os valores diferentes atribuídos a um atributo para verificar o conjunto de valores possíveis.
VMMI - Valores Máximos e Mínimos Incorretos	A consulta retorna, no caso de verificar valor máximo, o valor máximo atribuído a um atributo, no caso de valor mínimo, o valor mínimo atribuído a um atributo.
TI - Tamanho Incorreto	A consulta retorna os valores de um atributo ou o maior e menor valor de um atributo, quanto à quantidade de caracteres, juntamente com a quantidade de caracteres.
DI - Dígitos Incorreto	A consulta retorna os valores de um atributo ou o maior e menor valor de um atributo, quanto à quantidade de dígitos, juntamente com a quantidade de dígitos ou dígitos decimais.
SVI - Seqüência de Valores Incorreta	A consulta retorna os valores atribuídos a um atributo para verificar a seqüência de caracteres permitidas.
CEBI - Caracteres de Espaço em Branco Incorretos	A consulta retorna os valores atribuídos a um atributo para obter a quantidade de espaços em branco entre as palavras usando uma função que manipule os resultados.
Grupo 2 - Restrições de Definição	
continua na próxima página	

Tabela 4.2 – continuação da página anterior

Classe de Defeito	**Tipo de consulta**
UI - Uso Incorreto	A consulta retorna o número de atributos e de elementos associados a esses atributos para verificar se o atributo é opcional ou obrigatório.
UNI - Unicidade Incorreta	A consulta retorna valores duplicados atribuídos a um atributo e a quantidade total desse atributo com o valor atribuído, para verificar se os valores são únicos.
II - Identificador Incorreto	A consulta retorna o atributo identificador de um elemento, o número total dos atributos identificadores e dos elementos associados a esses identificadores, bem como, o número de atributos identificadores com valores atribuídos distintos e seus valores, para verificar os atributos identificadores.
Grupo 3 - Restrições de Relacionamento	
OI - Ocorrência Incorreta	A consulta retorna a quantidade de um elemento associado a outro elemento, para verificar qual é ocorrência máxima ou mínima.
ORI - Ordem Incorreta	A consulta retorna todos os atributos de um elemento para verificar a ordem.
AI - Associação Incorreta	A consulta retorna os atributos de um elemento x e de um elemento y relacionados por algum tipo de associação, seus valores e o número de ocorrências, para verificar essa associação.
Grupo 4 - Restrições Semânticas	
COI - Condição Incorreta	A consulta retorna o(s) conteúdo(s) do(s) atributo(s) relacionado(s) à condição que deve ser satisfeita, para verificar essa condição.

4.8 Casos de teste

Um caso de teste é formado pelo dado de teste e o resultado esperado. O dado de teste para o esquema de dados S é formado pelo par $(q,\ i)$, ou seja, os dados de teste são compostos por uma consulta q a determinado elemento ou atributo e uma instância de dados alternativa válida i. A consulta e as instâncias alternativas de dados estão relacionadas à mesma associação de Σ.

Considere a associação de defeito $(numcartao, G2-UI)$, um exemplo de dado de teste seria a consulta Q_1 gerada na Seção 4.7 e a instância de dados alternativa válida I_1 gerada na Seção 4.6.

Os defeitos no esquema em teste são revelados por meio da análise dos resultados obtidos das consultas às instâncias de dados alternativas em relação à especificação dos resultados esperados para o dado de teste, obtido na especificação dos dados do esquema ou da aplicação. Além dos resultados das consultas, o testador também pode analisar as instâncias de dados alternativas inválidas geradas. Isso pode auxiliar na detecção de defeitos, porque dados que deveriam ser válidos se o esquema estivesse correto, conforme a especificação dos dados, podem ter sido considerados inválidos para o esquema em teste.

Uma instância de dados original (I_0), fornecida pelo testador, sofre alterações de acordo com os padrões previamente estabelecidos pelas classes de defeito, formando o conjunto de instâncias de dados alternativas $(I_1,\ I_2,\ ...,\ I_i,\ ...,\ I_n)$. Dessa forma, o resultado esperado de um dado de teste formado por $(q,\ I)$, não é dado em um formato exato, mas por meio de uma especificação do resultado esperado baseada na especificação dos dados para a aplicação. Na Tabela 4.3, um exemplo de especificação de resultado esperado é ilustrado. Considere a associação de defeito $(numcartao, G2-UI)$; as consultas realizadas em "numcartao" segundo a classe de defeito *G2-UI* (Restrição de Definição - Uso Incorreto) têm como resultado esperado um atributo obrigatório.

Tabela 4.3: Exemplo de especificação do resultado esperado para o atributo "numcartao" (a_1)

Atributo	**Classe de defeito**	**Especificação do resultado esperado**
a_1 - numcartao	(*G2-UI*) Restrição de Definição - Uso Incorreto	$uso(a_1) = 'obrigatório'$

4.9 Processo de teste

O processo de teste da abordagem de teste genérica para esquemas de dados proposta neste trabalho é ilustrado na Figura 4.1 e descrito detalhadamente a seguir.

No processo de teste, o esquema em teste S e a instância de dados original I_0, válida para S, são disponibilizados pelo testador.

Inicialmente, o modelo formal que representa S é construído. Com base nesse modelo, o conjunto de associações de defeitos Σ é identificado. Essas associações são identificadas automaticamente ou manualmente, pelo testador. As associações são identificadas automaticamente por meio das regras P de S para detectar defeitos de definição de restrição incorreta. O testador identifica manualmente associações entre elementos ou atributos e classes de defeitos, que não tenham sido identificadas em S por meio das regras P, para revelar defeitos relacionados à definição de restrição ausente. As associações de defeitos Σ que serão exercitadas no teste são selecionadas pelo testador.

A instância de dados original I_0 válida para o esquema S é usada para gerar as instâncias de dados alternativas $(I_1,\ I_2,\ ...,\ I_i,\ ...,\ I_n)$, ou seja, a instância de dados original I_0 sofre uma alteração simples em um de seus elementos ou atributos de acordo com as associações de defeitos Σ selecionadas e com os padrões de alteração. Um exemplo de alteração é a modificação do valor assumido por determinado elemento que possui restrição de valor máximo para que seja verificado se o valor máximo definido no esquema está de acordo com a especificação do dado para o esquema (ver Tabela 4.1). As instâncias de dados alternativas geradas $(I_1,\ I_2,\ ...,\ I_i,\ ...,\ I_n)$ são separadas em válidas e inválidas para o esquema em teste.

As consultas Q são geradas e instanciadas para cada instância de dados alternativa válida I_i, considerando as associações Σ selecionadas.

Os dados de teste $(q,\ I)$ são executados. q é a consulta e I é uma instância de dados alternativa válida para o esquema em teste, q e I estão relacionados à mesma associação selecionada de Σ.

O testador compara os resultados obtidos com a execução das consultas com a especificação dos resultados esperados para verificar se foi revelado algum defeito no esquema com o conjunto de dados de teste executado. O testador também pode usar as instâncias de dados alternativas, consideradas inválidas para o esquema, durante a análise de resultados para auxiliar na detecção de defeito no esquema em teste. Se algum defeito foi revelado, o esquema de dados deve ser corrigido.

4.10 Considerações finais

Neste capítulo, a abordagem de teste de esquema de dados baseada em defeitos, denominada Análise de Instâncias de Dados Alternativas (AIDA), foi apresentada. Inicialmente, o modelo de dados utilizado na abordagem foi introduzido, esse modelo é usado para identificar as restrições em esquemas de dados e os esquemas que podem ser submetidos à abordagem de teste definida neste trabalho. O esquema em teste precisa ser representado de maneira formal para que seus elementos, atributos e restrições possam ser identificados e testados. Portanto, a representação formal usada no

trabalho foi introduzida e um exemplo de uso dessa representação foi ilustrado.

Além disso, as classes de defeito para esquemas de dados foram definidas e compreendem restrições de domínio, definição, relacionamento e semântica associadas a elementos ou atributos. A abordagem de teste inclui um conjunto de associações de defeitos formadas por elementos ou atributos e classes de defeitos, instâncias de dados alternativas e consultas a essas instâncias. As associações de defeitos guiam a produção de instâncias de dados alternativas e de consultas. As instâncias de dados alternativas representam os possíveis defeitos que podem ser revelados no esquema. As consultas são capazes de revelar os defeitos definidos nas classes de defeitos. As instâncias alternativas e as consultas são geradas com base em padrões especificados para cada classe de defeito.

A meta principal da abordagem de teste é evitar que dados incorretos possam ser considerados válidos ou que dados corretos possam ser considerados inválidos devido a um defeito na definição dos dados no esquema. E, conseqüentemente, evitar que esses dados causem falhas na aplicação que os manipula.

É importante ressaltar que a abordagem de teste AIDA proposta neste trabalho explora o teste de esquema de dados de uma forma diferente das outras abordagens de teste que têm a mesma finalidade, isto porque:

- a AIDA é uma abordagem baseada em defeito como as abordagens de Li e Miller [44] e Franzotte e Vergilio [32, 44]; no entanto, as alterações não são feitas no próprio esquema em teste, mas em uma instância de dados associada a esse esquema;

- a AIDA não é uma abordagem de teste associada a um contexto específico, ou seja, a um esquema de dados escrito em uma determinada linguagem;

- a AIDA gera o conjunto de dados de teste automaticamente, justamente porque as alterações são feitas em uma instância de dados válida para o esquema em teste.

Além disso, analogamente ao teste de perturbação de dados [55], descrito no capítulo anterior, no qual os dados de mensagens XML são perturbados ou modificados por pequenas alterações para testar a interação entre componentes Web, as instâncias de dados alternativas geradas pela AIDA podem ser utilizadas para testar a comunicação entre componentes Web ou para testar a aplicação Web ou a base de dados que utiliza o esquema em teste.

No entanto, a abordagem de teste AIDA possui limitações:

- a AIDA não pode ser completamente automatizada porque a atuação do testador é necessária para a análise dos resultados de teste que devem ser comparados com os resultados esperados; no entanto, esta é uma limitação de outras abordagens de teste (por exemplo, o testador determina mutantes equivalentes na Análise de Mutantes);

- os defeitos revelados pela aplicação da AIDA estão relacionados às classes de defeitos previamente identificadas, ou seja, somente são revelados defeitos cobertos pelas classes de defeitos, que podem ser estendidas.

No próximo capítulo é descrita, por meio de exemplos, a aplicação da AIDA em esquemas de dados de contextos diferentes de teste.

Capítulo 5

Contextos de Uso da Abordagem de Teste

A abordagem de teste AIDA pode revelar defeitos em esquemas de dados em diferentes contextos e tem o objetivo de assegurar a integridade dos dados definidos no esquema. Este capítulo apresenta alguns contextos de uso da abordagem de teste de esquema baseada em defeitos, proposta neste trabalho.

5.1 Teste de esquema XML

Como visto no Capítulo 2, os documentos XML, geralmente, são associados a um esquema XML. O esquema XML define a estrutura do documento XML, ou seja, o esquema estabelece os dados e as restrições aos dados presentes no documento XML. Um documento XML é bem-formado se segue as regras sintáticas definidas para XML. Esse documento é válido se é bem-formado e está em conformidade com as regras definidas no esquema. Assim sendo, a AIDA pode ser aplicada para testar esquemas XML utilizados em diferentes situações:

- Dados XML armazenados em arquivos XML, realizando o papel de uma base de dados;
- Dados XML armazenados em base de dados XML, criadas especificamente para armazenar dados XML (com suporte para DOM e linguagens de consulta XML);
- Dados XML armazenados em base de dados relacional;
- Mensagens XML usadas para trocar informações entre componentes em aplicações Web;
- Resultados de consultas a uma base de dados relacional fornecidos em formato XML;
- Documentos XML atualizados devido a alterações na especificação dos dados armazenados nesses documentos.

Essa abordagem de teste também pode ser usada para testar a comunicação baseada em XML entre componentes Web, tais como serviços Web (*Web services*). Um serviço Web é um conjunto de funções disponíveis à aplicações remotas por meio da Web. A comunicação entre o serviço Web e a aplicação ocorre basicamente por meio de SOAP, que é um protocolo baseado em XML para troca de informações. Portanto, a abordagem de teste de esquema poderia ser aplicada nos esquemas dessas mensagens, usadas para o intercâmbio de informações entre aplicações Web e serviços Web, com a meta de assegurar a integridade dos dados que estão sendo manipulados pelos serviços Web. E conseqüentemente, se os dados manipulados pelo serviço Web são confiáveis e acontecer algum erro na resposta obtida do processamento de um serviço Web, o teste de esquema poderia estar auxiliando na detecção de defeitos que possam ser revelados no processamento dos dados executado pelo serviço Web. A Figura 5.1 apresenta esse contexto de teste.

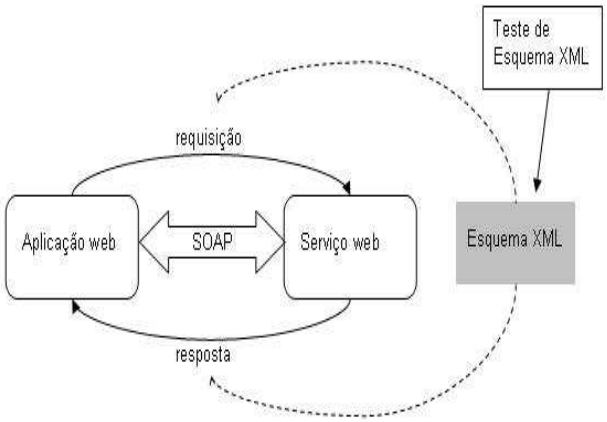

Figura 5.1: Contexto de teste de esquema XML no teste de serviços Web

5.1.1 XML *Schema*

Um esquema XML pode ser escrito, por exemplo, usando DTD (*Document Type Definition*) e XML *Schema* (XML *Language Schema*). A linguagem de definição de esquema denominada XML *Schema* é mais poderosa que a DTD devido a algumas características, tais como: ser escrita em XML, permitir o uso de tipos de dados e usar *namespaces*; e mais empregada atualmente. Portanto, para ilustrar o uso da abordagem de teste de esquema, a AIDA será aplicada em esquemas do tipo XML *Schema*.

A seguir é dado um exemplo de XML *Schema* denominado “disciplinas.xsd”. Esse esquema descreve dados referentes a disciplinas ofertadas em uma instituição, tais como: nome da disciplina, período em que a disciplina deve ser cursada e pré-requisitos para cursar a disciplina.

```
<?xml version="1.0"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
<xs:element name="disciplinas">
  <xs:complexType>
  <xs:sequence>
    <xs:element name="disciplina" minOccurs="0"
                maxOccurs="unbounded">
      <xs:complexType>
        <xs:sequence>
        <xs:element name="prerequisito" type="xs:string"
                    minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="nome" type="xs:string"
                      use="required"/>
        <xs:attribute name="periodo" type="xs:integer"
                      use="optional"/>
    </xs:complexType>
    </xs:element>
  </xs:sequence>
  </xs:complexType>
</xs:element>
</schema>
```

O documento XML apresentado a seguir é a instância de dados original para o XML *Schema* visto acima.

```
<?xml version="1.0"?>
<disciplinas xmlns:xsi="http://www.w3.org/2001/ XMLSchema-instance"
xsi:schemaLocation = "disciplinas.xsd">
    <disciplina nome="Algorithms I" periodo="1">
    </disciplina>
    <disciplina nome="Algorithms II" periodo="2">
        <prerequisito> Algorithms  I</prerequisito>
    </disciplina>
    <disciplina nome="Artificial Intelligence" periodo="6">
        <prerequisito> Introduction to Algebra </prerequisito>
    </disciplina>
</disciplinas>
```

O modelo de dados para o XML *Schema* “disciplinas.xsd” é apresentado na Figura 5.2. Esse modelo é uma instância do metamodelo MM definido no Capítulo 4. As classes são elementos de XML *Schema* e os atributos das classes podem ser elementos-filho de um elemento (classe) ou atributos do elemento (classe). Se for um atributo, o nome do atributo é precedido por (*A*).

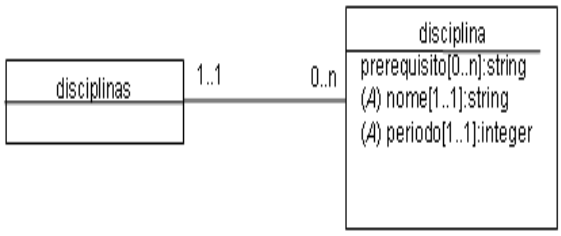

Figura 5.2: Modelo de dados do XML *Schema* “disciplinas.xsd”

O modelo de dados do XML *Schema* “disciplinas.xsd” pode ser representado por $S_{disciplinas} = (E,\ A,\ R,\ P)$, onde:

$E = \{disciplinas,\ disciplina,\ prerequisito\}$

$A = \{nome,\ periodo\}$

$R = \{ordem,\ tipo,\ ocorrência,\ uso\}$

$P = \{p_1(disciplinas,\ disciplina),\ p_2(disciplinas,\ ordem,\ disciplina),$
$p_3(disciplina,\ prerequisito),\ p_4(disciplina,\ nome),\ p_5(disciplina,\ periodo),$
$p_6(disciplina,\ ordem,\ prerequisito),\ p_7(disciplina,\ ocorrência),$
$p_8(prerequisito,\ tipo),\ p_9(prerequisito,\ ocorrência),$
$p_{10}(nome,\ tipo),\ p_{11}(nome,\ uso),$
$p_{12}(periodo,\ tipo),\ p_{13}(periodo,\ uso)\}$

A Tabela 5.1 apresenta as classes de defeitos, descritas no Capítulo 4, instanciadas para cobrir defeitos que poderiam ser revelados em esquemas escritos em XML *Schema*. Em XML *Schema*, os elementos podem assumir valores e, portanto, os elementos podem ser definidos por restrições de domínio.

