UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UMA ABORDAGEM DE TESTE ESTRUTURAL DE TRANSFORMAÇÕES M2T BASEADA EM HIPERGRAFOS

André da Silva Abade

Orientador: Prof. Dr. Fabiano Cutigi Ferrari

São Carlos – SP

2015

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UMA ABORDAGEM DE TESTE ESTRUTURAL DE TRANSFORMAÇÕES M2T BASEADA EM HIPERGRAFOS

André da Silva Abade

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação, área de concentração: Engenharia de Software
Orientador: Prof. Dr. Fabiano Cutigi Ferrari

São Carlos – SP

2015

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UMA ABORDAGEM DE TESTE ESTRUTURAL DE TRANSFORMAÇÕES M2T BASEADA EM HIPERGRAFOS

André da Silva Abade

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação, área de concentração: Engenharia de Software
Orientador: Prof. Dr. Fabiano Cutigi Ferrari

Aprovado em 05 de Janeiro de 2016.

Membros da banca:

Prof. Dr. Fabiano Cutigi Ferrari
(Orientador– DC-UFSCar)

Prof. Dr. Marcelo de Almeida Maia
(DC - UFU)

Profa. Dr. Valter Vieira de Carmargo
(DC - UFSCar)

São Carlos – SP

2015

Primeiramente, dedico esta pesquisa a Deus, o meu refúgio, a minha fortaleza. De modo especial, dedico este trabalho a minha Mãe Maria da G. Abade, pois desde o momento que fui tecido em seu ventre se fez rochedo guiando meus passos e intercedendo por mim. De modo singular, a minha Irmã e as duas crianças que moldam minha vida, meus amores, Júlia A. Abade (Filha) e Gabriela Abade Doneda (Sobrinha).

Agradecimentos

Agradeço ao meu Orientador Fabiano C. Ferrari, por ter me selecionado como orientando e oportunizado muito mais que uma conquista, a realização de um sonho. Ao meu Co-Orientador Daniel Lucrédio, por toda solicitude, humildade, gentileza, incentivo e motivação. Não há como externar minha admiração pelos senhores.

Agradeço também às instituições que fizeram com que esta trajetória fosse possível. Estas incluem a UFSCar - São Carlos, por ter sido minha casa durante estes anos. Também agradeço ao Instituto Federal de Educação, Ciência e Tecnologia do Mato Grosso, por ser provedor e financiador do meu afastamento para qualificação. Posso garantir que fui intenso e dei o meu melhor, espero ter correspondido a contento.

Agradeço ao LaPES (Laboratório de Pesquisa em Engenharia de Software). Tenho orgulho e extrema satisfação de ser parte do melhor e inesquecível laboratório de pesquisa da UFSCar.

Agradeço a Professora Dra. Sandra Fabbri, por sua receptividade, acolhida e por partilhar sua sapiência em nossos momentos de descontração e motivação a pesquisa. Agradeço também a família LaPES do meu convívio André DT, Fabio, Elis, Anderson, Erik, Tiago Gaspar, Tiago Jesus, Ana, Barbara. Aos colegas do Departamento de Computação que iniciaram esta jornada comigo, difícil nominá-los porém impossível esquecê-los.

A galera do Café, vocês são parte dos meus melhores momentos. A galera do Futebol e a moçada do churras, vocês foram importantes. Agradeço também aos colegas e amigos que colecionei nesta trajetória: Elias (Baiano), Fernando (Maranhão), Lucas Porto (Goias), Luiz Paccini, Roberto (Mineiro), Rener, Victor (Vitinho), Odair, Bento, Rafael Gastaldi, Cleiton (Cleitinho), Jaum, Alex Guido, Renato (Kiwi) e Ivan Ervlino.

Agradeço aos Funcionários do Departamento de Computação em nome dos servidores Augusto Cesar, Cristina Trevelin e Evelton. A competência, seriedade e préstimos de vocês são determinantes para qualquer aluno deste Departamento.

Aos meus amigos e companheiros de trabalho do IFMT - Campus Barra do Garças, em especial à Gleiner, Alexis, Jacinto, Josdyr e também aos amigos das Faculdades Cathedral de Barra do Garças.

Agradeço ao meu amigo e fonte de exemplos Professor Dr. José Nogueira e a minha sempre e eterna Professora Msc. Rosilene por seus conselhos e por partilharem comigo experiências de caráter, profissionalismo e afeto tipicas de espíritos evoluídos que são.

Ao meu amigo Professor Dr. Marco Campos, nossas conversas e apoio mutuo certamente é um dos pilares desta conquista. Ao meu irmão e melhor amigo Michelle Cozzolino Junior, você me visitaria até no pólo norte e sempre estará em meu coração e orações.

Agradeço também a Josilene Dália Alves minha companheira de jornada e namorada por partilhar meus momentos de angustias e aflições. Por viver comigo as desilusões e conquistas, por ser paciente e companheira e não menos importante por revisar todo meu texto.

Agradeço a minha irmã por ser parte da minha vida e conseqüentemente parte desta conquista. A Gabriela (sobrinha) por suas ligações, por me fazer chorar de saudade e me animar ao ouvir sua voz. A minha Filha Júlia por sua meiguice, carinho e pela certeza do seu perdão por minha ausência. Papai te ama.

Agradeço especialmente a Senhora minha Mãe Dona Maria G. Abade, não existe sinônimo que a qualifique. Não existe palavras que possam externar minha gratidão. Não existe pessoa sob a terra que eu mais admire e ame. Minha intercessora com sua preocupação constante, com seu zelo incessante e jeito singular de amar intensamente. Tudo que sou devo a pessoa que a senhora é.

Por fim, obrigado Senhor meu Deus por ser meu rochedo, minha fortaleza. Obrigado por iluminar meus caminhos, fortalecer minha alma e sob a tempestade acalmar meu coração.

"O meu olhar alcança o longe. Contempla o território que me separa da concretização de meu desejo. O destino final que o olhar já reconhece como recompensa, aos pés se oferece como lonjura a ser vencida. Mas não há pressa que seja capaz de diminuir esta distância. Estamos sob a prevalência de uma imposição existencial, regra que ensina, que entre o ser real e o ser desejado, há o senhorio inevitável do tempo das esperas."

Pe. Fábio de Melo

Não há nada bom nem mau a não ser estas duas coisas: a sabedoria que é um bem e a ignorância que é um mal. – PLATÃO

Resumo

Contexto: O MDD (Model-Driven Development ou Desenvolvimento Dirigido por Modelos) é um paradigma de desenvolvimento de software em que os principais artefatos são os modelos, a partir dos quais o código ou outros artefatos são gerados. Esse paradigma, embora possibilite diferentes visões de como decompor um problema e projetar um software para solucioná-lo, introduz novos desafios, qualificados pela complexidade dos modelos de entrada, as transformações e os artefatos de saída. ***Definição do Problema:*** Dessa forma, o teste de software é uma atividade fundamental para revelar defeitos e aumentar a confiança nos produtos de software desenvolvidos nesse contexto. Diversas técnicas e critérios de teste vêm sendo propostos e investigados. Entre eles, o teste funcional têm sido bastante explorado primordialmente nas transformações M2M (Model-to-Model ou Modelo para Modelo), enquanto que o teste estrutural em transformações M2T (Model-to-Text ou Modelo para Texto) ainda possui alguns desafios e carência de novas abordagens. ***Objetivos:*** O objetivo deste trabalho é apresentar uma proposta para o teste estrutural de transformações M2T, por meio da caracterização dos dados complexos dos modelos de entrada, templates e artefatos de saída envolvidos neste processo. ***Metodologia:*** A abordagem proposta foi organizada em cinco fases e sua estratégia propõe que os dados complexos (gramáticas e metamodelos) sejam representados por meio de hipergrafos direcionados, permitindo que um algoritmo de percurso em hipergrafos, usando combinatória, crie subconjuntos dos modelos de entrada que serão utilizados como casos de teste para as transformações M2T. Nesta perspectiva, realizou-se dois estudos exploratórios com propósito específico da análise de viabilidade quanto a abordagem proposta. ***Resultados:*** A avaliação dos estudos exploratórios proporcionou, por meio da análise dos critérios de cobertura aplicados, um conjunto de dados que demonstram a relevância e viabilidade da abordagem quanto a caracterização de dados complexos para os testes em transformações M2T. A segmentação das estratégias em fases possibilita a revisão e adequação das atividades do processo, além de auxiliar na replicabilidade da abordagem em diferentes aplicações que fazem uso do paradigma MDD.

Palavras-chave: Desenvolvimento Dirigido a Modelos, MDD, Modelo para Texto, Transformações de Modelos M2T, Teste em Transformações de Modelo, Teste Estrutural, Dados Complexos

Abstract

Context: MDD (Model-Driven Development) is a software development paradigm in which the main artefacts are models, from which source code or other artefacts are generated. Even though MDD allows different views of how to decompose a problem and how to design a software to solve it, this paradigm introduces new challenges related to the input models, transformations and output artefacts. ***Problem Statement:*** Thus, software testing is a fundamental activity to reveal defects and improve confidence in the software products developed in this context. Several techniques and testing criteria have been proposed and investigated. Among them, functional testing has been extensively explored primarily in the M2M (Model-to-Model) transformations, while structural testing for M2T (Model-to-Text) transformations still poses challenges and lacks appropriate approaches. ***Objective:*** This work aims to to present a proposal for the structural testing of M2T transformations through the characterisation of input models as complex data, templates and output artefacts involved in this process. ***Method:*** The proposed approach was organised in five phases. Its strategy proposes that the complex data (grammars and metamodels) are represented by directed hypergraphs, allowing that a combinatorial-based traversal algorithm creates subsets of the input models that will be used as test cases for the M2T transformations. In this perspective, we carried out two exploratory studies with the specific purpose of feasibility analysis of the proposed approach. ***Results and Conclusion:*** The evaluation of results from the exploratory studies, through the analysis of some testing coverage criteria, demonstrated the relevance and feasibility of the approach for characterizing complex data for M2T transformations testing. Moreover, structuring the testing strategy in phases enables the revision and adjustment of activities, in addition to assisting the replication of the approach within different applications that make use of the MDD paradigm.

Keywords: Model-Driven Development, MDD, Model-to-text, M2T Model Transformations, Testing Transformation Model, Structural Testing, Complex Data

Lista de Figuras

Lista de Tabelas

Lista de Códigos

Lista de Abreviaturas e Siglas

ADM – *Architecture-driven modernization*

ADT – *Abstract Data Types*

CIM – *Computation-Independent Model*

CORBA – *Common Object Request Broker Architecture*

CWM – *Common Warehouse Metamodel*

DSL – *Domain-Specific Language*

EBNF – *Extended Backus Naur Form*

EMF – *Eclipse Modeling Framework*

EMF – *Eclipse Modeling Framework*

GPML – *General-Purpose Modeling Languages*

HTML – *HyperText Markup Language*

JMI – *Java Metadata Interface*

M2C – *Model-to-Code*

M2M – *Model-to-Model*

M2T – *Model-to-Text*

MBE – *Model-Based Engineering*

MDA – *Model Driven Architecture*

MDD – *Model-Driven Development*

MDE – *Model Driven Engineering*

MDSD – *Model-Driven Software Development*

MOF – *Meta Object Facility*

OCL – *Object Constraint Language*

OMG – *Object Management Group*

PIM – *Platform-Independent Model*

PSM – *Platform Specific Model*

SPEM – *Software Process Engineering Metamodel*

SQL – *Structure Query Language*

UML – *Unified Modeling Language*

VHDL – *VHSIC Hardware Description Language*

XMI – *XML Metadata Interchange*

XML – *Extensible Markup Language*

Sumário

Capítulo 1

Introdução

1.1 Contextualização e Motivação

O MDD (*Model-Driven Development* ou Desenvolvimento Dirigido por Modelos) é um paradigma de desenvolvimento de *software* em que os principais artefatos são os modelos a partir dos quais o código ou outros artefatos são gerados. Modelos são artefatos primários do desenvolvimento e contêm informações que apoiam as várias fases do processo de desenvolvimento. Estes modelos podem capturar informações das diferentes fases do ciclo de vida de desenvolvimento, tais como: requisitos, projeto, implementação, análise de qualidade, validação e verificação (BEYDEDA; BOOK; GRUHN, 2005).

No MDD, os modelos de *software* são abstrações de alto nível, que são refinados e transformados para domínios específicos e, finalmente, transformados em código. Assim, garantir que esses modelos sejam eficazes e eficientes torna-se um dos grandes desafios para este paradigma, pois as transformações de modelos envolvem diferentes níveis de abstrações, estabelecendo que a complexidade do paradigma MDD esteja diretamente relacionada com três etapas envolvidas nestas transformações: os modelos de entrada; as regras de transformações e os artefatos de saída (GUERRA; SOEKEN, 2013; KLEPPE; WARMER; BAST, 2003).

A transformação é definida por um conjunto de regras que, juntas, descrevem como um modelo na linguagem origem pode ser transformado em um ou mais modelos na linguagem destino (KLEPPE; WARMER; BAST, 2003). As transformações, por sua vez, envolvem dois ou mais modelos e podem ser classificadas como transformação de modelo para modelo (M2M - *Model-to-Model*) e transformação de modelo para texto (M2T - *Model-to-Text*).

Para definir estas transformações, faz-se necessário uma ferramenta que possibilite aos engenheiros de *software* a construção de regras de mapeamento de modelo para modelo (M2M) ou de modelo para texto (M2T).

Os transformadores são programas que mapeiam os modelos de entrada conforme um meta-modelo de origem para artefatos de saída (GUERRA; SOEKEN, 2013). Além disso, deve-se possibilitar que as regras de mapeamento sejam especificadas de forma mais natural possível, uma vez que a construção de transformadores é uma tarefa complexa. Por fim, é necessário um mecanismo que aplique as transformações especificadas e que também possibilite a rastreabilidade do mapeamento definido, seja ele no modelo ou no texto (código fonte) gerado (LUCRÉDIO, 2009). Garantir que essas transformações possam acontecer, assegurando as propriedades esperadas, agrega uma complexidade peculiar para esta etapa do MDD.

As transformações M2T preenchem uma lacuna entre as linguagens de modelagem e as linguagens de programação, permitindo a geração de código fonte e produzindo documentação ou representações textuais do conteúdo do modelo. Os estudos de Czarnecki e Helsen (2006) afirmam que a geração de código é um dos principais objetivos do MDD, e que as transformações são de grande importância. Assim, a geração de artefatos com qualidade demanda métodos de testes que sejam capazes de ratificar que as transformações M2T estejam consoantes com os modelos de entrada e saída, ou seja, que código gerado corresponda com suas funcionalidades com a semântica do modelo.

O processo de desenvolvimento de software envolve uma série de atividades nas quais, apesar das técnicas, métodos e ferramentas empregadas, defeitos no produto ainda podem ocorrer. Um sistema não pode ser considerado confiável sem que seja realizado um processo de verificação e validação adequado. Mesmo sistemas desenvolvidos seguindo o paradigma MDD, no qual o código do *software* pode ser gerado automaticamente por meio dos modelos, não estão livres de defeitos. Nesse contexto, teste de software é uma das técnicas essenciais.

O teste consiste em uma análise dinâmica do produto, sendo relevante para a identificação e a eliminação de defeitos que persistem, representando a última revisão da especificação, projeto e codificação (MALDONADO, 1991; HARROLD, 2000; PRESSMAN, 2010). De maneira geral, as técnicas de teste podem ser divididas naquelas baseadas no código (caixa branca ou estrutural) e naquelas baseadas na especificação (caixa preta ou funcional). Nenhuma delas é completa pois visam identificar tipos diferentes de defeitos e sua utilização em conjunto pode elevar o nível de confiabilidade do software.

O teste de *software* pode ser uma abordagem eficaz para a validação das transformações de modelos. Entretanto, Fleurey, Steel e Baudry (2004) expõem que o teste de transformações de modelos é distinto do teste de implementações tradicionais, pois os dados de entrada são modelos complexos, especialmente quando comparados aos tipos de dados simples. O trabalho de Wimmer e Burgueo (2013) destaca a necessidade de abordagens de teste capazes de suportar e garantir a geração dos artefatos de *software* por meio das transformações M2T.

A testabilidade dos transformadores tem por objetivo primordial facilitar a geração de dados de testes e garantir uma melhor cobertura por meio das adequações dos critérios de teste. Os estudos de Harman (2008) afirmam que o teste de transformadores de modelos deixa aberta uma série de questões, que relacionam a combinação do teste de software e os tipos de transformações que ocorrem em MDD. Ainda que as questões observadas pelo autor não definam explicitamente a complexidade da caracterização dos dados de teste para os modelos de entrada e saída, essa questão fundamenta todos os problemas relatados em seu estudo. De acordo com Baudry et al. (2010), a complexidade dos dados manipulados pelas transformações de modelos trazem novos desafios, exigindo das técnicas de testes adaptações capazes de atender este paradigma de desenvolvimento.

1.2 Objetivos

O objetivo geral deste trabalho é a caracterização de dados complexos para testes estruturais em transformações M2T. Pretende-se formalizar uma solução que alinha a capacidade de ilustração com navegabilidade algorítmica capaz de representar atomicamente os dados complexos envolvidos no teste de transformações M2T. Adicionalmente, a representação proposta permitirá a geração dos casos de teste, o cômputo dos critérios de cobertura e a verificação dos *templates* quanto ao modelo de entrada e o código fonte gerado.

Para alcançar os objetivos almejados, foi proposta uma abordagem para caracterização de dados complexos baseada em hipergrafos. A contribuição pretendida se deu por meio da avaliação dos resultados obtidos entre dois estudos exploratórios realizados. Os conjuntos de casos de teste gerados permitiram exercitar os *templates* utilizados nas transformações e computar os critérios de cobertura definidos.

1.3 Metodologia

Uma pesquisa comumente é dividida em três grandes etapas, que podem ser conduzidas paralelamente dependendo do design de pesquisa adotado - fundamentação da proposta (revisão da literatura), desenvolvimento e por fim, avaliação e validação (LAKATOS; MARCONI, 2010). Assim, nesta seção serão descritos os procedimentos metodológicos que nortearam esta pesquisa.

Aborda-se, primeiramente, a caracterização da pesquisa que teve um caráter exploratório descritivo. Em seguida, tecem-se considerações acerca da técnica utilizada para a coleta e análise de dados.

A metodologia utilizada em consonância com os objetivos deste trabalho iniciou-se no levantamento do estado da arte por meio de um revisão sistemática da literatura sobre a temática investigada. A partir das conclusões e desfechos encontrados na revisão sistemática, traçou-se um planejamento para compor uma abordagem viável e que contemplasse com respostas as questões primordiais levantadas na literatura.

A abordagem proposta foi organizada em cinco fases e sua estratégia propõe que os dados complexos (gramáticas e metamodelos) sejam representados por meio de hipergrafos direcionados, permitindo que um algoritmo de percurso em hipergrafos, usando combinatória, criem subconjuntos dos modelos de entrada que serão utilizados como casos de teste para as transformações M2T.

Nesta perspectiva, realizou-se dois estudos exploratórios com propósito específico da análise de viabilidade quanto a abordagem proposta. O primeiro estudo denominado "Estudo Piloto" além de permitir a análise dos dados, subsidiou os aprimoramentos e ajustes necessários ao longo do processo de investigação. Já o segundo estudo permite consolidar as observações e conclusões por meio de uma avaliação dos resultados.

1.4 Organização

Neste capítulo foram apresentados o contexto no qual esta dissertação de mestrado está inserida, juntamente com a motivação e os objetivos pretendidos. No Capítulo 2 é feita uma revisão bibliográfica do trabalho, relacionando os temas pertinentes ao desenvolvimento do trabalho como os principais conceitos e técnicas do MDD, técnicas de teste de software, definições e classificações que fundamentam dados complexos e a representação por meio de hipergafos. No Capítulo 3 são discutidos o teste em transformações de

modelos e suas linhas gerais de estudos e aplicação. Adicionalmente, é apresentada uma revisão sistemática da literatura sobre testes em transformações M2T. No Capítulo 4 é proposta uma abordagem para caracterização de dados complexos baseada em hipergrafos. Já no Capítulo 5 são apresentados, de forma detalhada, os dois estudos exploratórios realizados, contendo desde a descrição de cada um a análise dos resultados obtidos. Por fim, no Capítulo 6 são apresentadas as conclusões, contribuições e trabalhos futuros.

Capítulo 2

Fundamentação Teórica

2.1 Considerações Iniciais

Neste capítulo são apresentados sucintamente os principais temas relacionados à fundamentação teórica desse trabalho. Na Seção 2.2, são apresentados os conceitos fundamentais relacionados ao MDD. Em seguida, na Seção 2.3, são apresentados os conceitos sobre transformadores de modelos. Ainda na Seção 2.3 são enfatizados os tipos de transformações de modelo para modelo (*Model-to-Model* ou M2M) e modelo para texto (*Model-to-Text* ou M2T), que serão foco da proposta apresentada neste trabalho, em particular as transformações M2T.

Na sequência são abordados as principais técnicas e critérios de teste inicialmente propostos para programas procedimentais, que passaram a ser investigadas no contexto de *software* desenvolvido sob outras abordagens, como por exemplo, o MDD. Nas Seções 2.4 e 2.5, são apresentados os conceitos básicos das técnicas tradicionais relacionados à atividade de teste de software. A Seção 2.6 de modo particular apresenta o embasamento teórico para caracterização de dados complexos e uma estratégia de representação destes dados utilizada neste trabalho por meio de hipergrafo.

2.2 Desenvolvimento Dirigido por Modelos

Um modelo é um conjunto coerente de elementos formais que descreve alguns objetos tais como: sistema, banco, telefone ou carro, construído com alguma finalidade, passível de uma forma particular de análise (MELLOR; CLARK; FUTAGAMI, 2003). Cada modelo é determinado pela especificação de um domínio que compartilha um determinado conjunto

de características. Da mesma forma, o domínio é definido por um conjunto de características que descreve uma família de problemas, para os quais uma determinada aplicação pretender conceber uma solução (PRIETO-DIAZ; ARANGO, 1991).

Modelos foram e são de grande importância nos contextos científicos. Basta pensar em física, química ou matemática: modelo padrão da física de partículas ou modelo atômico, são provavelmente simplificações da realidade, mas ao mesmo tempo são fundamentais para a compreensão dos conceitos básicos de teorias complexas, ainda que nunca sejam aplicadas no mundo real. A matemática e suas descrições formais têm sido tão importantes para todas as áreas da ciência, quanto a modelagem ou construção de modelos (JIH; REEVES, 1992).

O Desenvolvimento Dirigido por Modelos (*Model Driven Development* ou MDD) é uma continuação dessa tendência. Em vez de exigir que os engenheiros de *software* aprendam a soletrar cada detalhe de uma aplicação por meio das linguagens de programação, o MDD propõe níveis de abstração mais elevados, visando automatizar tarefas complexas de programação. Exemplos de tarefas complexas são: apoio à persistência, interoperabilidade e distribuição de aplicações que ainda são desenvolvidas manualmente (ATKINSON; KUHNE, 2003).

2.2.1 Conceitos do Desenvolvimento Dirigido por Modelos

Em Engenharia de Software, o MDD é um paradigma de desenvolvimento de *software* em que os principais artefatos de *software* são os modelos a partir dos quais o código ou outros artefatos são gerados. Do ponto de vista computacional um modelo é uma descrição de um sistema por meio de uma definição particular, omitindo detalhes irrelevantes para que as características mais relevantes sejam vistas com maior clareza (SWITHINBANK et al., 2005).

Muitas vezes essa definição particular é descrita por uma combinação de elementos gráficos e textuais, que seguem uma especificação formal, tendo como base uma linguagem com sintaxe e semântica bem definidas, chamada de linguagem de modelagem. Esta modelagem proporciona um conjunto de benefícios adicionais, sendo eles: validação sintática; verificação do modelo; simulação de modelos; transformações do modelo; execução e depuração do modelo (BRAMBILLA; CABOT; WIMMER, 2012).

Modelagem é a concepção de aplicações de *software* antes da codificação, essencial em projetos de pequeno, médio e grande porte. Um modelo desempenha um papel análogo no

desenvolvimento de *software* que plantas arquiteturais e outras representações, assim como mapas ou modelos físicos desempenham na construção de um edifício. A modelagem eleva o nível de abstração, possibilitando uma forma eficiente de visualizar o projeto e verificá-lo, quanto a especificação dos requisitos do domínio antes de implementar o código (OMG, 2011).

Na engenharia de *software* a linguagem de modelagem mais utilizada pelos profissionais da área é a *Unified Modeling Language (UML)* (BOOCH; RUMBAUCH; JACOBSON, 2005). A UML é definida pelo *Object Management Group (OMG)* e, atualmente, encontra-se na versão 2.4.1 de sua especificação.

De acordo com Brambilla, Cabot e Wimmer (2012), a necessidade de contar com os modelos de desenvolvimento de *software* é pautada em quatro fatores principais:

1. Os artefatos de *software* tornam-se cada vez mais complexos, portanto, precisam de concepções com diferentes níveis de abstração, dependendo do perfil dos atores envolvidos, da fase do processo de desenvolvimento e dos objetivos do projeto.

2. O *software* está presente na vida das pessoas. A expectativa é que a necessidade de novas soluções de *software* ou a evolução dos já existentes aumentem continuamente.

3. O mercado de trabalho sofre uma escassez contínua de profissionais com competências e habilidades no desenvolvimento de *software*.

4. Desenvolvimento de *software* é uma atividade colaborativa: muitas vezes impõe interações com *stakeholders* (por exemplo, clientes, gerentes, agentes de negócios, etc), que precisam de alguma mediação na descrição dos aspectos técnicos do desenvolvimento. A modelagem é uma ferramenta útil para lidar com todas essas necessidades.

2.2.2 Níveis de Abstração de Modelagem

Este tópico visa esclarecer que a modelagem pode ser aplicada em diferentes níveis de abstração, podendo até mesmo modelar seu próprio modelo. Na Figura 2.1 ilustra-se uma visão geral das principais características do MDD e resume como as questões primordiais são abordadas. Basicamente essas questões são organizadas em duas dimensões ortogonais: Conceituação (colunas da figura 2.1) e a implementação (linhas da figura 2.1).

As questões que tratam da implementação abordam o mapeamento dos modelos de

alguns sistemas já existentes ou futuros e são definidas por três importantes aspectos (BRAMBILLA; CABOT; WIMMER, 2012):

1. O *nível de modelagem* nos quais os modelos são definidos/descritos;
2. O *nível de realização* nas quais as soluções são implementadas por meio de artefatos que estão realmente sendo utilizados em ambientes computacionais, ou seja, plataformas típicas que incluem uma arquitetura de hardware, um sistema operacional e uma biblioteca de execução; e
3. O *nível de automação* no qual os mapeamentos da modelagem são transformados em artefatos(linhas de código).

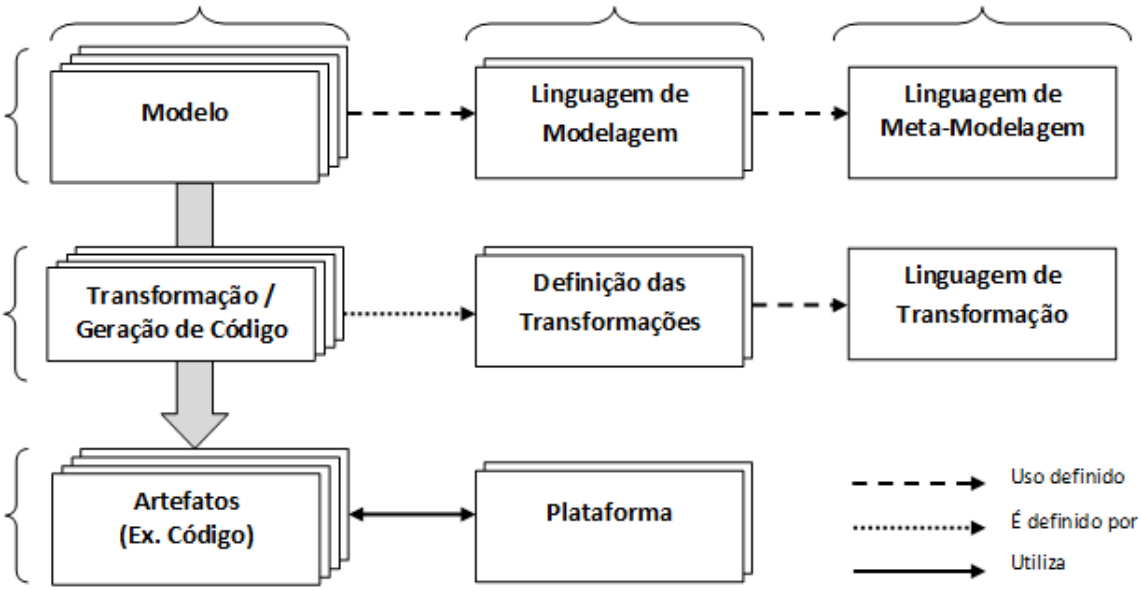


Figura 2.1: Estrutura da Metodologia MDE - (BRAMBILLA; CABOT; WIMMER, 2012)

A conceituação conduz à definição dos modelos conceituais a uma descrição da realidade. Podendo ser categorizada por três níveis:

1. nível de aplicação: são definidos os modelos de aplicação, executadas as regras de transformações e os componentes concretos são gerados;
2. nível de domínio de aplicação: são definidas as linguagens de modelagem, as transformações e as plataformas de implementação para um domínio específico; e
3. Metanível: são definidos a conceituação dos modelos e a linguagem de transformação.

O fluxo principal da Figura 2.1 ilustra a concepção do modelo de aplicação até a construção dos artefatos reais de *software*. O nível de automação é basicamente caracterizado pela definição e execução das transformações de modelos. Esta ordem de acontecimentos possibilita a reutilização de modelos para diferentes plataformas. Nas próximas subseções, serão descritos alguns detalhes sobre os níveis de abstração da modelagem, a arquitetura padronizada e as transformações dos modelos.

2.2.3 Linguagens de Modelagem

Cada modelo baseia-se em uma linguagem (formalismo), a qual define com precisão sua sintaxe e semântica. A sintaxe de uma linguagem pode ser entendida como concreta e abstrata. Enquanto a semântica pode ser compreendida como dinâmica e estática.

Como o grande desafio dos transformadores de modelo é refletir toda semântica da metaprogramação para modelo com precisão de sintaxe e semântica (CZARNECKI; HELSEN, 2006), estes ingredientes importantes das linguagens de modelagem, serão discutidos a seguir.

Sintaxe Abstrata O primeiro ingrediente é a definição dos conceitos de modelagem e suas prioridades, por meio da definição de metamodelos que desempenham um papel correspondente às gramáticas para linguagens textuais. Enquanto a gramática define todas as sentenças válidas de uma linguagem, o metamodelo define todos os modelos válidos de uma linguagem de modelagem. Os metamodelos são descritos com as linguagens de metamodelagem, que fazem uso do núcleo das linguagens de modelagem estruturais orientadas a objetos. O termo metamodelagem, teve origem quanto ao uso de linguagens de modelagem que representassem uma modelagem. Esse uso pode ser recursivo desde que esteja em conformidade com os níveis de abstração do domínio. Assim define-se a meta-metamodelagem. Portanto, como os modelos são sempre uma abstração de metamodelos, metametamodelos somente poderão ser definidos por sintaxes abstratas das linguagens que os representam. A linguagem de modelagem UML, possui algumas limitações quanto as suas representações estruturais, quando se trata de restrições para o modelo. Para adicionar restrições mais complexas as regras de validação do modelo pode-se utilizar uma linguagem de restrições. A OCL (*Object Constraint Language* ou Linguagem para Especificação de Restrições em Objetos) (OMG, 2012), por usar similaridade e integração com a UML é utilizada em metamodelos.

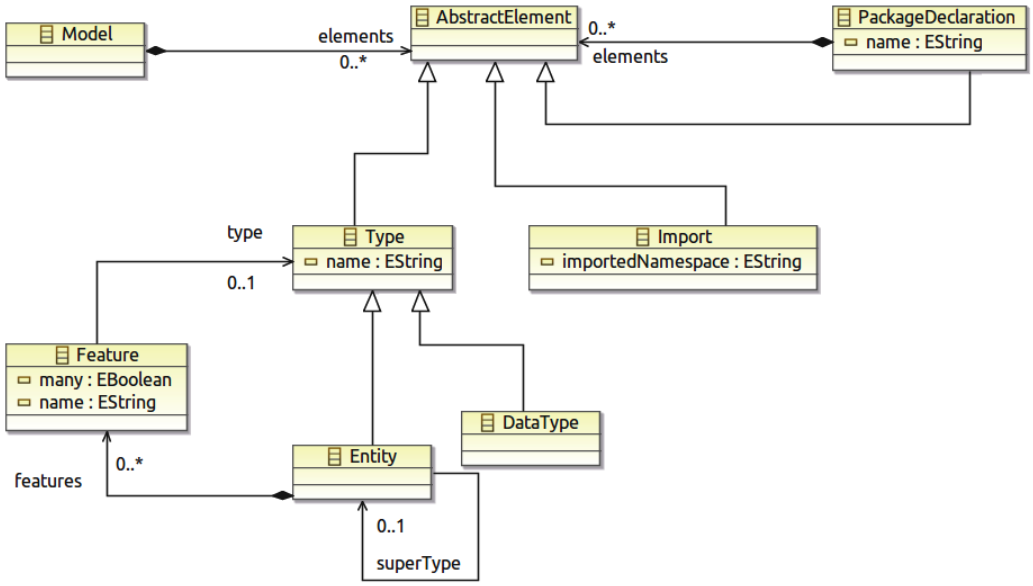


Figura 2.2: Sintaxe Abstrata - Metamodelo

Na Figura 2.2 ilustra-se a sintaxe abstrata por meio de *Frameworks* de metamodelagem que permitem a especificação de metamodelos usando editores dedicados, bem como a geração de editores de modelagem para definir e validar os modelos. Isso significa que os metamodelos são empregados: a) de forma construtiva, interpretando o metamodelo como um conjunto de regras de produção para modelos e b) analiticamente, interpretando o metamodelo como um conjunto de restrições de um modelo que tem de representar sua conformidade com o metamodelo (ATKINSON; KUHNE, 2003).

Sintaxe Concreta Como dito anteriormente, metamodelos definem apenas a sintaxe abstrata, e não possuem a capacidade de notação da linguagem concreta. Esta é uma diferença importante entre gramáticas baseadas em (*Extended Backus Naur Form* ou EBNF) que, por meio da sintaxe concreta das linguagens textuais definem todos elementos. Para as linguagens de modelagem, artefatos adicionais que se referem aos elementos do metamodelo são utilizados para especificar sua sintaxe concreta. Com esta separação entre sintaxe abstrata e concreta de uma linguagem, várias sintaxes concretas podem ser definidas para uma linguagem de modelagem por meio de elementos gráficos ou textuais.