Tabela 5.1: Classes de defeitos para esquemas em XML *Schema*

Classe de Defeito	**Descrição**
Grupo 1 - Restrições de Domínio	Defeitos associados às restrições de domínio de dados que podem ser assumidos por um elemento ou atributo
TDI - Tipo de Dado Incorreto	Os tipos de dados mais comuns em um XML *Schema* são: *string*, *decimal*, *integer*, *boolean*, *date* e *time*. O usuário pode definir tipos de dados simples e complexos. Os tipos de dados podem estar definidos incorretamente. Palavra-chave: *type*.
continua na próxima página	

Tabela 5.1 – continuação da página anterior

Classe de Defeito	**Descrição**
VI - Valor Incorreto	Um elemento pode ser associado a um valor, padrão ou fixo. O valor é definido como padrão, se deve ser automaticamente assumido pelo elemento quando outro valor não é atribuído. O valor é definido como fixo, se ele é o único valor permitido para o elemento. Essa definição pode estar incorreta. Além disso, o valor definido como fixo ou padrão pode estar incorreto. Palavras-chave: *default*, *fixed*.
VEI - Valores Enumerados Incorretos	Uma lista de valores aceitáveis para elementos e atributos pode ser definida por meio de valores enumerados. Essa lista pode estar definida incorretamente. Palavra-chave: *enumeration*.
VMMI - Valores Máximos e Mínimos Incorretos	O valor numérico de um atributo ou elemento pode ser limitado por valores máximos e mínimos. Os valores limites podem estar definidos incorretamente. Palavras-chave: *maxExclusive*, *minExclusive*, *maxInclusive*, *minInclusive*.
TI - Tamanho Incorreto	A quantidade de caracteres permitida para o conteúdo de atributos ou elementos pode estar definida incorretamente. Palavras-chave: *length*, *minLength*, *maxLength*.
DI - Dígito Incorreto	A quantidade total de dígitos ou de dígitos decimais para valores numéricos pode estar especificada incorretamente. Palavras-chave: *totalDigits*, *fractionDigits*.
SVI - Seqüência de Valores Incorreto	Uma seqüência de letras e números permitidos para o conteúdo de um atributo ou elemento pode estar definida incorretamente. Palavra-chave: *pattern*.
CEBI - Caracteres de Espaço em Branco Incorreto	O tratamento que deve ser dado aos caracteres de espaço em branco (tabulação, espaço, quebra de linha, retorno de carro) no conteúdo de atributos ou elementos pode estar especificado incorretamente. Palavra-chave: *whiteSpace*. Valores possíveis: *preserved* (preservado), *replace* (troca), *collapse* (remove).
Grupo 2 - Restrições de Definição	Defeitos relacionados à integridade de atributos.

continua na próxima página

Tabela 5.1 – continuação da página anterior

Classe de Defeito	Descrição
UI - Uso Incorreto	O uso (ocorrência) de um atributo pode ser definido incorretamente. Palavra-chave: *use*. Valores possíveis: *optional* (opcional), *required* (obrigatório).
Grupo 3 - Restrições de Relacionamento	Defeitos relacionados ao relacionamento entre elementos.
OI - Ocorrência Incorreta	Um número mínimo e máximo de ocorrências pode ser definido para um elemento, indicando quantas vezes o elemento pode ocorrer no documento XML. Essa definição pode estar incorreta. Palavras-chave: *maxOccurs*, *minOccurs*.
ORI - Ordem Incorreta	A ordem em que os elementos-filho de um elemento complexo devem aparecer no documento XML pode estar definida incorretamente. Palavra-chave: *complexType*. Valores possíveis: *all* (os elementos-filho podem estar em qualquer ordem), *sequence* (os elementos-filho devem obedecer a ordem de definição no esquema), *choice* (somente um elemento-filho deve aparecer).
AI - Associação Incorreta	Um elemento pode herdar propriedades de outro elemento por meio do relacionamento de generalização/especialização. Esse relacionamento pode estar definido incorretamente. Palavra-chave: *complexContent*. Valores possíveis: *restriction* (restringir o conteúdo de um tipo complexo), *extension* (estender o conteúdo de um tipo complexo).

Em $S_{disciplinas}$ pode ser observado que o atributo "periodo" está associado a restrição uso, definida nas restrições do metamodelo MM (Seção 4.2). Portanto, com base em $S_{disciplinas}$ a associação de defeito (periodo, *G2 - UI* (Restrição de definição - Uso Incorreto)) é identificada. De acordo com essa associação de defeito são geradas instâncias de dados alternativas modificando a instância de dados original, segundo os padrões de alteração definidos para a classe de defeito *G2 - UI*. Por exemplo, uma instância alternativa válida é gerada com a exclusão do atributo "periodo" em um dos elementos "disciplina" da instância de dados original, conforme visto a seguir.

```
<?xml version="1.0"?>
<disciplinas xmlns:xsi="http://www.w3.org/2001/ XMLSchema-instance"
```

```
xsi:schemaLocation = "disciplinas.xsd">
    <disciplina nome="Algorithms I">
    </disciplina>
    <disciplina nome="Algorithms II" periodo="2">
        <prerequisito> Algorithms  I</prerequisito>
    </disciplina>
    <disciplina nome="Artificial Intelligence" periodo="6">
        <prerequisito> Introduction to Algebra </prerequisito>
    </disciplina>
</disciplinas>
```

Além das instâncias alternativas, as consultas às instâncias válidas são geradas em XQuery. Essas consultas retornam o número de vezes que o atributo "periodo" aparece na instância alternativa válida, bem como, quantas vezes o elemento "disciplina", ao qual o atributo está associado, aparece. Se o número de vezes que o atributo e o elemento aparecem for igual em todos os resultados das consultas às instâncias válidas, o atributo é obrigatório, caso contrário é opcional.

A Tabela 5.2 apresenta a especificação do resultado esperado para as consultas realizadas nas instâncias alternativas válidas geradas segundo a associação de defeito (periodo, *G2 - UI*).

Tabela 5.2: Especificação do resultado esperado para o atributo "periodo"

Atributo	**Classe de defeito**	**Especificação do resultado esperado**
periodo	*G2 - UI*	$uso(periodo) = 'obrigatório'$

Com base nessa especificação do resultado esperado, o atributo "periodo" é obrigatório. No entanto, o resultado das consultas mostrou que esse atributo está definido como opcional no esquema, porque na consulta à instância alternativa válida apresentada anteriormente, o número de vezes que o atributo "periodo" aparece é menor que o número de vezes que o elemento "disciplina" aparece. Portanto, um defeito coberto pela classe de defeito *G2 - UI* é revelado na definição do atributo "periodo" e deve ser corrigido.

5.2 Teste em esquema associado à base de dados

O esquema de uma base de dados consiste na definição da estrutura lógica da base de dados, ou seja, é o projeto e a especificação dos dados. No contexto de esquemas de base de dados, a abordagem de teste pode ser aplicada em bases de dados que seguem o modelo relacional, que é mais comumente empregado.

5.2.1 Esquema associado a base de dados relacional

Inicialmente, a abordagem de teste AIDA está sendo aplicada para testar esquemas associados a bases de dados relacionais baseados no Modelo Entidade-Relacionamento (MER). Um exemplo de modelo entidade-relacional é apresentado na Figura 5.3. Esse exemplo descreve as entidades "curso", "aluno" e "disciplina" por seus atributos e relacionamentos. O MER é representado graficamente pelo diagrama entidade-relacionamento (DER).

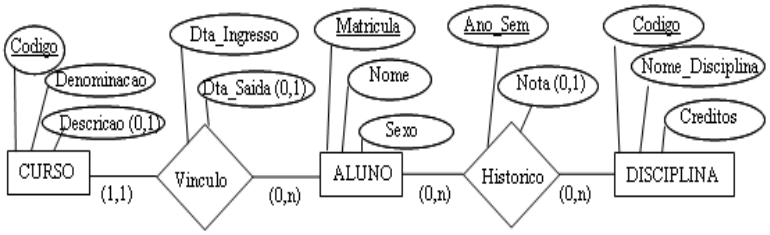

Figura 5.3: Diagrama Entidade-Relacionamento

O modelo de dados, que é instância do metamodelo MM descrito no Capítulo 4, para o DER da Figura 5.3, é ilustrado na Figura 5.4. Na Figura 5.4 as classes são entidades e os atributos das classes são atributos das entidades. Os atributos precedidos por (*I*) são identificadores.

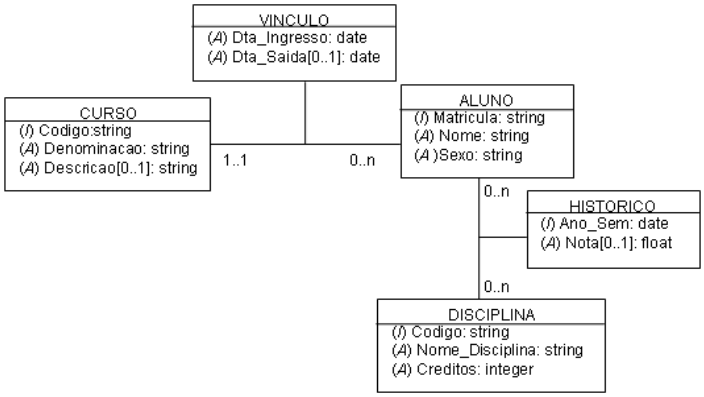

Figura 5.4: Modelo de dados para o esquema representado pelo DER da Figura 5.3

O modelo de dados visto na Figura 5.4 pode ser representado por $S_{curso} = (E,\ A,\ R,\ P)$ onde:
$E = \{CURSO,\ VINCULO,\ ALUNO,\ HISTORICO,\ DISCIPLINA\}$
$A = \{Codigo,\ Denominacao,\ Descricao,\ Dta\_Ingresso,\ Dta\_Saida,\ Matricula,$
$Nome,\ Sexo,\ Ano\_Sem,\ Nota,\ Codigo,\ Nome\_Disciplina,\ Creditos\}$
$R = \{tipo,\ identificador,\ uso,\ associacao\}$

$P = \{p_1(CURSO,\ Codigo),\ p_2(CURSO,\ Denominacao),\ p_3(CURSO,\ Descricao),$
$p_4(CURSO,\ associacao,\ ALUNO),$
$p_5(Codigo,\ tipo),\ p_6(Codigo,\ identificador),$
$p_7(Denominacao,\ tipo),$
$p_8(Descricao,\ tipo),\ p_9(Descricao,\ uso),$
$p_{10}(ALUNO,\ Matricula),\ p_{11}(ALUNO,\ Nome),\ p_{12}(ALUNO,\ Sexo),$
$p_{13}(ALUNO,\ associacao,\ CURSO),\ p_{14}(ALUNO,\ associacao,\ DISCIPLINA),$
$p_{15}(Matricula,\ tipo),\ p_{16}(Matricula,\ identificador),$
$p_{17}(Nome,\ tipo),$
$p_{18}(Sexo,\ tipo)$
$p_{19}(DISCIPLINA,\ Codigo),\ p_{20}(DISCIPLINA,\ Nome\_Disciplina),$
$p_{21}(DISCIPLINA,\ Creditos),\ p_{22}(DISCIPLINA,\ associacao,\ ALUNO),$
$p_{23}(Codigo,\ tipo),\ p_{24}(Codigo,\ identificador),$
$p_{25}(Nome\_Disciplina,\ tipo),$
$p_{26}(Creditos,\ tipo),$
$p_{27}(VINCULO,\ Dta\_Ingresso),\ p_{28}(VINCULO,\ Dta\_Saida),$
$p_{29}(VINCULO,\ associacao,\ CURSO,\ ALUNO),$
$p_{30}(Dta\_Ingresso,\ tipo),$
$p_{31}(Dta\_Saida,\ tipo),\ p_{32}(Dta\_Saida,\ uso),$
$p_{33}(HISTORICO,\ Ano\_Sem),\ p_{34}(HISTORICO,\ Nota),$
$p_{35}(HISTORICO,\ associacao,\ ALUNO,\ DISCIPLINA),$
$p_{36}(Ano\_Sem,\ tipo),\ p_{37}(Ano\_Sem,\ identificador),$
$p_{38}(Nota,\ tipo),\ p_{39}(Nota,\ uso)\}$

As classes de defeitos, descritas no Capítulo 4, instanciadas para tratar defeitos que poderiam ser revelados em esquemas baseados no modelo entidade-relacionamento são apresentadas na Tabela 5.3. Nesse caso, os itens definidos como elementos nas classes de defeitos são entidades.

Tabela 5.3: Classes de defeitos para esquema de base de dados baseado no MER

Classe de Defeito	**Descrição**
Grupo 1 - Restrições de Domínio	Defeitos associados às restrições de domínio de dados que podem ser assumidos por um atributo.
continua na próxima página	

Tabela 5.3 – continuação da página anterior

Classe de Defeito	**Descrição**
TDI - Tipo de Dado Incorreto	A definição do tipo de dado de um atributo pode estar incorreta.
VI - Valor Incorreto	Um atributo pode ser associado a um valor padrão. O valor definido como padrão pode estar incorreto.
VEI - Valores Enumerados Incorretos	Uma lista de valores aceitáveis para atributos pode estar definida incorretamente.
VMMI - Valores Máximos e Mínimos Incorretos	O valor numérico de um atributo pode ser limitado por valores mínimos e máximos. Os valores limites podem estar definidos incorretamente.
TI - Tamanho Incorreto	A quantidade de caracteres do conteúdo de um atributo pode estar definida incorretamente.
CEBI - Caracteres de Espaço em Branco Incorreto	O tratamento que deve ser dado aos caracteres de espaço em branco no conteúdo de atributos pode estar especificado incorretamente.
Grupo 2 - Restrições de Definição	Defeitos relacionados à integridade dos dados.
UI - Uso Incorreto	O atributo pode ser definido incorretamente como opcional ou obrigatório.
UNI - Unicidade Incorreta	A unicidade do conteúdo de um atributo pode estar definida incorretamente.
II - Identificador Incorreto	Um atributo ou um conjunto de atributos podem estar definidos incorretamente como identificador.
Grupo 3 - Restrições de Relacionamento	Defeitos relacionados ao relacionamento de entidades.
OI - Ocorrência Incorreta	O número de instâncias de uma entidade pode estar incorreto.
AI - Associação Incorreta	Entidades podem estar relacionadas por uma associação do tipo cardinalidade, generalização/especialização, agregação e elemento associativo definida incorretamente.
Grupo 4 - Restrições Semânticas	Defeitos relacionados à definição de condições para o conteúdo de atributos.

continua na próxima página

Tabela 5.3 – continuação da página anterior

Classe de Defeito	Descrição
COI - Condição Incorreta	Uma condição que deve ser satisfeita pelo conteúdo de um atributo pode estar definida incorretamente.

Suponha que na especificação dos dados para a base de dados representada na Figura 5.4 é definido que a data de saída ("Dta_Saida") de um aluno deve ser posterior a data de ingresso ("Dta_Ingresso") desse aluno. Portanto, o testador identifica a associação de defeito (Dta_Saida:Dta_Ingresso, *G4 - COI* (Restrição semântica - Condição Incorreta)). De acordo com essa associação de defeito são geradas instâncias de dados alternativas modificando a instância de dados original (base de dados original), com a inserção de registros com dados no atributo "Dta_Saida" iguais, anteriores ou posteriores aos dados do atributo "Dta_Ingresso". Um exemplo de instância alternativa é dado na Tabela 5.4, na qual é observada a inserção de um registro com o valor de "Dta_Saida" anterior ao valor de "Dta_Ingresso".

Tabela 5.4: Exemplo de valores de uma instância de dados alternativa relacionada à base de dados representada pelo DER da Figura 5.3

...	**Dta_Ingresso**	**Dta_Saida**
...	01/08/1999	31/07/2004
...	01/08/2000	31/07/2004
...	01/08/2005	01/07/2005

Além das instâncias alternativas, as consultas às instâncias válidas são geradas em SQL. Essas consultas retornam os conteúdos dos atributos "Dta_Saida" e "Dta_Ingresso" para que o testador compare os resultados obtidos com a especificação dos resultados esperados. A Tabela 5.5 apresenta a especificação do resultado esperado para as consultas realizadas nas instâncias alternativas válidas geradas segundo a associação de defeito (Dta_Saida:Dta_Ingresso, *G4 - COI*).

Tabela 5.5: Especificação do resultado esperado para o atributo referente a data de saída do aluno

Atributo	**Classe de defeito**	**Especificação do resultado esperado**
Dta_Saida	*G4 - COI*	$Dta\_Saida > Dta\_Ingresso$

Com base no resultado das consultas, um defeito coberto pela classe de defeito *G4 - COI* é revelado na definição do atributo "Dta_Saida" e deve ser corrigido. Isto porque foram considerados válidos, pelo esquema em teste, dados com a data de saída anterior ou igual a data de ingresso.

5.3 Considerações finais

Neste capítulo foram identificados alguns contextos de uso da abordagem de teste de esquema AIDA no âmbito de XML e de base de dados; de forma mais detalhada, o uso da abordagem de teste foi apresentado em dois contextos: esquemas escritos em XML *Schema* e esquemas de base de dados relacional baseados no Modelo Entidade-Relacionamento. Exemplos desses esquemas foram apresentados por meio do modelo de dados, instanciado do metamodelo MM definido no Capítulo 4, mostrando que as restrições previstas no metamodelo MM podem ser aplicadas nesses esquemas, e da representação formal previamente definida para que os esquemas possam ser processados algoritmicamente. Além disso, as classes de defeitos instanciadas para esquemas em XML *Schema* e esquemas de base de dados baseados no MER, segundo as restrições previstas para esses tipos de esquemas, foram ilustradas. Exemplos de uso e de possíveis defeitos a serem revelados pela AIDA também foram apresentados, mostrando como ela contribui e pode ser eficaz em revelar defeitos nesses contextos.

Neste capítulo também pôde ser observado que o esforço necessário para a adaptação da abordagem de teste em contextos diferentes está em instanciar as classes de defeitos definidas na AIDA para um contexto específico, ou seja, verificar quais classes de defeitos podem descrever possíveis defeitos de um determinado tipo de esquema. Isto porque alguma classe de defeito pode não ser necessária para a descoberta de defeitos em um determinado tipo de esquema, por não descrever um possível defeito desse tipo de esquema de dados. O esforço pode ser maior nessa tarefa se for necessário estender as classes de defeitos para um determinado contexto, no qual possa ser aplicada a abordagem de teste.

Uma ferramenta de suporte a abordagem foi implementada para automatizar e facilitar o processo de teste. E com o uso dessa ferramenta, a aplicabilidade da abordagem foi avaliada, assim como sua capacidade de revelar defeitos. Essa avaliação é assunto do próximo capítulo.

Capítulo 6

Avaliação da Análise de Instâncias de Dados Alternativas

A avaliação da AIDA tem por finalidade verificar a aplicabilidade da abordagem de teste e a sua capacidade de revelar defeitos em esquemas de dados. Para isso, foram realizados quatro estudos de caso, dois em esquemas XML e dois em esquemas associados a bases de dados relacionais.

A avaliação da eficácia da abordagem de teste foi executada em esquemas em desenvolvimento e em esquemas com defeitos semeados pela ferramenta XTM (ver Capítulo 3) ou por alunos, no caso em que não havia uma ferramenta de Análise de Mutantes disponível. Alguns passos realizados durante os estudos de caso são similares para XML e base de dados relacional.

Neste capítulo, uma breve descrição da ferramenta de teste desenvolvida por Nazar [54] para apoiar a AIDA é apresentada, os estudos de caso realizados são descritos e seus resultados são analisados. Além disso, uma estratégia de aplicação da abordagem de teste, baseada na análise dos resultados dos estudos de caso e das classes de defeitos definidas na AIDA, é explorada.

6.1 Ferramenta

A ferramenta denominada XTool [54] foi desenvolvida para apoiar a AIDA. Essa ferramenta realiza o teste de esquemas XML escritos em XML *Schema* e o teste de esquemas de base de dados relacional em PostGre SQL *Schema*.

A XTool foi implementada na linguagem Java. No contexto de esquemas XML a ferramenta emprega a API DOM (*Document Object Model*) [78], usada para manipular e validar o conteúdo de um documento XML, e o framework Kawa [3], que permite consultar documentos XML por meio da linguagem de consulta *XQuery* (XML *Query*) [40, 79]. No contexto de esquemas de base de dados relacional a ferramenta usa JDBC (*Java Database Connectivity*) [71] para manipular e consultar

informações em bases de dados PostGreSQL [59] por meio das linguagens DQL (Linguagem de Consulta de Dados) e DML (Linguagem de Manipulação de Dados) que são subconjuntos da SQL (*Structured Query Language*).

A execução da ferramenta XTool envolve: uma aplicação de software que manipule documentos XML relacionados a um esquema XML ou uma aplicação de base de dados relacional; classes de defeitos; instâncias de dados alternativas; consultas às instâncias de dados alternativas em uma linguagem apropriada ao tipo de esquema que está sendo testado; e o testador, que providencia o esquema em teste e uma instância de dados válida para o esquema, seleciona as associações de defeitos e analisa os resultados de teste com relação aos resultados esperados, de acordo com a especificação dos dados para a aplicação de software.

A Figura 6.1 ilustra as funcionalidades da ferramenta. A XTool é iniciada pelo testador, que escolhe o tipo de esquema que será testado (esquema XML ou esquema de base de dados relacional) e, em seguida, fornece o esquema que será testado e a instância de dados original, indicando o caminho e nome desses arquivos.

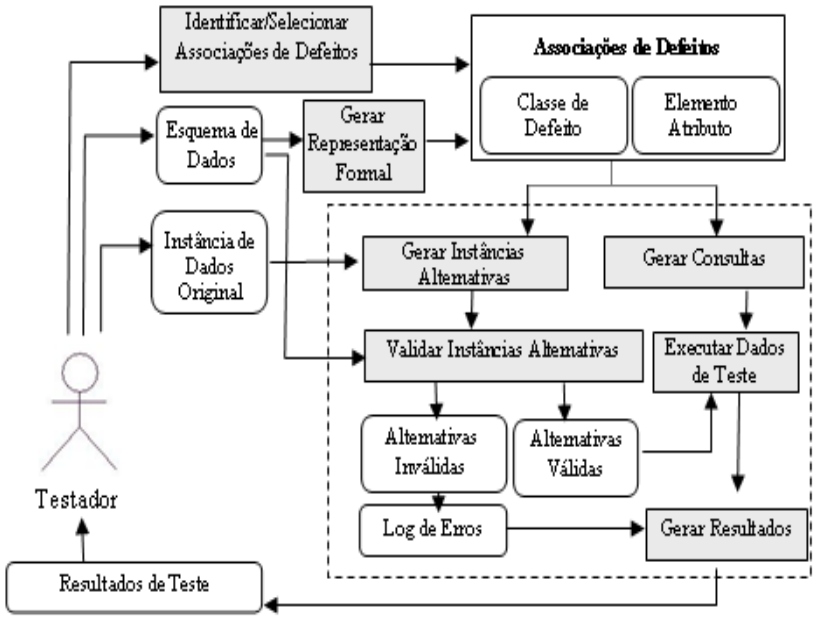

Figura 6.1: Ilustração das funcionalidades da XTool

A seguir é apresentada uma descrição sucinta das funcionalidades da ferramenta XTool:

- *Gerar representação formal* - o esquema em teste S é representado em termos de elementos E, atributos A, restrições R e regras de associação entre os elementos, atributos e restrições

P. A ferramenta processa o teste com base nessa representação, gerando as associações, as instâncias de dados alternativas e as consultas;

- *Gerar/Selecionar associações de defeitos* - o conjunto de associações de defeitos Σ relaciona elementos ou atributos do esquema às classes de defeitos introduzidas na Seção 4.4, por meio da representação formal ou manualmente pelo testador. As associações, que serão exploradas para revelar defeitos, devem ser selecionadas para que as instâncias de dados alternativas e as consultas sejam geradas;

- *Gerar alternativas* - as instâncias de dados alternativas são obtidas com base no conjunto de associações selecionadas e no padrão de alterações na instância de dados original, introduzido na Seção 4.6. As instâncias alternativas são validadas conforme o esquema em teste e separadas em válidas e inválidas;

- *Gerar consultas* - as consultas às instâncias de dados alternativas válidas são obtidas de acordo com as associações identificadas, segundo um padrão de consultas pré-estabelecido (ver Seção 4.7);

- *Executar dados de teste* - os dados de teste, formados pelas consultas e instâncias de dados alternativas válidas de acordo com as associações de defeitos, são executados pela ferramenta;

- *Gerar Resultados* - os resultados obtidos com a execução do teste são disponibilizados ao testador por meio de um relatório. Dessa forma, o testador verifica se os resultados obtidos estão em concordância com os resultados esperados, de acordo com a especificação dos dados. Um defeito no esquema de dados é revelado sempre que os resultados obtidos diferem dos resultados esperados.

6.2 Estudos de caso

Os estudos de caso foram realizados com o objetivo de avaliar a abordagem de teste de esquema de dados baseada em defeitos, proposta neste trabalho, analisando a aplicabilidade, o custo e a eficácia de seu uso. Esses estudos de caso foram executados com o auxílio da ferramenta XTool. Informações adicionais sobre os estudos de caso podem ser obtidas em Nazar [54].

6.2.1 Estudo de caso I - Testando esquemas XML

No primeiro estudo de caso foram testados esquemas de dois sistemas baseados na Web: Sistema de Matrícula e Sistema de Biblioteca. Esses sistemas foram desenvolvidos por dois grupos de alunos

do curso de mestrado em Ciências da Computação da Universidade Federal do Paraná. A meta dos alunos era o desenvolvimento de uma aplicação Web usando modelos UML. As especificações dos sistemas e dos esquemas foram passadas para cada grupo.

O sistema de matrícula tem dados que se referem aos estudantes e às disciplinas disponíveis no curso; baseado em dois esquemas XML *Schema*: "aluno.xsd" e "disciplina.xsd". O sistema de biblioteca contém dados que se referem aos usuários registrados e aos títulos disponíveis na coleção. Os documentos XML estão baseados em dois esquemas XML *Schema*: "usuario.xsd" e "obras.xsd". O Apêndice B contém as especificações desses sistemas, os esquemas, fragmentos dos documentos XML definidos pelos esquemas e outros dados referentes ao processo de teste.

Na Tabela 6.1 são apresentadas características dos esquemas, tais como: o número de elementos, atributos e restrições dos esquemas; a profundidade do esquema (considerando uma estrutura de árvore); e o número de registros dos documentos XML correspondentes. Tais características estão relacionadas com o número de instâncias de dados alternativas (documentos XML alternativos) e consultas que serão geradas com a aplicação da abordagem de teste.

Tabela 6.1: Características dos esquemas - Estudo de caso I

Esquema	**# Elementos**	**# Atributos**	**# Restrições**	**Profundidade**	**# Registros**
aluno	10	0	4	3	5
disciplina	10	0	4	3	6
obras	11	0	4	4	7
usuario	8	0	4	3	6

A XTool gerou a representação formal de cada esquema. A Tabela 6.2 apresenta o número de associações de defeitos identificadas e selecionadas. Nesse estudo de caso, o testador não adicionou associações de defeitos, foram selecionadas somente as associações identificadas pela XTool. Na Tabela 6.2 também são apresentados o número de instâncias alternativas geradas e o número de consultas produzidas pela XTool de acordo com as associações de defeitos selecionadas.

Tabela 6.2: Resultados de teste da XTool para cada esquema - Estudo de caso I

Esquema XML	**# Associações de defeitos**	**# Alternativas válidas/ inválidas**	**# Consultas geradas**
aluno	16	241/42	16
disciplina	16	283/43	16
obras	16	252/138	16
usuario	12	187/73	12

Na Tabela 6.2 pode ser observado que o número de associações de defeitos é igual ao número de

consultas geradas. Isto ocorre porque para cada associação de defeito somente uma consulta é gerada de acordo com o padrão de consultas. As instâncias de dados alternativas são geradas conforme um padrão de alterações e por isso, algumas alternativas geradas podem não estar em conformidade com as definições especificadas no esquema em teste. Essas alternativas são consideradas inválidas após o processo de validação das instâncias alternativas geradas.

Os dados de teste, compostos por uma consulta e uma instância de dados alternativa válida correspondente, foram executados. Na análise dos resultados de teste, o testador comparou os resultados obtidos com a especificação dos resultados esperados advindos da especificação dos dados. Os resultados do estudo de caso mostraram a presença de 13 defeitos nos esquemas testados. Os defeitos encontrados não foram semeados, eles foram detectados durante o desenvolvimento das aplicações pelos alunos. Esses defeitos estão relacionados à definição incorreta de ocorrência e tipo de dado. Na Tabela 6.3 é informado o defeito detectado e o número de defeitos revelados nos esquemas: "aluno.xsd", "disciplina.xsd", "obras.xsd" e "usuario.xsd" [1].

Tabela 6.3: Defeitos revelados - Estudo de caso I

Esquema XML	**Elemento**	**Defeito revelado**	**Número de defeitos revelados**
aluno	codigo, senha, disciplina_cursadas, disciplina_matriculadas	Restrições de Domínio - Tipo de Dado Incorreto	4
disciplina	codigo, carga_horaria, creditos	Restrições de Domínio - Tipo de Dado Incorreto	4
	codigo_professor	Restrições de Relacionamento - Ocorrência Incorreta	
obras	codigo, ano	Restrições de Domínio - Tipo de Dado Incorreto	2
usuario	codigo, pag_mensalidade, senha	Restrições de Domínio - Tipo de Dado Incorreto	3
Total	13	2	13

Com base nesse estudo de caso podem ser feitas as seguintes observações:

- o maior custo observado para a aplicação da abordagem está relacionado à análise dos resul-

[1] É importante observar que esses esquemas foram testados manualmente em um estudo de caso, descrito em [27], com o objetivo de validar essa abordagem de teste. No entanto, o número de defeitos revelados nesse estudo de caso não foi o mesmo detectado com o apoio da ferramenta XTool; isto porque no processo manual, o testador identificou e selecionou associações de defeitos relacionadas à definição de restrições ausentes, o que não foi realizado com o uso da XTool.

tados de teste realizada pelo testador, que compara os resultados obtidos com os resultados esperados;

- com respeito à eficácia, essa abordagem de teste com o apoio da XTool é eficaz em revelar defeitos cobertos pelas classes de defeitos em esquemas de dados XML.

6.2.2 Estudo de caso II - Testando esquemas XML com defeitos inseridos por operadores de mutação

A meta do segundo estudo de caso foi avaliar a eficácia da abordagem de teste baseada em defeitos com respeito a defeitos descritos por alguns operadores de mutação introduzidos por Franzotte e Vergilio [32].

Esse estudo de caso foi realizado com auxílio das ferramentas XTool e XTM [32], que implementa o teste de mutação. A idéia principal é gerar esquemas mutantes com a XTM e executar a XTool nesses mutantes para verificar se os defeitos representados pelos mutantes são revelados pelos dados de teste gerados pela XTool. Detalhes desse estudo de caso estão em Emer et al. [26].

Cinco esquemas XML foram usados no estudo de caso. Esses esquemas foram obtidos de diagramas produzidos pela aplicação *hyperModel* [7] e foram chamados de esquemas originais. Os esquemas são baseados em vocabulários relacionados a um catálogo de produtos (“catalog.xsd”) e a informações usadas no serviço de comércio eletrônico Amazon (“seller.xsd”, “cart.xsd”, “transaction.xsd”, “customer.xsd”). Um documento XML válido (instância de dados original) foi providenciado para cada esquema original. Esses documentos foram denominados documentos XML originais. O Apêndice C apresenta dois desses esquemas originais.

A Tabela 6.4 apresenta algumas características dos esquemas originais, como: o número de elementos, atributos, e restrições, a profundidade do esquema e o número de registros dos documentos XML originais usados no estudo de caso.

Tabela 6.4: Características dos esquemas originais - Estudo de caso II

Esquema original	# **Elementos**	# **Atributos**	# **Restrições**	**Profundidade**	# **Registros**
catalog	20	0	5	5	6
seller	20	1	8	5	1
cart	36	0	5	5	4
transaction	51	0	5	6	9
customer	21	0	7	5	1

O procedimento realizado no estudo de caso e os resultados obtidos em cada passo são apresentados a seguir.

1. Testar os esquemas originais com a XTool. A XTool foi executada com os esquemas originais usando o documento XML original correspondente. A Tabela 6.5 apresenta alguns resultados gerados pela XTool.

Tabela 6.5: Resultados de teste da XTool para os esquemas originais - Estudo de caso II

Esquema original	**# Associações de defeitos**	**# Consultas**	**# Instâncias de dados alternativas Válidas/Inválidas**
catalog	28	28	345/87
seller	45	45	174/60
cart	71	71	235/110
transaction	86	86	723/189
customer	46	46	139/63

2. Criar mutantes para cada esquema original usando a XTM. Os operadores de mutação introduzidos por Franzotte e Vergilio [32] aplicados nos esquemas originais pela XTM, foram: DT, LO, CSP, STE, RT (ver Seção 3.1.2). Os mutantes gerados foram validados sintaticamente e os mutantes inválidos foram descartados. Esses mutantes foram validados por meio de *parsers* comumente usados, como W3C XML *Schema validator* [80]. Os esquemas mutantes equivalentes aos esquemas originais também foram descartados. A Tabela 6.6 mostra o número de mutantes válidos gerados pela XTM para cada esquema original de acordo com os operadores de mutação empregados no estudo de caso.

Tabela 6.6: Número de mutantes gerados pela XTM por operador de mutação - Estudo de caso II

Esquema original	**DT**	**LO**	**CSP**	**STE**	**RT**	**Total**
catalog	56	20	20	52	39	187
seller	55	21	20	56	27	179
cart	27	36	36	160	32	291
transaction	35	51	51	136	46	319
customer	16	21	21	65	19	142
Total	189	149	148	469	163	1118

3. Testar os esquemas mutantes com a XTool. Para testar os esquemas mutantes utilizando a XTool, documentos XML válidos para cada esquema mutante foram gerados baseados nos

documentos XML originais correspondentes. Na Tabela 6.7, alguns resultados gerados pela XTool são apresentados. Nessa tabela, o número de associações de defeitos, documentos XML alternativos (ou instâncias de dados alternativas) válidos e inválidos são relacionados a todos os esquemas mutantes gerados por cada operador de mutação. Por exemplo, o operador STE gerou 52 esquemas mutantes para o esquema original "catalog" e o número 1456 é a soma das associações de defeitos identificadas e selecionadas desses mutantes. Observe que 1456 (número total de associações de defeitos) dividido por 52 (número de esquemas mutantes) é igual a 28; portanto, o número de associações de defeitos identificadas pela XTool para cada esquema mutante válido segundo o operador STE foi o mesmo gerado para o esquema original correspondente. O mesmo aconteceu para os outros mutantes, exceto para os mutantes criados pelo operador estrutural RT, porque este operador remove elementos do esquema original. O número de consultas geradas pela XTool para o esquema original e para os esquemas mutantes também foi igual, exceto para os mutantes gerados pelo operador RT.

4. Analisar os resultados de teste. Na análise dos resultados de teste, os resultados dos esquemas originais obtidos com a XTool foram considerados resultados esperados. Nesse caso, os resultados dos esquemas originais foram comparados automaticamente com os resultados de teste dos esquemas mutantes correspondentes. Um defeito representado por determinado mutante foi considerado revelado se o resultado de teste desse mutante fosse diferente do resultado de teste do esquema original correspondente. Isto significa que a abordagem de teste baseada em defeitos também é capaz de revelar defeitos descritos pelos operadores de mutação usados no estudo de caso. A Tabela 6.8 apresenta o número de defeitos revelados nos esquemas mutantes.