Semântica Define o significado dos elementos da sintaxe abstrata, e pode variar de acordo com o objeto desejado. No contexto do MDD, a semântica é definida em forma de ações a serem executadas por um interpretador automático. As abordagens para o tratamento da semântica MDD podem se enquadrar em ao menos

quatro categorias, como por exemplo: Denotativa, Operacional, Tradutiva e Pragmática (KLEPPE, 2007). Com isto, as regras semânticas podem ser classificadas em estáticas ou dinâmicas dependendo do momento em que são checadas (compilação ou em tempo de execução). A semântica estática mais precisamente pode ser denominada como *well-formedness* ou regras de boa formação, na qual está implícita a semântica dinâmica que representa as restrições ao conjunto de modelos válidos que podem ser expressos por meio de um formalismo (BEYDEDA; BOOK; GRUHN, 2005). Esses detalhes podem ser observados na Figura 2.3.

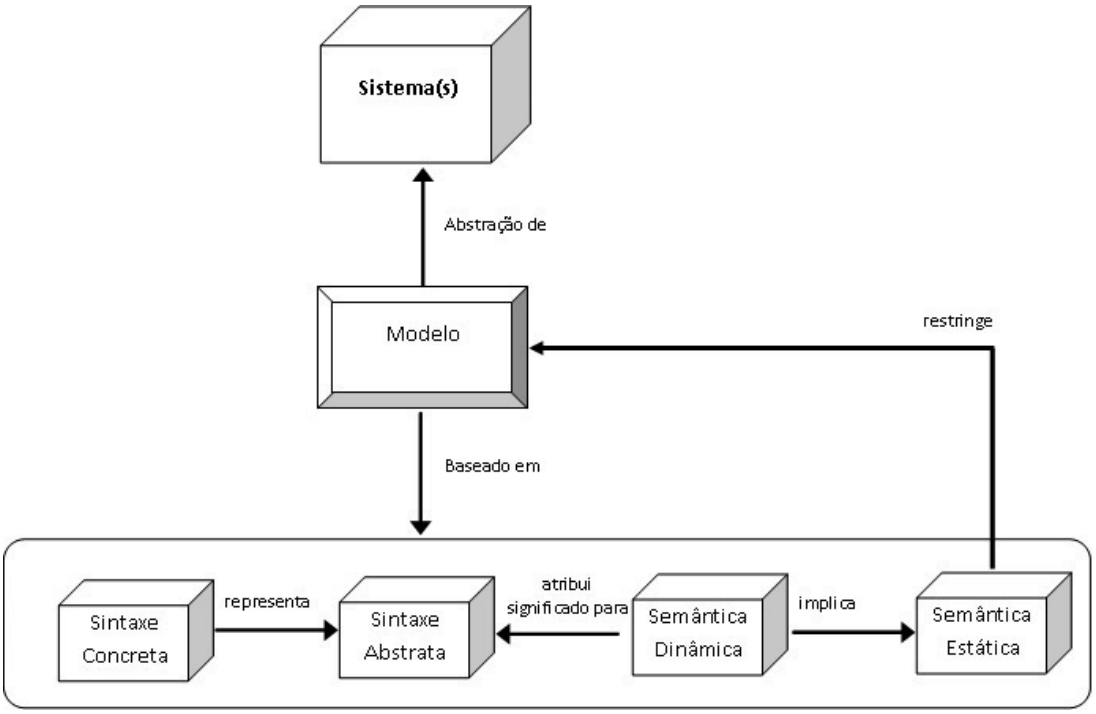


Figura 2.3: Sistemas, Modelos e Linguagens - Adaptado de (CAPLAT; SOURROUILLE, 2005)

As linguagens de modelagem são ferramentas que permitem a definição de uma representação concreta de um modelo conceitual, podendo consistir em representações gráficas e/ou descrições textuais. Nesse caso, elas são formalmente definidas e permitem a especificação de um projeto conforme a sintaxe da modelagem. De acordo com Harel e Rumpe (2004), duas grandes classes de linguagens podem ser descritas :

- Linguagens específicas de domínio (*Domain Specific Language* ou DSL) são linguagens projetadas especificamente para um determinado tipo de domínio, contexto ou negócio para facilitar as tarefas de pessoas que precisam descrever objetos nesse domínio.

- Linguagens de modelagem de uso geral (*General-Purpose Modeling Language* ou GPML), representam as ferramentas que podem ser aplicadas a qualquer negócio ou domínio para fins de modelagem. Um exemplo típico já citado na seção 2.2.1 é a linguagem UML.

2.2.4 Metamodelagem

A definição da linguagem normalmente requer um metamodelo que seja capaz de capturar os pontos comuns e variáveis do domínio, e não a mais comumente utilizada BNF. Isto ocorre pelo simples fato de que regras gramaticais BNF descrevem uma árvore, enquanto um metamodelo descreve um grafo, e árvores são subconjuntos de grafos (KLEPPE, 2007). Um metamodelo é uma estrutura similar a um diagrama de classes, e possui elementos como classes, atributos, associações e agregações. Assim, metamodelos constituem a definição de uma linguagem de modelagem, uma vez que proporcionam uma metodologia que descreve toda a classe de modelos que podem ser representados.

A *Meta Object Facility* (MOF) é uma tecnologia adotada pelo OMG para definição de metadados. Metadados é um termo geral para dados que descrevem informações. As informações, assim descritas, podem representar um sistema de computador, base de dados ou instâncias de programas em execução (OMG, 2008a). Em um sentido prático o MOF seria para modelos o que a BNF é para como uma notação para as gramáticas de linguagens de programação.

A MOF suporta tipos de metadados que podem ser descritos por meio de técnicas de modelagem de objetos. Esses metadados possibilitam a descrição de qualquer aspecto de um sistema e as informações nele contidas podendo, assim, descrever qualquer nível rico em detalhes, dependendo da especificação dos requisitos (OMG, 2003b).

Na Figura 2.4 ilustra-se a infraestrutura MOF de quatro camadas que sustentam as primeiras gerações da metodologia MDD. Esta infraestrutura consiste em uma hierarquia de camadas do modelo, na qual cada camada é caracterizada por ser uma instância de uma camada acima, com exceção da camada **M3**. A camada inferior **M0**, representa os dados reais, ou seja, objetos que são descritos e manipulados pelos engenheiros de *software*. A próxima camada **M1**, representa o modelo dos objetos por meio dos dados da camada **M0**. A camada **M2**, possui um modelo de informações da camada **M1**. Portanto, a camada **M2** caracteriza-se como um modelo do modelo, ou seja, metamodelo.

Finalmente, a camada **M3**, possui um modelo de informações da camada **M2**, e por

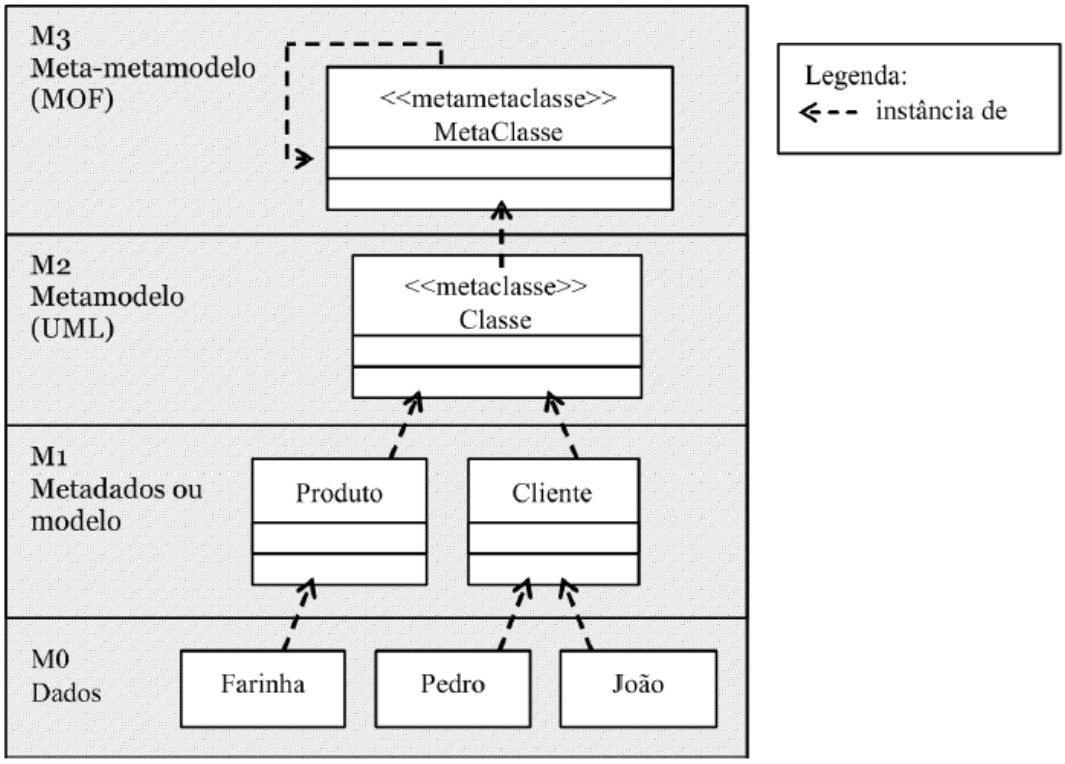


Figura 2.4: Infraestrutura MOF Tradicional de (LUCRÉDIO, 2009)

isso é chamada de meta-metamodelo. A MOF estabelece uma sintaxe abstrata comum para a definição de metamodelos e fornece um gerenciamento de metadados que permitem o desenvolvimento e a interoperabilidade de modelos e metamodelos. Como exemplos, cita-se ferramentas de modelagem e desenvolvimento padronizadas pela OMG, incluindo UML, CWM, SPEM, XMI e JMI (mapeamentos MOF para Java) (OMG, 2008a).

2.2.5 MDA - Arquitetura do OMG para MDD

A *Model Driven Architecture* (MDA) proposta pelo *Object Management Group* (OMG) é uma das concretizações do conceito de MDD, valorizando a importância dos modelos no processo, tornando-os o ponto-chave no desenvolvimento. Ela define que o processo de desenvolvimento de software deve ser direcionado pela atividade de modelagem do sistema, no nível conceitual, independente de qualquer plataforma/implementação, e por meio de transformações realizadas sobre esse modelo conceitual, novos modelos, com níveis de abstração cada vez mais específicos e ligados à implementação sejam gerados. Neste sentindo a MDA define-se como um conjunto de padrões *(standards)* internacionais que facilitam a implementação do MDD, descrevendo os tipos de modelos que podem ser utilizados, como os modelos podem ser preparados e as relações dos diferentes tipos de modelos (OMG, 2003a).

Quando uma nova tecnologia de aplicação está disponível, não é necessário desenhar novamente um sistema completamente do zero, uma vez que os modelos existentes e as regras das transformações horizontais e/ ou verticais correspondentes podem ser reutilizados (BRAMBILLA; CABOT; WIMMER, 2012). Portanto, a MDA concentra-se principalmente na funcionalidade e no comportamento de um sistema ou aplicação distribuída. Assim, independentemente da tecnologia na qual a MDA seja implementada, haverá não somente o encurtamento do ciclo de vida de desenvolvimento de *software*, mas também a melhora da legibilidade do sistema para seus *stakeholders*. Desta forma, os *stakeholders* poderão perceber as vantagens de um projeto composto por modelos, quando comparado àqueles baseados em linguagem de programação específica, os quais possuem uma visão meramente técnica (ZHAO; ZHANG, 2007).

A arquitetura MDA tem como metas três objetivos primordiais: a portabilidade, a interoperabilidade e reutilização. Para apoiar esta abordagem, o OMG definiu um conjunto específico de camadas e transformações que fornecem um quadro conceitual e um vocabulário para MDA. Pode-se notar na Figura 2.5 que o OMG identifica essas camadas como: Modelo Independente de Computação (*Computation-Independent Model* ou CIM); Modelo Independente de Plataforma (*Platform-Independent Model* ou PIM); e Modelo Específico para Plataforma (Platform-Specific Model ou PSM) (OMG, 2003a).

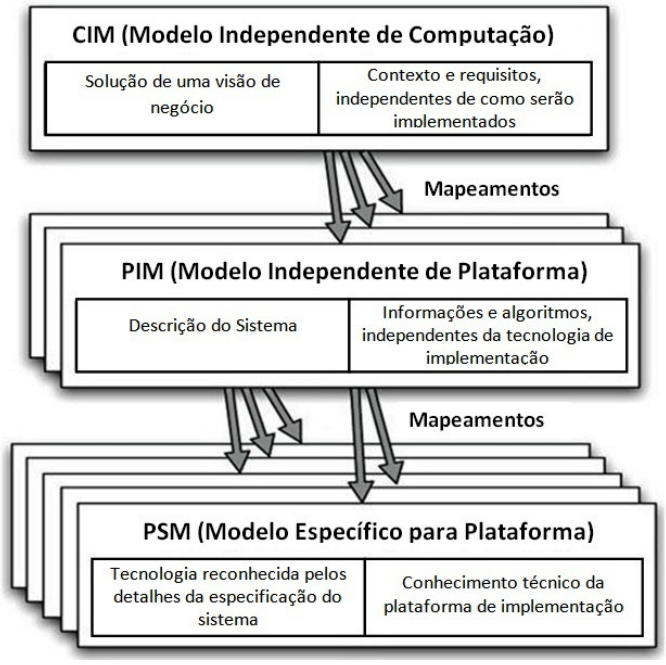


Figura 2.5: Os três níveis de abstração da modelagem - MDA - (BRAMBILLA; CABOT; WIMMER, 2012)

Um conjunto de mapeamentos entre cada nível e suas camadas subsequentes podem ser definidos por meio das transformações do modelo. Tipicamente, cada CIM pode mapear diferentes PIM, que por sua vez, podem ser mapeados para diferentes PSM.

A próxima seção abordará os principais conceitos envolvidos nas transformações de modelos.

2.3 Transformações

A transformação é um tema recorrente na ciência da computação. Afinal, computar dados nada mais é que transformar dados em informações pertinentes. Assim como computar dados básicos - valores numéricos, estruturas de dados como listas e árvores - é o coração da programação, a transformação do modelo tem sido caracterizada como o coração e alma do MDE (SENDALL; KOZACZYNSKI, 2003).

Quando o assunto é transformação de metadados, ou seja, dados que representam artefatos de *software*, esquemas de dados, programas, interfaces e modelos, entra-se nos domínios da metaprogramação. O principal desafio do uso da metaprogramação é fazer com que os metaprogramas respeitem a diversidade semântica dos metadados com que operam. Desta forma, a transformação do modelo é também um processo de metaprogramação e, portanto, deve enfrentar esse mesmo desafio (CZARNECKI; HELSEN, 2006).

A transformação de modelos pode ocorrer entre os mesmos níveis e entre diferentes níveis de abstração e também pode ocorrer entre os mesmos domínios e diferentes domínios. A transformação é a geração automática de um modelo destino, a partir de um modelo origem. A transformação é definida por um conjunto de regras que, juntas, descrevem como um modelo na linguagem origem pode ser transformado em um ou mais modelos na linguagem destino (KLEPPE; WARMER; BAST, 2003).

Na Figura 2.6 são mostrados os principais elementos para essa abordagem e como eles são combinados. Para possibilitar a geração de modelos, é necessária uma ferramenta de modelagem. Os engenheiros de *software*, por meio destas ferramentas produzem modelos que descrevem os conceitos do domínio. Estas ferramentas devem ser intuitivas e de fácil utilização. Ao mesmo tempo os modelos gerados precisam ser semanticamente completos e corretos, uma vez que devem ser compreendidos por transformadores, sensíveis as sutilezas e detalhes dos requisitos do domínio. Para definir estas transformações, faz-se necessária uma ferramenta que possibilite aos engenheiros de *software* a construção de regras de mapeamento de modelo para modelo ou de modelo para texto. Além disso,

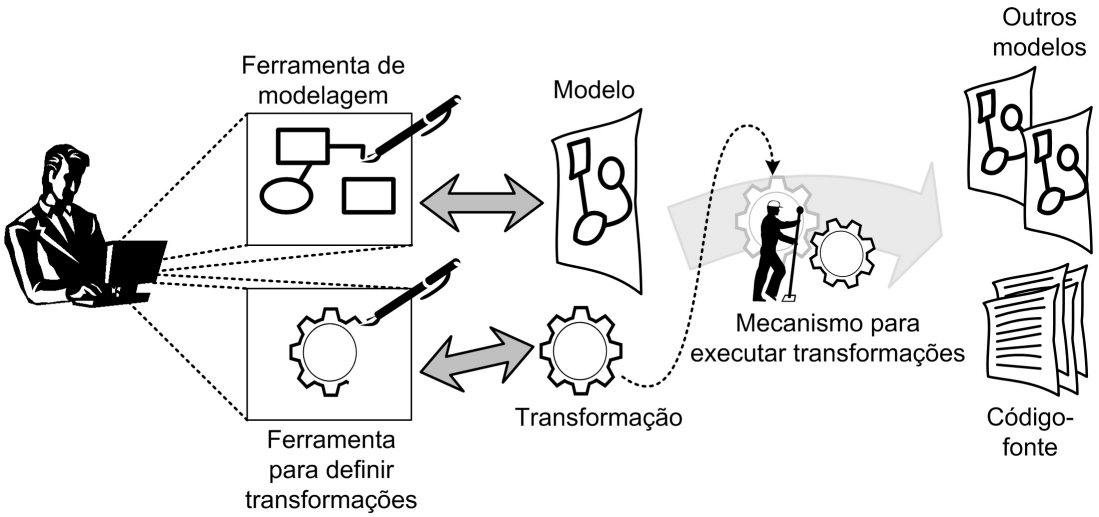


Figura 2.6: Principais Elementos Transformações MDD - (LUCRÉDIO, 2009)

devem possibilitar que as regras de mapeamento sejam especificadas de forma mais natural possível, uma vez que a construção de transformadores é uma tarefa complexa. Por fim, é necessário um mecanismo que aplique as transformações especificadas, e além de sua execução também possibilite a rastreabilidade do mapeamento definido, seja ele no modelo ou no texto (código fonte) gerado (LUCRÉDIO, 2009).

Existem duas atividades principais que acontecem com os modelos: refinamento e transformação (BRAMBILLA; CABOT; WIMMER, 2012). O refinamento do modelo ocorre por meio das mudanças graduais que acontecem para atender às necessidades dos requisitos do domínio. Assim, o modelo é refinado com mais informações, aprimorando os detalhes e aumentando o seu grau de qualidade quanto à fidelidade semântica dos requisitos descritos. Um modelo também pode ser refinado por razões puramente internas (refatoração).

Como vários modelos evoluem, os modelos dependentes também necessitam de aperfeiçoamentos para corresponderem a estas evoluções. Ao final de cada iteração do ciclo de desenvolvimento, todos os modelos devem ser consistentes entre si, mantendo seus mapeamentos. Os refinamentos podem ser manuais ou por meio de alguma forma de automação, assistida ou não. A automação pode ser expressa por meio das regras de refinamento do modelo, implementadas como padrões executáveis ou declarativos (BROWN; CONALLEN; TROPEANO, 2005).

As transformações envolvendo dois ou mais modelos podem ser classificadas como: transformação de modelo para modelo (model to model ou M2M) e transformação de modelo para texto (model to text ou M2T).

2.3.1 Transformações M2M

De modo geral, uma transfromação M2M é um programa que usa um ou mais modelos de entrada para produção de um ou mais modelos de saída (BROS; PIERS, 2013). Na maioria dos casos transformações de um-para-um, com um modelo de entrada e um modelo de saída, são suficientes. No entanto, há situações em que transformações um-para-muitos, muitos-para-um ou até mesmo muitos-para-muitos são necessárias.

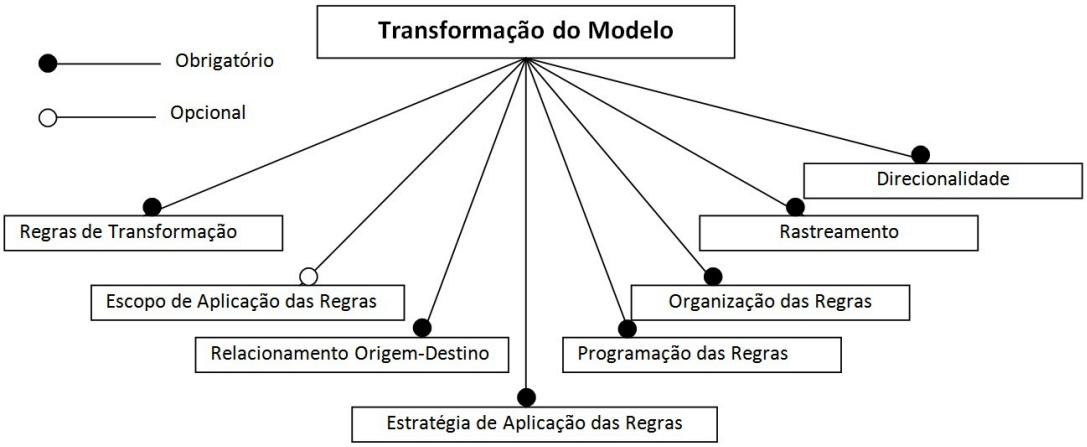


Figura 2.7: Diagrama de Características - Transformação do Modelo - (CZARNECKI; HELSEN, 2006)

Observa-se na Figura 2.7, definida nos trabalhos de Czarnecki e Helsen (2006) e de Mens, Czarnecki e Gorp (2006), uma hierarquia de características comuns e variáveis caracterizando um conjunto de instâncias deste conceito. As características proporcionam uma terminologia e uma representação das escolhas da abordagem do projeto de transformação do modelo.

De acordo com os domínios dos metamodelos Origem e Destino, o Transformador pode viabilizar: i) novos modelos no mesmo domínio, a fim de obter um grau maior de especificidade em relação ao modelo original. Esta transformação é chamada de *endogenous* ou *rephrasings* e; ii) novos modelos em domínios diferentes, a fim de se obter reuso de modelos para criação de novos modelos em diferentes domínios. Esta transformação é chamada de *exogenous* ou *translations* (MENS; CZARNECKI; GORP, 2006). Uma aplicação pode ser gerada automaticamente de uma especificação escrita em uma linguagem textual ou gráfica de um determinado domínio de problemas (CZARNECKI, 2005).

Na Figura 2.8 ilustra-se uma visão geral dos principais conceitos presentes na transformação do modelo para modelo. Esta ilustração demonstra um cenário simples de uma transformação tendo um modelo de entrada e um modelo de saída. Ambos os modelos

estão em conformidade com seus respectivos metamodelos (CZARNECKI; HELSEN, 2006).

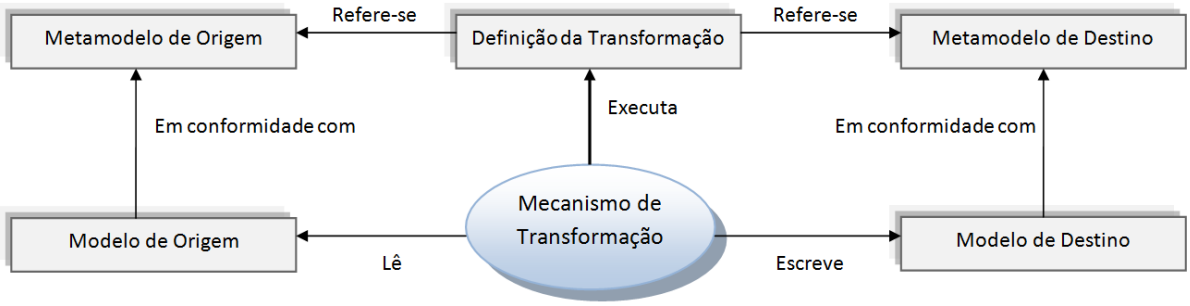


Figura 2.8: Conceitos Básicos da Transformação de Modelo para Modelo

Como as transformações são processos complexos, aplica-se uma técnica definida como Transformação em Cadeias, que permite a modelagem orquestrada de diferentes transformações do modelo. A transformação em cadeia é definida pela orquestração da linguagem de transformação que permite modelar as etapas de transformações mais simples sequencialmente. Transformações em cadeias mais complexas, também podem incorporar desvios condicionais, laços e outras estruturas de controle.

Transformação em cadeias não são apenas um meio para dividir a transformação a ser desenvolvida, mas também permitir a construção de transformações de modelos complexos de transformações já especificadas. Além disso, tratar transformações de forma segmentada em certos aspectos, pode permitir uma maior reutilização.

2.3.2 Transformações M2T

Vários conceitos, linguagens e ferramentas têm sido propostos na última década para automatizar a geração de texto por meio de modelos usando transformações. Um tipo importante de transformação do modelo é o modelo para texto (*model-to-text* ou M2T), que é usado para implementar o código, gerar documentos, implementar modelo serializados, modelo de visualização e exploração, automatizando várias tarefas da engenharia *software* (ROSE et al., 2012).

Na verdade, um dos principais objetivos do desenvolvimento dirigido a modelos está na geração de sistemas de *software* por meio dos modelos (WIMMER; BURGUEO, 2013). Assim, as transformações M2T estão mais preocupadas com a geração de código fonte por meio da transformação destes modelos.

Na Figura 2.9 apresenta-se uma visão simples da abordagem MDD, na qual destaca-se

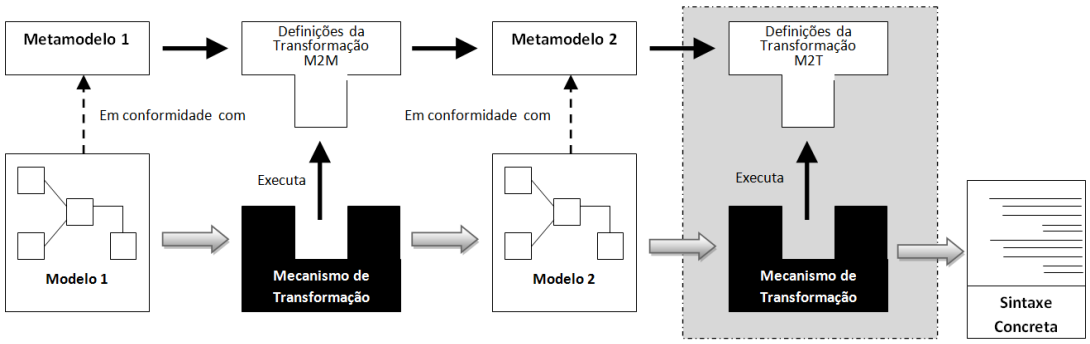


Figura 2.9: Visão Simples da Abordagem MDD Enfase M2T (LEDO; RAMALHO; MELO, 2012)

a etapa das transformações M2T como propósito final da abordagem.

Como a geração de código tem uma longa tradição na engenharia de *software*, deve-se sempre enfatizar a diferença de objetivos entre a geração de código em compiladores e a geração de código do MDD, em que a geração de código é um processo de transformação dos modelos em código fonte (AHO et al., 2006). Uma das questões primordiais é conceber quais partes do código podem ser geradas automaticamente por meio dos modelos e se esta geração é completa ou parcial (BRAMBILLA; CABOT; WIMMER, 2012). A especificação dos requisitos no caso das transformações M2T pode determinar qual a amplitude da transformação, ou seja, quais partes de código podem ser geradas por completo ou parcialmente.

A geração de código pode ser caracterizada como uma transição vertical, pois partindo de modelos em um nível alto de abstração pode-se gerar artefatos de níveis mais baixos. Uma transformação parcial pode significar que uma camada da aplicação é completamente gerada enquanto outra pode ser desenvolvida manualmente (BRAMBILLA; CABOT; WIMMER, 2012). Além disso, pode significar que uma camada é parcialmente gerada e algumas partes precisam ser implementadas manualmente. A geração parcial de código também pode referir-se ao nível do modelo, usando apenas a geração de código para determinadas partes de um modelo, enquanto outras não são tocadas pelo gerador de código e devem ser implementadas manualmente (LARA; GUERRA, 2005).

Basicamente podem ser definidas três abordagens que contemplem essas especificidades. A primeira permite ajustar ao máximo os detalhes de implementação e refinamento, com especificações mais ricas e detalhadas nos modelos para geração de código fonte. A segunda abordagem não demanda esse esforço adicional, porém as implementações e refinamentos adicionais somente poderão ser realizados após a transformação no código fonte gerado. É possível obter uma abordagem híbrida mesclando as duas abordagens anterio-

res, na qual as partes estáticas do modelo podem ser geradas por completo e as partes que exigem maiores detalhes de implementação e refinamento façam uso das gerações parciais.

O uso da geração parcial de código fonte por meio dos modelos, entretanto, demanda cuidados especiais, pois o sincronismo do código gerado com o modelo de origem deve ser preservado. Para tanto, inúmeras técnicas e abordagens são propostas e desenvolvidas. Outra questão tratada em transformações M2T refere-se à forma de implementar o gerador de código. De acordo com Czarnecki e Helsen (2006) e Rose et al. (2012), as abordagens mais comuns para realizar transformações M2T são as baseadas no ***Padrão Visitor (Visitor-Based Approaches)*** e no ***Método Template (Template-Based Approaches)***

- **Padrão Visitor**: representa uma operação a ser realizada sobre elementos da estrutura de um objeto. O Visitor permite que se crie uma nova operação sem que se mude a classe dos elementos sobre os quais ele opera. É um método de separar um algoritmo da estrutura de um objeto.

- **Método Template**: faz uso de *templates* que contêm uma combinação de artefatos de texto e pequenos trechos de código. Ao executar a transformação, o código incluído é executado e realiza uma consulta às informações contidas no modelo de origem e o resultado é inserido no artefato de texto gerado (OMG, 2003a).

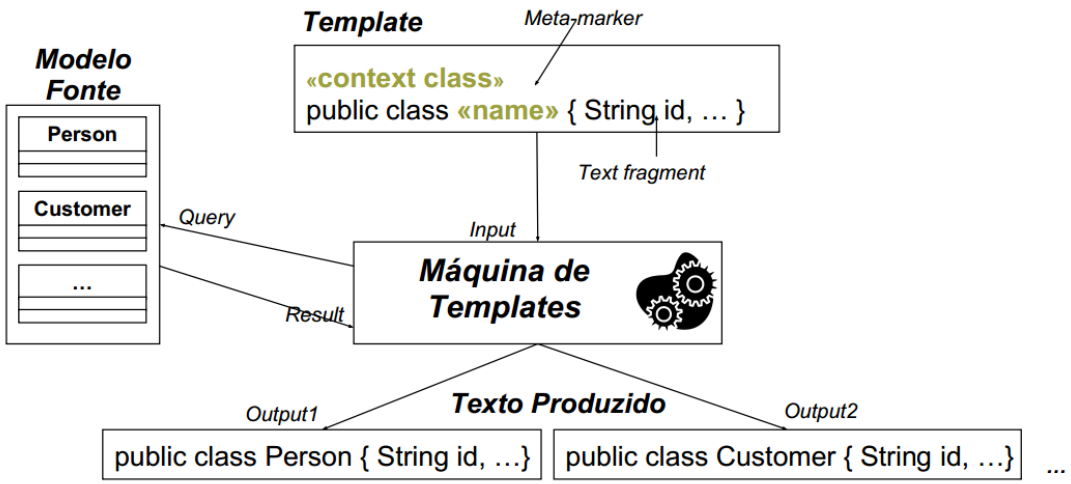


Figura 2.10: Visão geral abordagem baseada em Templates - Adaptado (BRAMBILLA; CABOT; WIMMER, 2012)

Como a arquitetura MDA também define as transformações M2T, o OMG expande a tecnologia MOF já citada na Seção 2.2.4 definindo um padrão para a implementação

de geradores de código por meio do método *templates*. Esse padrão é então denominado MOF *model-to-text* (MOFM2T), que descreve como transformar um modelo em vários artefatos de texto, como por exemplo: código fonte, especificações de implantação de *software*, relatórios e documentos (OMG, 2008b). Na Figura 2.10 ilustra-se uma visão geral de acordo com o padrão MOFM2T.

O MOFM2T apresenta recursos de linguagens com sintaxe concreta e abstrata para especificar transformações de modelos para texto, possibilitando a separação do código estático do dinâmico, a utilização de meta-marcadores e a representação explícita da estrutura do texto de saída e especificações de geração de código mais legíveis e reutilizáveis.

Existem diferentes transformadores que fazem uso do método Template para gerar texto por meio de modelos. Uma característica relevante para estas ferramentas é a capacidade de ler e interpretar os modelos diretamente e serializar os textos produzidos em arquivos. Assim, não é necessária nenhuma redefinição de leitura dos modelos, evitando retrabalho e ajustes manuais.

É possível demonstrar como estes conceitos são implementados na prática fazendo uso do EMF (Eclipse Modeling Framework). A abordagem do EMF permite a manipulação de modelos conforme seu correspondente metamodelo. O EMF segue um meta-metamodelo denominado Ecore, similar ao MOF, padrão estabelecido pela OMG. O Ecore surgiu como uma implementação do MOF, mas evoluiu para uma forma mais eficiente, a partir da experiência obtida após alguns anos na construção de ferramentas (STEINBERG et al., 2009).

Primeiramente utiliza-se o XText (ECLIPSE.ORG, 2013b) para desenvolver uma abordagem MDD usando a metodologia denominada "Grammar First" (BETTINI, 2013; BRAMBILLA; CABOT; WIMMER, 2012), inspirada pela EBNF esta abordagem inicia a definição da linguagem por meio do desenvolvimento da gramática que define a sintaxe abstrata e concreta de uma vez como uma única especificação, conforme o Código 2.1. Em uma etapa posterior, o metamodelo pode ser gerado automaticamente a partir da gramática, como ilustra-se na Figura 2.11.

```
grammar br.org.cursomde.modelo.Dsl with
org.eclipse.xtext.common.Terminals

generate dsl "http://www.org.br/cursomde/modelo/Dsl"

 Model:
   (elements += AbstractElement)*
;
PackageDeclaration:
   'package' name = QualifiedName '{'
     (elements += AbstractElement)*
   '}'
;
AbstractElement:
   PackageDeclaration | Type | Import
;
Feature:
   (many ?= 'many')? name = ID ':' type = [Type | QualifiedName]
;
Entity:
   'entity' name = ID
                ('extends' superType = [Entity | QualifiedName])?
   '{'
     (features += Feature)*
   '}'
;
DataType:
   'datatype' name=ID
;
Type:
   DataType | Entity
;
QualifiedName:
   ID ('.' ID)*
;
Import:
   'import' importedNamespace = QualifiedNameWithWildcard
;
QualifiedNameWithWildcard:
   QualifiedName '.*'?;
```

Código 2.1: Sintaxe Concreta/Abstrata representada por meio da Linguagem XText

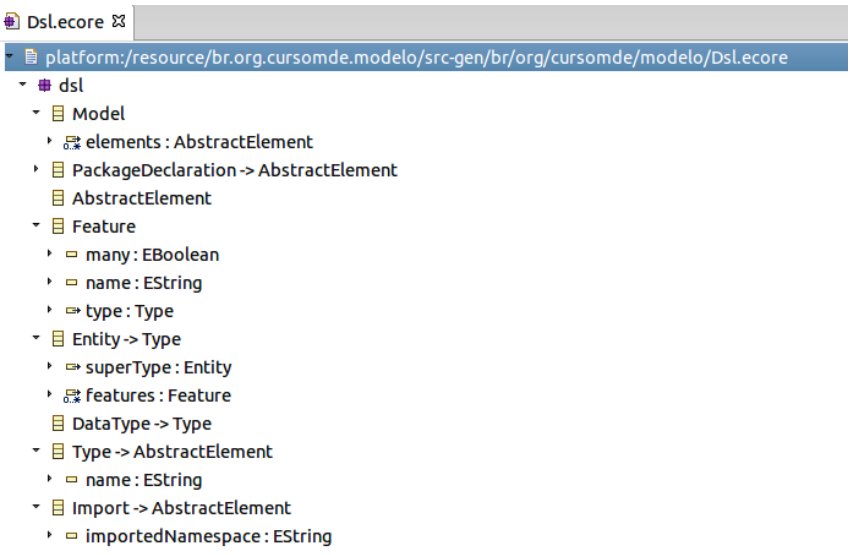


Figura 2.11: Metamodelo Ecore gerado por meio da Gramática XText

```
1  package java.lang {
2    datatype String
3  }
4
5  datatype int
6  datatype double
7
8  package java.util {
9    datatype Date
10 }
11
12   package bank.customer
13   {
14     import java.lang.*
15     import java.util.*
16     import bank.common.*
17
18     entity Customer {
19       nome : String
20       fone : String
21       cel  : String
22     }
23
24     entity Personal extends Customer {
25       cpf : int
26       dataNasc : Date
27     }
28
29     entity Institucional extends Customer {
30       cnpj : int
31       dataCriacao :Date
32     }
33   }
34
35   package bank.account
36   {
37     import java.lang.*
38     import java.util.*
39     import bank.common.*
40
41     entity Account {
42       numero : int
43       saldo : double
44     }
45
46     entity Savings extends Account {
47       percRendimento: double
48       valorAplicado : double
49     }
50
51     entity Current extends Account {
52       limite :double
53     }
54   }
55
56   package bank.common
57   {
58     import java.lang.*
59     import java.util.*
60
61     entity Bank {
62       codigo : int
63       nome : String
64       cnpj : int
65     }
66
67     entity Branches
68     {
69       prefixo: String
70       nome : String
71       fone : String
72     }
73   }
```

Código 2.2: Modelo definido em conformidade com Metamodelo

Após definido o metamodelo Ecore, define-se o modelo em conformidade com seu domínio de aplicação com suas entidades e propriedades inerentes, conforme o Código 2.2. Concluídas as etapas anteriores é possível executar transformação do modelo para texto.