É importante observar que os defeitos representados pelos esquemas mutantes válidos criados pela XTM não seriam revelados por *parsers* comumente usados para validar esquemas. Entretanto, pode ser observado na Tabela 6.8 que a abordagem de teste proposta neste trabalho é capaz de revelar todos os defeitos representados por esses mutantes. E, conforme o resultado desse estudo de caso, pode ser verificado que as associações de defeitos exercitadas cobriram todos os defeitos descritos pelos operadores de mutação aplicados na avaliação, ou seja, as classes de defeito da abordagem de teste cobriram os defeitos representados nos esquemas mutantes gerados pelos operadores de mutação empregados no estudo de caso.

6.2.3 Estudo de caso III - Testando esquema de base de dados relacional

O terceiro estudo de caso é baseado em uma aplicação de base de dados relacional que contém dados referentes ao histórico de alunos egressos de uma determinada Universidade, tais como: dados pessoais, dados acadêmicos, e dados profissionais. Originalmente, essa aplicação de base de dados foi

Tabela 6.7: Resultados de teste da XTool para cada esquema mutante gerado por operador de mutação - Estudo de caso II

Esquema mutante	**Operador de mutação**	**# Associações de defeitos**	**# Alternativas válidas/Inválidas**
catalog	STE	1456	17948/4516
	CSP	560	6564/1740
	DT	1400	16832/4768
	RT	904	7466/2138
	LO	560	6564/1740
seller	STE	2464	9688/3360
	CSP	880	3462/1359
	DT	2422	9256/3561
	RT	1082	4143/1423
	LO	924	3633/1260
cart	STE	11360	13440/2080
	CSP	2556	2989/433
	DT	1915	2285/370
	RT	2144	2425/419
	LO	2556	2984/428
transaction	STE	11696	15112/3416
	CSP	4386	5567/1181
	DT	3008	3889/881
	RT	3772	4765/996
	LO	4386	5561/1175
customer	STE	2990	3835/845
	CSP	966	1295/329
	DT	731	944/211
	RT	782	1006/224
	LO	966	1295/329

Tabela 6.8: Número de defeitos revelados - Estudo de caso II

Esquema mutante	**# Defeitos revelados**
catalog	187
seller	179
cart	291
transaction	319
customer	142
Total	1118

desenvolvida para o SGBD *MS SQL Server 2000*, que é uma aplicação proprietária. Para a execução

desse estudo de caso, a base de dados foi implementada usando PostGreSQL.

No DER da base de dados são encontradas 19 (dezenove) relações e 20 (vinte) relacionamentos (ver Apêndice D). A XTool identificou 19 entidades e 73 atributos na representação formal do esquema dessa base de dados. A Tabela 6.9 identifica as entidades, o número de atributos e de registros encontrados para cada entidade.

Tabela 6.9: Características do esquema de base de dados - Estudo de caso III

Entidades	# **Atributos**	# **Registros**
curso	4	6
tipo_curso	2	4
instituição	4	4
tipo_instituição	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_observação	2	10
cargo	4	5
nível	2	5
pessoa	8	8
emprego	8	6
observação	5	20
histórico_curso	6	4

As entidades e atributos, apresentadas na Tabela 6.9, foram associadas automaticamente às classes de defeitos, formando as associações de defeitos. Além disso, o testador identificou outras associações de defeitos para revelar defeitos quanto à definição de restrições ausentes. Todas as associações foram selecionadas, gerando instâncias de dados alternativas e consultas SQL de acordo com os padrões de alterações e de consultas definidos nas classes de defeitos. Somente instâncias de dados alternativas válidas para o esquema foram consultadas. A Tabela 6.10 mostra o número de associações de defeitos identificadas pela XTool e pelo testador e o número de consultas geradas.

A Tabela 6.11 apresenta um exemplo com associações de defeitos, o número de registros modificados na instância de dados original para gerar as instâncias alternativas e o número de consultas geradas para a entidade "histórico_curso" e seus atributos.

O testador comparou os resultados de teste, obtidos com a execução dos dados de teste, com os

Tabela 6.10: Número de associações de defeitos e de consultas geradas - Estudo de caso III

Identificador da associação	# **Associações de defeitos**	# **Consultas**
XTool	274	990
Testador	23	250
Total	297	1240

Tabela 6.11: Número de associações de defeitos, registros alterados e consultas geradas - Estudo de caso III

Associação de defeito		# **Registros**	# **Consultas**
histórico_curso	G3-Associação Incorreta (cardinalidade)	41	12
	G3-Associação Incorreta (elemento associativo)	6	1
	G2-Identificador Incorreto	1	1
	G2-Unicidade Incorreta	1	1
ID_histórico	G1-Tipo de Dado Incorreto	7	4
	G1-Dígito Incorreto	8	3
	G2-Uso Incorreto	4	1
ID_curso	G1-Tipo de Dado Incorreto	7	1
	G1-Dígito Incorreto	8	1
	G2-Uso Incorreto	4	1
ID_instituição	G1-Tipo de Dado Incorreto	7	1
	G1-Dígito Incorreto	8	1
	G2-Uso Incorreto	4	1
data_entrada	G1-Tipo de Dado Incorreto	7	1
	G2-Uso Incorreto	4	1
	G4-Condição Incorreta	48	25
ID_aluno	G1-Tipo de Dado Incorreto	7	1
	G1-Dígito Incorreto	8	1
	G2-Uso Incorreto	4	1
data_saída	G1-Tipo de Dado Incorreto	7	2
	G4-Condição Incorreta	48	25

resultados esperados, de acordo com a especificação dos dados da base de dados. Os registros das instâncias de dados alternativas inválidas também foram usados para verificar os resultados de teste. A Tabela 6.12 apresenta o número de defeitos revelados com os dados de teste gerados pela XTool.

Os defeitos revelados no processo de teste estão relacionados à definição incorreta das restrições de tipo de dados, tamanho e dígito; à ausência das restrições de uso e valores enumerados para atri-

Tabela 6.12: Defeitos revelados de acordo com as classes de defeitos - Estudo de caso III

Classes de defeitos selecionadas	# **Defeitos revelados**
G1-Tipo de dado incorreto	5
G1-Tamanho incorreto	3
G1-Dígito incorreto	4
G1-Valores enumerados incorretos	1
G2-Uso incorreto	13
G3-Associação incorreta (cardinalidade)	1
Total	27

butos; e, à associação incorreta de cardinalidade em um relacionamento. Os defeitos de ausência de restrições foram revelados com as associações de defeitos identificadas pelo testador. Esses defeitos podem ter sido introduzidos quando a base foi reescrita para PostGreSQL ou devido à interpretação incorreta dos diagramas que especificam o esquema.

É importante observar que a instância de dados original não é replicada pela XTool para gerar as instâncias alternativas. A instância original é atualizada com um único padrão de alteração, examinada pela consulta gerada e a alteração é desfeita.

Esse estudo de caso serviu para validar a abordagem de teste para esquema de dados no contexto de base de dados relacional, mostrando que a abordagem é capaz de auxiliar na detecção de defeitos em esquemas desse tipo. Com base no estudo de caso, também pode-se afirmar que o maior custo da aplicação da abordagem está na análise dos resultados de teste; dado que, essa análise é feita manualmente e envolve uma grande quantidade de dados devido ao número de consultas geradas.

6.2.4 Estudo de caso IV - Testando esquema de base de dados com inserção ad hoc de defeitos

O quarto estudo de caso teve por objetivo avaliar a eficácia da abordagem de teste em revelar defeitos semeados de maneira ad hoc em esquemas de bases de dados relacional.

No estudo de caso foram testados dois esquemas de bases de dados baseados no modelo entidade-relacionamento. O primeiro esquema especifica dados para gestão de cooperativa rural, armazenando informações sobre os cooperados [65]. O segundo define dados para gestão de bibliotecas, contendo informações sobre obras disponíveis na biblioteca bem como para controle de empréstimos e devoluções [30]. Os esquemas dessas bases de dados podem ser visualizados no Apêndice E. Para cada esquema em teste, uma instância de dados válida (instância de dados original) foi disponibilizada.

O estudo de caso foi realizado por um aluno do curso de graduação em Ciências da Computação, que ainda não conhecia a abordagem de teste de esquemas de dados. Além desse aluno, partici-

param do estudo de caso alunos do curso de mestrado em Ciências da Computação, que também não conheciam a abordagem de teste e as classes de defeitos implementadas pela XTool. Esses alunos atuaram no estudo de caso inserindo defeitos nos esquemas originais. Os passos realizados no estudo de caso e seus resultados são descritos a seguir. Mais detalhes do estudo de caso podem ser observados em Caxeiro [8].

1. Geração de diferentes versões de esquemas (mutantes) para cada esquema original, por meio de pequenas alterações. Os mutantes representam possíveis esquemas incorretos a serem utilizados na análise de eficácia. Foram gerados 19 (dezenove) mutantes para cada esquema original (cooperativa e biblioteca). Algumas alterações nos esquemas originais são descritas na Tabela 6.13.

Tabela 6.13: Exemplos de alterações aplicadas nos esquemas originais - Estudo de caso IV

Esquema	**Alteração**
cooperativa	Incluir visibilidade - Lote/Aquisição
	Incluir relacionamentos das tabelas lote e cooperado com a tabela compra
	Remover tabela pagamento
	Alterar campo “data_aquisição” para tipo inteiro
biblioteca	Inverter origem do relacionamento Listabiblioassunto - editora
	Alterar tipo de atributo - alterar campo “feriado” da tabela calendário de boolean para inteiro
	Alterar o tamanho do atributo “apelido” da tabela autor de 50 para 30
	Alterar campo “data_aquisição” para tipo inteiro (15) Remover chave primaria da tabela biblioteca

2. Teste do esquema original e dos esquemas mutantes com a XTool. A Tabela 6.14 apresenta o número total e a média de associações, instâncias alternativas e consultas geradas para o esquema original e a soma desses valores para todos os esquemas mutantes.

   Na Tabela 6.14, pode ser observado que o número médio de associações de defeitos identificadas nos esquemas mutantes está próximo do número total de associações identificadas para os esquemas originais correspondentes. Também podemos dizer que o mesmo ocorre para o número de instâncias de dados alternativas e consultas geradas.

3. Comparação dos resultados de teste obtidos. Os resultados obtidos com o teste dos esquemas originais são comparados com os resultados de teste obtidos com os respectivos mutantes. Se

Tabela 6.14: Número de associações, instâncias de dados alternativas e consultas geradas pela XTool para os esquemas originais e mutantes - Estudo de caso IV

Esquema		**Associações de defeitos**		**Instâncias alternativas**				**Consultas**	
				Válidas		Inválidas			
		Total	Média	Total	Média	Total	Média	Total	Média
Original	cooperativa	135	135	166	166	481	481	301	301
	biblioteca	168	168	520	520	663	663	668	668
Mutante	cooperativa	2617	138	2097	110	4858	256	4537	239
	biblioteca	4424	233	9430	496	11670	614	13940	734

os resultados de teste dos mutantes diferem dos resultados de teste do esquema original correspondente, o defeito descrito pelo mutante é considerado revelado pela abordagem de teste implementada na XTool. Dessa forma, foram revelados 36 defeitos (das 38 alterações inseridas) para os esquemas da cooperativa e da biblioteca. Duas alterações geraram mutantes equivalentes aos originais, essas alterações modificaram o nome de um atributo no esquema, o que não é considerado um defeito. Esses mutantes foram descartados. Assim, pode ser concluído que todos os defeitos descritos pelos mutantes gerados foram cobertos pelas classes de defeitos da abordagem de teste de esquemas de dados.

As classes de defeitos relacionadas aos defeitos revelados são: Restrição de Domínio - Tipo de Dado Incorreto, Valor Incorreto, Tamanho Incorreto, Valores Máximos e Mínimos Incorretos; Restrição de Definição - Uso Incorreto, Unicidade Incorreta, Identificador Incorreto; e Restrição de Relacionamento - Ocorrência Incorreta e Associação Incorreta. A classe de defeito que mais revelou defeito foi a Restrição de Relacionamento - Associação Incorreta.

A avaliação de custo da aplicação da abordagem não foi o objetivo desse estudo de caso; no entanto, pôde ser observado que a XTool gera os dados de teste automaticamente e que o custo da aplicação da abordagem de teste está na análise dos resultados de teste, na qual o testador deve comparar os resultados obtidos com os resultados esperados. No estudo de caso, o resultado esperado foi o do esquema original e essa comparação foi feita automaticamente. Entretanto, nem sempre isso é possível.

Com base na análise dos resultados obtidos no estudo de caso, pode ser concluído que a abordagem de teste de esquemas de dados é eficaz em revelar defeitos em esquemas de banco de dados relacional. Isto porque foram inseridos 38 defeitos em dois esquemas de banco de dados relacional, 36 foram cobertos pelas classes de defeitos da abordagem de teste e 2 foram descartados (mutantes equivalentes).

6.3 Uma estratégia de aplicação da abordagem de teste

Os estudos de caso possibilitaram a observação de que existe uma relação entre as classes de defeitos quanto ao padrão de alterações, para gerar instâncias de dados alternativas, e o padrão de consultas, para gerar consultas capazes de revelar os defeitos representados nas instâncias de dados alternativas. Nesta seção, esta relação é explorada com o intuito de definir critérios de teste que auxiliem na aplicação da abordagem de teste AIDA.

6.3.1 Relação entre as classes de defeitos

Com base na análise dos resultados de teste obtidos com os estudos de caso apresentados neste capítulo e do padrão de alterações e consultas estabelecidos para as classes de defeitos definidas na abordagem, percebe-se por inspeção que existe uma relação entre as classes de defeitos. Essa relação mostra que algumas classes de defeitos, quanto ao padrão de alterações e de consultas, englobam outras classes. Assim sendo, a relação entre as classes de defeitos pode ser utilizada para estabelecer uma hierarquia entre as classes, que permita a redução do custo de aplicação da abordagem mantendo sua eficácia em revelar defeitos. A seguir a relação de hierarquia entre as classes de defeitos é definida:

Uma classe de defeito X é dita hierarquicamente superior (CDS) a uma classe de defeito Y com respeito a um elemento ou atributo x se:

- o conjunto de instâncias de dados alternativas I_X gerado pelas alterações definidas para a classe X contém instâncias alternativas que representam os mesmos defeitos representados pelas instâncias alternativas I_Y geradas pelas alterações especificadas na classe Y; e,
- as consultas q_X geradas pela classe X têm a mesma probabilidade de revelar os mesmos defeitos revelados pelas consultas q_Y geradas pela classe Y.

Na Tabela 6.15 é apresentada a relação de hierarquia entre classes de defeitos definidas na Seção 4.4, de acordo com as alterações e consultas especificadas para essas classes.

6.3.2 Critérios de teste baseados em associações de defeitos

Os critérios de teste definidos na abordagem são baseados nas associações de defeitos identificadas no esquema de dados e nas classes de defeitos. Esses critérios requerem que as associações de defeitos sejam exercitadas por meio da execução das consultas às instâncias de dados alternativas válidas. Considere z um elemento ou atributo.

Critério Todas Restrições - exige que todas as associações de defeitos com respeito a (c.r.a.) z relacionadas às classes de defeitos dos grupos de restrições de domínio, definição, relacionamento

Tabela 6.15: Relação entre as classes de defeitos

Classe de Defeito Superior (CDS)	**Classe de Defeito Inferior** (CDI)
(*G1 - SVI*) Restrição de Domínio - Seqüência de Valores Incorreta	(*G1 - TI*) Restrição de Domínio - Tamanho Incorreto
	(*G1 - TDI*) Restrição de Domínio - Tipo de Dado Incorreto
(*G1 - VI*) Restrição de Domínio - Valor Incorreto	(*G1 - VEI*) Restrição de Domínio - Valores Enumerados Incorretos
(*G1 - VMMI*) Restrição de Domínio - Valores Máximos e Mínimos Incorretos	(*G1 - DI*) Restrição de Domínio - Dígito Incorreto
(*G2 - II*) Restrição de Definição - Identificador Incorreto	(*G2 - UI*) Restrição de Definição - Uso Incorreto
	(*G2 - UNI*) Restrição de Definição - Unicidade Incorreta
(*G1 - CEBI*) Restrição de Domínio - Caracteres de Espaço em Branco Incorretos	–
(*G3 - OI*) Restrição de Relacionamento - Ordem Incorreta	–
(*G3 - ORI*) Restrição de Relacionamento - Ocorrência Incorreta	–
(*G3 - AI*) Restrição de Relacionamento - Associação Incorreta	–
(*G4 - COI*) Restrição Semântica - Condição Incorreta	–

e semântica sejam exercitadas, executando cada consulta às instâncias de dados alternativas relacionadas a essas associações.

Critério Todas Restrições de Domínio - requer que todas as associações de defeitos c.r.a. z relacionadas às classes de defeitos do grupo de restrições de domínio sejam exercitadas, exigindo que todas as consultas às instâncias de dados alternativas relacionadas a essas associações sejam executadas.

Critério Todas Restrições de Definição - requer que todas as associações de defeitos c.r.a. z

relacionadas às classes de defeitos do grupo de restrições de definição sejam exercitadas, exigindo que todas as consultas às instâncias de dados alternativas relacionadas a essas associações sejam executadas.

Critério Todas Restrições de Relacionamento - requer que todas as associações de defeitos c.r.a. z relacionadas às classes de defeitos do grupo de restrições de relacionamento sejam exercitadas, exigindo que todas as consultas às instâncias de dados alternativas relacionadas a essas associações sejam executadas.

Critério Todas Restrições Semânticas - requer que todas as associações de defeitos c.r.a. z relacionadas à classe de defeito do grupo de restrições semânticas sejam executadas, exigindo que todas as consultas às instâncias de dados alternativas relacionadas a essas associações sejam executadas.

Critério Todas Restrições/CDS - exige que pelo menos uma associação de defeito c.r.a z relacionada às classes de defeitos hierarquicamente superiores (conforme visto na Tabela 6.15) seja exercitada, se a associação existir, executando a consulta às instâncias de dados alternativas relacionadas a essa associação.

Critério Todos Grupos de Restrições - exige que pelo menos uma associação de defeito c.r.a. z relacionada a cada grupo de restrições de domínio, definição, relacionamento e semântica seja exercitada, se a associação existir, executando a consulta às instâncias de dados alternativas relacionadas a essa associação.

Existem indícios nos estudos de caso de que: o custo na aplicação da AIDA possa ser reduzido com o emprego dos critérios Todas Restrições/CDS e Todos Grupos de Restrições; isto porque esses critérios selecionam apenas um subconjunto de associações de defeitos para ser exercitado.

6.4 Considerações finais

Neste capítulo foi apresentada a ferramenta XTool desenvolvida para auxilar o teste de esquemas de dados. A XTool implementa a abordagem de teste baseada em defeitos Análise de Instâncias de Dados Alternativas, descrita neste trabalho. A ferramenta gera a representação formal do esquema de dados, identifica associações de defeitos entre componentes do esquema e classes de defeitos previamente definidas, gera instâncias de dados alternativas e consultas a essas instâncias com o intuito de revelar defeitos no esquema de dados.

Estudos de caso para avaliar a abordagem de teste de esquemas de dados foram realizados com o auxílio da ferramenta XTool e apresentados neste capítulo. Esses estudos de caso mostraram que a abordagem de teste é capaz de detectar defeitos em esquemas de dados e que o uso da ferramenta XTool é imprescindível para apoiar o processo de teste, permitindo que sejam testados esquemas mais complexos, em tempo reduzido e com mais facilidade. No entanto, existe um custo na apli-

cação da abordagem relacionado à análise dos resultados de teste por parte do testador. Além disso, as características dos esquemas de dados (número de elementos e de atributos) e das instâncias de dados originais (quantidade de dados) influenciam diretamente o custo e o esforço de aplicação da abordagem; isto porque em dependência dessas características são identificadas as associações de defeitos e gerados os dados de teste (instâncias de dados alternativas e consultas). Certamente que quanto maior a quantidade de elementos e atributos, maior a quantidade de associações de defeitos a serem exercitadas para a descoberta de defeitos no esquema. No entanto, a quantidade de dados nas instâncias de dados originais também afeta o custo de aplicação da abordagem, porque as instâncias alternativas são geradas de acordo com os dados da instância original; portanto, quanto maior a quantidade de dados na instância original, maior será a quantidade de resultados de teste que devem ser analisados pelo testador manualmente.

Segundo a estratégia de teste, pode ser dito que o critério de teste Todas Restrições foi utilizado em todos os estudos de caso para selecionar as associações de defeitos que deveriam ser exercitadas. Novos estudos de caso podem ser realizados para avaliar o uso dos outros critérios de teste (tais como: todas restrições/CDS e todos grupos de restrições) e sua relação com a eficácia e o custo de aplicação da AIDA.

É importante ressaltar que a abordagem de teste contribui para a qualidade de aplicações de software que envolvem esquemas de dados, com a vantagem de que a geração dos dados de teste, formados por instâncias de dados e consultas a essas instâncias, pode ser feita automaticamente a partir dos padrões definidos nas classes de defeitos. Esses dados de teste, também podem ser empregados no teste das aplicações que manipulam os dados definidos no esquema em teste. Além disso, a AIDA pode ser 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 do teste.

Capítulo 7

Conclusões e Trabalhos Futuros

7.1 Síntese do trabalho

Este trabalho propõe uma abordagem de teste baseada em defeitos para esquemas de dados, denominada Análise de Instâncias de Dados Alternativas (AIDA), cuja meta é obter alta confiabilidade e assegurar a integridade dos dados definidos por esquemas e manipulados por aplicações de software para evitar falhas nessas aplicações.

Essa abordagem é genérica, emprega um modelo de dados baseado em um metamodelo MM introduzido por meio da especificação MOF para representar os esquemas de dados, e pode ser aplicada em esquemas de diferentes contextos que possam ser representados como uma instância do metamodelo MM. Além do modelo de dados, na abordagem é definida uma representação formal para o esquema, permitindo que o esquema seja processado algoritmicamente.

A AIDA é baseada em defeitos, classes de defeitos que descrevem defeitos comuns introduzidos no desenvolvimento ou evolução de esquemas são identificadas a partir de restrições aos dados do esquema definidas no metamodelo MM. As classes de defeitos guiam a geração de instâncias de dados alternativas e de consultas produzidas de acordo com padrões definidos para cada classe. As instâncias de dados representam os possíveis defeitos no esquema e as consultas são capazes de revelar esses defeitos.

Os defeitos que podem ser revelados pela abordagem estão relacionados à definição incorreta ou ausente de restrições aos dados no esquema. A idéia é evitar que dados incorretos possam ser considerados válidos ou que dados corretos possam ser considerados inválidos devido a um defeito na definição dos dados no esquema.

A AIDA pode ser empregada no contexto de XML para testar esquemas de: documentos XML que armazenam dados como uma base de dados, mensagens XML que são usadas para troca de informação em aplicações Web, resultados de consultas a bases de dados em formato XML, documentos

XML atualizados devido a alterações na especificação dos dados armazenados nesses documentos; e no contexto de esquemas de base de dados, para testar esquemas de base de dados relacional. Além disso, as instâncias de dados alternativas geradas na abordagem podem ser usadas, no contexto de XML, para testar aplicações que manipulam documentos em formato XML, interação entre componentes, bem como serviços Web; e no contexto de base de dados, para testar as aplicações de base de dados.

O desenvolvimento de uma ferramenta, denominada XTool [54], foi essencial para o uso dessa abordagem de teste, porque a aplicação manual da abordagem é custosa e poderia ser propensa a erros. No contexto de XML, a ferramenta providencia o teste de esquemas escritos em XML *Schema*, e no contexto de base de dados, a XTool apóia o teste de esquema de base de dados relacional em PostGreSQL *Schema*.

Estudos de caso foram realizados para validar a AIDA bem como para avaliar a eficácia e o custo de sua aplicação. O primeiro estudo de caso foi feito com esquemas XML obtidos do desenvolvimento de aplicações Web por parte de alunos de mestrado. Esse estudo de caso foi realizado para avaliar o custo e a eficácia da abordagem de teste em detectar defeitos. O segundo estudo de caso usou esquemas obtidos de diagramas produzidos pela aplicação hyperModel [7]. O objetivo desse estudo de caso foi avaliar a eficácia da abordagem de teste com respeito a defeitos descritos por alguns operadores de mutação introduzidos por Franzotte e Vergilio [32]. O terceiro estudo de caso envolveu um esquema de base de dados relacional e teve por objetivo avaliar a aplicabilidade e a eficácia da abordagem no contexto de base de dados. O quarto estudo de caso foi realizado em esquemas de bases de dados obtidos de Ruiz [65] e de Ferreira [30]. O objetivo foi avaliar a eficácia da abordagem de teste em revelar defeitos inseridos de maneira ad hoc em esquemas associados a bases de dados relacionais.

Os resultados dos estudos de caso permitem que sejam feitas algumas considerações em relação ao esforço, custo e eficácia da abordagem de teste:

- O maior custo está relacionado à análise dos resultados pelo testador, que compara o resultado do teste (resultado obtido) com o resultado esperado (conforme a especificação dos dados). Vale a pena ressaltar que esse é um problema de difícil solução inerente à atividade de teste relacionado à questão de automatização do oráculo, que é uma questão indecidível similar ao problema da parada [36];

- Ainda com relação ao custo e esforço de aplicação da abordagem, foi observado que as características dos esquemas de dados (número de elementos e atributos) e das instâncias de dados originais (quantidade de dados) afetam diretamente esses fatores, porque em dependência dessas características são identificadas as associações de defeitos e geradas as instâncias de

dados alternativas e as consultas a essas instâncias. Entre essas caracterísicas, é importante ressaltar a quantidade de dados da instância de dados original, que é uma característica determinante para aumentar o custo de aplicação da AIDA; isto porque, quanto maior a quantidade de dados na instância original, maior será a quantidade de resultados de teste que devem ser analisados manualmente pelo testador;

- Em relação à eficácia, a AIDA é capaz de revelar os defeitos cobertos pelas classes de defeitos em esquemas XML e em esquemas de bases de dados relacional;

- Quanto à avaliação de eficácia da abordagem de teste em relação aos defeitos descritos por alguns operadores de mutação ou por defeitos semeados de maneira ad hoc (sem o conhecimento das classes de defeitos), os casos de teste gerados pela abordagem, com o apoio da XTool, revelaram todos os defeitos, produzidos por operadores de mutação ou não, empregados nos estudos de caso.

Além dessas considerações, os resultados dos estudos de caso permitiram a visualização de uma estratégia de aplicação da abordagem de teste; pôde ser observado que existe uma relação de hierarquia entre as classes de defeitos no que diz respeito aos padrões de alterações e de consultas especificados para cada classe. Essa hierarquia possibilita a definição de critérios de teste para auxiliar a seleção dos elementos requeridos a serem exercitados no teste. Os elementos requeridos são as associações de defeitos, formadas por elementos ou atributos do esquema associados a uma classe de defeito. Esses elementos são exercitados por meio das consultas às instâncias de dados alternativas.

Uma vantagem importante da abordagem de teste é a geração automática de dados de teste a partir das classes de defeitos e de uma instância de dados válida para o esquema em teste. Além disso, essa abordagem pode testar esquemas mesmo que a aplicação de software não esteja disponível. No entanto, é necessário que a especificação dos dados da aplicação esteja disponível para a análise dos resultados do teste.

Pode ser apontada como uma limitação do trabalho não ter sido feita uma comparação dos resultados obtidos da aplicação da AIDA com resultados obtidos pelo teste tradicional da aplicação; por exemplo, com o teste funcional, para verificar se os mesmos defeitos são detectados ou não. No entanto, essa questão pode ser investigada em trabalhos futuros.

7.2 Contribuições

A principal contribuição deste trabalho é a abordagem de teste baseada em defeitos para esquemas de dados, denominada Análise de Instâncias de Dados Alternativas (AIDA), que auxilia na obtenção

de um alto nível de qualidade em aplicações de software que manipulam dados definidos por esquemas. Geralmente os dados dessas aplicações são validados de acordo com o esquema; no entanto, essa validação não é suficiente para garantir a qualidade dos dados, pois a definição dos dados no esquema pode estar incorreta em relação à especificação e, dessa forma, a aplicação pode manipular dados validados que estão incorretos e uma falha pode ser produzida, ou seja, o esquema incorreto permite que dados incorretos sejam validados ou, o contrário, que dados corretos sejam invalidados.

Existem outras contribuições inerentes à proposição da abordagem de teste, como a definição das classes de defeitos para esquemas de dados, que foram identificadas a partir de defeitos comuns que podem ser introduzidos em esquemas durante o seu desenvolvimento ou sua evolução (atualização). Esses defeitos estão relacionados às restrições aos dados definidos no esquema e identificadas no metamodelo MM. As classes podem ser usadas para revelar defeitos na definição incorreta ou ausente de restrições no esquema.

A definição do metamodelo de dados MM, para representar os esquemas de dados que podem ser verificados pela abordagem e identificar as restrições que podem ser exploradas no teste de esquemas, é outra contribuição do trabalho. Além disso, foram especificadas na abordagem de teste: a representação formal para identificar elementos, atributos e restrições aos dados definidos nos esquemas, permitindo que o esquema seja processado algoritmicamente; o padrão de alteração para gerar instâncias de dados alternativas a partir de uma instância de dados original associada ao esquema em teste, de modo que essas instâncias representam possíveis defeitos no esquema de dados; o padrão de consultas para determinar quais dados a consulta deve retornar para que os defeitos nas instâncias de dados alternativas possam ser revelados; e o conceito de associação de defeito para indicar os elementos requeridos a serem exercitados no teste.

Como resultado da realização dos estudos de caso foram propostos critérios de teste baseados nas associações de defeitos e em uma relação de hierarquia entre as classes de defeitos. Esses critérios possibilitam a sistematização do teste, auxiliando a seleção das associações de defeitos que devem ser exercitadas no processo de teste.

Por fim, também pode ser considerada uma contribuição deste trabalho de pesquisa a especificação da ferramenta XTool, que foi desenvolvida por Nazar [54] para apoiar a aplicação da abordagem de teste.

7.3 Trabalhos futuros

Existem muitas questões que ainda podem ser estudadas em relação à abordagem de teste proposta. Algumas dessas questões são abordadas a seguir:

- Explorar o uso da AIDA em outros contextos, como no contexto de serviços Web, bem como

em outras linguagens de definição de esquema para XML.

- Investigar a aplicabilidade da AIDA em relação a outros modelos, tal como um diagrama conceitual UML.
- Comparar os resultados obtidos da aplicação da AIDA com resultados obtidos pelo teste tradicional da aplicação, para verificar se os mesmos defeitos são detectados ou não.
- Implementar outras funcionalidades na XTool; por exemplo, o teste de outros tipos de esquemas de dados e suporte ao oráculo para facilitar a comparação dos resultados obtidos e esperados, mesmo que não seja possível realizar completamente a análise de resultados pela ferramenta.
- Investigar o uso das instâncias de dados alternativas geradas pela abordagem para o teste das aplicações que utilizam o esquema em teste, seguindo a idéia da perturbação de dados [55]. No contexto de XML, testar aplicações que manipulam documentos em formato XML, a interação entre componentes Web; e no contexto de base de dados, testar as aplicações de base de dados.
- Efetuar outros estudos de caso para avaliar o uso dos critérios de teste Todas Restrições, Todas Restrições/CDS e Todos Grupos de Restrições em relação ao custo e eficácia da abordagem.
- Realizar estudos de caso com esquemas de dados mais complexos e com o uso de instâncias de dados originais que diferem em relação à quantidade de dados armazenados, para analisar a relação entre a quantidade de dados e os fatores: eficácia e custo da abordagem.

As quatro primeiras questões mencionadas acima poderiam ser realizadas em trabalhos de mestrado porque seria preciso um estudo mais amplo para: adaptar a abordagem em outros contextos ou modelos, já que poderia ser necessário estender as classes de defeitos para o contexto ou modelo investigado; investigar se os resultados do teste tradicional de uma aplicação seriam os mesmos obtidos com a AIDA; e implementar outras funcionalidades na XTool, tal como suporte ao oráculo, que poderia ser obtido por meio da especificação do resultado esperado em uma linguagem formal. As outras questões poderiam ser abordadas em trabalhos de iniciação científica, porque exigem que sejam realizados novos estudos de caso para investigar o uso das instâncias de dados alternativas para testar aplicações de software que utilizam esquemas na definição dos dados, avaliar o emprego dos critérios de teste propostos em relação ao custo e eficácia da abordagem, explorar o uso de instâncias de dados originais com diferentes quantidades de dados armazenados em relação ao custo e eficácia da abordagem.

Referências Bibliográficas

[1] M. C. L. F. M. Aranha, N. C. Mendes, M. Jino, and C. M. T. Toledo. RDBTool: Uma Ferramenta de Apoio ao Teste de Bases de Dados Relacionais. In *XI CITS: Conferência Internacional de Tecnologia de Software*. agosto 2000.

[2] C. Atkinson. Meta-Modeling for Distributed Object Environments. IEEE Computer Society, 1997.

[3] Per Bothner. Qexo - The GNU Kawa implementation of XQuery. http://www.gnu.org/software/qexo/, acessado em 2005, 2005.

[4] T. A. Bruce. *Designing Quality Databases with IDEF1X Information Models*. Dorset House, New York, 1992.

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

[6] D. Carlson. *Modeling XML Application with UML - Pratical e-Business Applications*. Addison-Wesley, 2^{nd} edition, 2001.

[7] D. Carlson. HyperModel Application. http://www. xmlmodeling.com/models/index.html , acessado em 2006, 2006.

[8] P. V. Caxeiro. Utilizando uma Abordagem Baseada em Defeitos para o Teste de Esquemas de Bases de Dados: Resultados de um Estudo de Caso com a Ferramenta XTool. Trabalho de graduação, Departamento de Informática, Setor de Ciências Exatas, Universidade Federal do Paraná, julho 2007.

[9] M. L. Chaim. POKE-TOOL - Uma Ferramenta para Suporte ao Teste Estrutural de Programas Baseados em Análise de Fluxo de Dados. Tese de mestrado, Departamento de Engenharia de Computação e Automação Industrial, Faculdade de Engenharia Elétrica e de Computação, UNICAMP, abril 1991.

[10] M. Chan and S. Cheung. Testing Database Applications with SQL Semantics. In *Proceedings of the 2^{nd} Intl. Symp. on Cooperative Database Systems for Advanced Applications (CODAS'99)*, pages 364–375, March 1999.

[11] W. K. Chan, S. C. Cheung, and T. H. Tse. Fault-Based Testing of Database Application Programs with Conceptual Data Model. In *Proceedings of the 5^{th} International Conference on Quality Software*, pages 187–196, 2005.

[12] D. Chays, S. Dan, P. G. Frankl, F. I. Vokolos, and E. J. Weyuker. A Framework for Testing Database Applications. In *Proceedings of the 2000 ACM SIGSOFT International Symposium on Software Testing and Analysis ISSTA '00*, volume 25, August 2000.

[13] D. Chays and Y. Deng. Demonstration of AGENDA Tool Set for Testing Relational Database Applications. In *Proceedings of the 25^{th} International Software Engineering Conference - IEEE Computer Society*, pages 802–803, May 2003.

[14] D. Chays, Y. Deng, P. G. Frankl, S. Dan, F. I. Vokolos, and E. J. Weyuker. AGENDA: A Test Generator for Relational Database Applications. Technical report, Department of Computer Science, Polytechnic University, Brooklyn, Long Island, Westchester, August 2002.

[15] P. P. Chen. The Entity-Relationship Model - Toward a Unified View of Data. *ACM Transactions on Database Systems*, 1(1):9–36, 1976.

[16] E. F. Codd. A Relational Model for Large shared Data Banks. *Communications of the ACM*, 13(6):377–387, 1970.