Para concluir a etapa de geração de código é necessário definir qual linguagem de transformação será utilizada. Assim, como exemplo, foram utilizados dois transformadores em conjunto com o EMF, sendo eles:

- Xpand: Possibilita representações XML de transformações similares a scripts Ant (BAILLIEZ et al., 2013), encadeadas em um workflow. Apresenta uma sintaxe própria mínima, baseando-se na linguagem Xtend (ECLIPSE.ORG, 2013a) e em expressões de outras linguagens para completar sua sintaxe e semântica. Também provê extensões para programação orientada a aspectos.

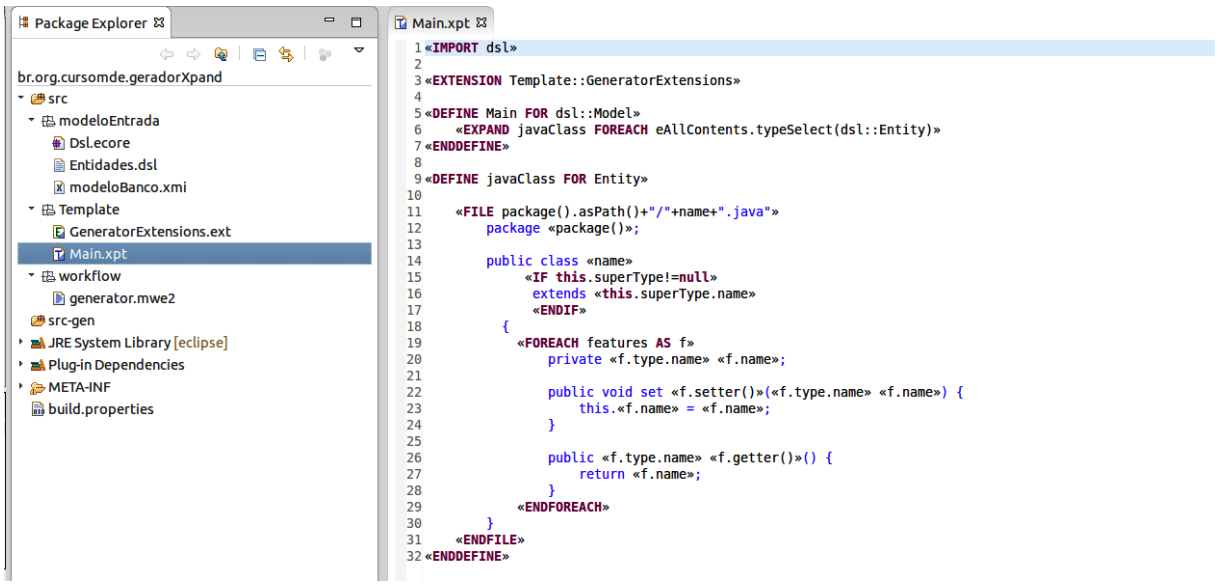


Figura 2.12: Projeto Gerador XPand

Ao analisar a Figura 2.12 é possivel identificar o modelo de entrada, o *template* com seu código exibido em detalhes do lado direito e o workflow no qual é configurado e engatilhada a transformação pela IDE.

- Java Emitter Templates (JET): Mecanismo tradicional de *templates* baseado no Java Server Pages (JSP) (Oracle, 2010) usado como padrão pelo próprio EMF. Os *templates* podem ser editados de forma bastante simples para gerar qualquer tipo de artefato, como Java, HTML e XML.

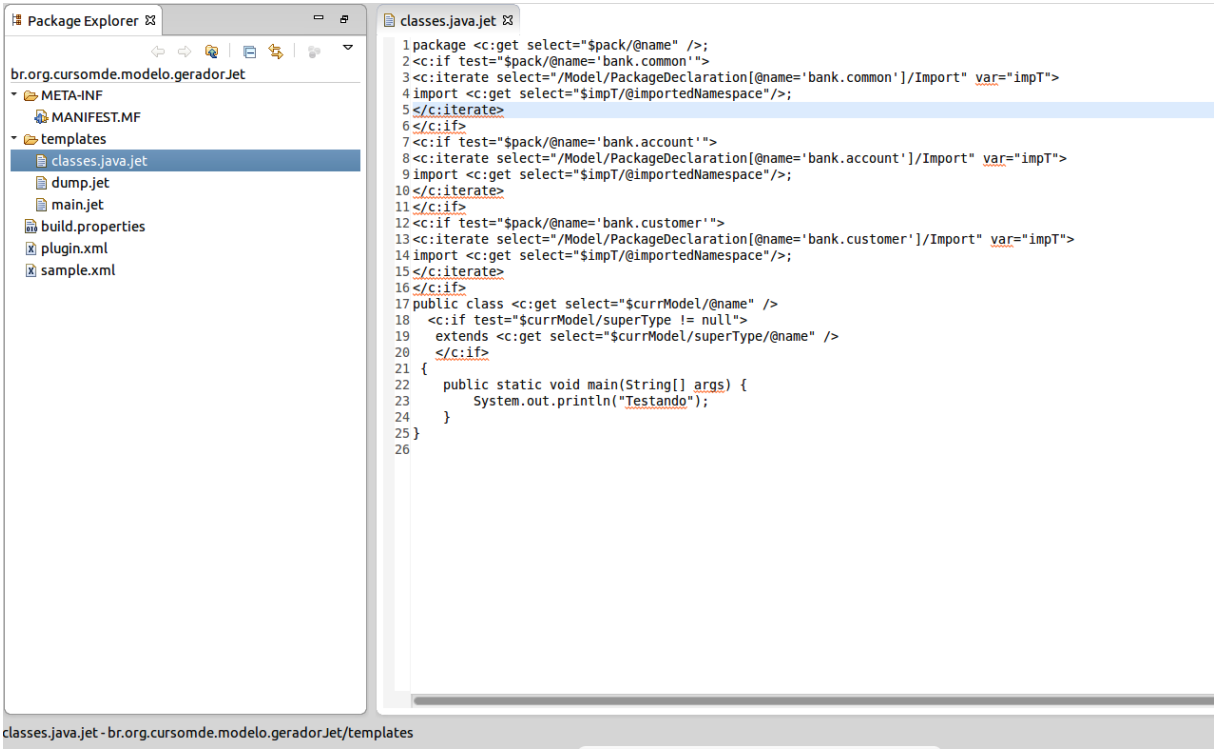


Figura 2.13: Projeto Gerador JET

Após a definição de cada Transformador (XPand e JET) e respeitada suas peculariedades, é possível observar na Figura 2.14 o projeto gerado após executadas as transformações M2T. O Código 2.3 ilustra com maiores detalhes a sua correspondência com o modelo especificado.

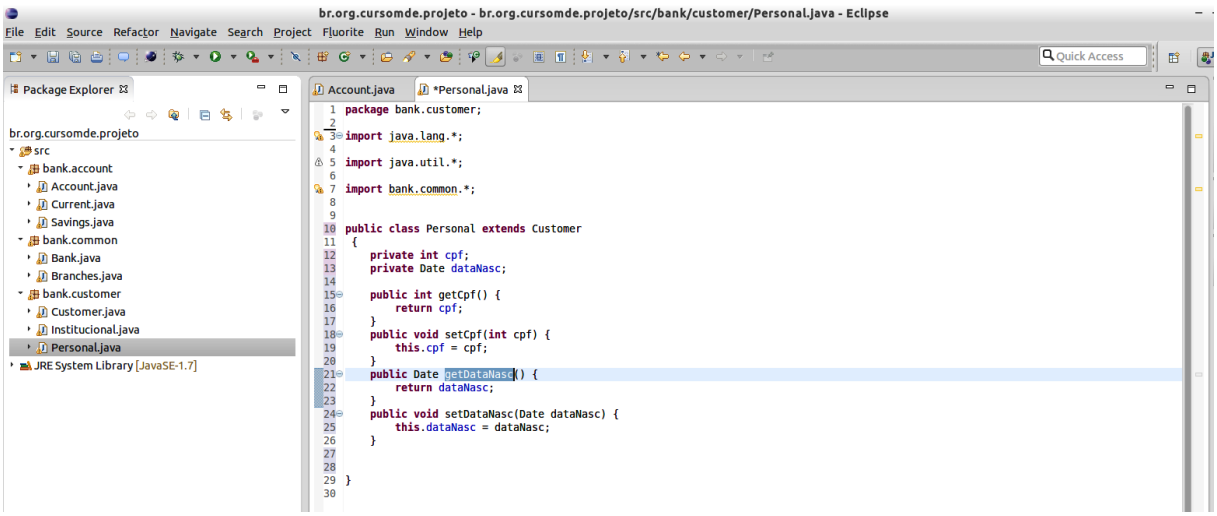


Figura 2.14: Código Gerado por Meio dos Transformadores XPand / JET

```
1  package bank.customer;
2
3  import java.lang.*;
4
5  import java.util.*;
6
7  import bank.common.*;
8
9
10 public class Personal extends Customer
11  {
12   private int cpf;
13   private Date dataNasc;
14
15   public int getCpf() {
16     return cpf;
17   }
18   public void setCpf(int cpf) {
19     this.cpf = cpf;
20   }
21   public Date getDataNasc() {
22     return dataNasc;
23   }
24   public void setDataNasc(Date dataNasc) {
25     this.dataNasc = dataNasc;
26   }
27
28
29 }
```

Código 2.3: Classe Personal especificada no Modelo de Entrada para o Transformador

2.4 Teste de Software

O teste de *software* tem como principal objetivo revelar defeitos presentes no *software*, sendo que um teste bem sucedido é aquele que revela a presença de um ou mais defeitos até então não encontrados (MYERS et al., 2004). Apesar de não ser possível por meio da atividade de teste provar que um programa está correto, o teste contribui para aumentar a confiança de que o *software* desempenha as funções especificadas, quando executado de forma sistemática e criteriosa (HARROLD, 2000; WEYUKER, 1996).

Neste trabalho as definições utilizadas para os termos falha, defeito, erro e engano estão alinhadas com terminologia em Engenharia de *Software* estabelecida pelo padrão IEEE 610.12-1990 (IEEE, 1990). Em consonância as definições, Ferrari (2007) afirma que um **defeito** (*fault*) trata-se de um passo, processo ou definição de dados incorretos e que é inserido no *software* devido a um **engano** (*mistake*) cometido durante o desenvolvimento do *software*. A existência de um defeito pode ocasionar a ocorrência de um **erro** (*error*) durante uma execução do programa, que se caracteriza por um estado inconsistente ou inesperado. Tal estado pode levar a uma **falha** (*failure*), ou seja, pode fazer com que o resultado produzido pela execução do *software* seja diferente do resultado esperado.

Fases de teste: Delamaro, Maldonado e Jino (2007) fundamentam que a atividade de teste é dividida em fases com objetivos distintos. De uma forma geral, pode-se estabelecer como fases o teste de unidade, de integração e de sistema, que são realizadas em sequência:

- **Teste de Unidade:** o teste de unidade tem como foco as menores unidades de um programa, que podem ser funções, procedimentos, métodos ou classes. Neste contexto, espera-se que sejam identificados defeitos relacionados a algoritmos incorretos, mal implementados ou simples enganos de programação. Cada unidade é testada separadamente, então o teste de unidade pode ser aplicado à medida que ocorre a implementação das unidades e pelo próprio desenvolvedor, sem a necessidade do sistema estar totalmente finalizado.

- **Teste de Integração:** após testadas as unidades do sistema, a ênfase é dada na construção da estrutura do sistema. À medida que as diversas partes do *software* são colocadas para trabalhar juntas, é preciso verificar a interação entre elas de maneira adequada. Neste caso é necessário um grande conhecimento das estruturas internas e das interações existentes entre as partes do sistema, por isso, o teste de integração tende a ser executado pela própria equipe de desenvolvimento.

- **Teste de Sistema:** após feito o teste de integração e o sistema estar completo, inicia-se o teste de sistema. O objetivo é verificar se as funcionalidades especificadas nos documentos de requisitos estão todas corretamente implementadas, como os requisitos funcionais e não funcionais.

Além dessas três fases de teste, destaca-se ainda o que se costuma chamar de "teste de regressão". Esse teste não se realiza durante o processo "normal" de desenvolvimento, mas sim durante a manutenção de *software*. A cada modificação efetuada no sistema, após a sua liberação, corre-se o risco de que novos defeitos sejam introduzidos.

Processo de teste: Independentemente da fase de teste, existem algumas etapas bem definidas para a execução da atividade de teste. Tais atividades são contempladas por processos de teste em geral. São elas:

1. Planejamento de testes: são definidos como os testes serão aplicados, os objetivos, os métodos e ferramentas adotadas para a realização dos testes (PRESSMAN, 2005);

2. Projeto de casos de teste: é projetada uma estratégia para a construção dos casos de testes com o objetivo de cobrir as condições de satisfação dos critérios estabelecidos;

3. Execução: com ou sem o auxílio de uma ferramenta, os casos de testes construídos na fase anterior são executados no programa em teste e sua saída comparada com a saída esperada. Caso seja encontrado algum defeito, este é documentado;

4. Análise dos resultados dos testes: os resultados dos testes realizados são registrados, organizados e apresentados na forma de relatórios.

Dentre essas atividades, Myers et al. (2004) citam o projeto de casos de teste como sendo a mais importante. O teste completo de um *software* é uma atividade geralmente impraticável devido ao seu alto custo computacional e financeiro. Com isso, o objetivo então passa a ser minimizar a não-completude do teste, e sugere a definição de conjuntos de casos de teste que apresentem alta probabilidade de encontrar a maioria dos erros com um mínimo de esforço e tempo.

Um caso de teste é um par ordenado $(d, S(d))$ tal que d é um elemento de um determinado domínio D, isto é, $d \in D$ e $S(d)$ é a saída esperada para uma dada função da especificação, quando d é utilizado como entrada. Entretanto, uma das principais questões relacionadas à atividade de teste refere-se ao tamanho do domínio de entrada, que pode ser muito grande para certas funcionalidades do *software*. Nesse contexto, critérios

de teste auxiliam o testador a obter subconjuntos desse domínio, reduzindo a quantidade de casos de teste que devem ser criados, além de auxiliarem na avaliação de conjuntos de casos de teste (FRANKL; WEYUKER, 2000).

Cada critério de teste está associado a uma técnica de teste específica, que se baseia em diferentes artefatos de *software* para derivar os requisitos de teste. Os critérios são utilizados como diretrizes para delimitar a quantidade de requisitos, determinando os elementos e características do *software* que devem ser exercitados. As principais técnicas e seus critérios são apresentados na próxima seção.

2.5 Técnicas e Critérios de Teste

Diferentes técnicas e critérios de testes podem ser empregados durante o processo de desenvolvimento de *software*, diferindo entre si primordialmente pelo tipo do artefato utilizado para criar um modelo que representa o *software* em teste. Exemplos destes artefatos são uma especificação de requisitos, um grafo de fluxo de controle, um diagrama de sequência UML e uma taxonomia de defeitos.

Uma ou mais técnicas de teste podem ser aplicadas em cada uma das fases de teste. Dentre as técnicas mais empregadas pela indústria de *software* e mais investigadas pela comunidade científica, destacam-se as técnicas funcional, estrutural e baseada em defeitos.

A seguir serão descritas cada uma dessas técnicas e os critérios utilizados para avaliação e construção dos casos de testes. O Teste Estrutural terá uma subseção específica, tendo em vista sua relevância quanto a situação problema.

Teste Funcional

O teste funcional é uma técnica utilizada para projetar casos de teste na qual o *software* é uma **caixa-preta** (MYERS et al., 2004). Para testá-lo, são fornecidas entradas e avaliadas as saídas geradas para verificar se estão em conformidade com os objetivos especificados. Nessa técnica, os detalhes de implementação não são considerados e o *software* é avaliado de acordo com o ponto de vista do usuário.

Em princípio, o teste funcional pode detectar todos os defeitos, submetendo o programa ou sistema a todas as possíveis entradas, o que é denominado *teste exaustivo* (MYERS et al., 2004). No entanto, o domínio de entrada pode ser muito grande ou até mesmo infinito, tornando o tempo da atividade de teste inviável.

Para particionar o domínio de entrada de dados e, consequentemente, viabilizar o

teste em termos de custo e esforço para realização, os seguintes critérios de teste funcional são tipicamente empregados:

- Particionamento de Equivalência: o domínio de entrada de um programa é identificado na especificação e organizado em classes de equivalência válidas e inválidas, sendo que são selecionados casos de testes 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 Valor Limite: tem por objetivo testar os limites superior e inferior de cada classe de equivalência. Esse critério é um complemento do critério do Particionamento de Equivalência, e é motivado pelo grande número de defeitos que tende a ocorrer nesses limites de entrada (PRESSMAN, 2010).

- 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.

Teste Baseado em Defeitos

É uma técnica de teste que utiliza informações sobre os erros mais comuns cometidos em um processo de desenvolvimento de *software*, os quais resultam na introdução de defeitos no mesmo. Parte-se da premissa que esses defeitos comumente inseridos no *software* são os tipos de defeitos que se deseja revelar com os testes, guiando então, o projeto dos casos de testes.

Dois critérios utilizados no teste baseado em defeitos são: *Semeadura de Erros e Análise de Mutantes* (DEMILLO; LIPTON; SAYWARD, 1978).

- Semeadura de Erros: consiste na inserção de uma quantidade determinada de defeitos no *software*, com propósito de derivar um conjunto de casos de testes que sejam capazes de detectar esses defeitos inseridos e obter uma taxa real de defeitos inseridos ou defeitos reais presentes no *software*.

- Análise de Mutantes: consiste na geração de várias versões modificadas (mutantes) de um *software*, com objetivo de revelar defeitos comumente inseridos durante o desenvolvimento. Neste critério de teste, os casos de teste também são avaliados quanto à sua capacidade de detectar esses defeitos.

2.5.1 Teste Estrutural

Pressman (2010) afirma que a técnica estrutural é vista como complementar à técnica funcional, porém baseia-se no conhecimento da estrutura interna da implementação. A técnica estrutural (também conhecida como teste **caixa-branca**) estabelece os requisitos de teste com base em uma dada implementação, requerendo a execução de partes ou de componentes elementares do programa. O teste de caixa-branca usa o conhecimento efetivo da aplicação para a especificação dos testes, com objetivo de proporcionar a máxima cobertura do código com menor esforço possível, por meio da seleção eficiente de casos de teste (GROSS et al., 2009).

A técnica estrutural utiliza como auxílio para definição dos requisitos uma representação do código fonte, denominada Grafo de Fluxo de Controle (GFC). Para representar o programa em um GFC é preciso decompor o mesmo em blocos, isto é, regiões de código sequencial sem qualquer salto de execução. Desse modo, todos os comandos contidos em um bloco, com exceção do primeiro e do último, possuem exatamente um comando predecessor e um sucessor.

A principal dificuldade em aplicar o teste estrutural encontra-se na determinação dos "caminhos não executáveis", que ocorrem quando um caminho não executável é um caminho do GFC que não pode ser coberto por nenhum valor do domínio de entrada.

Com base no GFC, alguns critérios são propostos como critérios baseados em fluxo de controle e critérios baseados em fluxo de dados. Esses critérios são sucintamente apresentados na sequência.

Critérios Baseados em Fluxo de Controle

De acordo com Myers et al. (2004) e Pressman (2010), os critérios baseados em fluxo de controle utilizam apenas as informações do controle da execução do programa, como comandos ou desvios, para determinar quais estruturas são necessárias. Os critérios mais conhecidos dessa classe são:

Todos-Nós: requer que os casos de testes executem ao menos uma vez cada vértice do GFC, ou seja, que todos os comandos sejam executados ao menos uma vez.

Todas-Arestas (ou Todos-Arcos): requer que cada desvio do fluxo de controle do programa seja exercitado pelo menos uma vez, ou seja, que cada aresta do grafo seja coberta.

Todos-Caminhos: requer que todos os caminhos possíveis do GFC sejam executados.

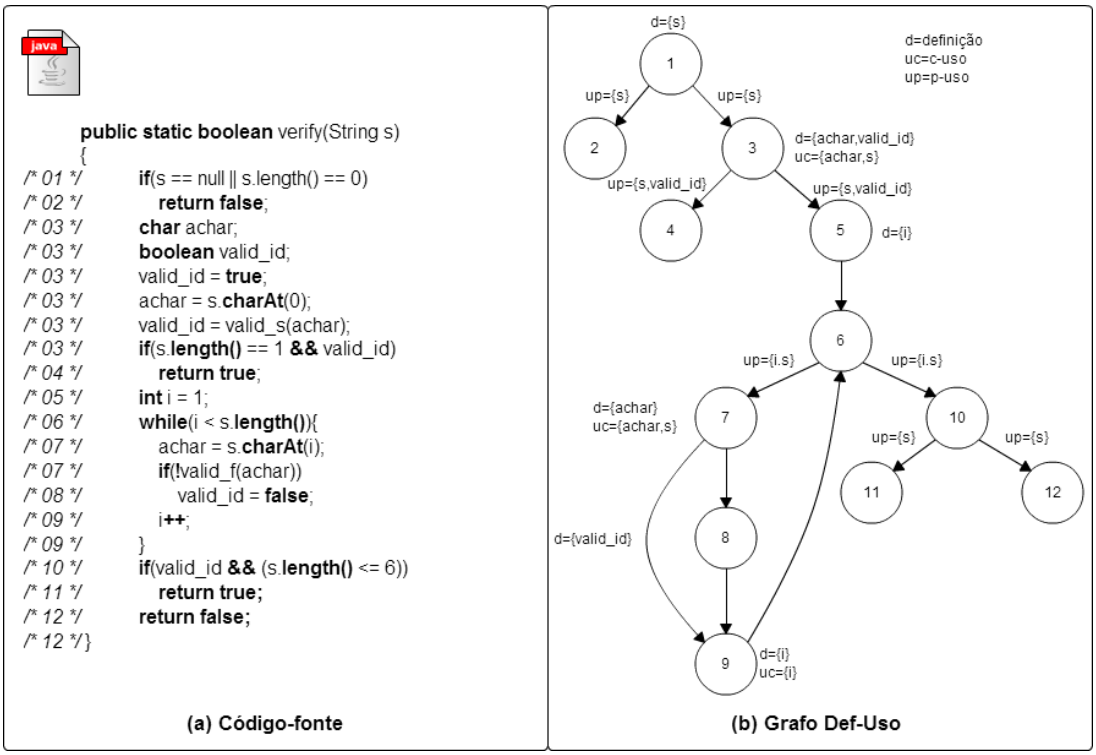


Figura 2.15: Grafo do Fluxo de Controle

É importante ressaltar que a cobertura do critério Todos-Nós é o mínimo esperado de uma boa atividade de teste, uma vez que o programa é executado na intenção de que todos os comandos presentes sejam executados ao menos uma vez. Além disso, executar todos os caminhos do programa é, na maioria dos casos, uma tarefa impraticável, pois na presença de laços, o número de caminhos de um programa pode ser muito grande ou até mesmo infinito.

Critérios Baseados em Fluxo de Dados

A motivação para a introdução dos critérios baseados em fluxo de dados é o fato de que, mesmo para programas pequenos, o teste baseado somente no fluxo de controle não é eficaz para revelar a presença de defeitos (DELAMARO; MALDONADO; JINO, 2007). Foram então propostos os critérios baseados em fluxo de dados, que levam em consideração as definições e usos de variáveis dentro de um programa. Os caminhos no fluxo de controle, que vão desde uma definição de uma dada variável até o uso dessa variável (sem que hajam redefinições do seu valor), constituem os requisitos de teste para esses critérios. Dentre os critérios de fluxo de dados, destacam-se os propostos por Rapps e J.Weyuker (1985) que utilizam uma extensão do GFC denominada Grafo Def-Uso.

Nesse grafo, além das arestas e nós criados no GFC, são inseridas também informações a respeito do fluxo de dados. Com isso, é possível identificar pontos em que um valor é atribuído a uma variável (caracterizando uma definição de variável) e pontos nos quais esses valores atribuídos são utilizados (caracterizando um uso de variável). O uso de uma variável pode ocorrer de duas formas: caso seja utilizado para determinar o desvio do fluxo de controle do programa, ele é chamado de uso predicativo (p-uso). Quando o valor da variável é utilizado em uma computação, ele é classificado como uso computacional (c-uso) (RAPPS; J.WEYUKER, 1985).

A premissa geral é que todo par definição-uso de uma variável deve ser exercitado por ao menos um caso de teste, para aumentar a confiança no correto funcionamento do programa. Os critérios apresentados a seguir têm essas características e fazem parte de duas famílias de critérios propostas por Rapps e J.Weyuker (1985) e por Maldonado (1991):

Todas-Definições: requer que cada variável definida em cada nó do GFC seja exercitada pelo menos uma vez. O critério é baseado na hipótese de que se um valor é atribuído a alguma variável, então essa variável deve ser utilizada posteriormente, caso contrário, não existe a necessidade de tal atribuição (RAPPS; J.WEYUKER, 1985).

Todos-Usos: requer que todas as associações entre uma definição de variável e seus subsequentes usos (c-usos e p-usos) sejam exercitados pelos casos de teste, por pelo menos um caminho livre de definição; ou seja, deve-se criar um conjunto de casos de teste que exercite cada caminho entre a definição de variável e todos os usos dessa variável, antes que ela seja redefinida (RAPPS; WEYUKER, 1982).

Todos-Du-Caminhos (ou Todos-P-Usos e Todos-C-Usos): requer que toda associação entre uma definição de variável e subsequentes p-usos ou c-usos dessa variável seja exercitada por todos os caminhos livres de definição e livres de laço que cubram essa associação (RAPPS; WEYUKER, 1982).

Todos-Potenciais-Usos: requer que os caminhos entre cada variável definida em cada nó do grafo e todos os nós alcançáveis do grafo sejam exercitados, antes que a variável seja redefinida (MALDONADO, 1991). A motivação para esse critério é que a não utilização de uma variável em um determinado ponto do programa pode representar um erro de implementação.

2.5.2 Geração de Dados de Teste

É impraticável testar o *software* com todos os possíveis valores de entrada ou executar todos os seus caminhos. Em geral, os critérios de teste dividem o domínio de entrada em subdomínios e requerem que pelo menos um ponto de cada subdomínio seja executado. Por exemplo, o critério estrutural Todos-Nós agrupa em um subdomínio todas as entradas que executam um determinado nó. A tarefa de geração de dados de teste é considerada um problema indecidível, sendo que existem muitas restrições inerentes às atividades de teste, que impossibilitam automatizar completamente essa etapa (EDVARDSSON, 1999; TIAN; GONG, 2014). Entre elas, destacam-se:

- **Correção Coincidente**: ocorre quando o programa em teste possui um defeito que é alcançado por um dado de teste, um estado de erro é produzido, mais coincidentemente um resultado correto é obtido.
- **Caminho Ausente**: corresponde à uma determinada funcionalidade requerida para o programa, mas que por engano não foi implementada, isto é, o caminho correspondente não existe no programa.
- **Caminhos Não-executáveis**: um caminho é dito não executável quando não existe um conjunto de valores atribuídos às variáveis de entrada do programa, parâmetros e variáveis globais que causam a sua execução. Um elemento requerido por dado critério estrutural é não executável se não existe caminho executável que o cobre.
- **Mutantes Equivalentes**: o programa mutante implementa a mesma função que o programa original e, portanto, não se pode gerar um dado de teste para diferenciá-los. Entretanto, detectar a equivalência destes programas também é, em geral, indecidível.

Apesar das limitações existentes, encontram-se na literatura diferentes categorias de técnicas que podem ser utilizadas para gerar dados de teste para satisfazer os diversos critérios. Dentre elas, as mais conhecidas são:

- geração aleatória de dados: pontos de domínio de entrada são selecionados aleatoriamente até que o critério de teste seja satisfeito (DURAN; NTAFOS, 1984; HAMLET; TAYLOR, 1990; BERTOLINO, 2007).

- geração com execução simbólica: auxilia a geração automática de dados de teste para um dado caminho ou conjunto de caminhos (BOYER; ELSPAS; LEVITT, 1975; RAMAMOORTHY; HO; CHEN, 1976; HOWDEN, 1977; CADAR; SEN, 2013).

- geração com execução dinâmica: está baseada na execução real do programa em teste, em métodos de minimização de funções e análise de fluxo de dados (KOREL, 1990; LI, 2005).

- geração de dados sensíveis a defeitos: fundamentos permitem escolher pontos do domínio de entrada relacionados a certos tipos de defeitos e, por isso, com alta probabilidade de que esses sejam revelados, são exemplos dessa técnica, Teste Domínios (WHITE; COHEN, 1980) e Testes Baseados em Restrições (DEMILLO; OFFUTT, 1991).

- geração utilizando técnicas de computação evolutiva: alguns autores propuseram o uso de técnicas de computação evolutiva para lidar com as limitações da atividade de geração de dados de teste, dando origem a uma nova área de pesquisa chamada Teste Evolucionário (WEGENER; BARESEL; STHAMER, 2001; WEI; RUILIAN; QUNXIONG, 2015).

2.6 Caracterização de Dados Complexos

Na ciência da computação, pode-se definir tipos de dados como um conjunto de valores e de operações que uma variável pode executar. Variáveis estas que indicam ao compilador ou interpretador as conversões necessárias, de acordo com a análise léxica, sintática e semântica, para obter os valores em memória durante a execução do programa.

Os programas construídos na época do desenvolvimento das primeiras versões do Fortran eram basicamente numéricos em sua natureza e simples se comparados com os projetos de *software* recentes. Pode-se fazer uso das evoluções desta linguagem para classificar os dados em ***tipos de dados primitivos*** e ***tipos de dados derivados*** (HAXAIRE et al., 2013). Essa classificação pode ser definida como:

- **Tipos de Dados Primitivos**: são atômicos, no sentido em que não podem ser decompostos em tipos mais simples. Alguns tipos primitivos relacionam-se diretamente à sua representação interna, como é o caso dos números inteiros. Outros, como o tipo caractere, precisam de um pequeno auxílio de hardware ou *software* para serem representados internamente.

O Fortran 77 já categorizava seus tipos de dados primitivos como: inteiros, reais, caracteres, lógicos e ***complexos***. Primordialmente, o uso desta linguagem foi destinada para cálculos científicos, no qual foi definido o tipo de ***números complexos***. Este tipo é concebido como um par de literais, os quais são inteiros ou reais, separados por vírgula "," e contidos entre parênteses "(" e ")". Os literais complexos representam números contidos no conjunto dos números complexos, isto é, números do tipo $z = x + iy$, onde i= raiz(-1), x é a parte real e y é a parte imaginária do número complexo z. Assim um literal complexo deve ser escrito:

(<parte real>, <parte imaginária>)

A necessidade de novas formas de abstração foram forjadas pelo Fortran, dada a necessidade de novas representações algébricas para conjuntos de valores particulares. Um avanço do Fortran 90 com relação ao seu predecessor Fortran 77 consiste na habilidade em definir novos tipos de variáveis, baseados nos tipos primitivos. Estes são os chamados ***tipos derivados ou estruturais***, os quais consistem em combinações de partes compostas por tipos intrínsecos e/ou outros tipos derivados, permitindo gerar estruturas de complexidade crescente.

- **Tipos de Dados Derivados**: são construídos a partir da aplicação de construtores a tipos mais simples. A possibilidade de utilizar construtores de tipos é importante porque permite ao programador definir novos tipos, devidamente voltados à solução do problema tratado. Na Figura 2.16 é apresentado um tipo de dado estruturado, considerado um agrupamento de dados heterogêneo.

```
tipo Aluno ≡ estrutura
    nome: texto
    matrícula: inteiro
    notas: arranjo de 3 real
    média: real
fim
```

Figura 2.16: Pseudocódigo Registro

Os elementos que compõem um registro são organizados em campos: cada elemento de dado que compõe o registro encontra-se associado a um campo do registro, que possui um nome de campo associado utilizado para identificá-lo individualmente.

A evolução computacional e a necessidade de desenvolver programas que contemplem os mais diversos problemas do mundo real tornam essa questão um grande desafio. A

capacidade de modelar problemas, separando os detalhes desnecessários, é chamada de abstração, que é a estruturação de um problema impreciso em entidades bem definidas, por meio da especificação de seus dados e operações.

Os trabalhos de Shlaer e Mellor (1988), Jalote (1989) e Rumbaugh et al. (1991) descrevem o paradigma da orientação a objetos como a programação de tipos abstratos e seus relacionamentos. Assim, como ilustrado na Figura 2.17, pode-se classificar neste processo de modelagem o ***tipo abstrato de dados***.

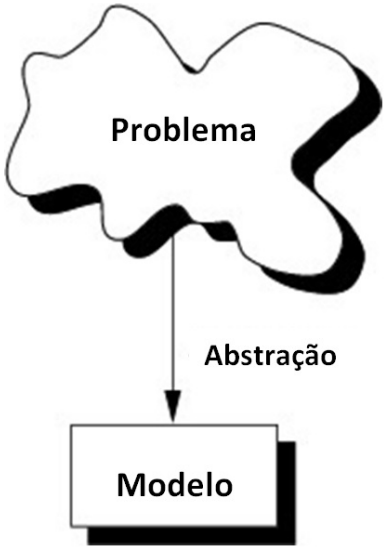


Figura 2.17: Criação de um Modelo por meio da Abstração de um Problema

- **Tipos Abstratos de Dados ou Abstract Data Types**: Um ***ADT*** é um objeto com uma descrição genérica independente dos detalhes de implementação. Esta descrição inclui uma especificação dos componentes a partir dos quais o objeto é feito e também os detalhes do comportamento deste objeto. A estas abstrações incluem-se objetos matemáticos (como números, polinômios, integrais, vetores), físicos (como polias, corpos flutuantes, mísseis), animais (cães, dinossauros, humanos) e objetos (como a pobreza, a honestidade, a inflação), que são abstratos mesmo no sentido de linguagem natural.

  De acordo com os trabalhos de Bergstra e Tucker (1982), semanticamente, um tipo abstrato de dados é identificado como um tipo de isomorfismo, uma álgebra de muitos ordenados $\boldsymbol{A}$ finitamente gerado pelo elemento nomeado na sua assinatura $\boldsymbol{Z}$. Essas estruturas são chamadas álgebras mínimas por não conterem subálgebras adequadas.

  Na Figura 2.18 é apresentado como caracterizar um ADT por suas propriedades: exporta um tipo; exporta um conjunto de operações (chamados de interface); as

operações desta interface são o único método de acesso à estrutura de dados; e os axiomas e precondições algébricas que definem o domínio de aplicação dos tipos.

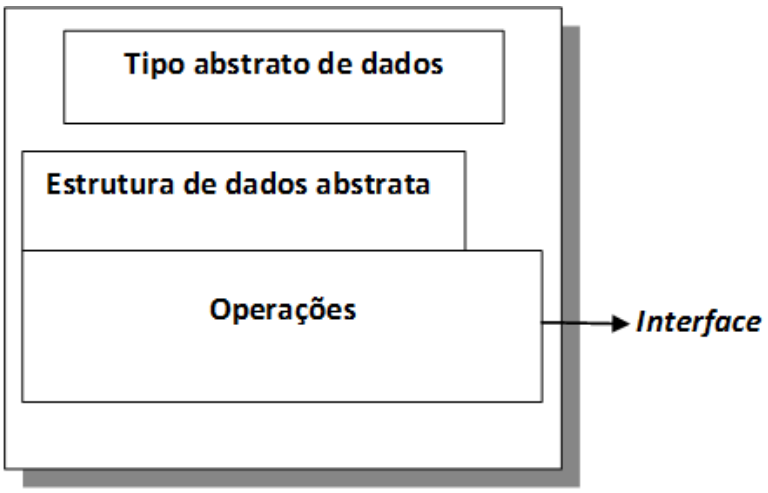


Figura 2.18: Um tipo abstrato de dados - ADT

O modelo define uma visão abstrata para o problema. Isto implica que o modelo objetiva apenas os detalhes relevantes do problema relacionado, delimitando assim, atributos e comportamentos inerentes à abstração do contexto. De acordo com Camardi (2012) os modelos computacionais são baseados principalmente na interação de duas estratégias, uma que visa à construção de tipos de dados e outra destinada à coleta de informações estatísticas. Assim, um modelo computacional pode ser definido por uma estrutura lógica que usa enormes recursos computacionais e processa dados estatísticos abrangentes para o estudo de sistemas complexos que não tem uma solução analítica. Neste sentido, novos arranjos de dados são abstraídos, na qual a teoria da informação desempenha um papel de codificação, avaliação e recodificação de dados, em diferentes fases da atividade de modelagem.

Em detrimento ao modelo analítico e ao modelo formal, o modelo computacional encapsula em sua ordem toda semântica da solução desenvolvida, ou seja, além dos ADT, os modelos computacionais caracterizam os relacionamentos, a cardinalidade, o sentindo e o fluxo de associações entre objetos pertencentes a um dado contexto (domínio). De acordo com Darmont et al. (2005), muitos pesquisadores em várias comunidades começam a afirmar que este arranjo de informações caracteriza dados complexos.

No entanto, esses conceitos emergentes de dados complexos não possuem uma unanimidade, mesmo dentro de uma única comunidade de pesquisa, como no caso a comunidade de banco de dados. Assim, o primeiro passo foi realizar uma breve revisão de literatura,

com diferentes tipos de pesquisas como mineração de dados, *data warehouse* e especificação algébrica dos tipos de dados, com propósito de descrever um consenso sintático da definição sobre ***dados complexos***.

- **Tipos de Dados Complexos**: é um conjunto de valores sem qualquer restrição sobre a forma como os valores são representados. Qualquer representação é permitida, com uma estrutura interna de complexidade arbitrária, incluindo texto, imagens, vídeos, gravações de som, formas geométricas e mapas. Porém, os operadores apropriados devem ser definidos para estas representações para que os valores possam ser acessíveis aos seus utilizadores. Os estudos preliminares de Darmont et al. (2005) foram capazes de concluir que os dados poderiam ser qualificados como complexos se eles possuíssem:

  - Múltiplos formatos: representados por vários formatos (base de dados, textos, imagens, sons e vídeos).
  - Múltiplas estruturas: representados por uma diversidade de estruturas (banco de dados relacionais e repositórios de documentos XML).
  - Múltiplas origens: com origem em diferentes fontes (Sistemas Distribuídos e a WEB).
  - Múltiplos meios: descritos de várias formas ou pontos de vista (radiografias, ultrassonografia e dados expressos em diferentes idiomas).
  - Múltiplas versões: que sofrem mudanças em termos de definição ou valor (banco de dados temporais).

  Esta primeira definição não é suficiente para abranger a grande variedade de dados complexos. Poderia realmente ser visto como um eixo de complexidade, entre outros eixos que tratam de semântica ou de processamento de dados, como é apresentado na Figura 2.19.

De acordo com as definições do Teste Estrutural da Seção 2.5.1, conhecer a estrutura interna da implementação é fundamental. Como a maioria das aplicações do mundo real utiliza estruturas de dados complexos, que são apresentados como objetos de agregação ou composição de outros objetos, faz-se necessário compreendê-los, tendo em vista que a eficácia do teste será determinada pela qualidade do subconjunto destes elementos complexos.

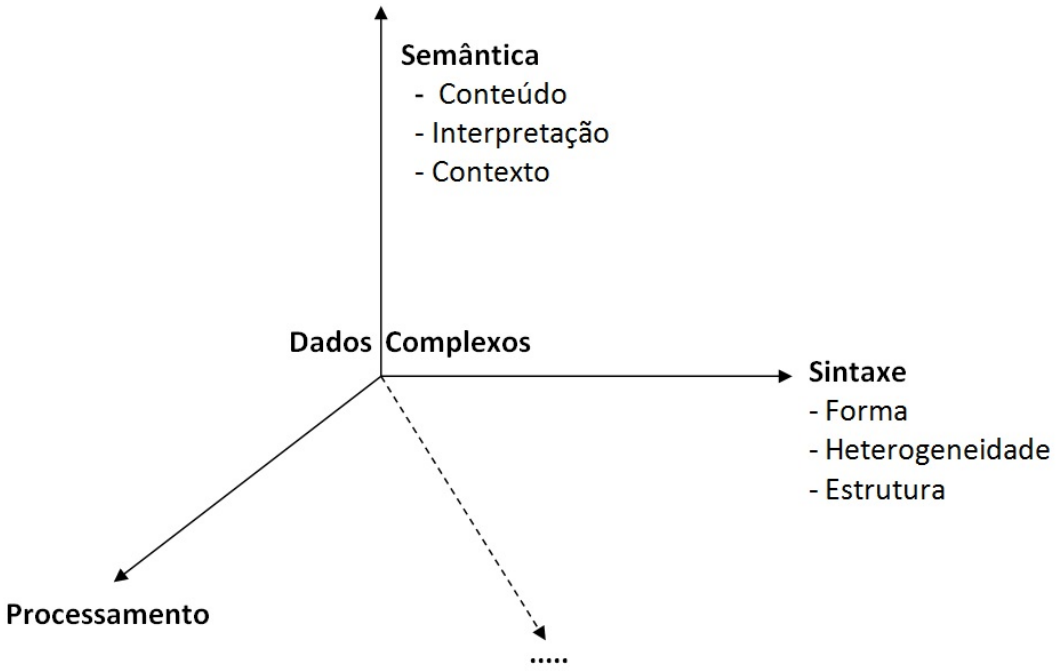


Figura 2.19: Eixos da Complexidade - (DARMONT et al., 2005)

No MDD, conforme a definição assumida para dados complexos, pode-se caracterizar um modelo ou seu subconjunto de elementos como um tipo complexo. Os mecanismos de transformação de modelos, quer o M2M ou M2T, fazem uso destes dados para executarem suas tarefas.

2.6.1 Teoria dos Grafos

A teoria dos grafos é um campo da matemática que estuda objetos combinatórios e ganhou popularidade em meados dos séculos XIX e XX, sobretudo por possibilitar a descrição de fenômenos de diferentes áreas, como: engenharia, indústria, organizações sociais, e especificamente a computação (STEEN, 2010).

Modelar problemas utilizando grafos se encaixa naturalmente com nossa capacidade de abstração, o que é particularmente interessante, pois esta representação não só estabelece o modo como pensamos os relacionamentos entre os objetos, como também permite formular questionamentos inerentes ao domínio representado. Deste modo, é possível fazer uso de todas as beneces da teoria dos grafos alinhadas com as técnicas e métodos de teste para compor uma abordagem de teste estrutural de transformações M2T.

2.6.2 Grafo e Grafo Direcionado

Um grafo G = (V,A) é um conjunto finito não vazio V (conjunto de vértices) e um conjunto A (conjunto de arestas) de pares não ordenados de elementos distintos de V. Assim, uma aresta $e \in A$ é um subconjunto de V com dois elementos (GUEDES, 2001).

É possível representar um grafo textual ou graficamente. Uma forma simples e comumente utilizada para representar grafo graficamente é utilizando círculos como padrão de representação dos vértices e linhas para representação padrão das arestas. Na Figura 2.20 são apresentadas as duas situações, a representação textual, tanto para o modo simples de arquivo texto quanto para a matriz de adjacência, e a representação gráfica do mesmo grafo.

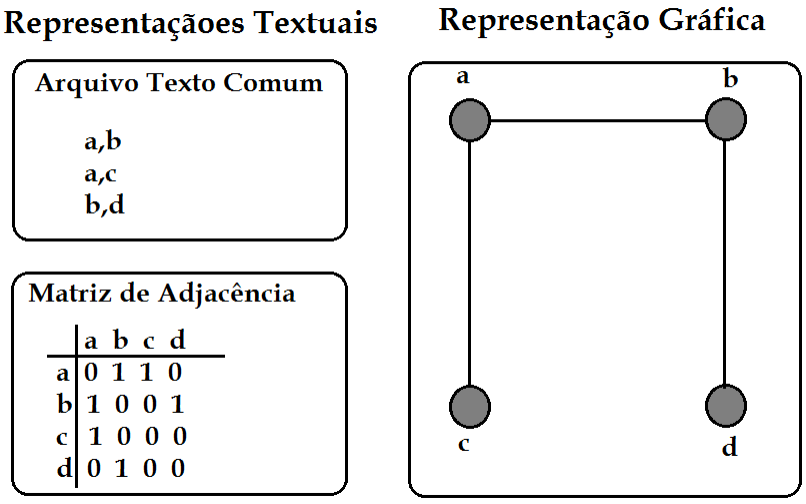


Figura 2.20: Tipos de Representações de Grafos

Um grafo direcionado D = (V,A) é um par onde V é um conjunto finito de vértices e A é um conjunto finito de arcos, onde um arco "**a**" $\in A$ é um par ordenado de elementos de V, ou seja, "**a**" $\in a\ VxV$. Na Figura 2.21 apresenta-se um exemplo simples de um grafo direcionado representado graficamente.

2.6.3 Hipergrafo e Hipergrafo Direcionado

O hipergrafo é uma generalização do conceito de grafos. Em um hipergrafo a cardinalidade das arestas pode ser diferente de dois. Um hipergrafo **H** igual a (V,A), no qual V é um conjunto finito de vértices e A é um conjunto finito de hiperarestas, no qual uma hiperaresta $a \in A$ é um subconjunto não vazio de V (AUSIELLO; D'ATRI; SACCà, 1983; GALLO et al., 1993; GUEDES, 2001).

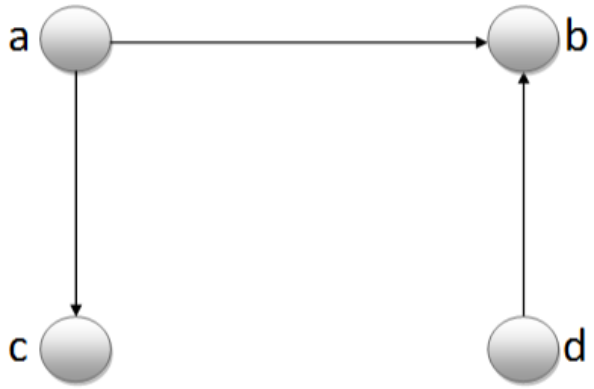


Figura 2.21: Exemplo de Grafo Direcionado

A generalização de conceitos também é estendida a hipergrafo direcionado, o qual faz uso dos preceitos de grafos direcionados e configura-se como uma especialização de hipergrafos. Assim, pode-se visualizar um hipergrafo direcionado como um conjunto de vértices e um conjunto de hiperarestas, nas quais as hiperarestas são pares de conjuntos disjuntos de vértices (X,Y). Ou seja, um hipergrafo direcionado $H = (V,A)$ é um par no qual V é um conjunto finito de vértices e A é um conjunto finito de hiperarestas, na qual uma hiperaresta "**a**" $\in A$ é um par ordenado (X,Y), nos quais X e Y são subconjuntos de elementos de V.

Os estudos de Guedes (2001) afirmam que o uso de hipergrafos direcionados constituem uma alternativa relevante para a caracterização de problemas em que as relações binárias usuais não são adequadas. Na Figura 2.22 é apresentado um exemplo de hipergrafo direcionado com sua representação textual e gráfica.

A literatura apresenta alguns problemas já caracterizados por hipergrafos direcionados, sendo eles:

- Em banco de dados relacionais a modelagem das dependências funcionais podem ser representadas por hipergrafos direcionais, e até mesmo os complexos relacionamentos de banco de dados NoSQL fazem uso dos conceitos de hipergrafos direcionados para implementar sua estrutura de armazenamento (IORDANOV, 2010). O problema a ser resolvido para estes casos é, dado um conjunto de dependências, encontrar um caminho mínimo que representa as mesmas dependências.

- Na área da lógica tem-se o cáculo proposicional e as variações do problema de satisfação (SAT). Neste tipo de problema, dado um conjunto de cláusulas desta forma:

  $\mathbf{C} = p_1 \vee p_2 \vee ... p_r \leftarrow p_{r+1} \wedge p_{r+2} \wedge ... \wedge p_q$, no qual, para $i = 1,...,q$, p_i é um literal,

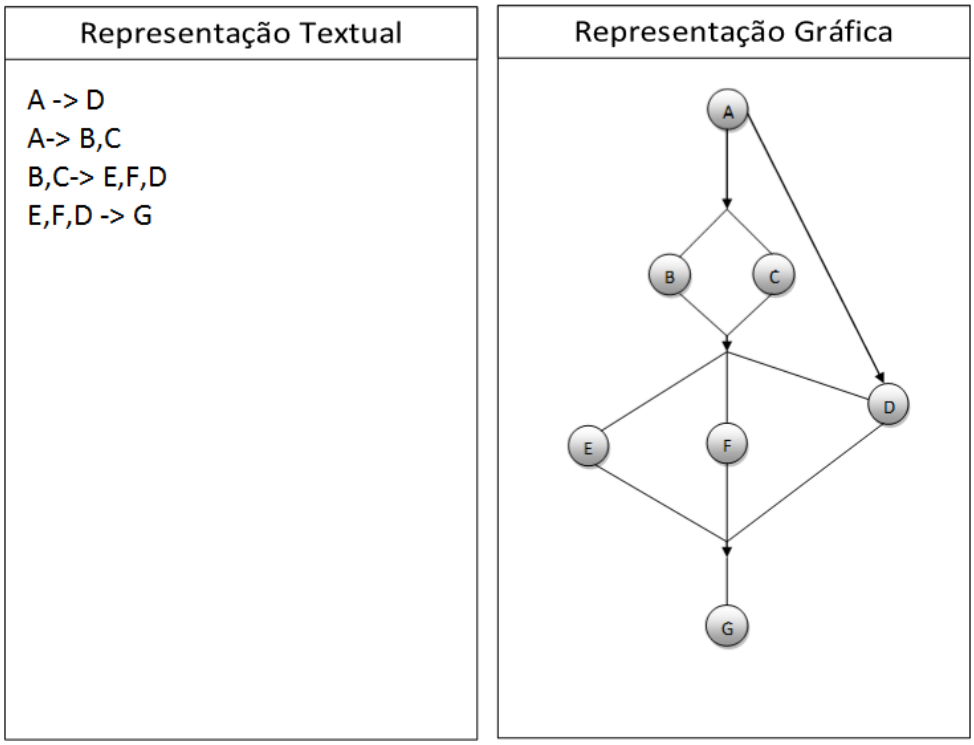


Figura 2.22: Exemplo de Grafo Direcionado

que pode ser verdadeiro ou falso. O significado de C é que pelo menos um literal $p_1, ...p_r$ deve ser verdadeiro quanto todos os literais $p_{r+1}, ...p_q$ o forem. Se isso for verdade, então a cláusula C é verdadeira, caso contrário é falsa.

Ainda em lógica, pode-se citar os grafos E/OU que podem, entre outras coisas, serem usados na resolução de problemas, por meio da redução por subproblemas (COURCELLE, 1993). Desta vez aparecem as relações entre um grupo de problemas (entrada) que podem ser resolvidos por um conjunto de subproblemas (saída).

- Na área de programação paralela algumas aplicações, como por exemplo, o estudo do comportamento de programas paralelos, podem ser caracterizadas por hiperarestas, nos mesmos moldes de uma rede de Petri, pois existe interdependências entre as tarefas (DING; SHEN; CAO, 2011).

- Um outro exemplo é a modelagem de sistemas especialistas que fazem uso de regras. Estas regras representam o conhecimento do sistema e podem ser caracterizadas como hiperarestas de hipergrafo direcionado (como cláusulas). Assim, verificações de consistência e completude podem ser executadas com maior facilidade (LAGUNA; MARQUES, 2009).

Todos os exemplos apresentados estão relacionados com conceitos de caminhos (hipercaminhos). Alguns buscam caminhos mínimos, ou todos os caminhos, e alguns usam

fechos transitivos (de caminhos). Mas, todos buscam encontrar caminhos entre determinados vértices.

Hipercaminhos Os hipercaminhos são estruturas flexíveis e são generalizações de caminhos em grafos. Os trabalhos de Guedes (2001) demonstram um esforço relevante quanto à definição do conceito de hipercaminhos e demonstram que apesar de algumas diferenças nas definições, os conceitos de conectividade são equivalentes. Problemas de hipercaminhos e algoritmos de solução são encontrados em várias publicações: Nguyen e Pallottino (1989), Marcotte e Nguyen (1998), Cominetti e Correa (2001) e Bell (2009).

Nos estudos de Gallo et al. (1993) pode-se encontrar os primeiros conceitos de B-caminho, F-caminho e BF-caminho. As letras B e F representam as expressões *backward* e *forward*, respectivamente. Um BF-caminho é ao mesmo tempo um B-caminho e um F-caminho.

Na Figura 2.23 pode-se observar um exemplo de hipercaminho. O hipercaminho de 1 a 9, $C = (a,c,f,g)$ tem origem $\{1,2\}$ e destino $\{9\}$. Ainda é possível observar que o hiperarco f pode ser retirado de C que o hipercaminho com a mesma origem e destino será preservado.

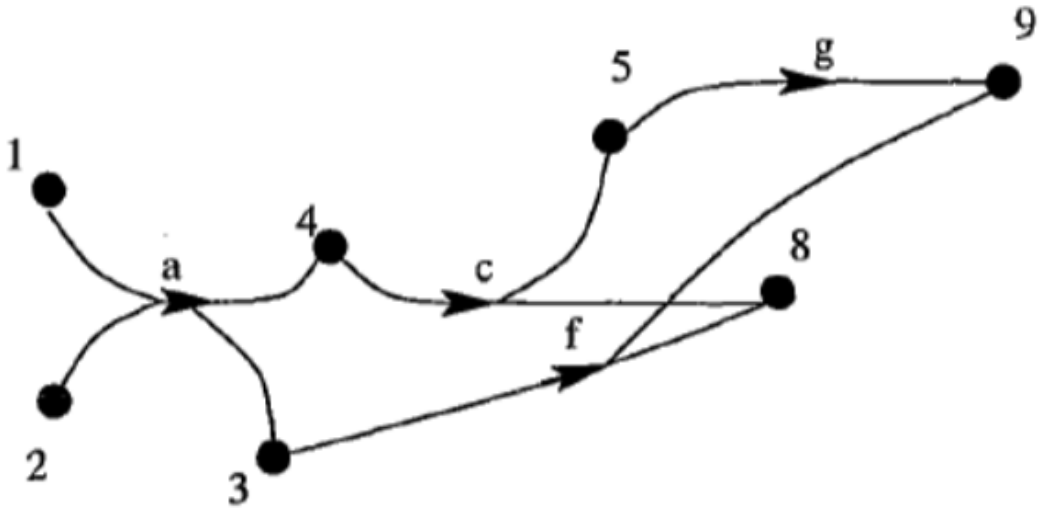


Figura 2.23: Hipercaminho C=(a,c,f,g) - (GUEDES, 2001)

Percursos Para resolver determinados problemas caracterizados por grafos, usualmente percorre-se um caminho, visitando os vértices e as arestas (arcos) em uma certa

ordem. Em grande parte destas situações, o percurso se faz por meio de buscas em profundidade ou largura.

Percursos em hipergrafos direcionados também têm grande importância na resolução destes problemas, como citam os trabalhos de Ausiello, Nanni e Italiano (1990) e Gallo et al. (1993), no quais os percursos são usados quase sempre para encontrar B-caminhos mínimos partindo de um determinado vértice.

Pode-se efetuar um percurso de forma geral em hipergrafos direcionados, por meio de conectividade simples (hiper-conectividade).

Inicialmente escolhe-se um vértice origem s, a partir do qual percorreremos vértices e hiperarcos conectados. Cada vértice escolhido é visitado (e marcado) e um hiperarco não explorado que tenha este vértice na origem é escolhido (e explorado). Coloca-se os vértices do destino deste hiperarco entre os vértices que podem ser escolhidos, e escolhe-se outro vértice. O algoritmo continua até que não existam mais vértices a escolher.

Na Figura 2.24 apresenta-se o algoritmo de BuscaSimples em hipergrafos, nota-se que sua estrutura é exatamente o algoritmo de busca em grafos já conhecido.

```
BuscaSimples
Dados H=(V,E) e s ∈ V
{
        Q ← { s }
        Enquanto Q ≠ ∅
        {
                Escolha x ∈ Q
                Visite x
                Escolha e ∈ FS(x) que não tenha sido explorado
                Se existir tal e então
                {
                        Explore e
                        Para y ∈ Dest(e)
                        {
                                Se y não foi visitado
                                Insira y em Q
                        }
                }
                Senão
                        Remova x de Q
        }
}
```

Figura 2.24: Algoritmo Busca Simples em Hipergrafos - (GUEDES, 2001)

Os algoritmos de percurso funcionam de forma incremental: a cada passo temos um conjunto de vértices que estão "conectados a s" e uma fronteira a explorar. Um novo vértice (ou hiperarco) é incluído, de forma a manter a conectividade. A conectividade simples tem as características necessárias. A propriedade "estar conectado a s" é monotônica, ou seja, todos os vértices encontrados em um hipercaminho de s a x estão conectados a s (GUEDES, 2001).

Hipergrafos de Fluxo O uso de grafos para aumentar a confiabilidade e a velocidade dos projetos de compiladores tem importante destaque nos estudos de compiladores e linguagens de programação. Neste contexto surgiram os grafos de fluxo e os intervalos (HECHT; ULLMAN, 1974), que são usados para estudar o fluxo de controle de um programa, ou para calcular o fluxo de dados. As definições de grafos de fluxo (HECHT; ULLMAN, 1972) permitem conceituar de modo equivalente hipergrafos de fluxo. Assim, Guedes (2001) define um hipergrafo de fluxo como uma tripla $H = (V, E, s)$, no qual (V, E) é um hipergrafo direcionado, $s \in V$ é o vértice origem, e existe um B-caminho de s para qualquer outro vértice em V.

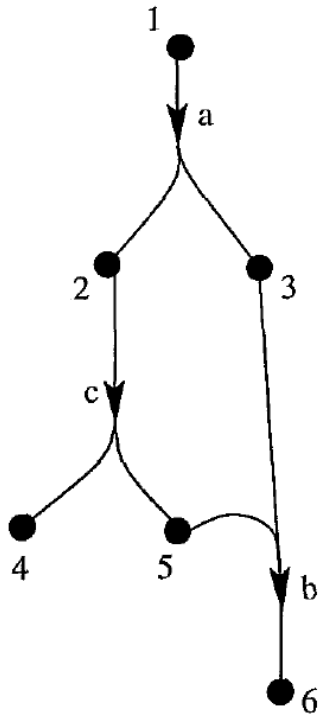


Figura 2.25: Hipergrafo de Fluxo com s=1 (GUEDES, 2001)

Essas definições e opiniões sobre os conceitos básicos da teoria de hipergrafos apresentadas até o momento são uma pequena amostra do que o leitor pode encontrar na

literatura, pois existem diversas tendências de pesquisa quanto à concepção de hipergrafos. Entretanto, é importante salientar que nesta dissertação as definições utilizadas anteriormente estão alinhadas com os estudos de Gallo et al. (1993) e Guedes (2001).

É possível encontrar alguns estudos que fazem uso de hipergrafos direcionados por sua capacidade de representar informações complementares que os grafos tradicionalmente não são capazes (GALLO; SCUTELLÀ, 1998). Na ciência da computação, estes estudos confirmam a viabilidade de uso dos hipergrafos direcionados para caracterizar informações complexas, primordialmente as envolvidas com gramáticas (BATORY, 2005), metamodelos (TVEIT, 2008; LAGUNA; MARQUES, 2009) e linhas de produto de software (LAGUNA; MARQUÉS; RODRÍGUEZ-CANO, 2011).

2.7 Considerações Finais

Neste capítulo foram apresentados os principais conceitos relativos ao contexto no qual este trabalho está inserido. Assim, foi discutido o uso do MDD como paradigma capaz de elevar o nível de abstração, no qual o desenvolvedor trabalha escondendo detalhes da plataforma de implementação e empregando as habilidades de automação do computador para ajudar nas tarefas de tradução entre problema e solução e nas atividades repetitivas.

A revisão da literatura apresentada dá ênfase ao processo de transformação dos modelos, atentando ao contexto em que este trabalho está inserido. Assim, mesmo com benefícios do MDD, testar as transformações M2M e M2T são tarefas complexas, que dificultam a garantia de corretude e qualidade dos artefatos de software gerados.

Na Seção 2.4, foram apresentados os principais conceitos e técnicas de teste de software e uma visão geral das principais técnicas e critérios de geração de dados de teste juntamente com suas necessidades e dificuldades de automatização. Posteriormente, a definição e possíveis classificações para dados complexos foi um aporte necessário para fundamentar a ênfase no contexto de testes em transformações de modelos.

Finalmente na Seção 2.6 abordou-se o processos de caracterização de dados complexos e as representações por meio de hipergrafos. Com relação a esta atividade foram descritos os conceitos relevantes, juntamente com métodos e critérios da literatura pertinentes ao contexto desta dissertação.

No próximo capítulo são apresentados os conceitos e peculiaridades sobre os tipos de transformações M2M e M2T juntamente com uma Revisão Sistemática da literatura,

conduzida para caracterizar o estado da arte de pesquisas que estão sendo desenvolvidas sobre a caracterização de dados complexos para transformações M2T.

Capítulo 3

Testes em Transformações de Modelos

3.1 Considerações Iniciais

O uso de modelos formais (executáveis) cada vez mais permeia os paradigmas da engenharia de software. Os modelos formais são utilizados como parte da especificação, como software de alto nível que descrevem a geração automática de código ou como ferramentas para verificação formal e testes baseados em modelos. Entretanto, o teste deve proporcionar novas abordagens e técnicas complementares, porque há sempre uma lacuna entre o modelo formal e o sistema real a ser verificada (BROY et al., 2005).

A testabilidade das transformações tem por objetivo primordial facilitar a geração de dados de testes e garantir uma melhor cobertura, por meio das adequações das técnicas e critérios de teste. Os estudos de Harman (2008) e Selim, Cordy e Dingel (2012) afirmam que o teste de transformações de modelos deixa em aberto uma série de questões que relacionam a combinação do teste de software e os tipos de transformações que ocorrem em MDD. A geração dos casos de testes e a definição dos critérios de cobertura constituem duas questões consideradas relevantes e abordadas durante esta investigação.

Neste capítulo são discutidos os testes em transformações de modelos, suas principais questões, tipos e desafios. Adicionalmente, apresenta-se uma revisão sistemática sobre testes em transformações M2T. Na Seção 3.2 são apresentadas algumas das questões consideradas de maior relevância sobre o teste de transformações de modelos. Nas Seções 3.3 e 3.4 são apresentados os tipos de testes de acordo com as peculiaridades de cada tipo das transformações de modelos. Por fim, na Seção 3.6, apresentam-se as considerações finais deste capítulo.

3.2 Desafios dos Testes em Transformações de Modelos

Na Seção 2.3 foram apresentadas as transformações de modelos como sendo o coração e a alma do MDD (SENDALL; KOZACZYNSKI, 2003), as quais devem ser consideradas como artefatos primários ao aplicar técnicas de MDD. Assim, como qualquer outro artefato de software, transformações de modelos devem sem concebidas, implementadas e consequentemente testadas. Entretanto, as transformações constituem uma classe de aplicações com características peculiares, o que torna testá-las um desafio. Ressalta-se que o teste de software pode ser uma abordagem eficaz para validação e verificação das transformações do modelo, para executar uma transformação e garantir sua integridade e semântica com os resultados reais esperados (HARMAN, 2008).

De modo superficial, pode parecer que testar transformações de modelos seja uma tarefa menos desafiadora do que testar o código fonte (FLEUREY; STEEL; BAUDRY, 2004). Entretanto, a natureza dos dados de entrada e de saída manipulados pelas transformações faz com que essa atividade englobe maiores níveis de complexidade. As transformações basicamente manipulam modelos, que são estruturas de dados complexos como já abordado na Seção 2.6. Assim, a complexidade dos dados de entrada e saída, a falta de ferramentas de gerenciamento de modelos e a heterogeneidade das linguagens de transformação são componentes especiais e desafiadores para o teste de transformações de modelos (WANG; KIM; CARRINGTON, 2008).

As regras de transformação de modelos são descrições de como uma ou mais especificações na linguagem de origem podem ser transformadas em uma ou mais especificações na linguagem de destino. Estes metamodelos descrevem uma estrutura estática dos modelos que são manipulados pela transformação. Os contratos das transformações de modelos são informações úteis para o desenvolvedor em MDD, por possibilitar a verificação de compatibilidade das transformações conforme as restrições de transformação do modelo.

Em muitos casos, as restrições (*constraints*) são adicionadas (geralmente expressas por OCL) à estrutura dos modelos ou metamodelos para garantirem a boa formação destes modelos. Tais regras são denominadas *well-formedness rules*.

No Capitulo 2 pode-se observar o metamodelo apresentado na Figura 2.11 sem qualquer regra de boa formação definida, posteriormente as Figuras 3.1 e 3.2 já apresentam o metamodelo com uma restrição (*constraint*) em OCL. Esta restrição garante que os nomes das entidades sejam sempre iniciados com caracteres maiúsculos para os modelos

instanciados, por meio do metamodelo anteriormente apresentado.

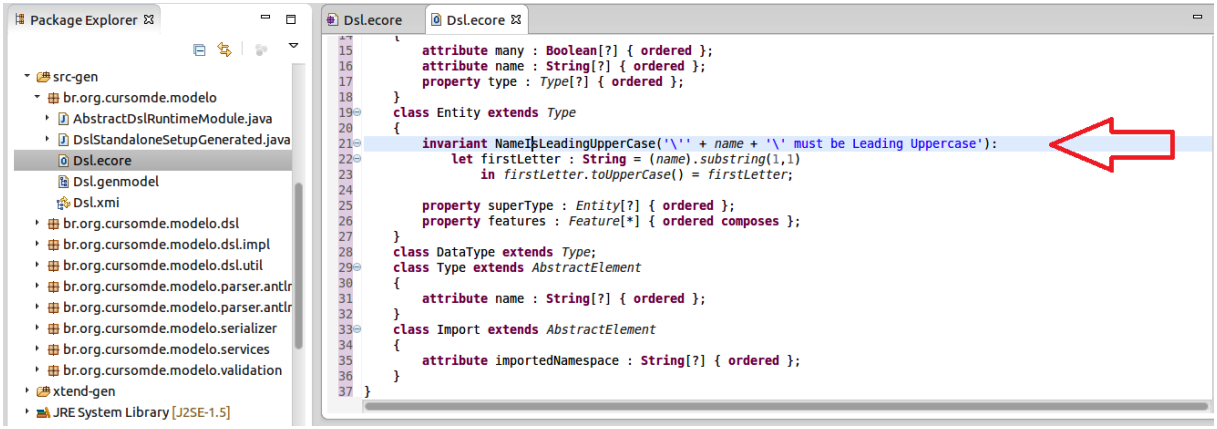


Figura 3.1: Constraint OCL no Metamodelo - OCL Editor

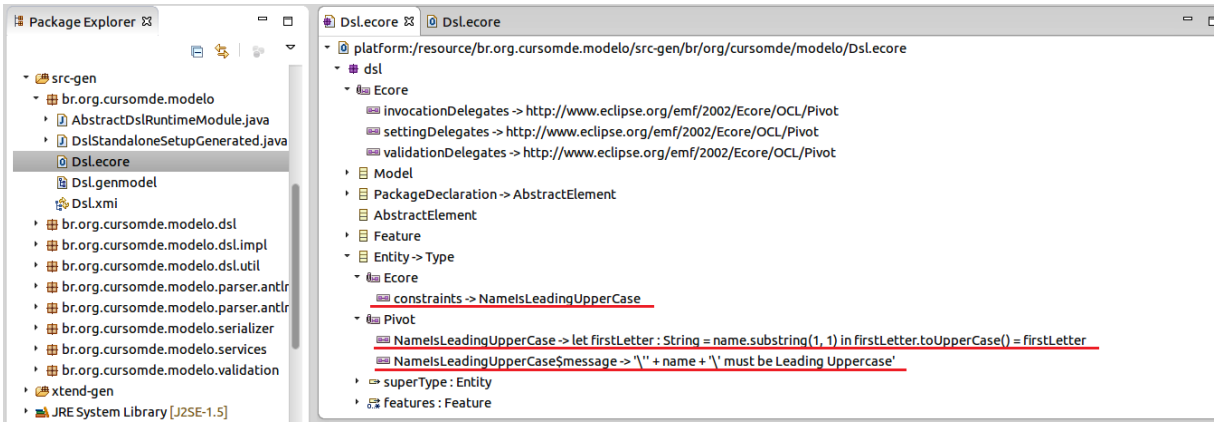


Figura 3.2: Metamodelo Ecore - Constraint OCL

Ainda que restrição em OCL seja apenas uma demonstração simples, pode-se notar a possibilidade de verificar a corretude da implementação por meio de uma validação sintática do metamodelo e, consequentemente, sua boa formação.