[17] J. Conallen. *Building Web Application with UML*. Addison-Wesley, 2002.

[18] R. A. De Millo. Hints on Test Data Selection: Help for the Practicing Programmer. *IEEE Computer*, April 1978.

[19] R. A. De Millo. Mutation Analysis as a Tool for Software Quality. In *Proceedings of the Annual International Computer Software and Applications Conference, COMPSAC 80*, October 1980.

[20] R. A. De Millo, D. C. Gwind, and K. N. King. An Extented Overview of the Mothra Software Testing Environment. In *Proceedings of the 2^{nd} Workshop on Software Testing, Verification and Analysis*, pages 142–151, July 1988.

[21] M. E. Delamaro. Proteum - Um Ambiente de Teste Baseado na Análise de Mutantes. Dissertação de mestrado, ICMSC/USP, outubro 1993.

[22] Y. Deng, P. Frankl, and D. Chays. Testing Database Transactions with AGENDA. In *Proceedings of the* 27^{th} *International Conference on Software Engineering - ACM Press*, May 2005.

[23] G.A. Di Lucca and M. Di Penta. Considering Browser Interaction in Web Application Testing. In *Proceedings of the* 5^{th} *IEEE Intl. Workshop on Web Site Evolution*. IEEE Computer Society Press, 2003.

[24] G.A. Di Lucca, A.R. Fasolino, F. Faralli, and U. De Carlini. Testing Web applications. In *Proceedings of the International Conference on Software Maintenance*, pages 3–6. IEEE Press, October 2002.

[25] G.A. Di Lucca, A.R. Fasolino, F. Pace, P. Tramontana, and U. De Carlini. WARE: A tool for the Reverse Engineering of Web Applications. In *Proceedings of the VI Eur. Conf. on Software Maintenance and Reengineering*. IEEE Computer Society Press, March 2002.

[26] M. C. F. P. Emer, I. F. Nazar, S. R. Vergilio, and M. Jino. Evaluating a Fault-Based Testing Approach for XML Schemas. In *Proceedings of the* 8^{th} *IEEE Latin-American Test Workshop, LATW 2007*. March 2007.

[27] M. C. F. P. Emer, S. R. Vergilio, and M. Jino. A Testing Approach for XML Schemas. In *Proceedings of the* 29^{th} *Annual International Computer Software and Applications Conference, COMPSAC 2005 - QATWBA 2005*. IEEE Press, July 2005.

[28] S. C. P. F. Fabbri, J. C. Maldonado, M. E. Delamaro, and P. C. Masiero. Proteum/FSM – uma ferramenta para apoiar a validação de máquinas de estado finito pelo critério análise de mutantes. In *IX Simpósio Brasileiro de Engenharia de Software*, pages 475–478, outubro 1995.

[29] S. C. P. F. Fabbri, J. C. Maldonado, and P. C. Masiero. Aplicação do Critério Análise de Mutantes na Validação de Especificações Baseadas em Statecharts. In *XI Simpósio Brasileiro de Engenharia de Software*, outubro 1997.

[30] M. R. Ferreira. Projeto Biblos - Proposta de Banco de Dados. Monografia de especialização, Departamento de Ciência da Computação/Universidade Federal de Lavras, abril 2004.

[31] F. G. Frankl. The use of Data Flow Information for the Selection and Evaluation of Software Test Data. Phd thesis, Department of Computer Science, New York University, October 1987.

[32] L. Franzotte and S. R. Vergilio. Applying Mutation Testing to XML Schemas. In *Proceedings of the* 18^{th} *Intl. Conference on Software Engineering and Knowledge Engineering, SEKE 2006*, July 2006.

[33] A. Gerber and K. Raymond. MOF to EMF: there and back again. In *Proceedings of the 2003 OOPSLA - Workshop on Eclipse Technology EXchange*, October 2003.

[34] P. Hnetynka and F. Plásil. Distributed Versioning Model for MOF. In *Proceedings of the Winter International Symposium on Information and Communication Technologies, ACM International Conference Proceeding Series*, volume 58, pages 1–6, 2004.

[35] J. R. Horgan and S. London. ATAC - Automatic Test Coverage Analysis for C Programs. Technical report, Bellcore Internal Memorandum, June 1990.

[36] W. E. Howden. Methodology for the generation of program test data. *IEEE Transactions on Software Engineering*, SE-24(5):554–559, May 1975.

[37] W. E. Howden. Weak Mutation Testing and Completeness of Test Sets. *IEEE Transactions on Software Engineering*, 8(4):371–379, July 1982.

[38] IEEE. IEEE Standard for Conceptual Modeling Language Syntax and Semantics for IDEF1X. New York: IEEE, 1998.

[39] G. M. Kapfhammer and M. L. Soffa. A Family of Test Adequacy Criteria for Database-driven Applications. In *Proceedings of the* 9^{th} *European Software Engineering Conference held jointly with* 11^{th} *ACM SIGSOFT International Symposium on Foundations of Software Engineering ESEC/FSE-11*, volume 28, September 2003.

[40] H. Katz. *XQuery from the Experts-Guide to the W3C XML Query Language*. Addison-Wesley, 2003.

[41] D.C. Kung, C.H. Liu, and P. Hsia. An Object-Oriented Web Test Model for Testing Web Applications. In *Proceedings of the* 24^{th} *Annual International Computer Software and Applications Conference, COMPSAC 2000*, pages 537–542. IEEE Press, 2000.

[42] S.C. Lee and J. Offutt. Generating test cases for XML-based web component interactions using mutation analysis. In *Proceedings of the* 12^{th} *International Symposium on Software Reliability Engineering*, pages 200–209. IEEE Press, November 2001.

[43] P. S. J. Leitão, P. R. S. Vilela, and M. Jino. Mapping Faults to Failures in SQL Manipulation Commands. In *Proceedings of the* 3^{rd} *ACS/IEEE International Conference on Computer Systems and Applications (AICCSA-05)*, January 2005.

[44] J. B. Li and J. Miller. Testing the Semantics of W3C XML Schema. In *Proceedings of the* 29^{th} *Annual International Computer Software and Applications Conference, COMPSAC 2005*. IEEE Press, July 2005.

[45] C.H. Liu, D.C. Kung, and P. Hsia. Object-based Data Flow Testing of Web Applications. In *Proceedings of the* 1^{st} *Asia-Pacific Conference on Quality Software*, pages 7–16. IEEE Press, 2000.

[46] C.H. Liu, D.C. Kung, P. Hsia, and C.T. Hsu. Structural Testing of Web Applications. In *Proceedings of the* 11^{th} *International Symposium on Software Reliability Engineering*, pages 84–96. IEEE Press, 2000.

[47] J. C. Maldonado. Critérios Potenciais Usos: Uma Contribuição ao Teste Estrutural de Software. Tese de doutorado, Departamento de Engenharia de Computação e Automação Industrial, Faculdade de Engenharia Elétrica e de Computação, UNICAMP, julho 1991.

[48] J. C. Maldonado, M. Chaim, and M. Jino. Briding the gap in the presence of infeasible paths: Potencial uses testing criteria. In *Proceedings of the XII Intl. Conference of The Chilean Science Computer Society*, pages 323–340, October 1992.

[49] J. C. Maldonado, A. M. R. Vincenzi, E. F. Barbosa, S. do R. S. de Souza, and M. E. Delamaro. Aspectos Teóricos e Empíricos de Teste de Cobertura de Software. Notas Didáticas 31, Instituto de Ciências Matemáticas de São Carlos, junho 1998.

[50] A. P. Mathur and W. E. Wong. An empirical comparison of data flow and mutation-based test adequacy criteria. *The Journal of Software Testing, Verification and Reliability*, 4(1):9–31, March 1994.

[51] T. McCabe. A Software Complexity Measure. *IEEE Transactions on Software Engineering*, 2:308–320, December 1976.

[52] L. J. Morell. A Theory of Fault-based Testing. *IEEE Transactions on Software Engineering*, 16(8):844–857, August 1990.

[53] G. Myers. *The Art of Software*. Wiley, New York, 1979.

[54] I. F. Nazar. XTool - Uma Ferramenta para Teste de Esquemas de Estruturas de Dados. Dissertação de mestrado, Departamento de Informática, Setor de Ciências Exatas, Universidade Federal do Paraná, março 2007.

[55] J. Offutt and W. Xu. Generating Test Cases for Web Services Using Data Perturbation. In *Proceedings of the TAV-WEB*, volume 29. ACM SIGSOFT SEN, September 2004.

[56] OMG. MetaObject Facility (MOF) Specification version 1.4. http://www.omg.org/docs/formal/02-04-03.pdf, acessado em 2006, April 2002.

[57] OMG. MetaObject Facility Core Specification version 2.0. http://www.omg.org/cgi-bin/doc?formal/2006-01-01, acessado em 2006, January 2006.

[58] OMG. Object Constraint Language - OMG Available Specification version 2.0. http://www.omg.org/docs/formal/06-05-01.pdf, acessado em 2006, May 2006.

[59] PostGreSQL Global Development Group. PostGreSQL. http://www.postgresql.org, acessado em 2006, 2006.

[60] R. S. Pressman. *Software Engineering*. McGraw-Hill, New York, 6^{th} edition, 2006.

[61] S. Rapps and E. Weyuker. Data Flow analysis techniques for test data selection. In *Proceedings of the International Conference on Software Engineering*, September 1982.

[62] S. Rapps and E. J. Weyuker. Selecting Software Test Data Using Data Flow Information. *IEEE Transactions on Software Engineering*, 11(4), April 1985.

[63] F. Ricca and P. Tonella. Analysis and Testing of Web Applications. In *Proceedings of the* 23^{rd} *International Conference on Software Maintenance*, pages 25–34. IEEE Press, May 2002.

[64] M. A. Robbert and F. J. Maryanski. Automated Test Plan Generator for Database Application Systems. In *Proceedings of the ACM SIGSAMLL/PC Symposium on Small Systems*, pages 100–106, 1991.

[65] D. D. Ruiz. Transformação de modelos conceituais em modelos de implementação de bancos de dados. http://www.sbbd-sbes2005.ufu.br/arquivos/MiniCursoSBBD2005_Mapeamento.pps, outubro 2005.

[66] F. Ruiz, A. Vizcaino, F. Garcia, and M. Piattini. Using XMI and MOF for representation and interchange of software process. In *Proceedings of the* 14^{th} *International Workshop*, pages 739–744, September 2003.

[67] A. Silberschatz, H. F. Korth, and S. Sudarshan. *Database System Concepts*. McGraw-Hill, 5^{th} edition, 2005.

[68] J. B. Silva. Proteste+: Ambiente de Validação Automática de Qualidade de Software através de Técnicas de Teste e de Métricas de Complexidade. Tese de mestrado, CPGCC-UFRGS, 1995.

[69] A. S. Simão and J. C. Maldonado. Proteum-RS/PN: Uma Ferramenta para Apoiar a Edição, Simulação e Validação de Redes Petri Baseada no Teste de Mutação. In *XIV Simpósio Brasileiro de Engenharia de Software*, outubro 2000.

[70] E. S. Spoto. Critério de Teste Estrutural para Programas de Aplicação de Base de Dados Relacional. Tese de doutorado, Departamento de Engenharia de Computação e Automação Industrial, Faculdade de Engenharia Elétrica e de Computação, UNICAMP, dezembro 2000.

[71] SUN. Java Database Connectivity (JDBC). http://java.sun.com /javase/technologies/database/, acessado em 2006, 2006.

[72] M. J. Suárez-Cabal and J. Tuya. Using a SQL Coverage Measurement for Testing Database Applications. In *Proceedings of the* 12^{th} *Intl. Symp. on the Foundations of Engineering*, November 2004.

[73] E. Tittel. *Schaum's Outline of Theory and Problems of XML*. McGraw-Hill, 2002.

[74] S. R. Vergilio, J. C. Maldonado, and M. Jino. Infeasible paths within the context of data flow based criteria. In *Proceedings of the VI Intl. Conference on Software Quality*, pages 310–321, October 1996.

[75] G. Vossen. *Data models, database languages and database management systems*. Addison-Wesley, 1991.

[76] W3C. Extensible Markup Language (XML) 1.0 (Third Edition)- W3C recommendation. www.w3.org/TR/2004/REC-xml-20040204/, acessado em 2005, February 2004.

[77] W3C. XML Schema, Working Draft. www.w3.org/XML/Schema, acessado em 2005, July 2004.

[78] W3C. DOM - Document Object Model. http://www.w3.org/DOM, acessado em 2005, 2005.

[79] W3C. XML Query Language (XQuery). http://www.w3c.org/XML/Query/, acessado em 2005, 2005.

[80] W3C. Validator for XML Schema. http://www.w3.org/2001/03/webdata/xsv, acessado em 2006, 2006.

[81] E. Weyuker, S. N. Weiss, and Hamlet. Comparison of program testing strategies. In *Proceedings of the* 4^{th} *Symposium on Software Testing, Analysis and Verification*, pages 154–164, 1991.

[82] W. E. Wong. On Mutation and Data Flow. Phd thesis, Department of Computer Science, Purdue University, December 1993.

[83] W. Xu, J. Offutt, and J. Luo. Testing Web Services by XML Pertubation. In *Proceedings of the* 16^{th} *IEEE International Symposium on Software Reliability Engineering*. IEEE, 2005.

[84] J. Zhang, C. Xu, and S. C. Cheung. Automatic Generation of Database Instances for White-box Testing. In *Proceedings of the* 25^{th} *Annual International Computer Software and Applications Conference, 2001. COMPSAC 2001*, pages 161 – 165, October 2001.

Apêndice A

OCL

Nesse apêndice, a sintaxe da linguagem de restrição de Objeto - OCL (*Object Constraint Language*) utilizada na definição do metamodelo para esquemas de dados é brevemente descrita.

A.1 Alguns Conceitos de OCL

Um diagrama UML não é refinado o suficiente para representar todos os aspectos de uma especificação. Muitas vezes existe a necessidade de escrever restrições adicionais sobre os objetos no modelo. Essas restrições geralmente são escritas em linguagem natural, mas isso não é o mais apropriado, pois pode gerar ambigüidades. A OCL [58] foi desenvolvida para tentar resolver esse problema.

A OCL é uma linguagem para descrever expressões e restrições em modelos orientados a objetos. Uma expressão é uma especificação de um valor e uma restrição é uma limitação em um ou mais valores de um modelo orientado a objeto. A OCL é uma linguagem de especificação, não é uma linguagem de programação, ou seja, não é possível escrever lógica de programa ou fluxo de controle com OCL.

Os tipos de restrições de OCL, são: invariantes de classe, pré e pós-condições de operações, e guardas de transições. A restrição do tipo invariante de classe é usada para a definição do metamodelo para esquemas de dados e é abordada a seguir.

Invariante

Uma invariante é uma condição que se aplica a todas as instâncias de uma classe, tipo ou interface. As invariantes devem ser verdadeiras sempre.

O contexto de uma expressão OCL especifica o elemento (classe, interface, ...) do modelo para o qual a expressão é definida. O contexto de uma invariante é definido por um objeto que pode ser

referenciado por *self* ou pode ser explicitamente nomeado.

Uma restrição invariante tem a seguinte sintaxe:

context <contexto>

inv:<expressão>

A Figura A.1 ilustra um diagrama de classes, que é usado nos exemplos de invariantes.

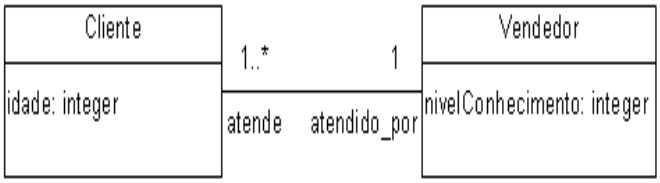

Figura A.1: Diagrama de classes usado para exemplificar invariantes

Exemplo 1: Invariantes em atributos - o valor do atributo idade da classe Cliente possui uma restrição.

context Cliente

inv: self.idade >= 18

ou

context c.Cliente

inv: c.idade >= 18

Exemplo 2: Invariantes em associações - o valor do atributo nivelConhecimento da instância associada da classe Vendedor é restrito por uma condição que deve ser satisfeita sempre.

context Cliente

inv: atendido_por.nivelconhecimento >= 5

Apêndice B

Estudo de Caso I

Neste apêndice são disponibilizados os esquemas e alguns resultados de teste referentes ao Estudo de Caso I.

B.1 Especificação dos dados, esquemas e documentos XML dos sistemas de Matrícula e de Biblioteca

Especificação dos dados

Sistema de Matrícula

Descrição do Sistema: O aluno poderá acessar o sistema com login 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 contida no Sistema de Matrícula:
o Aluno: código, nome, login, senha, curso, período, disciplinas cursadas, disciplinas matriculadas;
o 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.

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 contida no Sistema de Biblioteca:
o Obras: código, título, descrição, conteúdo, autor, editora, local, ano, ISBN.
o 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.