Revisitando os conceitos, a estrutura e os objetivos da transformação de modelos é, possível compreender como Baudry et al. (2010) segmentam sua abordagem em quatro fases complementares que descrevem o processo de teste em transformação de modelos. A primeira fase estabelece a geração de casos de teste, a qual envolve a criação de um conjunto de testes que possam checar a conformidade com o metamodelo de origem para testar as transformações decorrentes. Portanto, definir critérios adequados é relevante para garantir que o conjunto de teste será efetivo, capaz de estabelecer uma cobertura satisfatória quando o modelo for utilizado pelos testes gerados.

A segunda fase consiste em avaliar o conjunto de testes gerados. Um conjunto de testes que alcança uma avaliação positiva provavelmente será capaz de expor falhas que

possam ocorrer em uma transformação. Já um conjunto de testes com avaliação negativa pode ser aprimorado, adicionando modelos relevantes aos critérios necessários ao conjunto de testes.

A construção do oráculo é a terceira fase, na qual sua função é comparar uma saída real de uma transformação com uma saída esperada e avaliar o quanto esta transformação foi executada corretamente (FLEUREY; STEEL; BAUDRY, 2004).

A quarta e última fase descreve a execução das transformações com o conjunto de testes gerados, avaliando as saídas reais por meio da função de oráculo. Para cada modelo do conjunto de testes, a função de oráculo detecta as discrepâncias entre a sua correspondente saída esperada, permitindo ao testador analisar as transformações e corrigir as eventuais falhas encontradas.

É possível observar que os desafios dos testes em transformações de modelos residem em cada uma das fases descritas por Baudry et al. (2010). Entretanto, cada fase possui peculiaridades singulares de acordo com o tipo de transformação envolvida. Como a arquitetura MDA define as transformações, cada qual,com suas pecularied ades, basicamente as abordagens de teste de transformações são agrupadas em teste de transformações M2M e teste de transformações M2T. Ambos os grupos são discutidos a seguir, com ênfase no teste de transformações M2T, como forma de fundamentar a abordagem deste trabalho.

3.3 Testes em Transformações M2M

Em transformações de M2M, um conjunto de modelos de entrada é transformado em um conjunto de modelos de saída. Um defeito comum neste tipo de transformação é quando ocorre uma falha ao transformar alguns elementos do modelo de origem em elementos para o modelo de destino. Esta perda de informação pode causar erros nas etapas subsequentes da arquitetura MDA, tais como as transformações M2T, nas quais um modelo de origem é transformado em código fonte. Assim, um defeito nas transformações M2M pode comprometer todo o fluxo de desenvolvimento de uma aplicação.

Existem dois desafios relevantes para abordagens de teste em transformações M2M. O primeiro desafio é a automatização da geração de casos de teste, que consiste em uma atividade trabalhosa, tendo em vista toda complexidade das estruturas de dados e comportamentos, os quais devem estar de acordo com as restrições definidas no metamodelo de origem. O segundo é a comparação dos oráculos de teste com os modelos de saída, uma tarefa nada trivial, pois requer uma comparação entre dois modelos sintaticamente

diferentes mesmo que possam ser semanticamente equivalentes (BAUDRY et al., 2006).

Seguindo os desafios elencados e suas características, muitas abordagens e métodos de teste com propósito de garantir a integridade, boa formação e corretude das transformações em M2M vêm surgindo. Serão apresentadas a seguir algumas abordagens com técnicas agrupadas em Geração de Casos de Testes para Modelos e a Construção da Função de Oráculo de Teste.

Geração de Casos de Testes M2M Abordagens para a geração de modelos de teste vêm adaptando técnicas de teste convencionais que possam gerar casos de teste com coberturas aceitáveis. Uma estratégia amplamente utilizada é a definição dos critérios de cobertura conforme o metamodelo definido, isto é, cada metaclasse de origem dever ser instanciada pelo menos uma vez, em pelo menos um caso de teste gerado. Além disso, as propriedades desta meta classe devem ser representadas com valores significativos ao domínio de aplicação testado.

Os estudos de Wang, Kim e Carrington (2006) propõem uma abordagem de cobertura do metamodelo com base na infraestrutura MOF. Uma vez que todas as linguagens de modelagem façam uso da MDA, a cobertura do metamodelo pode ser definida para cada camada da infraestrutura MOF. Abordagem semelhante é definida por Fleurey et al. (2009), que adicionalmente acrescentam que a mesma técnica pode ser realizada em outros metamodelos, com padrões similares aos descrito pela tecnologia MOF.

Existem formas bem conhecidas para definir os critérios de cobertura, por exemplo, Particionamento de Equivalência, já descrito na Seção 2.5, que podem ser aplicadas no contexto das transformações de modelos. Novamente, os estudos de Fleurey et al. (2009) aplicam o teste de partição-categoria para decompor um metamodelo de origem em classes de equivalência, e posteriormente escolher modelos representativos de teste para cada uma dessas classess. Os estudos de Stuermer et al. (2007) seguem uma abordagem semelhante, pois definem uma técnica de teste de particionamento denominada *classification-tree method*.

Aplicar simplesmente as técnicas de partição de equivalência nas transformações M2M notoriamente não seria suficiente, pois a posologia também incorre em efeitos colaterais problemáticos. A técnica pode gerar um grande número de casos de testes (modelos de teste) como também a transformação pode objetivar apenas partes do metamodelo de origem. Assim, apenas estes dois fatos podem subsidiar situações nas quais a técnica seria ineficaz, evidenciando que seria inócua a geração de casos de testes para todo metamodelo

(FLEUREY et al., 2009; BAUDRY et al., 2010).

Alguns estudos predizem que é possível atenuar a ineficácia da geração de casos de testes por meio da técnica de partição de equivalência, calculando ou examinando toda a especificação envolvida na transformação, como, por exemplo, OCL de pré-condições, ou por meio de uma análise estática da implementação do código da transformação. A abordagem de Fleurey et al. (2009) propõe que este problema seja tratado por meio da análise das regras especificadas em OCL. Já as abordagens de Lamari (2007) e Guerra et al. (2010), Guerra e Soeken (2013) propõem que o metamodelo efetivo seja calculado a partir de especificações de transformações em linguagens de transformações especializadas.

Ainda que as abordagens mencionadas transmitam a ideia de que um metamodelo efetivo possui um tamanho reduzido e adequado a geração dos casos de testes, este pensamento minimalista pode incorrer em quebra dos estados hierárquicos dos modelos para os casos de testes gerados. Adicionalmente, são encontradas dificuldades de compreensão dos testadores, uma vez que com a geração de casos de testes otimizados, as partes do metamodelo considerado eficaz não correspondem intuitivamente ao significado semântico.

Existem abordagens que tratam destas questões por meio da aplicação de técnicas de testes de mutação. Os estudos de Mottu, Baudry e Traon (2006a) definem um conjunto de operadores de mutação que são utilizados juntamente com uma ferramenta denominada *Omogen* para avaliar a cobertura do metamodelo. Os operadores de mutação modificam uma das definições da transformação por meio da introdução de falhas. Caso um conjunto de testes não possa detectar a falha introduzida, os casos de testes são incompletos e devem ser atualizados.

A abordagem proposta por Darabos, Pataricza e Varró (2006) define um framework que, por meio dos operadores de mutação, gere casos de testes (modelos de teste) com defeitos típicos encontradas ao escrever transformações gráficas. Já os estudos de Küster e Abd-El-Razik (2007) geram mutantes para os conjuntos de testes a partir de especificações declarativas das transformações (e.g. especificação relacional QVT).

Adicionalmente, outras técnicas são pesquisadas e desenvolvidas. A literatura recente evidencia abordagens que fazem uso da lógica de primeira ordem para resolverem problemas de satisfatibilidade (SAT). O problema SAT tem por objetivo determinar se existe uma interpretação que satisfaça uma determinada fórmula booleana, ou seja, se as variáveis desta fórmula se mantém consistentes, desde que seus valores possam ser substituídos por Verdadeiro ou Falso de tal forma que a fórmula seja avaliada como Verdadeira (BABIC; BINGHAM; HU, 2006; GOMES et al., 2008). Modernos resolvedores SAT (SAT Solvers)

auxiliam as técnicas de Teste Funcional (teste caixa-preta) para resolver problemas com estruturas complexas, com mais de um milhão de variáveis e restrições (constraints). Abordagens como as de Sen, Baudry e Mottu (2008, 2009), Arcaini, Gargantini e Riccobene (2011), Famelis, Salay e Chechik (2012) e Abad et al. (2013) fazem uso dos resolvedores SAT para gerar casos de testes que porventura atinjam níveis satisfatórios de cobertura dos metamodelos.

Para resumir esta seção, pode-se concluir que a geração de casos de teste para transformações M2M recebeu notória atenção dos pesquisadores recentemente. Muitas técnicas têm sido aplicadas para gerar automaticamente casos de testes, concentrando-se principalmente na construção de casos de testes relevantes que proporcionem uma cobertura eficiente do metamodelo.

Construção da Função de Oráculo de Teste Um oráculo de teste de software tem por objetivo determinar se o resultado de um caso de teste está correto. A função do oráculo é comparar o resultado real com o resultado esperado e validar sua corretude (FLEUREY; STEEL; BAUDRY, 2004).

Há vários métodos para construir um oráculo para transformação de modelos. Caso os modelos de saída esperados estejam disponíveis, a função do oráculo é comparar o modelo real com o modelo esperado ou diferenciá-los (LIN; ZHANG; GRAY, 2004; KOLOVOS; PAIGE; POLACK, 2006). No entanto, se os modelos de saída esperados não estiverem disponíveis, então a função do oráculo é validar a saída da transformação em relação a saída pré definida por meio das linguagens de especificação ou contratos das transformações (CARIOU et al., April 2004; MOTTU; BAUDRY; TRAON, 2006b; GOGOLLA; VALLECILLO, 2011).

Um levantamento apresentado por Mottu, Baudry e Traon (2008) resume os principais tipos de oráculos que podem ser utilizados em testes de transformações de modelo. Estes tipos estão agrupados nas seguintes categorias:

1. Transformação por referência, na qual o artefato (modelo ou código) da transformação correta é conhecido e referenciado pelas mesmas funcionalidades da transformação testada.

2. Transformação inversa, expressa por t o t^{-i} (para uma transformação em teste t), a qual é verificada identidade da transformação.

3. Transformação com modelo de saída esperado, na qual o usuário (manualmente) informa o resultado esperado da transformação.

4. Transformações por Contratos, na qual os testes são validados pelas ações de pré- e pós-condições ou por meio das linguagens especializadas.

Na literatura, as categorias 3 e 4 possuem um destaque por possuírem o maior número de abordagens que contemplem estas características para construção do oráculo de testes para transformação de modelos. Alguns destes estudos analisam a transformação, verificando sua saída com relação às fontes pré-existentes de conhecimento informadas como restrições (*constraints*), como, por exemplo, pós-condições ou invariantes relativas à linguagem de saída. Os estudos de Baudry et al. (2006) fazem uso desta abordagem manipulando instruções em OCL para especificar as restrições necessárias como base de conhecimento ao oráculo de testes. Uma característica relevante é que muitas destas abordagens conseguem prover um conhecimento limitado ao oráculo, permitindo que seja verificada apenas a boa formação (*well-formedness*)s do metamodelo alvo.

Atentos a esta limitação, Whittle e Gajanovic (2005) propõem uma abordagem que adiciona invariantes para restringir o metamodelo de saída de alguma forma. A abordagem faz uso de uma linguagem de domínio específico denominada NASA e tem por objetivo a geração de código. Dado este contexto, propriedades relacionadas à estrutura do código gerado eram conhecidas, então capturadas e abstraídas como invariantes para restrições OCL, assim, poderia ser verificado facilmente a conformidade do código gerado com a estrutura conhecida, permitindo que o oráculo fosse além das questões de boa formação.

Uma abordagem semelhante é proposta por Kolovos, Paige e Polack (2006), que fazem uso de uma linguagem específica denominada *Epsilon Comparison Language* (ECL), que é usada para especificar as restrições entre o modelo de origem e o modelo de saída. O uso da ECL permite a especificação das restrições por meio de dois metamodelos diferentes. Estudos de Lano e Clark (2008) também propõem uma abordagem similar que faz uso de um conjunto de restrições para verificar a correção sintática e semântica das transformações do modelo.

Todas as abordagens descritas anteriormente fazem uso de linguagens textuais para expressar as restrições necessárias ao oráculo de testes das transformações. Entretanto, a literatura também reporta trabalhos que fazem uso de padrões gráficos para expressar restrições e definir as combinações esperadas para instâncias do metamodelo. Assim, os oráculos podem verificar se um modelo de destino corresponde a um padrão esperado.

Este tipo de oráculo é abordado nos trabalhos de Orejas e Wirsing (2009), que faz uso dos padrões gráficos, nos quais as propriedades a serem verificadas podem ser expres-

sas por atributos variáveis para corresponderem aos modelos de origem e de saída das transformações. Cabe ressaltar que esta abordagem é muito semelhante aos trabalhos que fazem uso da ECL, uma vez que definem uma restrição declarativa entre o modelo de origem e o modelo de saída. A diferença está na expressividade da linguagem usada para definir as restrições, pois padrões gráficos são naturalmente mais intuitivos.

A análise das abordagens que tratam sobre a construção dos oráculos de teste permite compor duas questões importantes sobre a verificação das transformações de modelos. A primeira questão versa sobre as restrições que permitem apenas uma verificação parcial das transformações, uma vez que somente é possível verificar se o modelo de saída atende a determinados contratos ou padrões com estruturas conhecidas. Assim, as linguagens que expressam essas regras por muitas vezes não possuem recursos suficientes para verificar as propriedades semânticas de uma transformação.

A segunda questão remete à necessidade dos desenvolvedores programarem todas as restrições necessárias, muitas vezes incorrendo em regras equivocadas ou falhas. Abordagens que descrevem os oráculos categorizados no item 3 requerem um esforço ainda maior, tendo em vista que um modelo de saída esperado deve ser construído para cada caso de teste. Adicionalmente, esta categoria exige um algoritmo de comparação sofisticado para verificar os resultados das transformações com relação as saídas esperadas, tanto quanto para equivalência sintática e semântica.

3.4 Testes em Transformações M2T

O teste em transformações M2M tornou-se um importante tema nos estudos da engenharia dirigida a modelos (VALLECILLO et al., 2012; AMRANI et al., 2012). Assim, as transformações M2T preenchem uma lacuna entre as linguagens de modelagem e as linguagens de programação, permitindo a geração de código fonte e produzindo documentações ou representações textuais do conteúdo do modelo.

Czarnecki e Helsen (2006) afirmam que a geração de código é um dos principais objetivos da MDD, e que as transformações são de grande importância. Assim, a geração de artefatos com qualidade demanda métodos de testes que sejam capazes de assegurar que as transformações M2T estejam em conformidade com os modelos de entrada e saída.

A importância das transformações M2T é novamente enfatizada no trabalho de Rose et al. (2012), que complementa os estudos de Czarnecki e Helsen (2006), definindo um modelo de características que apoia a decisão no uso das linguagens de transformações

M2T.

Motivado pelas oportunidades de pesquisas determinadas pelas inúmeras questões que envolvem o teste em transformações de modelos, o trabalho de Selim, Cordy e Dingel (2012) faz um recorte temporal definido, com objetivo de reconhecer as principais investigações, identificando temáticas e abordagens dominantes e emergentes. Ainda que este trabalho tenha sido realizado no ano de 2012 e os relatos do estudo não abordem exclusivamente os Testes em Transformações M2T, três fases são ressaltadas e necessitam de mais pesquisas e investigações, sendo elas: Geração de dados de testes, Critérios de adequação e a Definição de casos de teste.

3.5 Revisão Sistemática

O trabalho de Wimmer e Burgueo (2013), destaca a necessidade de abordagens de testes capazes de suportar e garantir a geração dos artefatos de *software* por meio das transformações M2T. Assim, com o intuito de caracterizar o estado da arte em testes em transformações de M2T, conduziu-se neste trabalho uma Revisão Sistemática (RS) da literatura, de modo a sumarizar as evidências científicas relacionadas ao tema investigado.

A revisão sistemática é um tipo de pesquisa científica, que tem por finalidade apresentar uma avaliação não tendenciosa a respeito de um tópico de pesquisa, fazendo uso de metodologia de revisão que seja confiável, rigorosa e que permita auditagem (KITCHENHAM, 2004; FABBRI et al., 2013). Portanto, este trabalho propôs identificar quais técnicas de teste estrutural e seus critérios estão sendo utilizadas, quais destas técnicas caracterizavam dados complexos e qual padrão comportamental predominante foi utilizado nas transformações.

Todos os detalhes de condução e metodologia utilizados nesta revisão sistemática são apresentados no trabalho de Abade, Ferrari e Lucrédio (2015), e serão discutidos a seguir apenas os resultados da seleção final. Na próxima seção, apresenta-se uma visão geral da seleção de estudos primários e concentra-se na análise e nas comparações entre as abordagens e as conclusões alcançadas.

Seleção Preliminar a Final – Visão Geral: O processo de seleção preliminar retornou um total de 202 estudos primários dos três repositórios analisados. Após a aplicação dos critérios de inclusão e exclusão a cada um dos estudos, por meio da apreciação de seus títulos e "Abstract", o conjunto de estudos primários avaliados pertinentes somaram

um total de 49 estudos. Nesta fase foram incluídos 21 estudos por indicação de especialistas consultados durante a condução desta etapa, oriundos de congressos e periódicos, totalizando 70 estudos. Os estudos aceitos na etapa de seleção preliminar, passaram por uma leitura completa e foram novamente classificados com base nos critérios de inclusão e exclusão, configurando um conjunto de 09 estudos alinhados aos interesses desta investigação.

Tabela 3.1: Seleção Final dos Estudos Primários - (ABADE; FERRARI; LUCRÉDIO, 2015)

Ano	Fonte	ID	Título	Autores	Caracteriza Dados Complexos	Teste Estrutural	
						Padrão Visitor	Método Template
2008	Specialist	S1	Unit Testing Model Management Operations	(POLACK et al., 2008)	Não	-	X
	Scopus	S2	Using model transformation to support model-based test coverage measurement	(NASLAVSKY; ZIV; RICHARDSON, 2008)	Não	-	X
2010	ACM	S3	Multi-level Testes for Model Driven Web Applications	(FRATERNALI; TISI, 2010)	Não	-	X
2011	Scopus	S4	Model-driven testing: Transformations from test models to test code	(LAMANCHA et al., 2011)	Não	-	X
2013	Specialist	S5	A Method for Testing Model to Text Transformations	(TISO; REGGIO; LEOTTA, 2013)	Parcialmente	-	X
	Specialist	S6	An Approach to Testing Java Implementation against Its UML ClassModel	(CHAVEZ et al., 2013)	Não	X	-
	ACM	S7	Transformation Rules for Platform Independent Testing: An Empirical Study	(ERIKSSON; LINDSTRöM; OFFUTT, 2013)	Parcialmente	X	-
	Scopus	S8	MTC Flow: A Tool to Design, Develop and Deploy Model Transformation Chains	(ALVAREZ; CASALLAS, 2013)	Não	-	X
	Specialist	S9	Testing M2T/T2M transformations	(WIMMER; BURGUEO, 2013)	Não	-	X

Na Tabela 3.1 são apresentados os trabalhos que compõem o conjunto final de estudos primários selecionados. Os estudos estão ordenados por ano de publicação e consequentemente dispostos por Fonte de Busca, Identificador (ID) e Autores. Os estudos listados na Tabela 3.1 foram classificados de acordo com um conjunto de características pertinentes às questões de pesquisa e especificadas em abordagens utilizadas nas transformações de modelo.

Análise O processo descritivo sobre estes estudos foi organizado por meio das facetas determinadas pelo conjunto de características, as quais são pertinentes às questões de pesquisa elaboradas. É importante destacar que o padrão comportamental utilizado

nas transformações, bem como a caracterização dos dados complexos, impactam sobre a implementação de técnicas e abordagens para testar transformações M2T. Por exemplo, o Método Templates pode fazer uso de uma implementação de referência, permitindo que critérios de cobertura possam ser definidos em relação aos modelos de entrada, os *templates* e o código gerado.

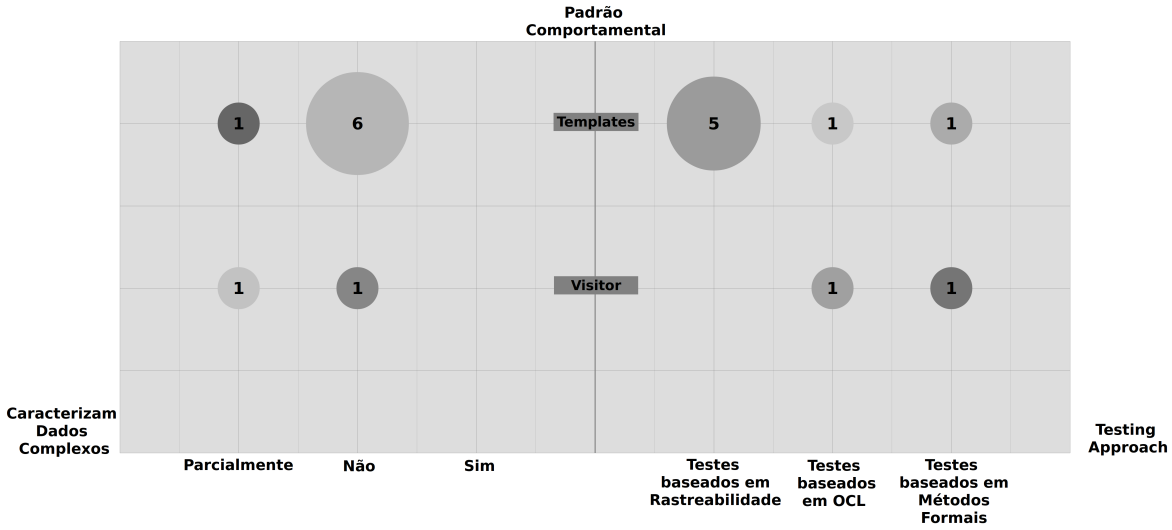


Figura 3.3: Gráfico de Bolhas - Distribuição Resultados por Faceta - (ABADE; FERRARI; LUCRÉDIO, 2015)

Na Figura 3.3 apresenta-se a distribuição dos estudos nas facetas, o que possibilita identificar as quantidades de estudos associados às questões de pesquisa definidas para a revisão.

Faceta 1 - Padrão Comportamental - Esta faceta agrupa os estudos pelo padrão *(pattern)* comportamental da concepção do transformador que são qualificados de acordo com as afirmações de Czarnecki e Helsen (2006) e Rose et al. (2012). Após analisadas as características das linguagens de transformações utilizadas em cada estudo, obteve-se os seguintes grupos: a) abordagens com o ***Padrão Visitor*** (S6, S7) e b) abordagens com o ***Método Templates*** (S1, S2, S3, S4, S5, S8, S9).

Os resultados apresentados na Figura 3.3 vão ao encontro dos esforços de padronização do MOFM2T do OMG. É possível evidenciar dentre os estudos analisados que a maioria, 78% dos estudos, faz uso da abordagem por meio do Método Templates, enquanto apenas 22% utilizam o Padrão Visitor. Com esta perspectiva, o trabalho de Rose et al. (2012) define um modelo de características que apoia o processo de decisão quanto ao uso das linguagens de transformações M2T, baseado nos padrões MOF.

Faceta 2 - Caracterização dos Dados das Transformações - Basicamente a complexidade das transformações é definida pelo alto nível de abstração utilizado nos modelos de entrada, pela complexidade das regras de transformação e pela diversidade de artefatos de software que podem ser gerados (BAUDRY et al., 2010). Em detrimento ao modelo analítico e ao modelo formal, o modelo computacional encapsula em sua ordem toda semântica da solução desenvolvida, ou seja, além dos tipos de dados abstratos, os modelos computacionais caracterizam os relacionamentos, a cardinalidade, o sentindo e o fluxo de associações entre objetos pertencentes a um dado contexto.

De acordo Baudry et al. (2010) a complexidade dos dados manipulados pelas transformações de modelos trazem novos desafios, exigindo adaptações das técnicas de testes. Neste sentindo, obteve-se que 78% dos estudos não caracterizam dados complexos nas transformações, sendo eles os estudos (S1,S2,S3,S4,S6,S8,S9). Somente 22% dos estudos caracterizam parcialmente dados complexos nas transformações do modelo por algum método de representação (S5,S7).

Faceta 3 - Abordagem de Teste - O Teste de software consiste em uma análise dinâmica do produto, sendo relevante para a identificação e a eliminação de defeitos que persistem, representando a última revisão da especificação, projeto e codificação. De modo geral, as técnicas de teste podem ser divididas naquelas baseadas no código (caixa branca ou estrutural) e naquelas baseadas na especificação (caixa preta ou funcional).

Este estudo teve como objetivo investigar as técnicas de teste estrutural utilizadas em transformações M2T. Assim, foi possível selecionar estudos que utilizam técnicas para examinar a estrutura interna do software e compor os requisitos de teste. Esta faceta agrupou os estudos primários que investigam abordagens baseadas em teste estrutural com as seguintes características: a) *Traceability-based Test* com 56% (S1,S2,S3,S4, S8); b) *OCL-based Test* com 22% (S6,S9) e c)*Formal Method-based Test* com 22% (S5,S7).

Discussão dos Resultados Os resultados da análise elaborada são agrupados conforme as características comuns das abordagens de teste alinhadas com as facetas da pesquisa. Para isto, três grupos são definidos, sendo eles: A) Teste Estrutural com Padrão *Visitor*; B) Teste Estrutural com Método *Templates* e C) Caracterização de Dados Complexos para Testes em Transformações. Na sequência desta organização, são discutidos os estudos selecionados dos grupos (A) e (B) com base nas categorias incluídas da terceira faceta já apresentadas na Figura 3.3, identificadas como: *Traceability-based Testing*, *OCL-based Testing* e *Formal Method-based Testing*.

A - Teste Estrutural com Padrão *Visitor*

Apenas duas abordagens de teste possuem as características definidas para este grupo. Estas são identificas como os estudos (S6 e S7), apresentados na Tabela 3.1. Cabe observar que este grupo não inclui qualquer estudo primário que utilize a técnica de rastreabilidade em sua abordagem de teste.

OCL-based Testing

O estudo S6 de Chavez et al. (2013), propõe uma abordagem inovadora chamada CCUJ. Os autores descrevem como automatizar uma abordagem que possibilita a verificação de conformidade (well-formedness) entre um projeto UML e uma implementação Java. A abordagem CCUJ verifica se a implementação Java é consistente com as especificações OCL de pré e pós-condições associadas aos diagramas de classe do modelo. Neste trabalho os autores não definem explicitamente os critérios de cobertura para o teste estrutural uma vez que, boa parte da metodologia preocupou-se com a verificação de conformidade. O estudo descreve a metodologia, porém não aborda qualquer questão sobre a caracterização dos dados complexos de teste.

Formal Method-based Testing

A abordagem S7 de Eriksson, Lindström e Offutt (2013) propõe que o teste estrutural para transformações M2T e seus critérios de cobertura sejam definidos por meio de arranjos compostos por predicados e suas cláusulas. Um dos processos destas abordagens é quantificar quantas cláusulas compõem um predicado, para estimar o número de requisitos de teste baseados nas expressões lógicas. Esta abordagem discute como resolver a discrepância do número de requisitos de teste entre o modelo e o código fonte gerado, tal abordagem é considerada em média 60% maior em suas avaliações empíricas. A proposta de Eriksson, Lindström e Offutt (2013) caracteriza parcialmente os dados complexos de entrada/saída envolvidos nas transformações M2T. Assim, este estudo consegue contemplar basicamente todas a questões que definem os problemas para os testes em Transformações M2T. Entretanto, esta abordagem destaca que apenas um compilador foi utilizado para as transformações, invalidando assim, seus resultados para outros transformadores.

B - Teste Estrutural com Método Template

Os estudos S1, S2, S3, S4, S5, S8 e S9 apresentados na Tabela 3.1, abordam o teste em transformações M2T usando o padrão Método Templates. A seguir estes estudos serão discutidos de acordo com suas características.

Traceability-based Testing

A abordagem S1 de Polack et al. (2008) faz uso das técnicas de teste de unidade para garantir a corretude das operações do modelo em relação ao código fonte gerado. Para tanto, é proposto um protótipo que especifica a execução desta transformação, o qual é implementado sobre a plataforma do Epsilon. Esta abordagem faz uso do know-how dos testes em transformações M2M para contemplar as transformações M2T. Ainda que o teste de unidade não seja um conceito novo para o desenvolvimento de software, revela-se como uma importante estratégia para os desenvolvedores melhorarem a qualidade do código, enquanto compreendem os requisitos funcionais de uma classe ou método (OSHEROVE, 2009). Muitas das abordagens de Teste Estrutural em Transformações de Modelo, primordialmente as M2M, fazem uso desta técnica para estabelecer a boa formação dos modelos e critérios de corretude.

Os trabalhos S2 de Naslavsky, Ziv e Richardson (2008) e S4 de Lamancha et al. (2011) contribuem para os testes em transformações M2T aplicando *Model Based Testing* (MBT). MBT é uma técnica para gerar automaticamente casos de teste baseados em modelos extraídos de artefatos de software. Assim, seu objetivo principal é a criação de artefatos que descrevem os requisitos e comportamentos do sistema, visando automatizar o processo de teste. Uma vez desenvolvido o modelo, este pode ser usado de várias formas e para vários estágios do processo de desenvolvimento de software, tais como, especificação de requisitos, geração de código, análise de confiabilidade do sistema e execução de testes (BROY et al., 2005).

A abordagem proposta no estudo S2 utiliza uma estratégia com mapeamentos e técnica de rastreabilidade, por meio de anotações no código, para mensurar a cobertura alcançada pelos casos de testes exercitados pelos modelos derivados com a abordagem MBT. O estudo S4 propõe um *framework* para automatizar os testes no contexto de MBT. Este *framework* possibilita que os modelos de teste sejam gerados para os dois tipos de transformações M2M e M2T. Nesta abordagem, para gerar os casos de teste é utilizado o *MOFScript* seguindo os padrões da *OMG*. A contribuição deste estudo é pautada primordialmente na automatização da geração dos casos de teste, por meio de mapeamentos e técnica de rastreabilidade.

O estudo S3 de Fraternali e Tisi (2010), se concentra em questões de ambientes de teste. Ambiente de teste é toda a infraestrutura sobre a qual os ensaios de teste serão executados, incluindo, configurações de hardware, software, ferramentas de automação, equipes de testes, aspectos organizacionais, suprimentos e documentação da rede. Sua

finalidade é propiciar a realização de testes em condições conhecidas e controladas (BASTOS et al., 2007). Nesta perspectiva, o estudo S3 aborda problemas de definição, gestão e execução de testes para aplicações web sob o paradigma MDD, em seus vários níveis de abstração. A abordagem está alinhada com o fluxo de transformações MDA e código fonte. A metodologia proposta faz uso do *WebML* e a plataforma *WebRatio* para implementar um *framework* responsável pelos casos de testes, mantendo a rastreabilidade entre os níveis MDA.

O trabalho S8 de Alvarez e Casallas (2013) descreve uma ferramenta denominada *MTC Flow* que permite aos desenvolvedores criarem aplicações segundo o paradigma MDD. A ferramenta fornece uma DSL gráfica para especificar o modelo *MTC workflow*, independente da tecnologia de transformação. Outra característica importante é a gestão da execução alternativa de caminhos usando um sistema de *Tag* para marcar os elementos no *MTC Design*. Estas marcas mais tarde irão orientar os mecanismos de execução do MTC. Nesta abordagem, a integração das diferentes camadas da MDA possibilita a interoperabilidade. As Técnicas de teste disponíveis no ambiente da ferramenta subsidiam a validação da corretude e boa formação dos modelos.

OCL-based Testing

A abordagem S9 de Wimmer e Burgueo (2013), faz uso do know-how herdado dos testes em transformações M2M para compor uma proposta de teste M2T/T2M independente das linguagens utilizadas, tanto para os modelos de entrada quanto para os artefatos de saída. O estudo S9, especifica os contratos e regras das transformações por meio de uma extensão da OCL, permitindo que testes de unidade sejam realizados, afim de validar os contratos e regras estabelecidas.

Formal Method-based Testing

A abordagem S5 de Tiso, Reggio e Leotta (2013), apreseta um método para testar transformações M2T que utiliza um conjunto de abordagens integradas. O desenvolvimento das transformações do modelo é parte deste método. Nesta proposta existe um conjunto de regras que caracterizam a boa formação e corretude dos modelos, além da caracterização dos dados por meio de estereótipos dos diagramas UML para seleção dos modelos de entrada. Assim, o método proposto tem por objetivo adequar os casos de teste para estabelecer critérios de cobertura do modelo.

C - Caracterização de Dados Complexos para Teste em Transformações M2T

A maioria das aplicações do mundo real utilizam estruturas de dados complexos, que são apresentadas como objetos de agregação ou composição de outros objetos. Em um contexto de teste estrutural, são necessários conhecimentos sobre a estrutura interna da implementação para elaborar os dados de entradas relevantes ao testes, a compreensão de tais estruturas de dados é fundamental. Esta compreensão deve possibilitar a definição efetiva dos dados de teste que determinará a alta cobertura e consequentemente a alta confiança da corretude do software.

Uma vez que se torna possível representar adequadamente os elementos dos modelos de entrada em uma abordagem MDD, os testadores são capazes de compor conjuntos de dados de entrada complexos que exercitem partes específicas das transformações. Estas partes específicas estão relacionadas, por exemplo, com um tipo particular de artefato que será gerado. Por exemplo, enquanto um conjunto de elementos de entrada pode exercitar no transformador a geração de artefatos Java, outro conjunto particular pode exercitar na transformação a geração de artefatos SQL.

Nesta revisão sistemática, os estudos S5 de Tiso, Reggio e Leotta (2013) e S7 de Eriksson, Lindström e Offutt (2013) caracterizam parcialmente dados complexos em transformações M2T usando alguma estratégia de representação de dados. A abordagem S7 usa a lógica de predicados para compor arranjos de dados complexos, a fim de resolver a discrepância no número de requisitos de teste entre o modelo de origem e o código gerado. Uma das limitações relatadas pelos autores foi a utilização de apenas um transformador com padrão comportamental Visitor e a falta de relatos quanto a efetiva caracterização de dados complexos para diferentes tipos de artefatos que podem ser gerados conforme o paradigma MDD.

Como já discutido anteriormente, o estudo S5 tem como objetivo adequar os casos de teste para estabelecer critérios de cobertura de teste do modelo. Entretanto, a caracterização de dados complexos descrita nesta abordagem possui uma natureza genérica. Os autores não abordam particularmente a representação de dados complexos para os modelos textuais, embora a característica genérica da abordagem permita personalizações para transformações M2T.

3.6 Considerações Finais

Neste capítulo foram apresentadas os conceitos e técnicas do teste em transformações de modelos e uma visão geral das principais técnicas e critérios utilizados. Adicionalmente, uma breve revisão da literatura dá ênfase ao teste em transformações M2M, atentando para as técnicas de geração de casos de testes, construção da função do oráculo de teste e as dificuldades e desafios que motivam inúmeras pesquisas.

Posteriormente, apresentou-se a Revisão Sistemática da Literatura sobre os Testes em Transformações M2T, na qual foi possível elaborar um levantamento dos estudos existentes a respeito das técnicas e critérios de teste estrutural aplicadas as transformações M2T. Os resultados deste levantamento possibilitaram a categorização das características comuns encontradas nas abordagens, bem como seus problemas e limitações recorrentes.

Em uma análise de alto nível é possível concluir que as soluções relacionadas ao Teste de Transformações M2T atualmente já começam a mudar o foco inicial das abordagens que primavam pela boa formação e corretude dos modelos. Algumas abordagens fazem propostas interessantes quanto ao uso de técnicas que estabeleçam critérios de cobertura para os testes e outros já tentam resolver a testabilidade dentro das inúmeras linguagens de transformações existentes. Ou seja, as soluções encontradas na literatura ainda não podem ser consideradas como estabelecidas e o tópico de pesquisa precisa ser melhor explorado.

Com relação às questões da pesquisa, foi possível perceber que as abordagens para teste estrutural nesse contexto estão evoluindo, principalmente com a utilização de mapeamentos e rastreabilidade das transformações. Entretanto, a maioria das abordagens avaliadas tratam exclusivamente dos problemas de *well-formdness* e corretude dos modelos. Não foi possível observar, por meio do estudo realizado, abordagens que caracterizavam a complexidade dos dados manipulados pelas transformações de modelos e que otimizassem de alguma forma os critérios de cobertura por meio da definição dos casos de testes.

Embora existam abordagens que testem as transformações M2T, os principais dilemas ainda carecem de muita pesquisa e investigação. A complexidade dos dados de testes que envolve o modelo de entrada, o transformador e os artefatos de saída, configuram-se como um gap relevante para investigações futuras.

No próximo capítulo é apresentado como proposta deste trabalho uma abordagem para caracterização de dados complexos baseada em hipergrafos. A abordagem é estruturada em fases e discutida em detalhes.

Capítulo 4

Abordagem para Caracterização de Dados Complexos Baseada em Hipergrafo

4.1 Considerações Iniciais

Neste capítulo é apresentada uma abordagem para caracterização de dados complexos baseada em hipergrafo. Apresenta-se também uma visão geral na Seção 4.2, sua estrutura na Seção 4.3 e o detalhamento de cada fase entre as Seções 4.4 e 4.8. Por fim, na Seção 4.9 apresentam-se as considerações finais deste capítulo.

4.2 Visão Geral da Abordagem

A pesquisa foi realizada para formular uma abordagem tendo como escopo os testes de transformações M2T e concentrou seus esforços primordialmente nos estudos e questionamentos encontrados na literatura, com base nas indagações de Harman (2008) sobre os testes de transformações quanto à combinação do teste de software e os tipos de transformações e nos trabalhos de Sendall e Kozaczynski (2003), Biehl (July 2010) e Guerra e Soeken (2013), os quais estabelecem que a complexidade do paradigma MDD está diretamente relacionada a três etapas envolvidas nas transformações de modelos, sendo elas: a complexidade dos modelos de entrada, a complexidade das regras de transformações e a complexidade dos artefatos de saída.

Existem também problemas específicos quanto à aplicação das técnicas de teste estrutural e a definição de critérios de cobertura que contemplem as diferenças entre a estrutura do modelo e o código-fonte gerado, uma vez que a definição dos casos de teste gerados como modelos de entrada não implicam necessariamente a cobertura completa do código-

fonte gerado. Nesse contexto, Kirner (2009) e Baudry et al. (2010) identificaram algumas questões relevantes quanto a aplicação das técnicas de teste estrutural em transformações M2T, sendo elas:

1. Os diferentes tipos de linguagem de modelagem e seus estilos singulares.
2. As linguagens de modelagem podem utilizar abstrações complexas que omitam detalhes da implementação, tornando complicado mapear as entidades estruturais entre modelo e código fonte gerado.
3. A geração de código pode ser parametrizável, ou seja, os modelos semânticos dependem das configurações para geração do código fonte.
4. A complexidade dos dados manipulados pelas transformações M2T.

E as conclusões obtidas por meio da Revisão Sistemática apresentada na Seção 3.5, permitem assegurar que as abordagens de teste M2T começam a mudar o seu foco inicial, antes pautados primordialmente na boa formação *(well-formedness)* e corretude dos modelos. É possível constatar também que a maioria das abordagens de teste M2T fazem uso do padrão comportamental método *template*, alinhados aos esforços do *OMG* para normatizar as transformações M2T, na tentativa de aumentar a garantia da qualidade por meio do teste de software.

Diante das inúmeras questões e das oportunidades de pesquisa constatadas, a abordagem proposta neste capítulo consiste em caracterizar o modelo de entrada de uma transformação M2T, de modo a gerar casos de testes que exercitem os *templates* envolvidos nas transformações e garantir níveis satisfatórios quanto aos critérios de cobertura definidos para os testes.

4.3 Estrutura da Abordagem

A estratégia da abordagem propõe que os dados complexos (gramáticas e metamodelos) sejam representados por meio de hipergrafos direcionados, permitindo que um algoritmo de percurso em hipergrafos, usando combinatória, crie subconjuntos dos modelos de entrada que serão utilizados como caso de teste para as transformações M2T. O transformador exercita cada caso de teste gerado, fornecendo juntamente com o código arquivos com mapeamentos e rastreabilidade de cada transformação. Posteriormente, estes arquivos são analisados a fim de possibilitarem o cômputo dos critérios de cobertura considerados para abordagem. Nas Figuras 4.1 e 4.2 apresenta-se a visão geral da estrutura proposta. Basicamente a abordagem está dividida em cinco fases, sendo elas:

1. A Representação Baseada em Gramáticas e Metamodelos.
2. A Caracterização com Hipergrafos.
3. A Geração dos Casos de Teste.
4. A Execução dos Casos de Teste.
5. A Análise e Cômputo dos Critérios de Cobertura.

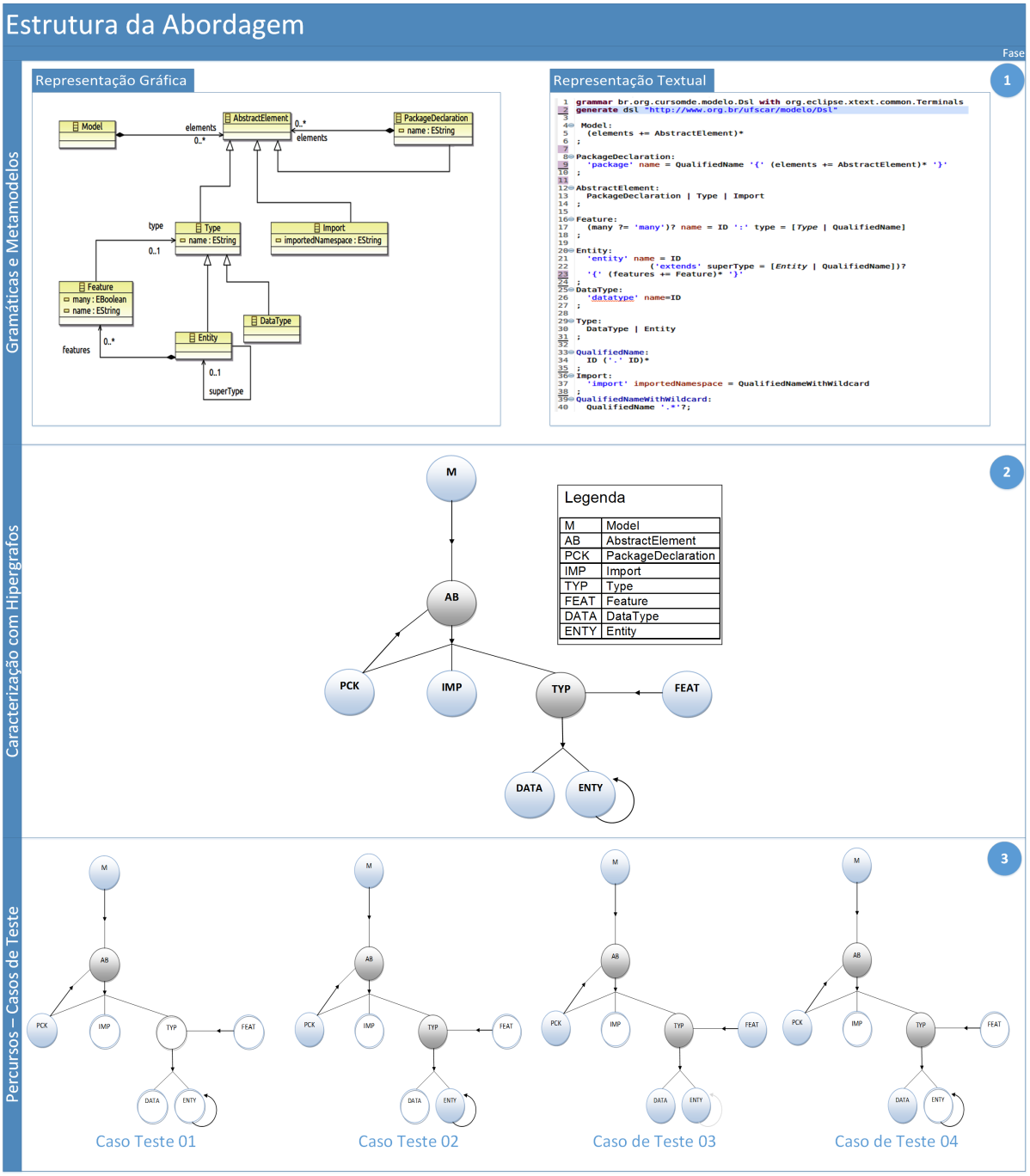


Figura 4.1: Estrutura da Abordagem - Parte 01

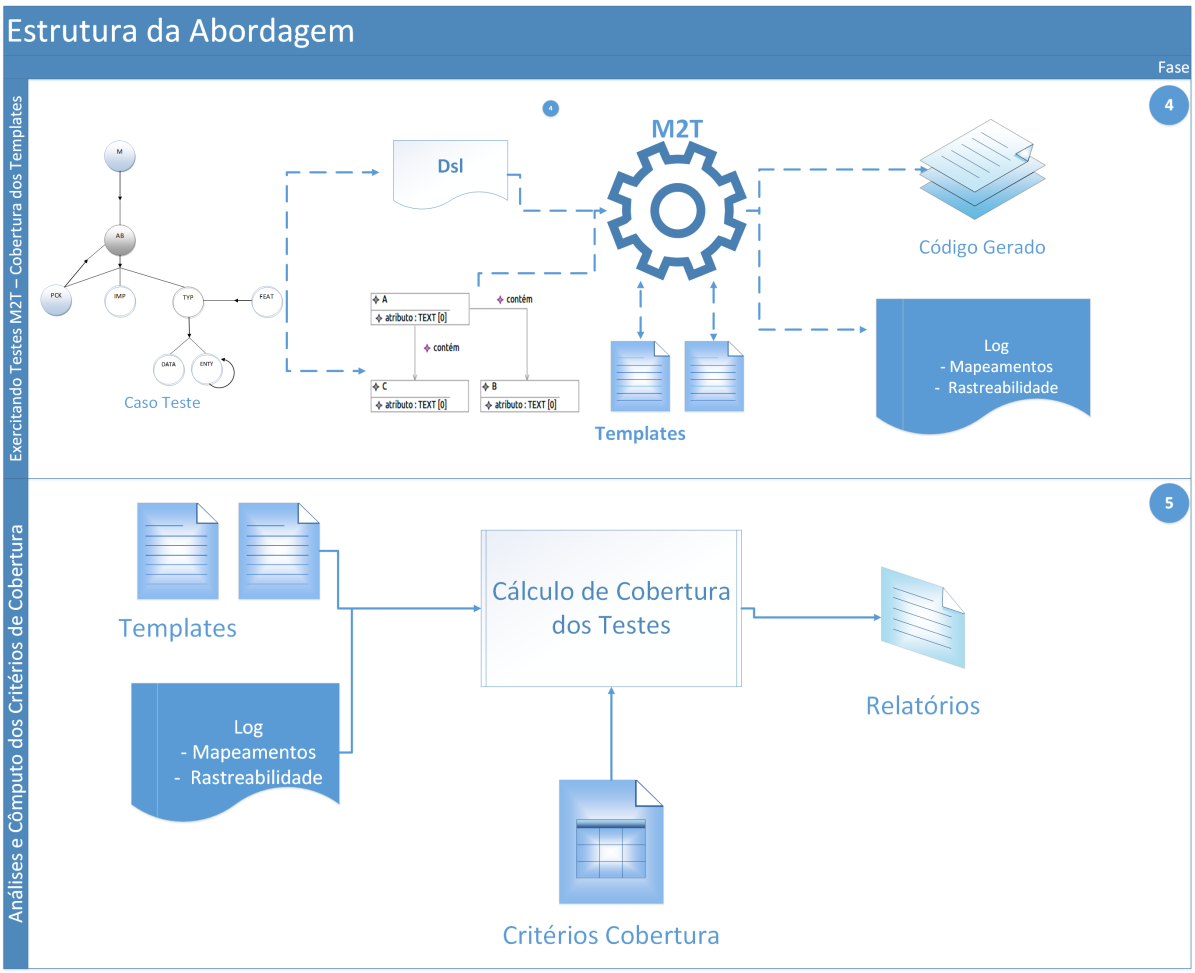


Figura 4.2: Estrutura da Abordagem - Parte 02

A Fase 1 aborda o quanto é importante o processo de representação de estruturas complexas como gramáticas e metamodelos, por meio de estratégias que mantenham ao máximo suas características e peculiaridades. Neste momento da abordagem a preocupação está alinhada às questões de Kirner (2009) e Baudry et al. (2010), que alertam sobre os tipos de linguagens de modelagem e suas singularidades. Na segunda fase aplica-se o processo de caracterização dos dados complexos por meio de hipergrafos direcionados. Todo processo é fundamentado nos trabalhos sobre a representação de Editores de Diagramas (MINAS; VIEHSTAEDT, 1995), Gramáticas (BATORY, 2005) e Diagramas de *Features* (LAGUNA; MARQUÉS; RODRÍGUEZ-CANO, 2011).

Após a representação dos elementos por meio dos hipergrafos direcionados, a Fase 3 demonstra a geração de percursos no hipergrafo e volta toda sua preocupação no trato e validação semântica, conforme as restrições da gramática ou do metamodelo utilizado. Os percursos validados são eleitos como potenciais casos de teste. Ainda na Fase 3, para cada percurso gerado e validado, cria-se um modelo de acordo com as características dos

elementos presentes no percurso traçado. Na Fase 4, transformações M2T são exercitadas por meio dos casos de teste gerando os artefatos necessários para as análises e cômputos executados na fase seguinte.

Por fim, na Fase 5 é analisada e computada a cobertura de cada caso de teste exercitado pelas transformações M2T, conforme os critérios de cobertura definidos. Assim, de modo sistemático, a abordagem conclui sua avaliação propondo uma solução que almeja responder cada uma das questões levantadas na Seção 4.2.

Nas próximas Seções serão explicadas as metodologias utilizadas em cada uma das fases da abordagem.

4.4 A Representação Baseada em Gramáticas e Metamodelos

Fazer uso de representações que facilitem a compreensão de dados complexos em aplicações de software é uma necessidade com destaque na literatura, especialmente em sistemas com linguagens visuais. Assim, usar algum tipo de formalismo com representação gráfica das estruturas de dados complexos constitui uma estratégia promissora.

A estratégia de representação utilizada neste trabalho considerou essencialmente dois tipos de abordagens relevantes no processo de investigação da literatura, sendo elas: Abordagens baseadas em Gramáticas (BETTINI, 2013) e as abordagens baseadas em Metamodelos (OMG, 2013). Assim, de modo natural, a estratégia adotou tecnologias interoperáveis e com o máximo de compatibilidade. Para as abordagens baseadas em gramáticas, utilizou-se o Xtext (ECLIPSE.ORG, 2013b). Para abordagens baseadas em metamodelos selecionou-se o GMF (ISTRIA et al., 2013).

O Xtext é uma ferramenta que permite definir gramáticas com sintaxe própria adaptadas do padrão EBNF. Possui um conjunto de APIs que permitem descrever os diferentes aspectos relevantes aos modelos e suas validações sintáticas e semânticas. Adicionalmente, possui a característica de integrar-se em tempo de execução aos componentes pautados no *Eclipse Modeling Framework (EMF)* (STEINBERG et al., 2009), permitindo, assim, que uma quantidade maior de recursos e funcionalidades sejam utilizados por outros frameworks como, por exemplo, *Graphical Modeling Project - GMF* (ECLIPSE.ORG, 2013b).

O *Graphical Modeling Framework (GMF)* fornece integração entre o gerador *Eclipse Modeling Framework (EMF)* e o *Graphical Editing Framework (GEF)*. Esta integração

permite relacionar o conjunto de elementos que compõem o diagrama com um modelo de domínio, criando, assim, uma representação visual do diagrama para um modelo aplicável a qualquer domínio (ISTRIA et al., 2013).

O modelo *Ecore EMF* (STEINBERG et al., 2009) basicamente consiste em uma estrutura de objetos instanciados denominados *EPackage*. Cada *EPackage* contém outros elementos denominados *EClass*, *EDataTypes* e *EEnum*, que possuem a capacidade de integração com abordagens baseadas em gramáticas (Xtext) ou metamodelos (GMF). Utilizando o *Xtext* é possível inferir modelos Ecore por meio de uma gramática, como também importar modelos já existentes. No caso do GMF, o uso de seu editor permite a relação entre os objetos Ecore por meio de uma diagramação visual. Essas características permitem que as duas abordagens sejam mescladas, por meio dos modelos Ecore, tanto inferindo uma gramática quanto diagramando visualmente um modelo. Assim, é possível a reutilização de abstrações existentes mantendo a flexibilidade e a capacidade de adequações dos modelos *Ecore EMF*.

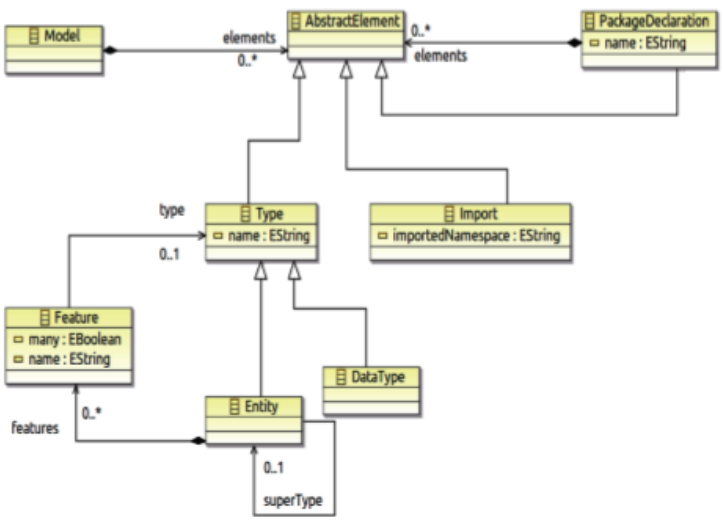
A integração entre as tecnologias Xtext e GMF pode ser vista na Figura 4.3. O exemplo apresentado na figura demonstra como as duas tecnologias podem ser utilizadas e evidencia a fiel correspondência entre as representações textuais e gráficas, sejam elas gramáticas ou metamodelos. Essa representação permite que sejam geradas classes Java para entidades que são definidas no modelo de domínio para as transformações M2T. Este mesmo exemplo será utilizado para ilustrar as próximas fases da abordagem.

4.5 A Caracterização com Hipergrafos Direcionados

Modelar problemas usando grafos está diretamente relacionado à capacidade de abstração do ser humano, o que é particularmente interessante, pois essa forma de caracterização não só estabelece relações entre os objetos, mas também permite formular questões inerentes a um domínio representado (STEEN, 2010). Imaginando um tráfego rodoviário entre cidades podem-se formular questões que vão além das relações de distância e vizinhança, como: qual a ordem de um determinado percurso executado entre as cidades? Todas as entregas podem ser realizadas? Assim, além dos tipos de dados abstratos os modelos computacionais encapsulam toda semântica para propor soluções algorítmicas que respondam a estes questionamentos.

Como já visto nas Subseções 2.6.1 e 2.6.3, um hipergrafo é uma generalização de grafos, cujas arestas podem conectar mais de dois vértices e são chamadas de hiperarestas.

Metamodelo (GMF)



Gramática (Xtext)

```
1  grammar br.org.cursomde.modelo.Dsl with org.eclipse.xtext.common.Terminals
2  generate dsl "http://www.org.br/ufscar/modelo/Dsl"
3
4  Model:
5    (elements += AbstractElement)*
6  ;
7
8  PackageDeclaration:
9    'package' name = QualifiedName '{' (elements += AbstractElement)* '}'
10 ;
11
12 AbstractElement:
13   PackageDeclaration | Type | Import
14 ;
15
16 Feature:
17   (many ?= 'many')? name = ID ':' type = [Type | QualifiedName]
18 ;
19
20 Entity:
21   'entity' name = ID
22               ('extends' superType = [Entity | QualifiedName])?
23   '{' (features += Feature)* '}'
24 ;
25 DataType:
26   'datatype' name=ID
27 ;
28
29 Type:
30   DataType | Entity
31 ;
32
33 QualifiedName:
34   ID ('.' ID)*
35 ;
36 Import:
37   'import' importedNamespace = QualifiedNameWithWildcard
38 ;
39 QualifiedNameWithWildcard:
40   QualifiedName '.*'?;
```

Figura 4.3: Abordagens Baseadas em Gramáticas (Xtext) e Metamodelos (GMF)

Nesta fase da abordagem, o processo de caracterização dos dados complexos por meio de hipergrafos leva em consideração a equivalência entre os elementos representados com as tecnologias do Xtext e GMF. Assim, a abordagem faz uso das características mescladas das duas tecnologias e toda capacidade de interoperabilidade existente entre elas. Por exemplo, os elementos *EClass* do modelo Ecore tornam-se os vértices do hipergrafo, enquanto os relacionamentos e associações entre as *EClass* são representadas como hiperarestas.

Na Seção 4.4 foi apresentado um exemplo de gramática/metamodelo que também será utilizado para descrever o processo de caracterização desenvolvido nesta fase. Nas Figuras 4.4 e 4.5 apresenta-se a caracterização dos dados da gramática ou metamodelo conforme o modelo Ecore, por meio de um hipergrafo direcionado.

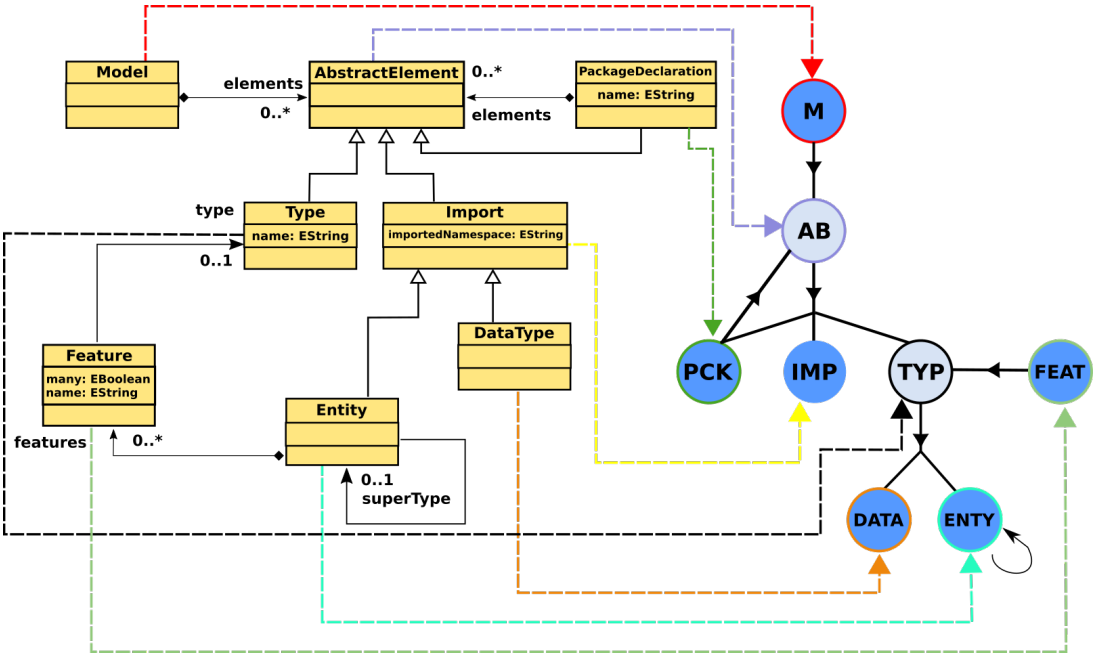


Figura 4.4: Caracterização Metamodelo por meio de Hipergrafos

Os relacionamentos são elementos semânticos de extrema importância nos modelos, dada a capacidade de estabelecerem associações entre os elementos que o compõem. Assim, nesta fase da abordagem as hiperarestas representam estes relacionamentos e acumulam em suas propriedades informações sobre a cardinalidade mínima e máxima e uma propriedade peculiar do *EMF Core* denominada *"Containment"* para indicar as associações do tipo composição, ou seja, a parte não existe sem o todo. Já os vértices são representados pela associação de todos os *EDataTypes* presentes no elemento *EClass*. Adicionalmente, a propriedade denominada *isAbstract* foi acrescentada aos vértices para determinar quais elementos podem ser ou não adicionados pelo algoritmo de busca implementado. Algumas abstrações semânticas que não podem ser expressas pelo formalismo dos hipergrafos

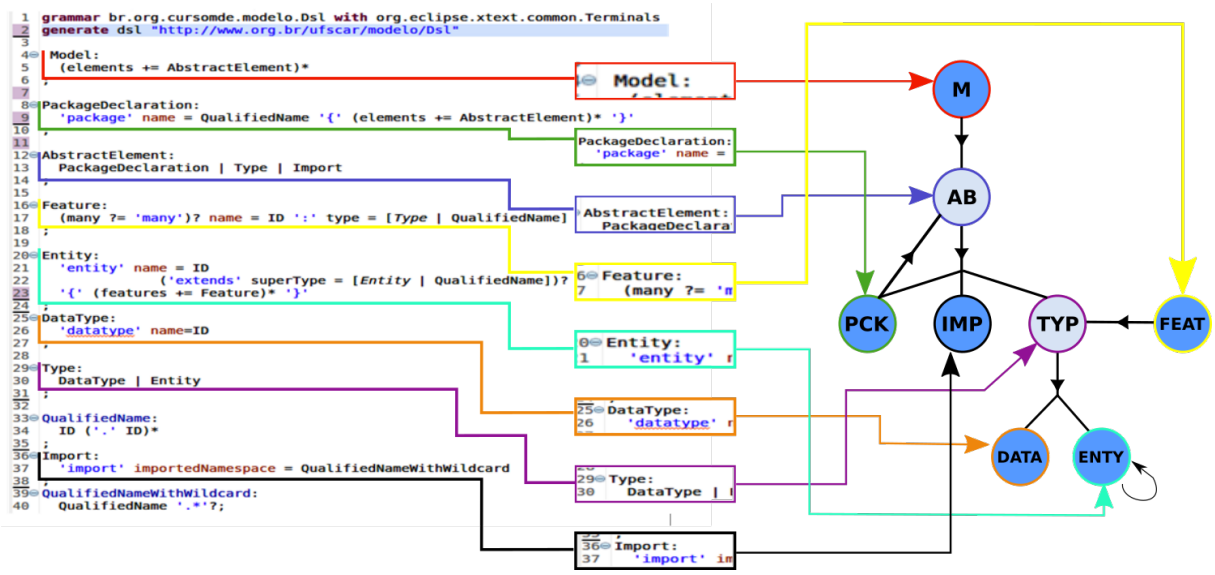


Figura 4.5: Caracterização Gramática por meio de Hipergrafos

podem compor restrições adicionais que auxiliam tanto na especificação semântica do modelo quanto no algoritmo de busca que geram os caminhos em hipergrafos, garantindo a maior representação possível dos dados abstraídos do modelo.

Cabe ressaltar que as propriedades como "*Containment*" e "*isAbstract*" não são enfaticamente representadas nas ilustrações dos hipergrafos, porém são implementadas nos algoritmos utilizados. Algumas características salientes e outros tipos de relações encontradas entre os elementos *EClass* são descritas e implementadas por meio de restrições aplicados ao algoritmo de percurso implementado. Na Tabela 4.1 apresenta-se três dessas restrições implementadas para o modelo ilustrado na Figura 4.3.

Tabela 4.1: Restrições Semânticas

A- isRoot	Validar a presença do vértice origem como raiz para todos os hipercaminhos.
B- isAbstract	Validar a obrigatoriedade de um vértice abstrato e implementar pelo menos um de seus vizinhos contidos na hiperaresta.
C- isContainment	Validar se um par vértice na hiperaresta com relacionamento do tipo *"Containment=true"*, torna sua cardinalidade impositiva(*i.e.*, o vértice origem devem implementar um de seus vizinhos).

4.6 Geração dos Casos de Teste - Algoritmo de Percurso em Hipergrafos

Nas fases anteriores a abordagem formulou-se uma estratégia de representação e caracterização dos dados complexos por meio do formalismo dos hipergrafos. Nesta fase,

todas as preocupações são voltadas para as questões que tratam da definição do conjunto de dados de teste.

Os casos de testes que serão gerados por meio da abordagem e possuem um propósito específico de exercitar os *templates* utilizadas nas transformações M2T. Portanto, eles devem ser gerados observando as peculiaridades e características dos critérios de cobertura utilizados em teste caixa branca.

Boa parte das soluções que fazem uso de transformações com Método Templates utilizam implementações de referência. As mudanças e atualizações constantes nestas implementações podem gerar *templates* com estruturas desatualizadas e segmentos de código que jamais serão utilizados pelos modelos de entrada ou até mesmo *templates* com segmentos de código desnecessários, os quais podem causar anomalias na aplicação. A testabilidade dos *templates*, aliada a bons conjuntos de dados de testes, pode aumentar a detecção de defeitos que vão além da incorretude e má formação.

Para avaliar a viabilidade e a eficácia da estratégia adotada sobre a definição do conjunto de dados de teste, a abordagem pautou-se na metodologia descrita por Myers et al. (2004), cujos os estudos definiam um conjunto de regras para projetar casos de teste com maior probabilidade de detecção de erros. Uma dessas recomendações preditas por Myers et al. (2004) afirma que a metodologia menos eficaz de todas são conjuntos de dados de teste gerados aleatoriamente, pois selecionam subconjuntos de todos os possíveis valores de entrada. Em termos probabilísticos para detecção de defeitos, os dados de testes selecionados de forma aleatória possuem poucas chances de formarem um conjunto ideal ou próximo do ideal. Entretanto, uma quantidade substancial de trabalhos como, por exemplo, os estudos de Arcuri, Iqbal e Briand (2012) e Shamshiri et al. (2015) vêm investigando a geração aleatória de dados de testes com intuito de comprovar sua relevância como uma técnica de teste útil. A simplicidade desta técnica permite análises matemáticas tratáveis e, consequentemente, alguns resultados demonstram que existem situações práticas em que a geração de dados de testes aleatórios configura-se como uma opção viável.

Neste sentido, a abordagem proposta reafirma os preceitos de Myers et al. (2004) e não faz uso da geração de dados de testes aleatórios, tendo em vista a complexidade dos dados envolvidos nas transformações de modelos. Para tanto, a abordagem faz uso dos conceitos da análise combinatória aplicadas às estruturas de hipergrafos, implementando e adaptando um algoritmo de busca capaz de retornar pelo menos uma estrutura combinatória como, por exemplo, hipercaminhos (GALLO et al., 1993), que atenda às restrições e requisitos semânticos abstraídos da representação, seja ela uma gramática ou metamodelo.

A definição do algoritmo de busca para estabelecer os percursos entre os elementos, ou seja, dados complexos caracterizados por meio dos hipergrafos, faz uso dos alargamentos conceituais propostos por Guedes, Markenzon e Faria (2011), que tratam situações em que estruturas representadas podem não atender às definições tradicionais de hipergrafos direcionados. Assim, Guedes, Markenzon e Faria (2011) definem essas estruturas como uma extensão de hipergrafo equivalente, ou seja, estruturas que permitem ciclos e atribuições de pesos entre os possíveis caminhos.

A implementação e adaptação do algoritmo de busca exigiu a superação de alguns desafios, uma vez que grande parte dos estudos publicados que abordavam hipercaminhos em hipergrafos tratavam de problemas relacionados à otimização de rotas de menor percurso e outros tipos de conectividade (NIELSEN; ANDERSEN; PRETOLANI, 2005; AUSIELLO; FRANCIOSA; FRIGIONI, 2001; PRETOLANI, 2000).

As adaptações implementadas no algoritmo foram guiadas pelos estudos realizados por Pandy e Chawla (2003) e Thakur e Tripathi (2009) que abordavam os conceitos de hipercaminhos e hiperciclos para hipergrafos. A abordagem de Nielsen, Andersen e Pretolani (2005) propõe soluções para hipercaminhos mínimos com restrições (*K shortest hyperpaths*) enquanto a solução proposta por Nortje, Britz e Meyer (2012) usa algoritmos padrões de hipercaminhos para extrair módulos de localidade sintáticas para ontologias SROIQ. Estes estudos possibilitaram a adaptação do algoritmo para identificar e tratar os ciclos encontrados em cada hiperarco e aplicar as validações semânticas por meio de restrições (*constraints*) necessárias aos desafios da proposta deste trabalho.

O Algoritmo 1 *SB-Visita*, adaptado de (GUEDES; MARKENZON; FARIA, 2011), implementa um percurso em hipergrafos que considera os problemas inerentes ao conceito de hipercaminhos, no qual a partir de um vértice de origem s em um hipergrafo H é possível encontrar todos os vértices que são a ele conectados. A solução por meio de uma variação do algoritmo de busca em profundidade para grafos direcionados procura identificar todos os vértices vizinhos que pertencem ao vértice-origem da hiperaresta. Adicionalmente, a solução foi adaptada para tratar a detecção de ciclos, eliminar caminhos redundantes e validar cada percurso por meio da aplicação das restrições semânticas abstraídas do modelo.