Esquemas XML do Sistema de Matrícula

---

aluno.xsd

```
<?xml version="1.0" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified">
<xs:element name="alunos">
<xs:complexType>
<xs:sequence>
   <xs:element name="aluno" maxOccurs="unbounded">
   <xs:complexType>
   <xs:sequence>
 <xs:element name="codigo" type="xs:string"
              minOccurs="1"/>
 <xs:element name="nome" type="xs:string" minOccurs="1"/>
 <xs:element name="login" minOccurs="1" maxOccurs="1"/>
 <xs:element name="senha" type="xs:string" minOccurs="1"/>
 <xs:element name="curso" type="xs:string" minOccurs="1"/>
 <xs:element name="periodo" minOccurs="1" maxOccurs="1"/>
 <xs:element name="disciplina_cursadas" type="xs:string"
              minOccurs="0" maxOccurs="unbounded"/>
 <xs:element name="disciplina_matriculadas" type="xs:string"
```

```
              minOccurs="1" maxOccurs="unbounded"/>
   </xs:sequence>
   </xs:complexType>
   </xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
```

---

disciplina.xsd

```
<?xml version="1.0" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified">
<xs:element name="disciplinas">
<xs:complexType>
<xs:sequence>
  <xs:element name="disciplina" maxOccurs = "unbounded">
  <xs:complexType>
  <xs:sequence>
 <xs:element name="codigo" type="xs:string" minOccurs="1"/>
 <xs:element name="nome" type="xs:string" minOccurs="1"/>
 <xs:element name="carga_horaria" type="xs:string" minOccurs="1"/>
 <xs:element name="creditos" type="xs:string" minOccurs="1"/>
 <xs:element name="periodo" type="xs:string" minOccurs="1"/>
 <xs:element name="ementa" type="xs:string" minOccurs="1" />
 <xs:element name="codigo_professor" minOccurs="0" maxOccurs="1"/>
      <xs:element name="codigo_prerequisito" minOccurs="0"
                  maxOccurs="unbounded"/>
    </xs:sequence>
    </xs:complexType>
    </xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
```

Esquemas XML do Sistema de Biblioteca

---

obras.xsd

```
<?xml version="1.0" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified">
<xs:element name="obras">
<xs:complexType>
<xs:sequence>
   <xs:element name="obra" maxOccurs="unbounded">
   <xs:complexType>
   <xs:sequence>
 <xs:element name="codigo" type="xs:string" minOccurs="1"
              maxOccurs="1"/>
 <xs:element name="titulo" type="xs:string" minOccurs="1"
              maxOccurs="1"/>
 <xs:element name="descricao" minOccurs="1" maxOccurs="1"/>
 <xs:element name="conteudo" type="xs:string" minOccurs="1"
              maxOccurs="1"/>
 <xs:element name="autores">
 <xs:complexType>
  <xs:sequence>
    <xs:element name="nome" type="xs:string" minOccurs="1"
                maxOccurs="unbounded"/>
   </xs:sequence>
  </xs:complexType>
  </xs:element>
 <xs:element name="editora" minOccurs="1" maxOccurs="1"/>
 <xs:element name="local" minOccurs="1"  maxOccurs="1"/>
 <xs:element name="ano" type="xs:integer" minOccurs="1"
              maxOccurs="1"/>
   </xs:sequence>
   </xs:complexType>
   </xs:element>
```

```
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
```

---

usuario.xsd

```
<?xml version="1.0" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified">
<xs:element name="usuarios">
<xs:complexType>
<xs:sequence>
   <xs:element name="usuario" maxOccurs="unbounded">
   <xs:complexType>
   <xs:sequence>
      <xs:element name="codigo" type="xs:string" minOccurs="1"
                  maxOccurs="1"/>
 <xs:element name="nome" type="xs:string" minOccurs="1"
              maxOccurs="1"/>
 <xs:element name="login" minOccurs="1" maxOccurs="1"/>
 <xs:element name="senha" type="xs:string" minOccurs="1"
              maxOccurs="1"/>
 <xs:element name="pag_mensalidade" type="xs:string"
              minOccurs="1" maxOccurs="1"/>
 <xs:element name="qdade_acesso" minOccurs="1" maxOccurs="1"/>
   </xs:sequence>
   </xs:complexType>
   </xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
```

Fragmento exemplo de instância de dados original - documento XML do Sistema de Matrícula

---

aluno.xml

```
<?xml version="1.0" ?>
  <alunos xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="aluno.xsd">
    <aluno>
      <codigo>000001</codigo>
      <nome>Joao Augusto</nome>
      <login>joaoaugu</login>
      <senha>jjj21*</senha>
      <curso>Informatica</curso>
      <periodo>2</periodo>
      <disciplina_cursadas>001</disciplina_cursadas>
      <disciplina_cursadas>003</disciplina_cursadas>
      <disciplina_matriculadas>002</disciplina_matriculadas>
      <disciplina_matriculadas>004</disciplina_matriculadas>
    </aluno>
  ...
  ...
    <aluno>
      <codigo>000002</codigo>
      <nome>Maria Augusta</nome>
      <login>mariaugu</login>
      <senha>mmm42%</senha>
      <curso>Informatica</curso>
      <periodo>4</periodo>
      <disciplina_cursadas>001</disciplina_cursadas>
      <disciplina_cursadas>003</disciplina_cursadas>
      <disciplina_cursadas>002</disciplina_cursadas>
      <disciplina_cursadas>004</disciplina_cursadas>
      <disciplina_matriculadas>005</disciplina_matriculadas>
    </aluno>
  </alunos>
```

---

disciplina.xml

```
<?xml version="1.0" ?>
<disciplinas xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="disciplina.xsd">
  <disciplina>
     <codigo>000001</codigo>
     <nome>Algoritmos I</nome>
     <carga_horaria>60</carga_horaria>
     <creditos>5</creditos>
     <periodo>1</periodo>
     <ementa>Estudo de algoritmos</ementa>
     <codigo_professor>041</codigo_professor>
  </disciplina>
  ...
  ...
  <disciplina>
     <codigo>000002</codigo>
     <nome>Algoritmos II</nome>
     <carga_horaria>60</carga_horaria>
     <creditos>5</creditos>
     <periodo>2</periodo>
     <ementa>Estudo de algoritmosXXXX</ementa>
     <codigo_professor>052</codigo_professor>
     <codigo_prerequisito>000001</codigo_prerequisito>
  </disciplina>
</disciplinas>
```

Fragmento exemplo de instância de dados original - documento XML do Sistema de Biblioteca

---

obras.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<obras
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
xsi:noNamespaceSchemaLocation="obras.xsd">
<obra>
<codigo>000001</codigo>
<titulo>Building Web Applications Whith UML</titulo>
<descricao>Como construir aplicacoes Web com UML</descricao>
<conteudo>Tecnologias relacionadas a aplicacoes Web
          e modelamento</conteudo>
<autores>
 <nome>Jim Conallen</nome>
</autores>
<editora>Addison Wesley</editora>
<local>Boston</local>
<ano>2002</ano>
</obra>
  ...
  ...
<obra>
<codigo>000004</codigo>
<titulo>Code Complete</titulo>
<descricao>Livro sobre a construcao de software</descricao>
<conteudo>Desenvolvimento de software</conteudo>
<autores>
 <nome>Steven C. McConnell</nome>
</autores>
<editora>Microsoft Press</editora>
<local>Washington</local>
<ano>1993</ano>
</obra>
</obras>
```

---

usuarios.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<usuarios
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="usuario.xsd">
```

```
<usuario>
<codigo>000001</codigo>
<nome>Maria Perez</nome>
<login>mariap</login>
<senha>mmu67i</senha>
<pag_mensalidade>022005</pag_mensalidade>
<qdade_acesso>1</qdade_acesso>
</usuario>
...
...
<usuario>
<codigo>000006</codigo>
<nome>Ronaldo Pedroso</nome>
<login>ronalp</login>
<senha>mrr90i</senha>
<pag_mensalidade>022005</pag_mensalidade>
<qdade_acesso>1</qdade_acesso>
</usuario>
</usuarios>
```

B.2 Representação formal

Na Tabela B.1 é apresentada a representação formal dos esquemas “aluno.xsd” e “disciplina.xsd” do Sistema de Matrícula e “obras.xsd” e “usuario.xsd” do Sistema de Biblioteca.

Tabela B.1: Representação formal dos esquemas do Estudo de Caso I

Esquema XML	**Representação do esquema de dados XML**
aluno	$E = \{alunos,\ aluno,\ codigo,\ nome,\ login,\ senha,\ curso,$ $periodo, disciplina\_cursadas,\ disciplina\_matriculadas\}$ $A = \{\ \}$ $R = \{order,\ occur,\ type\}$ $P = \{p_1(alunos,\ aluno),\ p_2(alunos,\ order,\ aluno),$
continua na próxima página	

Tabela B.1 – continuação da página anterior

Esquema XML	**Representação do esquema de dados XML**
	$p_3(aluno,\ codigo)$, $p_4(aluno,\ nome)$, $p_5(aluno,\ login)$, $p_6(aluno,\ senha)$, $p_7(aluno,\ curso)$, $p_8(aluno,\ periodo)$, $p_9(aluno,\ disciplina\_cursadas)$, $p_{10}(aluno,\ disciplina\_matriculadas)$, $p_{11}(aluno,\ order,\ codigo,\ nome,\ login,\ senha,\ curso,\ periodo,\ disciplina\_cursadas,\ disciplina\_matriculadas), p_{12}(aluno,\ occur)$, $p_{13}(codigo,\ occur)$, $p_{14}(codigo,\ type)$, $p_{15}(nome,\ occur)$, $p_{16}(nome,\ type)$, $p_{17}(login,\ occur)$, $p_{18}(senha,\ occur)$, $p_{19}(senha,\ type)$, $p_{20}(curso,\ occur)$, $p_{21}(curso,\ type)$, $p_{22}(periodo,\ occur)$, $p_{23}(disciplina\_cursadas,\ occur)$, $p_{24}(disciplina\_cursadas,\ type)$, $p_{25}(disciplina\_matriculadas,\ occur)$, $p_{26}(disciplina\_matriculadas,\ type)\}$
disciplina	$E = \{disciplinas,\ disciplina,\ codigo,\ nome,\ carga\_horaria,\ creditos,\ periodo,\ ementa,\ codigo\_professor,\ codigo\_prerequisito\}$ $A = \{\ \}$ $R = \{order,\ occur,\ type\}$ $P = \{p_1(disciplinas,\ disciplina),\ p_2(disciplinas,\ order,\ disciplina),\ p_3(disciplina,\ codigo),\ p_4(disciplina,\ nome),\ p_5(disciplina,\ carga\_horaria),\ p_6(disciplina,\ creditos),\ p_7(disciplina,\ periodo),\ p_8(disciplina,\ ementa),\ p_9(disciplina,\ codigo\_professor),\ p_{10}(disciplina,\ codigo\_prerequisito),\ p_{11}(disciplina,\ order,\ codigo,\ nome,\ carga\_horaria,\ creditos,\ periodo,\ ementa,\ codigo\_professor,\ codigo\_prerequisito),\ p_{12}(disciplina,\ occur),\ p_{13}(codigo,\ occur),\ p_{14}(codigo,\ type),\ p_{15}(nome,\ occur),\ p_{16}(nome,\ type),\ p_{17}(carga\_horaria,\ occur),\ p_{18}(carga\_horaria,\ type),\ p_{19}(creditos,\ occur),\ p_{20}(creditos,\ type),$
continua na próxima página	

Tabela B.1 – continuação da página anterior

Esquema XML	**Representação do esquema de dados XML**
	$p_{21}(periodo,\ occur)$, $p_{22}(periodo,\ type)$,
	$p_{23}(ementa,\ occur)$, $p_{24}(ementa,\ type)$,
	$p_{25}(codigo\_professor,\ occur)$,
	$p_{26}(codigo\_prerequisito,\ occur)\}$
obras	$E = \{obras,\ obra,\ codigo,\ titulo,\ descricao,\ conteudo,\ autores,$
	$nome,\ editora,\ loca,\ ano\}$
	$A = \{\ \}$
	$R = \{order,\ occurs,\ type\}$
	$P = \{p_1(obras,\ obra)$, $p_2(obras,\ order,\ obra)$,
	$p_3(obra,\ codigo)$, $p_4(obra,\ titulo)$, $p_5(obra,\ descricao)$,
	$p_6(obra,\ conteudo)$, $p_7(obra,\ autores)$, $p_8(obra,\ editora)$,
	$p_9(obra,\ local)$, $p_{10}(obra,\ ano)$,
	$p_{11}(obra,\ order,\ codigo,\ titulo,\ descricao,\ conteudo,\ autores,\ editora,$
	$local,\ ano)$, $p_{12}(obra,\ occur)$,
	$p_{13}(codigo,\ occur)$, $p_{14}(codigo,\ type)$,
	$p_{15}(titulo,\ occur)$, $p_{16}(titulo,\ type)$,
	$p_{17}(descricao,\ occur)$,
	$p_{18}(conteudo,\ occur)$, $p_{19}(conteudo,\ type)$,
	$p_{20}(autores,\ nome)$, $p_{21}(autores,\ order,\ nome)$,
	$p_{22}(nome,\ occur)$, $p_{23}(nome,\ type)$,
	$p_{23}(editora,\ occur)$,
	$p_{24}(local,\ occur)$,
	$p_{25}(ano,\ occur)$, $p_{26}(ano,\ type)\}$
usuario	$E = \{usuarios,\ usuario,\ codigo,\ nome,\ login,\ senha,$
	$pag\_mensalidade,\ qdade\_acesso\}$
	$A = \{\ \}$
	$R = \{order,\ occur,\ type\}$
	$P = \{p_1(usuarios,\ usuario)$, $p_2(usuarios,\ order,\ usuario)$,
	$p_3(usuario,\ codigo)$, $p_4(usuario,\ nome)$, $p_5(usuario,\ login)$,
	$p_6(usuario,\ senha)$, $p_7(usuario,\ pag\_mensalidade)$,
	$p_8(usuario,\ qdade\_acesso)$,
	$p_9(usuario,\ order,\ codigo,\ nome,\ login,\ senha,$

continua na próxima página

Tabela B.1 – continuação da página anterior

Esquema XML	Representação do esquema de dados XML
	$pag\_mensalidade$, $qdade\_acesso$), $p_{10}(usuario$, $occur)$, $p_{11}(codigo$, $occur)$, $p_{12}(codigo$, $type)$, $p_{13}(nome$, $occur)$, $p_{14}(nome$, $type)$, $p_{15}(login$, $occur)$, $p_{16}(senha$, $occur)$, $p_{17}(senha$, $type)$, $p_{18}(pag\_mensalidade$, $occur)$, $p_{19}(pag\_mensalidade$, $type)$, $p_{20}(qdade\_acesso$, $occur)\}$

B.3 Associações de defeitos

Na Tabela B.2 são vistas as associações, entre elementos ou atributos e restrições, geradas pela XTool para os esquemas "aluno.xsd", "disciplina.xsd", "obras.xsd" e "usuario.xsd". É importante relembrar que o elemento que será raiz do documento XML (no caso do esquema "aluno.xsd", o elemento raiz é o elemento alunos) pode ser associado à classe de defeito *G3 - ORI* (Ordem Incorreta), mas essa associação não é explorada pela ferramenta nos casos em que esse elemento é o elemento raiz e só possui um elemento-filho, portanto a ordem entre os elementos-filho não pode ser diferente.

Tabela B.2: Associações geradas para os esquemas do Estudo de Caso I

Esquema XML	Elemento	Classe de defeito
aluno	alunos	*G3 - ORI* - Ordem Incorreta
	aluno	*G3 - ORI* - Ordem Incorreta, *G3 - OI* - Ocorrência Incorreta
	codigo	*G3 - OI* - Ocorrência Incorreta, *G1 - TDI* - Tipo de Dado Incorreto
	nome	*G3 - OI* - Ocorrência Incorreta, *G1 - TDI* - Tipo de Dado Incorreto
	login	*G3 - OI* - Ocorrência Incorreta
	senha	*G3 - OI* - Ocorrência Incorreta, *G1 - TDI* - Tipo de Dado Incorreto
continua na próxima página		

Tabela B.2 – continuação da página anterior

Esquema XML	Elemento	Classe de defeito
	curso	*G3 - OI* - Ocorrência Incorreta, *G1 - TDI* - Tipo de Dado Incorreto
	periodo	*G3 - OI* - Ocorrência Incorreta
	disciplina_cursadas	*G3 - OI* - Ocorrência Incorreta, *G1 - TDI* - Tipo de Dado Incorreto
	disciplina_matriculadas	*G3 - OI* - Ocorrência Incorreta, *G1 - TDI* - Tipo de Dado Incorreto
disciplina	disciplinas	*G3 - ORI* - Ordem Incorreta
	disciplina	*G3 - ORI* - Ordem Incorreta, *G3 - OI* - Ocorrência Incorreta
	codigo	*G3 - OI* - Ocorrência Incorreta, *G1 - TDI* - Tipo de Dado Incorreto
	nome	*G3 - OI* - Ocorrência Incorreta, *G1 - TDI* - Tipo de Dado Incorreto
	carga_horaria	*G3 - OI* - Ocorrência Incorreta, *G1 - TDI* - Tipo de Dado Incorreto
	creditos	*G3 - OI* - Ocorrência Incorreta, *G1 - TDI* - Tipo de Dado Incorreto
	periodo	*G3 - OI* - Ocorrência Incorreta, *G1 - TDI* - Tipo de Dado Incorreto
	ementa	*G3 - OI* - Ocorrência Incorreta, *G1 - TDI* - Tipo de Dado Incorreto
	codigo_professor	*G3 - OI* - Ocorrência Incorreta
	codigo_prerequisito	*G3 - OI* - Ocorrência Incorreta
obras	obras	*G3 - ORI* - Ordem Incorreta
	obra	*G3 - ORI* - Ordem Incorreta, *G3 - OI* - Ocorrência Incorreta
	codigo	*G3 - OI* - Ocorrência Incorreta, *G1 - TDI* - Tipo de Dado Incorreto
	titulo	*G3 - OI* - Ocorrência Incorreta, *G1 - TDI* - Tipo de Dado Incorreto
	descricao	*G3 - OI* - Ocorrência Incorreta

continua na próxima página

Tabela B.2 – continuação da página anterior

Esquema XML	Elemento	Classe de defeito
	conteudo	*G3 - OI* - Ocorrência Incorreta, *G1 - TDI* - Tipo de Dado Incorreto
	autores	*G3 - ORI* - Ordem Incorreta
	nome	*G3 - OI* - Ocorrência Incorreta, *G1 - TDI* - Tipo de Dado Incorreto
	editora	*G3 - OI* - Ocorrência Incorreta
	local	*G3 - OI* - Ocorrência Incorreta
	ano	*G3 - OI* - Ocorrência Incorreta, *G1 - TDI* - Tipo de Dado Incorreto
usuario	usuarios	*G3 - ORI* - Ordem Incorreta
	usuario	*G3 - ORI* - Ordem Incorreta, *G3 - OI* - Ocorrência Incorreta
	codigo	*G3 - OI* - Ocorrência Incorreta, *G1 - TDI* - Tipo de Dado Incorreto
	nome	*G3 - OI* - Ocorrência Incorreta, *G1 - TDI* - Tipo de Dado Incorreto
	login	*G3 - OI* - Ocorrência Incorreta
	senha	*G3 - OI* - Ocorrência Incorreta, *G1 - TDI* - Tipo de Dado Incorreto
	pag_mensalidade	*G3 - OI* - Ocorrência Incorreta, *G1 - TDI* - Tipo de Dado Incorreto
	qdade_acesso	*G3 - OI* - Ocorrência Incorreta

Apêndice C

Estudo de Caso II

Neste apêndice são apresentados dois esquemas utilizados no Estudo de Caso II.