Neste algoritmo a escolha de qual hiperarco percorrer é um ponto crucial que será detalhado. Cabe ressaltar que é preciso "desfazer" um percurso já feito para que outro caminho seja possível, pois precisa-se garantir que os vértices visitados só são usados uma vez como destino e uma vez como origem para cada ramo do percurso. Isso é feito na

Algoritmo 1: SB-Visita Adaptado de (GUEDES; MARKENZON; FARIA, 2011)

Entrada: Uma tripla $H = (V, E, s)$, no qual (V, E) é um hipergrafo direcionado, $s \in V$ é o vértice origem, e existe um B-caminho de s para qualquer outro vértice em V
Saída: Lista de Percursos - Hipercaminhos
Dados: $H = (V, E)$ e $s \in V$

```
1  início
2      para cada v ∈ V faça
3          B_v ← F_v ← 0 ;
4          B_s ← λ;
5          para cada e ∈ E faça
6              P_e ← 0 ;
7              Empilhe s em Q;
8              enquanto Q ≠ ∅ faça
9                  x ← Topo de Q ;
10                 se existir e ∈ FS(x) tal que P_e = 0 então
11                     para todo v ∈ Org(e) faça
12                         B_v ≠ 0 ;
13                         F_v = 0 ;
14                     fim
15                     para todo w ∈ Dest(e) faça
16                         B_w = 0 ;
17                     fim
18                     para todo v ∈ Org(e) faça
19                         F_v ← e ;
20                     fim
21                     para todo w ∈ Dest(e) faça
22                         B_w ← e ;
23                     fim
24                     P_e = x ;
25                     Empilhe cada w ∈ Dest(e) em Q;
26                 senão
27                     Desempilhe x de Q;
28                     a ← B_x ;
29                     B_x ← 0 ;
30                     se a ≠ λ então
31                         para todo w ∈ Dest(a) faça
32                             B_w = 0 ;
33                         fim
34                         para todo v ∈ Org(a) faça
35                             F_v = 0
36                         fim
37                     fim
38                     para todo b ∈ FS(x) tal que P_b = x faça
39                         P_b ← 0
40                     fim
41                 fim
42             fim
43         fim
44     fim
45 fim
```

linha 3 do Algoritmo 1 com dois rótulos, B_v e F_v, indicando respectivamente o hiperarco usado para chegar em v e o hiperarco usado para sair de v. O rótulo λ representa um hiperarco fictício na inicialização da linha 4 do Algoritmo 1.

Para garantir que um vértice v pode ser usado como origem de um hiperarco a ser percorrido, basta verificar se $B_v \neq 0$ e $F_v = 0$, ou seja, entre os hiperarcos percorridos neste ramo existe um hiperarco que chega em v, mas não existe um hiperarco que sai de v. E para saber se um vértice w pode ser usado como destino, basta que $B_w = 0$. As linhas de 10 a 17 permitem saber se o hiperarco pode ser percorrido, verificando se os vértices de $Org(e)$ podem ser usados como origem e os vértices de $Dest(e)$ como destino.

Ao percorrer um hiperarco e, os rótulos são atualizados de forma que $F_v = e$ para todo $v \in Org(e)$ e $B_w = e$ para todo $w \in Dest(e)$. Isso indica que os vértices de $Org(e)$ já não podem mais ser usados como origem, e que os vértices de $Dest(e)$ passam a ser usados como origem. Nas linhas 18 a 23 pode-se observar esta atualização.

O trecho das linhas 28 a 34 desfaz um percurso por hiperarco quando o vértice x é retirado da pilha e $B_x = 0$, ou seja, x volta a ser usado como destino. O hiperarco usado para chegar em x, a deve ser analisado, e se todos no seu destino já saíram da pilha, então para cada $v \in Org(a)$ deve-se atribuir $F_v = 0$. Esta operação indica que o percurso está retornando e permite que o hiperarco a possa ser percorrido outra vez.

As operações com rótulos B e F do algoritmo ajudam a ter informações sobre os vértices apenas. Estas informações são úteis para decidir se um hiperarco pode ser percorrido ou não. Assim, para poder percorrer um hiperarco mais de uma vez, mas sem que o algoritmo fique indefinidamente repetindo o mesmo hiperarco, usa-se o rótulo P_e, que indica qual vértice foi utilizado para alcançar o hiperarco e. Na linha 6 é possível observar a inicialização de $P_e = 0$, indicando que o hiperarco e pode ser percorrido. Já na linha 10 do algoritmo é realizado o teste para saber se o hiperarco pode ser percorrido, agora usando o rótulo P_e.

Para que o hiperarco e não seja percorrido duas vezes seguidas partindo do mesmo vértice, o valor de P_e só será modificado (para zero) quando o vértice P_e sair da pilha. Assim, não existe o risco de repetir um hiperarco sem que a pilha tenha mudado e, portanto, a partir de um vértice x, cada hiperarco de $FS(x)$ é percorrido no máximo uma vez enquanto x está no topo da pilha. O trecho do algoritmo é apresentado no intervalo das linhas 28 a 40 e não impede que um hiperarco venha ser percorrido mais de uma vez. Se o vértice x voltar ao topo da pilha, o mesmo hiperarco poderá ser percorrido, porém a pilha terá outra configuração e consequentemente um outro percurso.

Os Algoritmos 2 - Detecção de Ciclos, 3 - B-Visita, 4 - Achar Vértice, 5 - Busca Ciclos e 6 Valida Restrições *(Constraints)* complementam a solução e serão descritos brevemente por suas funcionalidades primordiais. Estes algoritmos fazem uso das seguintes estruturas de dados auxiliares:

- Lista dos Vértices (hiper-nós) (HL): é uma lista com todos os vértices do hipergrafo. A lista é construída com uma operação de remoção de vértices duplicados em HL.
- Matriz de Vértices (VM): é o conjunto binário que representa os hiper-nós com seus respectivos nós simples, ou seja, $VM =\mid HL \mid \times \mid V \mid$ de zeros(0) e uns(1).

- Lista de Adjacência (AS): é o conjunto das listas de adjacências de todos os vértices (hiper-nós) em HL.
- Super Lista (SL): é uma lista com o conjunto das listas de hiper-nós para cada hiper-nó pertencente a HL. Pode ser construída por meio da Matriz de Vértices.

O Algoritmo 2 é a chamada principal que indica o percurso do tipo busca em profundidade no hipergrafo, chamando o procedimento *visita*() para cada vértice não visitado. No trecho do algoritmo das linhas 1 a 11 pode-se observar a inicialização da pilha, dos vértices não visitados e a chamada para próximo procedimento.

Algoritmo 2: DetectaHiperCiclo

Entrada: Uma tripla $H = (V,E,s)$, no qual (V,E) é um hipergrafo direcionado, $s \in V$ é o vértice origem, e existe um SB-caminho de s para qualquer outro vértice em V
Saída: Lista dos hiperciclos ao longo dos hipercaminhos com origem em s
Dados: $H = (V,E)$ e $s \in V$, Lista Superset SL, Lista Adjacência AL, Lista Hiper-nós HL

```
1  início
2  |  S ← ∅ Inicializando a Pilha;
3  |  para cada v ∈ HL faça
4  |  |  visitado[v] = false
5  |  fim
6  |  para cada v ∈ HL faça
7  |  |  se visitado[v] = false então
8  |  |  |  visita(v);
9  |  |  fim
10 |  fim
11 fim
```

O procedimento B-Visita do Algoritmo 3 recebe um parâmetro e o empilha em S (linha 3), iniciando o percurso pelo vértice w. Caso o vértice o já exista na pilha, um hiperciclo é detectado e removido (linhas 5-8). Nas linhas de 10 a 15 cada elemento não visitado de $SL(w)$ é visitado chamando o método recursivamente. Finalmente, todos os elementos não visitados em $AS(w)$ são visitados pela chamada recursiva do procedimento *visita*(v). Nas linhas 21 e 22, já no final do percurso w é marcado como visitado e retirado da pilha.

O Algoritmo 4 com o procedimento AchouVertice é invocado pelo procedimento $B-$*Visisa* com o vértice w como parâmetro e retornando o índice da primeira ocorrência w no SL na pilha.

O procedimento do Algoritmo 5 é invocado pelo método *visita*(u) e enumera o hiperciclo entre a origem o e o destino u, que são passados como parâmetros. Assim, pares consecutivos de vértices na pilha são analisados. Caso exista uma hiperaresta no hipergrafo com dois vértices escolhidos com origem e destino, estes são adicionados à lista de arestas que participam do hiperciclo.

Finalmente, o último procedimento do Algoritmo 6 valida as restrições semânticas definidas para a gramática/metamodelo, sendo aplicada cada restrição à lista de percursos

Algoritmo 3: B-Visita

```
1  Procedimento visita()
   Entrada: Um hiper-nó w
2  início
3      Empilhe w em S;
4      o = isPresent(w) ;
5      se o ≠ null então
6          buscaCiclo(o,w);
7          visita(o);
8      fim
9      l1 = SL(w);
10     para cada v ∈ l1 faça
11         se visitado[v] = falso então
12             visita(v);
13         fim
14         l2 = AL(w);
15     fim
16     para cada v ∈ l2 faça
17         se visitado[v] = falso então
18             visita(v) ;
19         fim
20     fim
21     visitado[w] = verdadeiro;
22     Desempilhe w de S;
23 fim
```

Algoritmo 4: achouVertice

```
1  Procedimento achouVertice()
   Entrada: Um hiper-nó w
   Saída: Primeiro elemento e na pilha tal que w ∈ e
2  início
3      Desempilhe temp de S;
4      S1 ← ∅ Inicializando nova Pilha;
5      enquanto S ≠ ∅ faça
6          Desempilhe x de S;
7          Empilhe x em S1;
8          se w ∈ x então
9              enquanto S1 ≠ ∅ faça
10                 S.push(S1.pop())
11             fim
12             Empilhe temp em S;
13             retorna x
14         fim
15     fim
16     enquanto S1 ≠ ∅ faça
17         S.push(S1.pop())
18     fim
19     Empilhe temp em S;
20     retorna null;
21 fim
```

livres de hiperciclos já processada. Nas linhas 13 a 23 cada percurso é analisado conforme as restrições definidas e marcações de verdadeiro ou falso são retornadas, validando cada restrição por percurso livre de hiperciclos.

Originalmente, o custo computacional do algoritmo *SB-VISIT* aplicado a hipergrafos direcionados com ausência de ciclos é $O(size(H))$ para encontrar uma ordenação válida, assumindo que $size(H)$ é a somatória da cardinalidade do hiperarco. Entretanto, as adaptações implementadas executam uma busca em profundidade para cada iteração que ocorre nos hiperarcos conhecidos para detectar os possíveis ciclos e eliminar caminhos

Algoritmo 5: buscaCiclo

```
1  Procedimento encontraCiclo()
   Entrada: Um hiper-nó Org(a) e Dest(u)
   Saída: Hcomciclo[o,u]resolvido
2  início
3      C ← ∅ Lista de arestas detectadas com Ciclo;
4      Desempilhe prev de S;
5      enquanto true faça
6          Desempilhe curr de S;
7          se (e = newEdge(curr, prev)) ∈ E então
8              Insere e de C
9          fim
10         se curr = 0 então
11             break
12         fim
13         prev = curr
14     fim
15     paraCiclo(C)
16 fim
```

Algoritmo 6: validarRestrições

```
1  Procedimento validarRestrições()
   Entrada: H livre de ciclos * Lista de hipercaminhos
   Saída: Lista de hipercaminhos validados
2  início
3      para cada v ∈ V faça
4          P_v ← ∅;
5      fim
6      para cada e ∈ E faça
7          K_e ← 0;
8      fim
9      Q ← s ;
10     R ← c Lista de restrições semânticas P_s ← λ ;
11     repita
12         Escolha e Remova x ∈ Q;
13         para cada e ∈ FS(x) faça
14             K_e ← K_e + 1;
15             se K_e = |Org(e)| então
16                 para cada w ∈ Dest(e)talqueP_w = ∅ faça
17                     se confirmarRestrição(R,w)=verdadeiro então
18                         P_w ← e;
19                         Insira w em Q;
20                         isTrue = verdadeiro;
21                     senão
22                         Insira w em Q;
23                         isTrue = falso;
24                     fim
25                 fim
26             fim
27         fim
28     até Q = ∅;
29 fim
```

redundantes. Como não é possível garantir qual será a ordem escolhida de exploração dos vértices, nem se alguma ordem de escolha implica ou não em caminhos redundantes, o algoritmo deve permitir que um hiperarco possa ser percorrido mais de uma vez. Com isso, a solução deixa de ter complexidade de tempo polinomial e passa a ter complexidade exponencial.

Os estudos de Thakur e Tripathi (2009) afirmam que a complexidade computacional para traçar hipercaminhos em hipergrafos com ciclos é um problema NP-Completo. Por

outro lado, os estudos também demonstram que o uso de restrições aplicadas ao algoritmo pode diminuir seu custo. Gallo et al. (1998) propõem uma abordagem que faz uso destas restrições para resolver problemas de satisfatibilidade em hipergrafos direcionados valorados. A abordagem proposta neste trabalho também faz uso de restrições alinhadas ao seu contexto de aplicação. Por exemplo, ao identificar um ciclo no percurso, a adaptação do algoritmo força a solução a seguir o fluxo até o último vértice-folha da hiperarco, evitando assim que o hiperarco seja percorrido indefinidamente.

Cabe ressaltar que neste primeiro momento não houve preocupações quanto ao custo computacional das adaptações implementadas no algoritmo. A melhor solução utilizada pelos métodos anteriores foi definida como aquela que está mais próxima da raiz, ou seja, a que é obtida com a menor quantidade de aplicações de regras. A utilização dessas regras não está associada a qualquer custo. Isto porque nos problemas estudados não havia interesse em se descobrir o custo da solução. Entretanto, há muitos problemas em que o custo de uma solução desempenha um papel fundamental para sua aceitação.

Depois que os problemas dos ciclos e percursos redundantes foram resolvidos no algoritmo e um conjunto de caminhos distintos foi obtido, cada caminho é validado conforme as restrições semânticas do modelo apresentadas na Tabela 4.1. O resultado da execução do algoritmo completo é apresentado na Figura 4.6, na qual é possível perceber os percursos gerados pelo algoritmo e uma visualização do percurso graficamente por meio das representações em hipergrafos, conforme o exemplo adotado na Seção 4.4.

4.7 Exercitando Transformações M2T por meio dos Casos de Testes

Usando os percursos válidos gerados pelo algoritmo descrito na Seção 4.6 é possível injetar dados de acordo com a caracterização dos elementos do modelo presente em cada hipercaminho. Assim, casos de testes são gerados para cada percurso válido e posteriormente as transformações M2T são exercitadas possibilitando a análise e cômputo do percentual de cobertura dos testes para cada variação dos casos de testes gerados.

De acordo com Baudry et al. (2006) a combinação das características dos dados injetados para os casos de teste gerados é um fator de impacto relevante quanto à determinação da eficácia dos critérios de cobertura definidos. A abordagem proposta neste trabalho não objetivou neste primeiro momento automatizar ou debater a injeção dos dados conforme

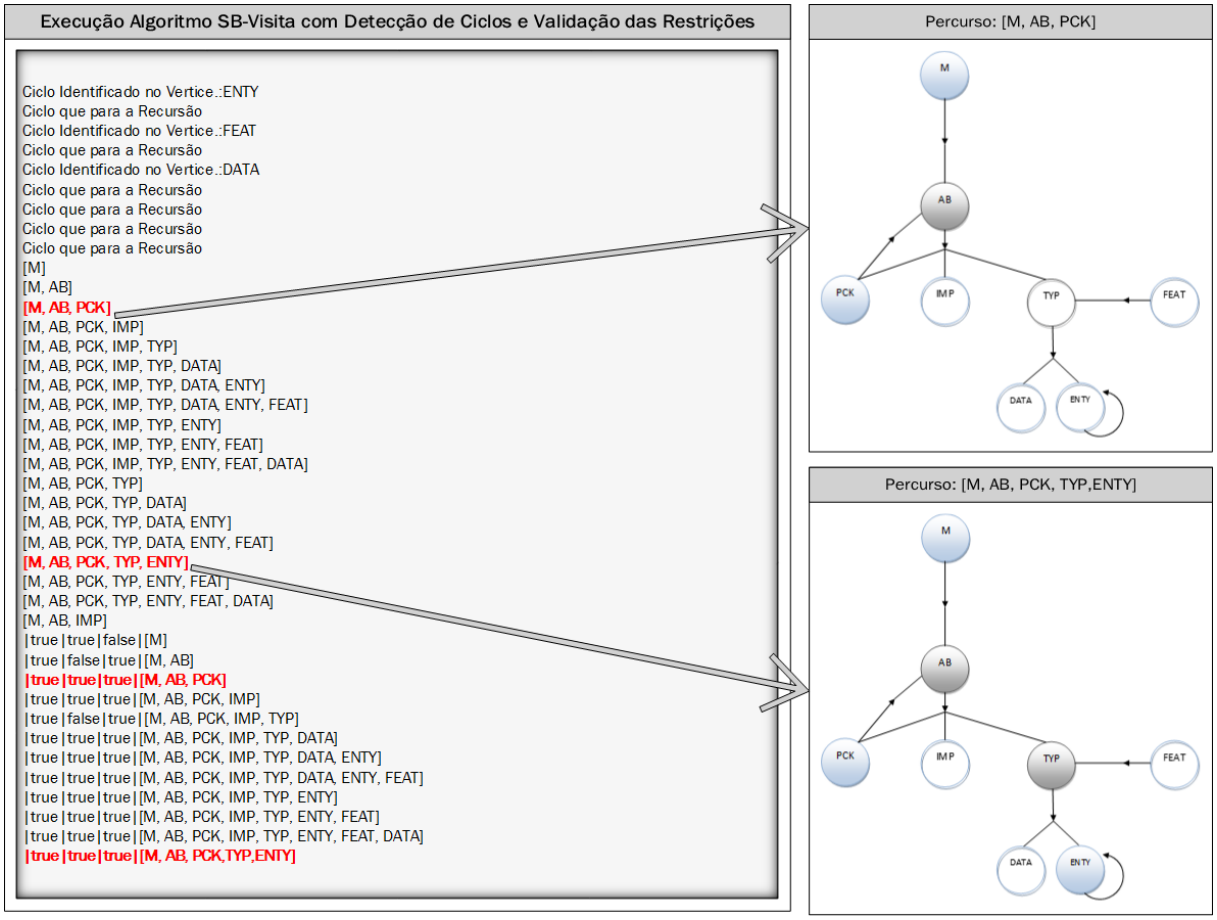


Figura 4.6: Execução da Algoritmo Completo - Percursos traçados e representação do Hipergrafos

os modelos caracterizados. Entretanto, é possível definir e ajustar os critérios de cobertura por meio das características dos dados injetados para cada caso de teste.

A literatura apresenta um vasto número de abordagens que tratam de soluções de automatização e injeção de dados para geração de casos de testes. O estudo apresentado por Sen, Baudry e Mottu (2009) apresenta uma estratégia de automação para casos de testes aplicados a modelos e usa as características da aplicação para delimitar a cardinalidade dos elementos presentes no modelo. Outra estratégia interessante e com bons resultados é a injeção dos dados por meio de algoritmos genéticos que refinam o conjunto de dados para os elementos do modelo, conforme os critérios de cobertura (TIAN; GONG, 2014). Assim, é possível perceber que não existe uma solução ótima para tratar este problema, e que cada abordagem faz uso de recursos que mais satisfazem ao contexto de aplicação das técnicas e critérios dos testes utilizados.

Nesta fase, a abordagem apresentada neste trabalho faz uso do transformador *JET* para exercitar as transformações M2T e gerar além do código-fonte outros artefatos que

possibilitem a análise e cômputo dos critérios de cobertura. Cabe ressaltar que o transformador utilizado possui personalizações e adequações de uso propostas pelas abordagens de Lucrédio (2009) e Possatto (2013).

O trabalho de Lucrédio (2009) especifica e detalha um processo para tratar das inconsistências entre os *templates* e a implementação de referência, de forma a garantir o reuso em MDD. Já os estudos de Possatto (2013), especificam um processo de migração automática de código em MDD. Os autores personalizam o transformador para gerar um arquivo que faz o mapeamento entre os *templates* e o código-fonte, minimizando a inconsistência entre os *templates* e a implementação de referência.

A estratégia da abordagem proposta neste trabalho faz uso das personalizações que geram o arquivo de mapeamento e adicionalmente habilita a opção de rastreabilidade do transformador *JET* para produzir um log em arquivo após cada transformação realizada. Este log é capturado e utilizado no processo de análise e cômputo juntamente com o arquivo de mapeamento.

Como a estratégia da abordagem proposta não objetiva a automatização da geração dos dados para os casos de testes, é permitido ao testador definir a cardinalidade dos elementos segundo o número de ocorrências e relevância para o domínio da aplicação analisado.

No exemplo apresentado na Seção 4.4 os dados injetados para a criação dos casos de testes foram pautados por meio de uma avaliação empírica, na qual foram gerados casos de testes com um, dois, três, quatro ou cinco ocorrências para cada elemento presente no hipercaminho. A avaliação permitiu concluir que só ocorreram alterações significativas quando nos casos de teste com 1(uma) ocorrência ou casos de testes com 3 (três) ocorrências por elemento. As outras cardinalidades avaliadas não apresentaram alterações que justificassem qualquer ganho de cobertura.

Na Figura 4.7 apresenta-se como os casos de testes são gerados de acordo com a cardinalidade dos elementos. Cinco pontos foram marcados e enumerados na figura e são descritos para uma melhor compreensão do processo.

A primeira marcação da figura diz respeito ao hipercaminho (percurso) traçado pelo algoritmo, parte da estratégia desta abordagem. Para cada elemento contido no percurso representado são injetados dados conforme as características descritas na gramática/metamodelo.

Na segunda marcação da figura é possível observar o uso da DSL para especificar

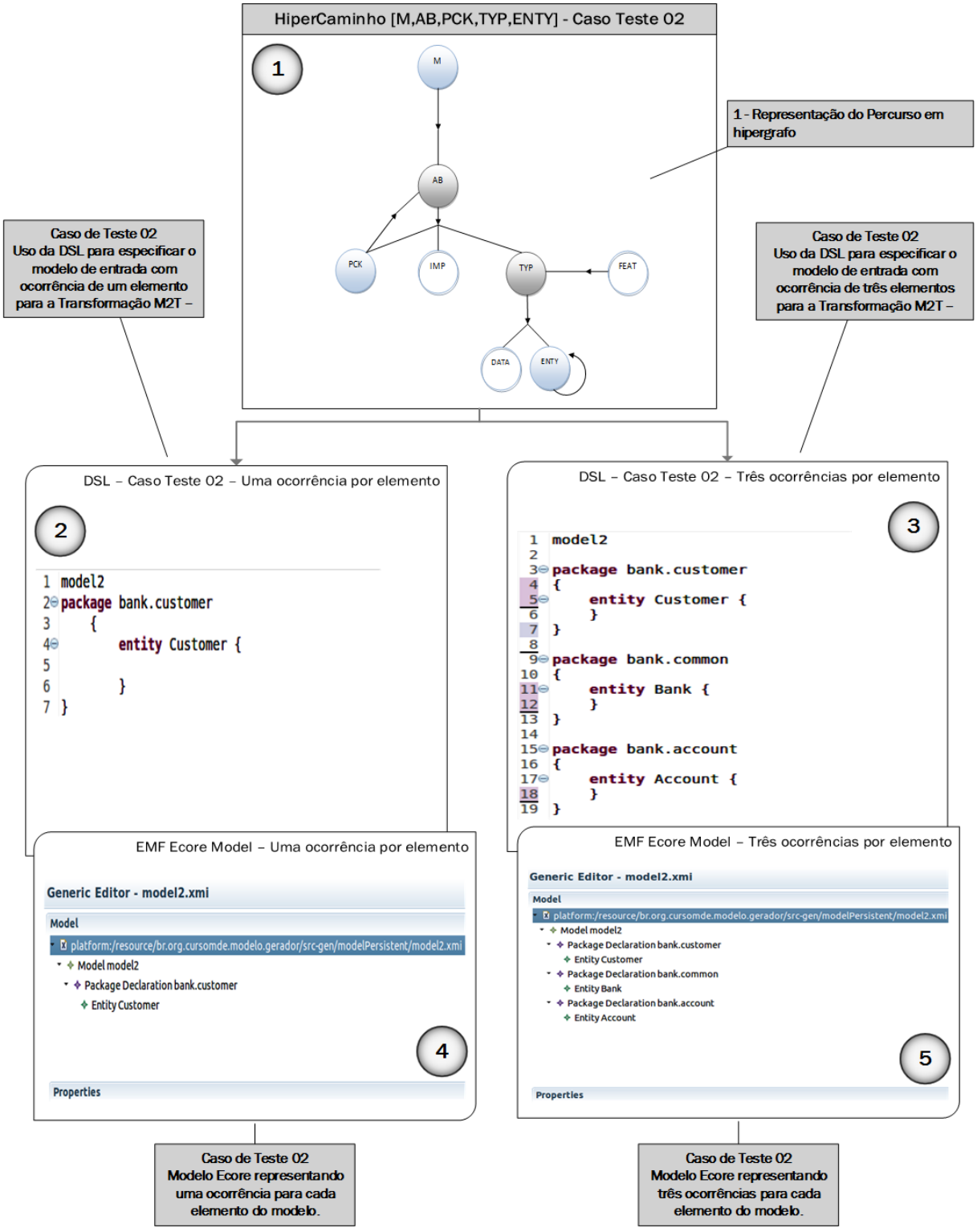


Figura 4.7: Injeção Dados de Testes - Hipercaminhos

o modelo de entrada com ocorrência de um elemento, compondo assim o caso de teste identificado como 2(dois) no processo de injeção de dados. A terceira marcação também faz uso da DSL para especificar o modelo de entrada, porém agora com três ocorrências para cada elemento do percurso.

A quarta e quinta marcação apontam para o modelo Ecore gerado e persistido com padrão XMI. Na quarta marcação o modelo Ecore representa o caso de teste 2 com uma ocorrência por elemento. Já na quinta marcação o modelo Ecore representa o caso de teste 2 com três ocorrências por elemento.

Cabe destacar que os casos de teses com 3 (três) ocorrências por elemento apresentaram melhores resultados quanto aos critérios de cobertura definidos, dadas as observações empíricas realizadas.

4.8 Análise e Cômputo dos Critérios de Cobertura

Tanto os apelos encontrados na literatura (WIMMER; BURGUEO, 2013; FLEUREY et al., 2009; BAUDRY et al., 2006) quanto os resultados conclusivos da revisão sistemática (ABADE; FERRARI; LUCRÉDIO, 2015) demonstram a necessidade de abordagens para testar transformações M2T com técnicas estruturais. Assim, a presente proposta concilia as características da orientação a objetos presentes no MDD e implementa sua estratégia utilizando o teste estrutural como técnica para avaliar as transformações realizadas.

Critérios para selecionar e avaliar conjuntos de casos de teste são fundamentais pra o sucesso da atividade de teste. Tais critérios devem fornecer indicação de quais casos de teste devem ser utilizados de forma a aumentar as chances de revelar erros ou, quando erros não forem revelados, estabelecer um nível elevado de confiança na correção do programa. Um caso de teste consiste de um par ordenado $(d;S(d))$, no qual $d \in D$ e $S(d)$ é a respectiva saída esperada (DELAMARO; MALDONADO; JINO, 2007). Um caso de teste CT_i deve especificar as entradas e a saída esperada, e um oráculo compara a saída obtida com a esperada definida no CT_i, e decide se o teste passou ou falhou.

A estratégia da abordagem proposta define como oráculo a implementação de referência. As saídas geradas (código fonte) dada a complementaridade, a inclusão e a pertinência do conjunto dos casos de teste devem ser estruturalmente iguais. Neste tipo de verificação, constroem-se casos de teste para a análise da implementação interna dos *templates*. A cobertura é expressa em termos da porcentagem de caminhos executados nos *templates* exercitados pelas transformações.

Cabe ressaltar que para a estratégia elaborada é pressuposto que o modelo de entrada avaliado tenha sido testado quanto a sua corretude e boa formação. Consequentemente, a abordagem não trata estes critérios de testes por compartilhar o entendimento de alguns autores (FLEUREY; STEEL; BAUDRY, 2004; HARMAN, 2008; TISO; REGGIO; LEOTTA, 2012; WIMMER; BURGUEO, 2013), os quais versam sobre inúmeras abordagens que testam transformações de modelos avaliando sua corretude e boa formação.

Com base na análise preliminar das transformações exercitadas e os relatos dos especialistas Lucrédio (2009) e Possatto (2013) quanto às aplicações desenvolvidas e avaliadas, os critérios definidos neste trabalho priorizaram cobrir e computar o quanto do código dos *templates* era efetivamente escrito pela transformação na geração de código-fonte, quantas marcações (como por exemplo a *Tag marker*, que representa a efetiva substituição de um elemento do modelo de entrada por uma parte especifica do código-fonte), foram efetivamente realizadas pela transformação na geração do código-fonte e também quantas instruções, isto é, comandos de condição ou repetição considerados relevantes no processo de mapeamento e rastreabilidade de código, são efetivamente exercitadas pelos *templates* durante a transformação. Assim, nesta fase da estratégia, 3 (três) critérios de cobertura que contemplavam estas necessidades são definidos, sendo eles:

1. Cobertura do número de caracteres exercitado no *template* pela transformação que gera código-fonte;

2. Cobertura do número de marcadores (*tags*) do *template* substituídos por elementos do modelo de entrada;

3. Cobertura do número de instruções exercitadas no *template* pela transformação.

Os critérios de cobertura são analisados e computados fazendo uso das personalizações do transformador reportadas na Seção 4.7 que geram os artefatos de mapeamentos e rastreamento das transformações realizadas. Nas Figuras 4.8 e 4.9 são apresentados os arquivos de mapeamento e rastreabilidade gerados a cada transformação exercitada pelos casos de teste.

Na Figura 4.8, na qual apresenta-se o arquivo de mapeamentos deve-se destacar que, ao invés de produzir anotações de mapeamento no código gerado cria-se fisicamente um novo arquivo em paralelo, contendo a relação das posições de cada mapeamento, obtida por meio da associação dos arquivos de *templates* com as referências dos modelos de entrada e com o código correspondente gerado. Os detalhes dos campos do arquivo de

Arquivo Mapping JET

mapping_mod.txt

```
templates/main.jet:4194,4195:null:117,118:null:null
templates/main.jet:4267,4272:null:118,123:null:if
templates/main.jet:4369,4370:null:123,124:null:if
templates/classes.java.jet:0,8:src/bank/customer/Customer.java:0,8:null:null
templates/classes.java.jet:8,38:src/bank/customer/Customer.java:8,21:get:null
templates/classes.java.jet:38,40:src/bank/customer/Customer.java:21,23:null:null
templates/classes.java.jet:237,238:src/bank/customer/Customer.java:23,24:null:null
templates/classes.java.jet:437,438:src/bank/customer/Customer.java:24,25:null:null
templates/classes.java.jet:479,480:src/bank/customer/Customer.java:25,26:null:if
templates/classes.java.jet:567,575:src/bank/customer/Customer.java:26,34:null:iterate
templates/classes.java.jet:575,617:src/bank/customer/Customer.java:34,45:get:iterate
templates/classes.java.jet:617,619:src/bank/customer/Customer.java:45,47:null:iterate
templates/classes.java.jet:567,575:src/bank/customer/Customer.java:47,55:null:iterate
templates/classes.java.jet:575,617:src/bank/customer/Customer.java:55,66:get:iterate
templates/classes.java.jet:617,619:src/bank/customer/Customer.java:66,68:null:iterate
templates/classes.java.jet:567,575:src/bank/customer/Customer.java:68,76:null:iterate
templates/classes.java.jet:575,617:src/bank/customer/Customer.java:76,89:get:iterate
templates/classes.java.jet:617,619:src/bank/customer/Customer.java:89,91:null:iterate
templates/classes.java.jet:631,632:src/bank/customer/Customer.java:91,92:null:if
templates/classes.java.jet:639,653:src/bank/customer/Customer.java:92,106:null:null
templates/classes.java.jet:653,688:src/bank/customer/Customer.java:106,114:get:null
templates/classes.java.jet:688,690:src/bank/customer/Customer.java:114,116:null:null
templates/classes.java.jet:800,814:src/bank/customer/Customer.java:116,130:null:null
templates/classes.java.jet:871,884:src/bank/customer/Customer.java:130,143:null:iterate
```

Figura 4.8: Artefatos Personalizados Transformador JET - Arquivo Mapping

Arquivo Tracer JET

```
JET Transformation [JET Transformation] - Aug 23, 2015 7:23:42 PM
    Trace: completed action.
templates/classes.java.jet(22,4):  <c:iterate select='$currModel/Feature' var='anAttribute'>
    Trace: processing loop body
templates/classes.java.jet(23,13):  <c:get select='$anAttribute/type/@name'>
    Trace: completed action.
templates/classes.java.jet(24,6):  <c:get select='$anAttribute/@name'>
    Trace: completed action.
templates/classes.java.jet(22,4):  <c:iterate select='$currModel/Feature' var='anAttribute'>
    Trace: finished loop
templates/main.jet(23,6):  <java:class name='{$currModel/@name}' template='templates/classes.java.jet'>
    Trace: completed action.
templates/main.jet(22,4):  <c:iterate select='/Model/PackageDeclaration[@name=''bank.common'']/Entity' var='currModel'
    Trace: finished loop
templates/main.jet(29,1):  <c:if test='isVariableDefined(''org.eclipse.jet.resource.project.name'')'>
    Trace: processing body
templates/dump.jet(2,1):  <c:dump select='/*' format='true' entities='true'>
    Trace: completed action.
templates/main.jet(30,5):  <ws:file template='templates/dump.jet' path='{$org.eclipse.jet.resource.project.name}/dump.
    Trace: completed action.
Successful Execution
```

Figura 4.9: Artefatos Personalizados Transformador JET - Arquivo Tracer

associação criados pelo *JET* podem ser visualizados na figura, onde cada linha representa

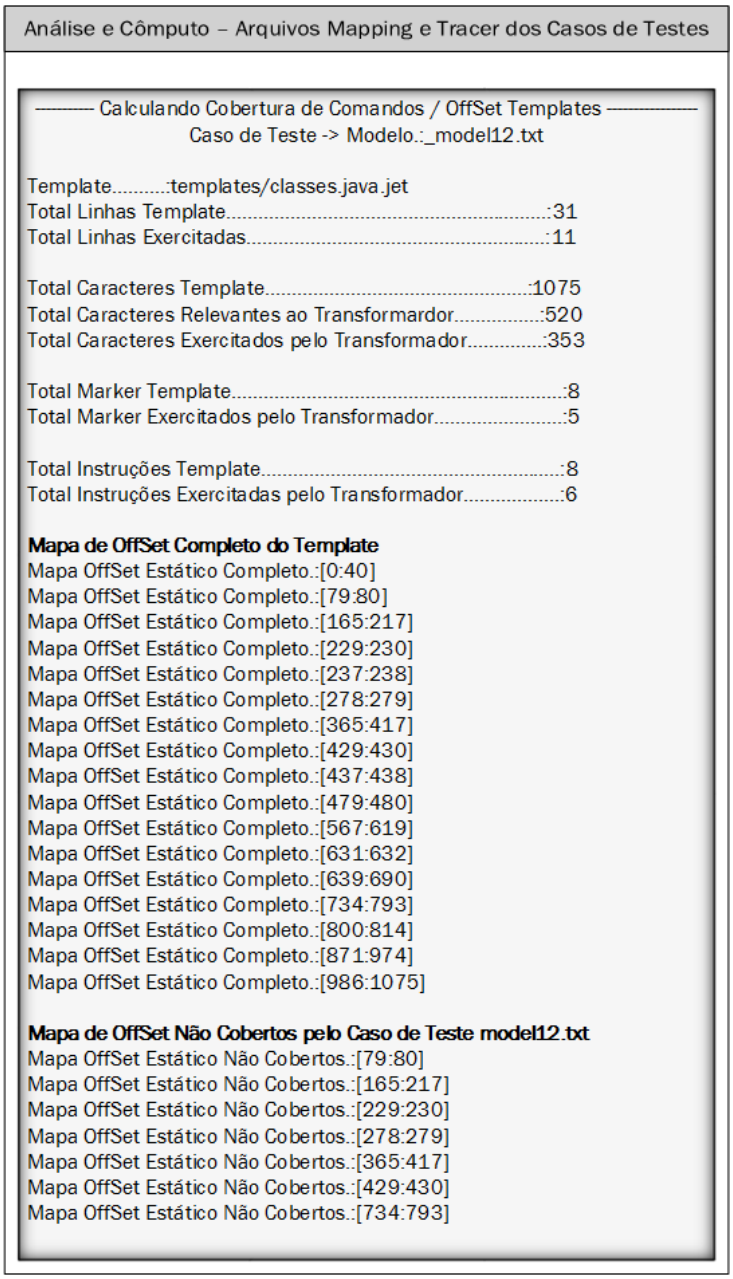


Figura 4.10: Relatório de Análises e Cômputos - Critérios e Cobertura dos Testes M2T

um mapeamento. Os campos são delimitados com caractere ":" e seguem o seguinte formato: caminho e nome do arquivo de *template*; posições "*off-set*" de início e fim de cada segmento de código do *template*; caminho e nome do arquivo gerado pela transformação por meio do *JET*; posições "*off-set*" de início e fim de cada segmento de código gerado mapeados a cada *template* e, por último, as informações da origem do código produzido e do tipo de mapeamento.