C.1 Esquema de catálogo de produtos

A seguir é apresentado o esquema de catálogo de produtos extraído de Carlson [7].

Esquema XML "catalog.xsd"

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Catalog">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="name" type="xs:string"/>
         <xs:element name="description" type="xs:string"/>
         <xs:element name="startDate" type="xs:dateTime"/>
         <xs:element name="endDate" type="xs:dateTime"/>
         <xs:element name="item" minOccurs="0" maxOccurs="unbounded">
           <xs:complexType>
              <xs:sequence>
                <xs:element name="name" type="xs:string"/>
                <xs:element name="description" type="xs:string"/>
                <xs:element name="listPrice" type="xs:float"/>
                <xs:element name="sku" type="xs:string"/>
                <xs:element name="globalIdentifier" type="xs:string"/>
                <xs:element name="tipoProduto">
                  <xs:complexType>
```

```
                    <xs:choice>
                     <xs:element name="unitOfMeasure">
                      <xs:simpleType>
                        <xs:restriction base="xs:string">
                          <xs:enumeration value="each"/>
                          <xs:enumeration value="dozen"/>
                          <xs:enumeration value="meter"/>
                          <xs:enumeration value="kilogram"/>
                        </xs:restriction>
                      </xs:simpleType>
                     </xs:element>
                     <xs:element name="unitOfTime">
                      <xs:simpleType>
                        <xs:restriction base="xs:string">
                           <xs:enumeration value="hour"/>
                           <xs:enumeration value="day"/>
                           <xs:enumeration value="week"/>
                           <xs:enumeration value="month"/>
                           <xs:enumeration value="year"/>
                        </xs:restriction>
                      </xs:simpleType>
                     </xs:element>
                    </xs:choice>
                  </xs:complexType>
                </xs:element>
                <xs:element name="feature" minOccurs="0" maxOccurs="unbounded">
                  <xs:complexType>
                    <xs:sequence>
                      <xs:element name="value" type="xs:string" minOccurs="0"
                      maxOccurs="unbounded"/>
<xs:element name="type">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element name="name" type="xs:string"/>
                          <xs:element name="description" type="xs:string"/>
                          <xs:element name="multivalued" type="xs:boolean"/>
                        </xs:sequence>
                      </xs:complexType>
                    </xs:element>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
```

```
          </xs:complexType>
 </xs:element>
      </xs:sequence>
   </xs:complexType>
  </xs:element>
</xs:schema>
```

C.2 Esquema do serviço de comércio eletrônico Amazon

A seguir é apresentado o esquema *seller* do Amazon extraído de Carlson [7].

Esquema XML "seller.xsd"

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
 elementFormDefault="qualified">
 <xs:element name="sellers">
  <xs:complexType>
  <xs:sequence>
  <xs:element name="seller" maxOccurs="unbounded">
  <xs:complexType>
   <xs:sequence>
     <xs:element name="sellerName" type="xs:string" minOccurs="0"/>
     <xs:element name="nickName" type="xs:string" minOccurs="0"/>
     <xs:element name="glancePage" type="xs:string" minOccurs="0"/>
     <xs:element name="about" type="xs:string" minOccurs="0"/>
     <xs:element name="moreAbout" type="xs:string" minOccurs="0"/>
     <xs:element name="averageFeedbackRating" type="xs:decimal" minOccurs="0"/>
     <xs:element name="totalFeedback" type="xs:nonNegativeInteger" minOccurs="0"/>
     <xs:element name="totalFeedbackPages" type="xs:nonNegativeInteger"
                 minOccurs="0"/>
     <xs:element name="location" minOccurs="0">
      <xs:complexType>
       <xs:sequence>
           <xs:element name="city" type="xs:string" minOccurs="0"/>
           <xs:element name="state" type="xs:string" minOccurs="0"/>
           <xs:element name="country" type="xs:string" minOccurs="0"/>
 </xs:sequence>
```

```
</xs:complexType>
     </xs:element>
     <xs:element name="sellerFeedback" minOccurs="0">
      <xs:complexType>
       <xs:sequence>
           <xs:element name="feedback" minOccurs="1" maxOccurs="unbounded">
            <xs:complexType>
             <xs:sequence>
                <xs:element name="rating" type="xs:nonNegativeInteger"
                            minOccurs="0"/>
                <xs:element name="comment" minOccurs="0">
  <xs:simpleType>
            <xs:restriction base="xs:string">
                      <xs:maxLength value="100"/>
                      <xs:whiteSpace value="preserve"/>
                    </xs:restriction>
                  </xs:simpleType>
</xs:element>
                <xs:element name="date" type="xs:date" minOccurs="0"/>
                <xs:element name="ratedBy" type="xs:string" minOccurs="0"/>
       </xs:sequence>
      </xs:complexType>
           </xs:element>
 </xs:sequence>
</xs:complexType>
     </xs:element>
   </xs:sequence>
   <xs:attribute name="sellerID" use="required">
     <xs:simpleType>
       <xs:restriction base="xs:string">
         <xs:pattern value="[a-zA-Z0-9]{8}"/>
       </xs:restriction>
     </xs:simpleType>
   </xs:attribute>
  </xs:complexType>
 </xs:element>
 </xs:sequence>
 </xs:complexType>
 </xs:element>
</xs:schema>
```

Apêndice D

Estudo de Caso III

O Apêndice D apresenta o diagrama de entidade-relacionamento e um fragmento do esquema correspondente do Estudo de Caso III.

D.1 DER da base de dados de alunos egressos

A Figura D.1 ilustra o DER que representa o esquema da aplicação de base de dados referente ao histórico de alunos egressos.

D.2 Esquema de alunos egressos

A seguir é apresentado um fragmento do esquema de alunos egressos.

Esquema de alunos egressos escrito em DDL

```
CREATE TABLE Aluno (
       id_aluno             int NOT NULL,
       data_nascimento      datetime NULL,
       cpf                  varchar(11) NULL,
       sexo                 varchar(1) NULL,
       RA                   smallint NOT NULL,
       status               bit NOT NULL,
       RG                   char(10) NULL
)
go
ALTER TABLE Aluno
```

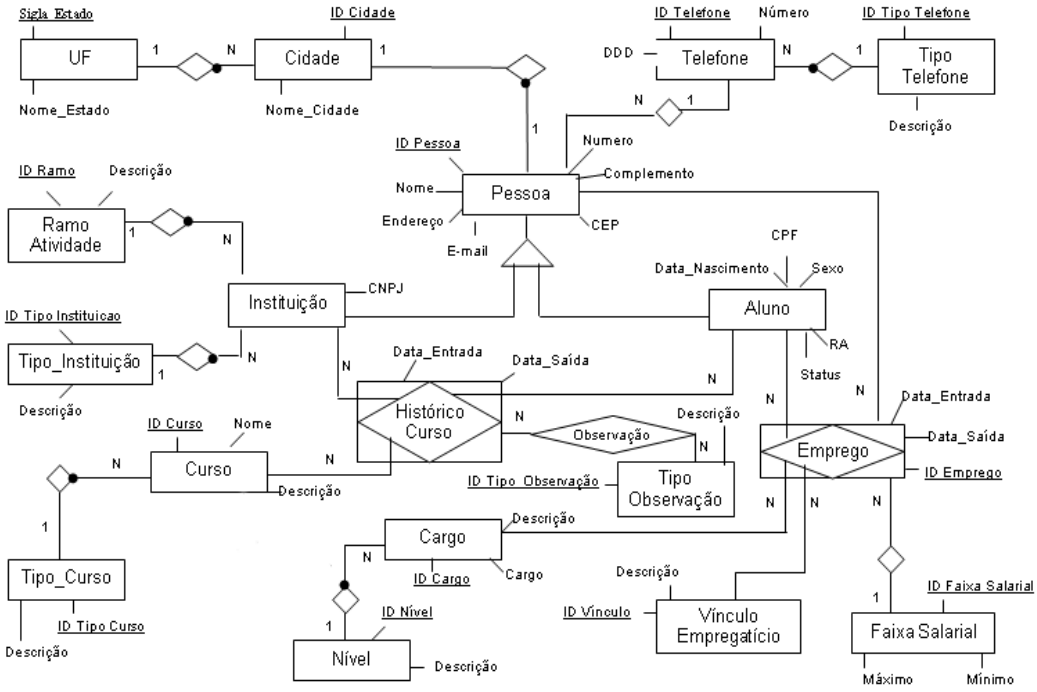

Figura D.1: Diagrama de Entidade-Relacionamento da base de dados relacional - Estudo de Caso III

```
	ADD PRIMARY KEY NONCLUSTERED (id_aluno)
go
CREATE TABLE Cargo (
	id_cargo		int IDENTITY,
	cargo			varchar(50) NOT NULL,
	descricao		varchar(60) NULL,
	id_nivel		int NOT NULL
)
go
ALTER TABLE Cargo
	ADD PRIMARY KEY NONCLUSTERED (id_cargo)
go
CREATE TABLE Cidade (
	id_cidade		int IDENTITY,
	sigla_estado		char(2) NOT NULL,
	nome_cidade		varchar(60) NOT NULL
)
go
ALTER TABLE Cidade
	ADD PRIMARY KEY NONCLUSTERED (id_cidade)
go
```

```
CREATE TABLE Curso (
       id_curso             int IDENTITY,
       nome_curso           varchar(40) NOT NULL,
       id_tipo_curso        int NOT NULL,
       duracao              int NOT NULL
)
go

...

ALTER TABLE Usuario
       ADD FOREIGN KEY (id_curso)
                           REFERENCES Curso
go
ALTER TABLE Historico_Curso
add constraint historico_curso_check_entrada_saida check
((data_saida>data_entrada)or((data_saida = '')))
ALTER TABLE Emprego
add constraint emprego_check_entrada_saida check
((data_saida>data_entrada)or((data_saida = '')))
ALTER TABLE faixa_salarial
add constraint emprego_check_faixa_salarial check
(maximo>minimo)
go
```

Apêndice E

Estudo de Caso IV

Este apêndice mostra os diagramas de entidade-relacionamento das bases de dados do Estudo de Caso IV e fragmentos dos respectivos esquemas.

E.1 DER da base de dados de cooperativa rural

A Figura E.1 apresenta o diagrama de entidade-relacionamento que representa o esquema da aplicação de base de dados referente a cooperativa rural do Estudo de Caso IV.

E.2 Esquema da cooperativa rural

A seguir é apresentado um fragmento do esquema da cooperativa rural escrito em DDL.

Esquema da cooperativa rural

```
CREATE TABLE aquisicao (
    data_aquisicao timestamp without time zone NOT NULL,
    numero_fiscal integer,
    matricula character(9) NOT NULL,
    numero_lote integer NOT NULL,
    codigo_aquisicao numeric NOT NULL
);
CREATE TABLE comercializa (
    codigo character(8) NOT NULL,
    numero_lote integer NOT NULL
);
```

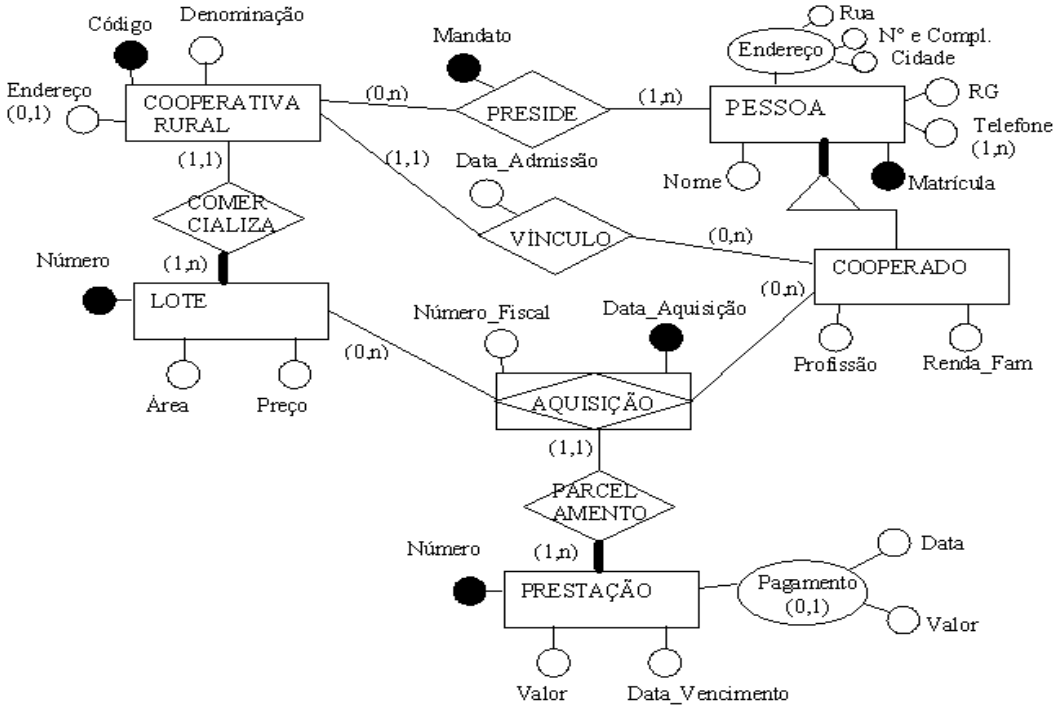

Figura E.1: Diagrama de Entidade-Relacionamento da cooperativa rural - Estudo de Caso IV

```
CREATE TABLE cooperado (
    profissao character varying(60),
    rendafam double precision,
    matricula character(9) NOT NULL
);
CREATE TABLE cooperativa_rural (
    codigo character(8) NOT NULL,
    denominacao character varying(60) NOT NULL,
    endereco character varying(255)
);
CREATE TABLE endereco (
    rua character varying(120) NOT NULL,
    cidade character varying(120) NOT NULL,
    numero character(20) NOT NULL,
    matricula character(9) NOT NULL
);
CREATE TABLE lote (
    numero integer NOT NULL,
    area integer NOT NULL,
    preco double precision NOT NULL
);
```

```
CREATE TABLE pagamento (
    data timestamp without time zone,
    valor double precision,
    numero integer NOT NULL
);
...
```

E.3 DER da base de dados de biblioteca

A Figura E.2 apresenta um fragmento do diagrama de entidade-relacionamento que representa o esquema da aplicação de base de dados referente a biblioteca do Estudo de Caso IV.

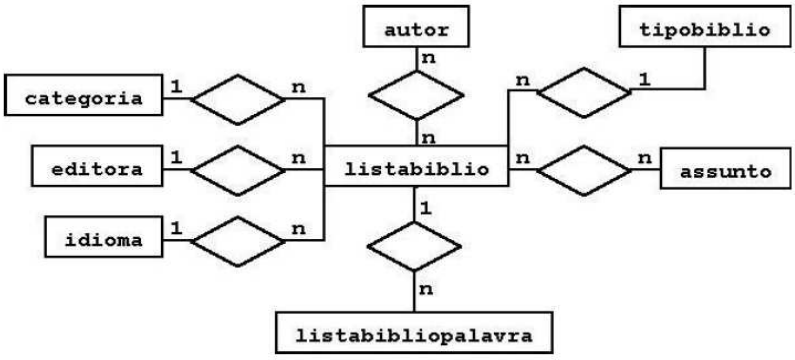

Figura E.2: Diagrama de Entidade-Relacionamento da biblioteca - Estudo de Caso IV

E.4 Esquema da biblioteca

A seguir é apresentado um fragmento do esquema da biblioteca escrito em DDL.

Esquema da biblioteca

```
CREATE TABLE acervo (
    id_acervo integer DEFAULT nextval(('seq_acervo'::text)::regclass) NOT NULL,
    is_listabiblio integer NOT NULL,
    tombo character varying(15),
    dttombo date,
    tipoaquisicao smallint NOT NULL,
    dtbaixa date,
```

```
    tipobaixa smallint,
    status smallint NOT NULL,
    dados character varying(70),
    codigo character varying(20),
    is_biblioteca integer NOT NULL,
    CONSTRAINT reg_status CHECK (((status >= 1) AND (status <= 6))),
    CONSTRAINT reg_tipoaquisicao CHECK (((tipoaquisicao >= 1)
               AND (tipoaquisicao <= 3))),
    CONSTRAINT reg_tipobaixa CHECK (((tipobaixa >= 1) AND (tipobaixa <= 3)))
);
CREATE TABLE assunto (
    id_assunto integer DEFAULT
               nextval(('seq_assunto'::text)::regclass) NOT NULL,
    assunto character varying(100) NOT NULL
);
CREATE TABLE autor (
    id_autor integer DEFAULT
                     nextval(('seq_autor'::text)::regclass) NOT NULL,
    autor character varying(100) NOT NULL,
    apelido character varying(50)
);
CREATE TABLE biblioteca (
    id_biblioteca integer DEFAULT
                  nextval(('seq_biblioteca'::text)::regclass) NOT NULL,
    biblioteca character varying(100) NOT NULL,
    is_usuario integer,
    telefone1 character varying(15),
    telefone2 character varying(15)
);
CREATE TABLE calendario (
    id_data date NOT NULL,
    descricao character varying(50) NOT NULL,
    feriado boolean,
    observacao text
);
...
```