O log de rastreamento habilitado no transformador *JET* gera uma estrutura de anotações sobre cada linha exercitada pela transformação. Na Figura 4.9 é possível observar

a estrutura das anotações que têm o seguinte formato: caminho e nome do arquivo de *template*; linha e coluna do segmento de instruções avaliado; segmento de instruções avaliadas pelo transformador; e a linha seguinte, que representa o status da instrução e sua ação conforme o modelo de entrada exercitado.

Juntamente com os artefatos de mapeamento e rastreabilidade gerados, os *templates* são analisados conforme as características definidas pelos critérios de cobertura. Em seguida, são consequentemente computadas as coberturas realizadas para cada caso de teste exercitado pelas transformações.

Na Figura 4.10 é apresentado um relatório formatado que pode ser utilizado para subsidiar o processo de tomada de decisão dos projetistas e desenvolvedores quanto à testabilidade das transformações M2T e a consequente adequação dos casos de testes gerados, possibilitando os refinamentos necessários para aumentar a cobertura dos testes realizados.

4.9 Considerações Finais

Neste capítulo apresentou-se uma abordagem de caracterização de dados complexos baseada em hipergrafo para teste, definida no contexto das transformações M2T, para geração de casos de testes por meio de gramáticas/metamodelos. A abordagem, está ancorada nos princípios do teste estrutural e utiliza os percursos traçados sob os as representações por meio de hipergrafos como base para a geração de dos casos de testes.

A estratégia para abordagem proposta foi organizada em cinco fases que explicitam cada peculiaridade envolvida no processo, desde a caracterização dos dados ao cômputo da cobertura dos casos de testes.

No próximo capítulo são descritos dois estudos de caso seguindo a abordagem proposta. Os estudos fazem uso de gramáticas e metamodelos para transformações M2T, por meio do Método *Templates* e seus resultados são apresentados e discutidos em detalhes.

Capítulo 5

Estudos de Exploratórios

5.1 Considerações Iniciais

Neste capítulo são descritos dois estudos exploratórios conduzidos neste trabalho, sendo o primeiro deles no formato de um estudo piloto. Os dois estudos fazem uso de gramáticas e metamodelos para transformações M2T, por meio do Método *Templates*. O primeiro estudo, denominado estudo piloto, é utilizado de modo formativo, ajudando a investigar e a definir o instrumento de avaliação, as tecnologias específicas a serem empregadas na coleta final de dados e a articulação final das proposições teóricas do estudo. O segundo estudo, por sua vez, envolve uma aplicação dirigida por modelos para reutilização de software desenvolvida e fornecida por Lucrédio (2009).

Nas Seções 5.2 e 5.3 são apresentados os estudos descritos e estruturados com base no trabalho de Wohlin et al. (2000). Para a contextualização, faz-se uma breve descrição de cada um dos estudos e, em seguida, são apresentados o propósito do estudo, o contexto de aplicação, a forma pela qual os estudos foram conduzidos, a apresentação dos resultados obtidos e a análise dos resultados. Por fim, na Seção 5.4, apresentam-se as considerações finais deste capítulo.

5.2 Estudo Piloto: Gramática/Metamodelo Entidades Java Beans

Este estudo piloto consiste na caracterização dos dados de testes dos modelos utilizados em transformações M2T, por meio de hipergrafos direcionados. A avaliação faz uso da

abordagem apresentada no Capítulo 4 e tem com propósito a análise e o cômputo dos resultados, a fim de demonstrar e validar a aplicabilidade da abordagem em transformações M2T.

O estudo piloto concilia as características da orientação com objetos presentes no MDD e implementa sua estratégia fazendo uso do teste estrutural como técnica para avaliar as transformações realizadas. Utilizaram-se os critérios eleitos conforme justificativas apresentadas na Seção 4.8, sendo eles: Cobertura do número de caracteres exercitados no *template* pela transformação que gera código-fonte; Cobertura do número de marcadores do *template* substituídos por elementos do modelo de entrada e Cobertura do número de comandos efetivamente exercitados pelos *templates* na transformação. A seguir, tem-se uma descrição da gramática/metamodelo (extraída do documento da Eclipse.org (2013b)) que fora utilizado como exemplo ao longo desta abordagem.

5.2.1 Contexto e Definição do Estudo Piloto: Gramática/Metamodelo Entidades Java Beans

Java Beans são componentes de software escritos na linguagem de programação Java. Segundo a especificação da Oracle, os *Java Beans* são "componentes reutilizáveis de software que podem ser manipulados visualmente com a ajuda de uma ferramenta de desenvolvimento" (Oracle, 2010).

O estudo piloto faz uso de uma gramática formulada para o tutorial do *XText* denominada *"grammar Domain Model"*, pautada sob o metamodelo Ecore cujas regras de interpretação são tratadas como uma espécie de plano de construção para a criação dos *EObjects* que formam o modelo semântico. Assim, por meio de uma DSL, é possível especificar modelos utilizados pelos transformadores e gerar artefatos *Java Beans* conforme o domínio específico.

Cabe ressaltar que esta gramática é utilizada ao longo deste trabalho como referência e exemplo de aplicação prática e, consequentemente, eleita como estudo exploratório piloto. Na Seção 2.3.2 é possível observar o Código 2.1, que apresenta a descrição da gramática e seu metamodelo equivalente na Figura 2.11.

5.2.2 Planejamento

O estudo piloto está inserido no contexto de testes de transformações M2T, o qual envolve a testabilidade dos artefatos gerados por meio da caracterização dos dados com-

plexos de teste e a avaliação dos critérios de cobertura sob os *templates* utilizados nas transformações. O objeto de estudo é uma gramática *XText* (descrita na Seção 5.2.1, denominada *"grammar Domain Model"*. A escolha desse objeto de estudo deu-se por ser um exemplo simples e por sua relevância quanto ao reuso e geração de código-fonte.

O intuito do procedimento descrito a seguir é verificar a viabilidade de aplicação da abordagem proposta quanto a caracterização dos dados complexos de testes em transformações M2T. Adicionalmente, verifica-se o grau de adequação dos casos de teste gerados com base nos percursos de hipergrafos direcionados em relação aos critérios estruturais. Para isso, segue-se cada fase especificada pela abordagem proposta.

O estudo exploratório é composto dos seguintes materiais:

- Ubuntu 14.04 (64 bits);
- JRE 1.7.0_71 (64 bits)
- Eclipse Luna Service Release 1 (4.4.1) ($\boldsymbol{Buildid}: 20140925 - 1800$);
- Eclipse Web Tools 3.6.1.*;
- Eclipse Modeling Tools 4.4.1;
- JET 1.1.1.* (modificado);
- Xpand SDK 2.0.0.v201406030414;
- Xtext Complete SDK 2.7.2.v201409160908;
- Graphical Modeling Framework (GMF) Tooling SDK 3.2.1.201409171321;
- Algoritmo SB-Visita (adaptado) para traçar percursos em hipergrafos; direcionados tratando ciclos e validando restrições semânticas do modelo;
- Casos de teste baseados nos percursos gerados e validados pelo Algoritmo;
- Algoritmo de apoio para análise e cômputo da cobertura dos *templates* e dos critérios estruturais, alvo deste trabalho.

5.2.3 Operação

Nesta Seção é apresentada a condução do estudo piloto. Com base na gramática/ metamodelo, os dados complexos (elementos do modelo) são representados por meio de um

hipergrafo. Posteriormente, esta representação é implementada pelo Algoritmo 1 *SB-Visita*, adaptado de (GUEDES; MARKENZON; FARIA, 2011), gerando 28 percursos que são base para construção dos casos de teste. Para a criação dos casos de teste são utilizadas duas estratégias de injeção de dados, segundo a cardinalidade dos elementos presentes em cada percurso do algoritmo. A primeira estratégia utilizou a injeção de uma ocorrência por elemento presente no percurso designado para o caso de uso. Posteriormente, na segunda estratégia, utilizou-se a injeção de dados com três ocorrências por elemento presente no percurso designado.

Cabe destacar que a utilização dessas estratégias de injeção de dados foi pautada em avaliações preliminares, com observações empíricas de injeções de uma a cinco ocorrências por elemento. A avaliação permitiu concluir que só ocorreram alterações significativas quando nos casos de teste com uma ocorrência ou casos de teste com três ocorrências por elemento. As outras cardinalidades avaliadas não apresentaram alterações que justificassem ganho de cobertura.

Após a elaboração do conjunto de casos de teste, cada um deles é aplicado como modelo de entrada para o transformador *JET*. Cada transformação realizada gera os arquivos de mapeamento e rastreabilidade. Posteriormente, utiliza-se um algoritmo que interpreta os arquivos de mapeamento e rastreabilidade juntamente com a estrutura dos *templates*. Na Figura 4.1 da Seção 4.3 é apresentada a estrutura da abordagem proposta. Assim, o estudo piloto segue cada fase da estratégia apresentada pela abordagem para obtenção e representação dos dados coletados. Para garantir a validade dos dados, a execução dos testes é monitorada, certificando que todos os casos de teste que compõem os conjuntos são executados.

5.2.4 Coleta de Dados

Os dados coletados nesse estudo piloto são:

- Os percursos gerados por meio do Algoritmo 1 *SB-Visita* e as devidas adaptações para detecção de ciclo e validação das restrições semânticas. É preciso destacar que a restrições semânticas utilizadas para este estudo piloto são apresentadas na Figura 4.1 da Seção 4.5.

  Na Tabela 5.1 são apresentados os percursos gerados pelo Algoritmo 1 *SB-Visita*. Cada percurso (hipercaminho) é identificado por um número de ordem apresentado na tabela. As colunas identificadas como A, B e C na tabela são correspondentes às restrições implementadas e validadas pelo algoritmo proposto. Os percursos são

selecionados conforme a validação das restrições, sendo premissa para seleção que todas as restrições sejam validadas. Na coluna identificada como "Status Seleção" é possível observar os percursos selecionados.

Tabela 5.1: Conjunto de Hipercaminhos.

Restrições				Hipercaminhos	Status Seleção
Nº ID	A	B	C		
1	true	true	false	[M]	Não Selecionado
2	true	false	true	[M, AB]	Não Selecionado
3	true	true	true	[M, AB, PCK]	Selecionado
4	true	true	true	[M, AB, PCK, IMP]	Selecionado
5	true	false	true	[M, AB, PCK, IMP, TYP]	Não Selecionado
6	true	true	true	[M, AB, PCK, IMP, TYP, DATA]	Selecionado
7	true	true	true	[M, AB, PCK, IMP, TYP, DATA, ENTY]	Selecionado
8	true	true	true	[M, AB, PCK, IMP, TYP, DATA, ENTY, FEAT]	Selecionado
9	true	true	true	[M, AB, PCK, IMP, TYP, DATA, ENTY, ENTY, FEAT]	Selecionado
10	true	true	true	[M, AB, PCK, IMP, TYP, ENTY]	Selecionado
11	true	true	true	[M, AB, PCK, IMP, TYP, ENTY, FEAT, DATA]	Selecionado
12	true	false	true	[M, AB, PCK, TYP]	Não Selecionado
13	true	true	true	[M, AB, PCK, TYP, DATA]	Selecionado
14	true	true	true	[M, AB, PCK, TYP, DATA, ENTY]	Selecionado
15	true	true	true	[M, AB, PCK, TYP, DATA, ENTY, FEAT]	Selecionado
16	true	true	true	[M, AB, PCK, TYP, ENTY]	Selecionado
17	true	true	true	[M, AB, PCK, TYP, ENTY, FEAT]	Selecionado
18	true	true	true	[M, AB, PCK, TYP, ENTY, FEAT, DATA]	Selecionado
19	true	true	true	[M, AB, IMP]	Selecionado
20	true	false	true	[M, AB, IMP, TYP]	Não Selecionado
21	true	true	true	[M, AB, IMP, TYP, DATA]	Selecionado
22	true	true	true	[M, AB, IMP, TYP, DATA, ENTY]	Selecionado
23	true	true	true	[M, AB, IMP, TYP, DATA, ENTY, FEAT]	Selecionado
24	true	true	true	[M, AB, IMP, TYP, ENTY]	Selecionado
25	true	true	true	[M, AB, IMP, TYP, ENTY, FEAT]	Selecionado
26	true	true	true	[M, AB, IMP, TYP, ENTY, FEAT, DATA]	Selecionado
27	true	false	true	[M, AB, TYP]	Não Selecionado
28	true	false	true	[M, AB, TYP, DATA]	Não Selecionado
29	true	false	true	[M, AB, TYP, DATA, ENTY]	Não Selecionado
30	true	false	true	[M, AB, TYP, DATA, ENTY, FEAT]	Não Selecionado
31	true	false	true	[M, AB, TYP, ENTY]	Não Selecionado
32	true	false	true	[M, AB, TYP, ENTY, FEAT]	Não Selecionado
33	true	false	true	[M, AB, TYP, ENTY, FEAT, DATA]	Não Selecionado
34	true	true	true	[M, AB, IMP, PCK]	Selecionado
35	true	false	true	[M, AB, IMP, PCK, TYP]	Não Selecionado
36	true	true	true	[M, AB, IMP, PCK, TYP, ENTY]	Selecionado
37	true	true	true	[M, AB, IMP, PCK, TYP, ENTY, FEAT]	Selecionado
38	true	true	true	[M, AB, IMP, PCK, TYP, ENTY, FEAT, DATA]	Selecionado
39	true	true	true	[M, AB, IMP, PCK, TYP, DATA]	Selecionado
40	true	true	true	[M, AB, IMP, PCK, TYP, DATA, ENTY]	Selecionado
41	true	true	true	[M, AB, IMP, PCK, TYP, DATA, ENTY, FEAT]	Selecionado

O Algoritmo 1 *SB-Visita* processou 41 percursos para o hipergrafo que caracterizava a gramática/metamodelo estudada, sendo que 13 percursos obtiveram status como "Não Selecionados" e 28 percursos foram processados como "Selecionados".

- Casos de teste criados com base nos percursos selecionados. Ressalta-se que, para este estudo piloto, como são utilizadas duas estratégias de injeção de dados para

os casos testes, um rótulo foi atribuído a cada modelo de entrada para facilitar sua identificação e associação quanto ao percurso e cardinalidade utilizados. Este rótulo é atribuído ao nome do arquivo *XMI* correspondente ao modelo de entrada gerado.

- A cobertura dos *templates* pelos critérios estruturais definidos pela estratégia da abordagem já explicitados na Seção 4.8.

5.2.5 Análise

Nesta seção apresenta-se uma análise quantitativa, de acordo com cada um dos tópicos de dados coletados para o estudo piloto.

I) ***Os percursos e casos de teste gerados***

Como pode ser observado na Tabela 5.1 a representação da gramática/metamodelo, por meio do hipergrafo direcionado, gerou 41 (100%) percursos seguindo o princípio SB-Conectividade e tratando as redundâncias de elementos e ciclos. Após aplicar as restrições, obteve-se 28 (68%) percursos selecionados e 13 (32%) percursos não selecionados.

Caracterizar dados complexos para geração de casos de teste em transformações M2T não é uma tarefa trivial (BAUDRY et al., 2010). Não é possível fazer uso das técnicas convencionais e tampouco lidar com a semântica envolvida nas gramáticas e metamodelos. Assim, caso fosse adotada uma estratégia simplista para geração de subconjuntos (como, por exemplo, a Permutação Circular, que ocorre em situações quando há grupos com m elementos distintos formando uma circunferência), o número de combinações seria dado pela fórmula: $P_c(m) = (m-1)!$, onde $(8-1)! = 5.040$ combinações possíveis, assumindo que neste estudo piloto são caracterizados oito elementos distintos que compõem a gramática e consequentemente o hipergrafo.

Ainda que restrições combinatórias possam ser inseridas para otimizar o número de subconjuntos possíveis, não se encontra na literatura uma solução que alinhe a capacidade de ilustração com a navegabilidade algorítmica possibilitada por meio dos hipergrafos. Outro fator relevante quanto à abordagem proposta são os conceitos de conectividade amplamente discutidos pelos autores Gallo e Scutellà (1998, 2002) e Guedes, Markenzon e Faria (2011), que conduzem a solução algorítmica proposta quando delimita em sua entrada que uma tripla $H = (V, E, s)$, na qual (V, E) é um hipergrafo direcionado, $s \in V$ é o vértice origem, e existe um SB-caminho de s para qualquer outro vértice em V. Assim, percursos obtidos e selecionados podem garantir

o relacionamento entre os elementos, incorporando ainda mais sentido semântico à base de criação dos casos de teste.

O uso desta estratégia pela abordagem proposta é uma solução relevante, pois não incorre em um processo minimalista que muitas vezes quebra os estados hierárquicos dos modelos para os casos de teste gerados como já relatado pela literatura (BAUDRY et al., 2010). Adicionalmente, a utilização do percurso como base para criação do caso de teste ameniza as dificuldades de compreensão dos testadores, uma vez que com a geração de casos de teste otimizados, as partes do metamodelo considerado eficaz correspondem intuitivamente ao significado semântico validado por meio das restrições e do hipercaminho traçado.

Na Seção 4.7, que descreve a fase 4 da abordagem proposta é explicitado o processo de injeção de dados para cada percurso gerado que consequentemente dará origem a um caso de teste. Para este estudo piloto foram gerados dois conjuntos de casos de teste para cada percurso, sendo um com uma ocorrência de cada elemento, e o outro com três ocorrências por elemento presente no percurso.

II) ***A cobertura dos* templates *pelos critérios estruturais definidos***

O processo de avaliação e cômputo da cobertura dos *templates*, por meio das transformações, utilizou 28 casos de teste.

Para cada transformação exercitada são gerados dois arquivos que possibilitam a extração das informações que subsidiam todo processo de análise. Ao todo foram processados 56 arquivos, sendo que 28 destes são referentes aos mapeamentos entre instruções dos *templates* e seus correspondentes trechos de código-fonte gerados. Os demais arquivos contêm as informações extraídas do *tracer* do transformador *JET*. Na Seção 4.8 é apresentada a estrutura e detalhes destes arquivos por meio das Figuras 4.8 e 4.9.

Neste estudo piloto são analisados dois arquivos de *templates*, sendo eles: *Main.jet* e *Classes.jet*. A *template Main* possui uma estrutura de segmentos predominantemente dinâmica, com poucos segmentos ditos estáticos. Já a *template* Classes possui uma estrutura com segmentos de maioria estáticos.

As transformações realizadas dão origem a um relatório analítico que permite a sumarização dos dados e viabilizam o processo de análise e interpretações dos resultados. Na Figura 5.1 é apresentado um exemplo do relatório gerado para um caso de teste avaliado.

****

Caso de Teste ->Modelo_41_28_01

Template.....:templates/classes.java.jet

Total Caracteres Template.....:1530
Total Caracteres Relevantes ao Transformardor.....:975
Total Caracteres Exercitados pelo Transformador.....:808

Total Marker Template.....:11
Total Marker Exercitados pelo Transformador.....:8

Total Instruções Template.....:8
Total Instruções Exercitadas pelo Transformador.....:6

Mapa OffSet Completo		Mapa OffSet Estático Completo	Mapa OffSet Estático Não Coberto
[0:8]	[688:690]	[0:40]	[79:80]
[8:38]	[690:734]	[79:80]	[165:217]
[38:40]	[734:793]	[165:217]	[229:230]
[40:79]	[793:800]	[229:230]	[278:279]
[79:80]	[800:814]	[237:238]	[365:417]
[80:165]	[814:871]	[278:279]	[429:430]
[165:217]	[871:884]	[365:417]	[734:793]
[217:229]	[884:926]	[429:430]	
[229:230]	[926:927]	[437:438]	
[230:237]	[927:964]	[479:480]	
[237:238]	[964:982]	[567:619]	
[238:278]	[982:1024]	[631:632]	
[278:279]	[1024:1029]	[639:690]	
[279:365]	[1029:1082]	[734:793]	
[365:417]	[1082:1104]	[800:814]	
[417:429]	[1104:1141]	[871:1436]	
[429:430]	[1141:1185]	[1448:1530]	
[430:437]	[1185:1238]		
[437:438]	[1238:1240]		
[438:479]	[1240:1282]		
[479:480]	[1282:1284]		
[480:567]	[1284:1321]		
[567:575]	[1321:1343]		
[575:617]	[1343:1380]		
[617:619]	[1380:1383]		
[619:631]	[1383:1420]		
[631:632]	[1420:1436]		
[632:639]	[1436:1448]		
[639:653]	[1448:1530]		
[653:688]			

Template.....:templates/main.jet

Total Caracteres Template.....:1437
Total Caracteres Relevantes ao Transformardor.....:216
Total Caracteres Exercitados pelo Transformador.....:74

Total Marker Template.....:0
Total Marker Exercitados pelo Transformador.....:0

Total Instruções Template.....:4
Total Instruções Exercitadas pelo Transformador.....:4

Mapa OffSet Completo		Mapa OffSet Estático Completo	Mapa OffSet Estático Não Coberto
[0:58]	[851:855]	[0:126]	[0:58]
[58:60]	[855:867]	[178:180]	[60:124]
[124:126]	[867:870]	[203:206]	[544:550]
[126:178]	[870:885]	[241:244]	[584:587]
[178:180]	[885:891]	[330:334]	[672:676]
[180:203]	[891:924]	[426:432]	[767:773]
[203:206]	[924:927]	[510:514]	[851:855]
[206:241]	[927:1011]	[526:529]	[867:870]
[241:244]	[1011:1015]	[544:550]	[885:891]
[244:330]	[1015:1105]	[584:587]	[924:927]
[330:334]	[1105:1111]	[672:676]	[1011:1015]
[334:426]	[1111:1189]	[767:773]	[1105:1111]
[426:432]	[1189:1193]	[851:855]	[1189:1193]
[432:510]	[1193:1205]	[867:870]	[1205:1208]
[510:514]	[1205:1208]	[885:891]	[1223:1228]
[514:526]	[1208:1223]	[924:927]	[1240:1241]
[526:529]	[1223:1228]	[1011:1015]	[1254:1255]
[529:544]	[1228:1240]	[1105:1111]	[1327:1332]
[544:550]	[1240:1241]	[1189:1193]	[1429:1430]
[550:584]	[1241:1254]	[1205:1208]	
[584:587]	[1254:1255]	[1223:1228]	
[587:672]	[1255:1327]	[1240:1241]	
[672:676]	[1327:1332]	[1254:1255]	
[676:767]	[1332:1429]	[1327:1332]	
[767:773]	[1430:1437]	[1429:1430]	
[773:851]			

Figura 5.1: Relatório de Análise e Cômputo dos Critérios de Cobertura - Caso de Teste 28 - 01 ocorrência por elemento do percurso

A seguir será apresentada a avaliação de complementaridade, pertinência e inclusão do conjunto dos casos de testes quanto aos critérios de cobertura analisados. Destaca-se que a estratégia da abordagem proposta para geração dos casos de teste é pautada em um algoritmo de percursos em hipergrafos por meio da análise combinatória. Assim, a avaliação desses conjuntos de casos de teste por meio das relações

de complementaridade, pertinência e inclusão são essenciais para a abordagem de teste proposta. Tais critérios podem fornecer indicação de quais casos de teste devem ser utilizados de forma a aumentar as chances de revelar defeitos ou, quando não forem revelados, estabelecer um nível elevado de confiança na correção do programa (MALDONADO et al., 1998).

É importante ressaltar que, dada a natureza combinatória e a integridade dos estados hierárquicos dos modelos para o conjunto dos casos de teste gerados, existe uma tendência primordial de relação inclusiva entre o conjunto de casos de teste. Entretanto, a complementaridade é evidenciada quando observado a cardinalidade de cada elemento, ou seja, o número de ocorrências para cada elemento presente na estrutura do caso de teste.

Como neste estudo foram utilizados duas estratégias de injeção de dados para criação dos casos de teste, estes resultados serão apresentados em dois grupos distintos, conforme suas características.

III) ***Avaliação de Cobertura dos Casos de Teste - Uma ocorrência por elemento do percurso***

Com base nesses resultados, ainda que preliminarmente, foi elaborada uma análise que avalia a cobertura de todo conjunto de casos de teste gerados com propósito de constatar a eficácia de cobertura. Cada caso de teste foi analisado levando em consideração as seguintes informações: O total de caracteres relevantes dos *templates*, o mapa completo de *off-set* relevantes avaliados, o mapa de *off-set* cobertos pelos casos de teste e o mapa de *off-set* não cobertos.

A título de ilustração, considerando os critérios estruturais adotados pela abordagem proposta, são identificadas nas Tabelas 5.2 e 5.3 o processo de cômputo da avaliação de cobertura dos casos de teste para os *templates* main.jet e classes.jet. Foram selecionados como amostra cinco casos de teste com maior relação de complementaridade, pertinência e inclusão. O mapa de *off-set* cobertos pelos casos de teste estão destacados na cor amarela, enquanto as linhas em cor cinza representam os segmentos dinâmicos não avaliados nesta abordagem. Os segmentos que não foram cobertos pelos casos de teste estão com a cor de fundo branca.

Ao final do processo, os segmentos do conjunto de 28 casos de teste são sumarizados juntamente às informações obtidas por meio dos relatórios de análise e cômputo apresentados na Figura 5.1 e consequentemente calculada a cobertura dos *templates* conforme o propósito de cada critério aplicado.

Tabela 5.2: Avaliação do Conjunto de Casos de Teste - Main.jet

Arquivo Template Main.jet		Modelo 03	Modelo 07	Modelo 08	Modelo 09	Modelo 38	Modelo 41
Mapa OffSet Completo	Mapa OffSet Estático	Caso Teste 01	Caso Teste 04	Caso Teste 05	Caso Teste 06	Caso Teste 25	Caso Teste 28
[0:58]	[0:58]	[0:58]	[0:58]	[0:58]	[0:58]	[0:58]	[0:58]
[58:60]	[58:60]	[58:60]	[58:60]	[58:60]	[58:60]	[58:60]	[58:60]
[60:124]	[60:124]	[60:124]	[60:124]	[60:124]	[60:124]	[60:124]	[60:124]
[124:126]	[124:126]	[124:126]	[124:126]	[124:126]	[124:126]	[124:126]	[124:126]
[126:178]							
[178:180]	[178:180]	[178:180]	[178:180]	[178:180]	[178:180]	[178:180]	[178:180]
[180:203]							
[203:206]	[203:206]	[203:206]	[203:206]	[203:206]	[203:206]	[203:206]	[203:206]
[206:241]							
[241:244]	[241:244]	[241:244]	[241:244]	[241:244]	[241:244]	[241:244]	[241:244]
[244:330]							
[330:334]	[330:334]	[330:334]	[330:334]	[330:334]	[330:334]	[330:334]	[330:334]
[334:426]							
[426:432]	[426:432]	[426:432]	[426:432]	[426:432]	[426:432]	[426:432]	[426:432]
[432:510]							
[510:514]	[510:514]	[510:514]	[510:514]	[510:514]	[510:514]	[510:514]	[510:514]
[514:526]							
[526:529]	[526:529]	[526:529]	[526:529]	[526:529]	[526:529]	[526:529]	[526:529]
[529:544]							
[544:550]	[544:550]	[544:550]	[544:550]	[544:550]	[544:550]	[544:550]	[544:550]
[550:584]							
[584:587]	[584:587]	[584:587]	[584:587]	[584:587]	[584:587]	[584:587]	[584:587]
[587:672]							
[672:676]	[672:676]	[672:676]	[672:676]	[672:676]	[672:676]	[672:676]	[672:676]
[676:767]							
[767:773]	[767:773]	[767:773]	[767:773]	[767:773]	[767:773]	[767:773]	[767:773]
[773:851]							
[851:855]	[851:855]	[851:855]	[851:855]	[851:855]	[851:855]	[851:855]	[851:855]
[855:867]							
[867:870]	[867:870]	[867:870]	[867:870]	[867:870]	[867:870]	[867:870]	[867:870]
[870:885]							
[885:891]	[885:891]	[885:891]	[885:891]	[885:891]	[885:891]	[885:891]	[885:891]
[891:924]							
[924:927]	[924:927]	[924:927]	[924:927]	[924:927]	[924:927]	[924:927]	[924:927]
[927:1011]							
[1011:1015]	[1011:1015]	[1011:1015]	[1011:1015]	[1011:1015]	[1011:1015]	[1011:1015]	[1011:1015]
[1015:1105]							
[1105:1111]	[1105:1111]	[1105:1111]	[1105:1111]	[1105:1111]	[1105:1111]	[1105:1111]	[1105:1111]
[1111:1189]							
[1189:1193]	[1189:1193]	[1189:1193]	[1189:1193]	[1189:1193]	[1189:1193]	[1189:1193]	[1189:1193]
[1193:1205]							
[1205:1208]	[1205:1208]	[1205:1208]	[1205:1208]	[1205:1208]	[1205:1208]	[1205:1208]	[1205:1208]
[1208:1223]							
[1223:1228]	[1223:1228]	[1223:1228]	[1223:1228]	[1223:1228]	[1223:1228]	[1223:1228]	[1223:1228]
[1228:1240]							
[1240:1241]	[1240:1241]	[1240:1241]	[1240:1241]	[1240:1241]	[1240:1241]	[1240:1241]	[1240:1241]
[1241:1254]							
[1254:1255]	[1254:1255]	[1254:1255]	[1254:1255]	[1254:1255]	[1254:1255]	[1254:1255]	[1254:1255]
[1255:1327]							
[1327:1332]	[1327:1332]	[1327:1332]	[1327:1332]	[1327:1332]	[1327:1332]	[1327:1332]	[1327:1332]
[1332:1429]							
[1429:1430]	[1429:1430]	[1429:1430]	[1429:1430]	[1429:1430]	[1429:1430]	[1429:1430]	[1429:1430]
[1430:1437]							
1437 Caract.	216 Caract.	64 Caract.	74 Caract.	74 Caract.	74 Caract.	74 Caract.	74 Caract.

- ***Cobertura do número de caracteres exercitado no* template *pela transformação que gera código-fonte***

  Ao analisar as informações devidamente computadas e sumarizadas é possível

Tabela 5.3: Avaliação do Conjunto de Casos de Teste - Classes.jet

Arquivo Template Classes.jet		Modelo 03	Modelo 07	Modelo 08	Modelo 09	Modelo 38	Modelo 41
Mapa OffSet Completo	Mapa OffSet Estático	Caso Teste 01	Caso Teste 04	Caso Teste 05	Caso Teste 06	Caso Teste 25	Caso Teste 28
[0:8]	[0:8]	[0:8]	[0:8]	[0:8]	[0:8]	[0:8]	[0:8]
[8:38]	[8:38]	[8:38]	[8:38]	[8:38]	[8:38]	[8:38]	[8:38]
[38:40]	[38:40]	[38:40]	[38:40]	[38:40]	[38:40]	[38:40]	[38:40]
[40:79]							
[79:80]	[79:80]	[79:80]	[79:80]	[79:80]	[79:80]	[79:80]	[79:80]
[80:165]							
[165:217]	[165:217]	[165:217]	[165:217]	[165:217]	[165:217]	[165:217]	[165:217]
[217:229]							
[229:230]	[229:230]	[229:230]	[229:230]	[229:230]	[229:230]	[229:230]	[229:230]
[230:237]							
[237:238]	[237:238]	[237:238]	[237:238]	[237:238]	[237:238]	[237:238]	[237:238]
[238:278]							
[278:279]	[278:279]	[278:279]	[278:279]	[278:279]	[278:279]	[278:279]	[278:279]
[279:365]							
[365:417]	[365:417]	[365:417]	[365:417]	[365:417]	[365:417]	[365:417]	[365:417]
[417:429]							
[429:430]	[429:430]	[429:430]	[429:430]	[429:430]	[429:430]	[429:430]	[429:430]
[430:437]							
[437:438]	[437:438]	[437:438]	[437:438]	[437:438]	[437:438]	[437:438]	[437:438]
[438:479]							
[479:480]	[479:480]	[479:480]	[479:480]	[479:480]	[479:480]	[479:480]	[479:480]
[480:567]							
[567:575]	[567:619]	[567:619]	[567:619]	[567:619]	[567:619]	[567:619]	[567:619]
[575:617]							
[617:619]							
[619:631]							
[631:632]	[631:632]	[631:632]	[631:632]	[631:632]	[631:632]	[631:632]	[631:632]
[632:639]							
[639:653]	[639:690]	[639:690]	[639:690]	[639:690]	[639:690]	[639:690]	[639:690]
[653:688]							
[688:690]							
[690:734]							
[734:793]	[734:793]	[734:793]	[734:793]	[734:793]	[734:793]	[734:793]	[734:793]
[793:800]							
[800:814]	[800:814]	[800:814]	[800:814]	[800:814]	[800:814]	[800:814]	[800:814]
[814:871]							
[871:884]	[871:884]	[871:884]	[871:884]	[871:884]	[871:884]	[871:884]	[871:884]
[884:926]	[884:926]	[884:926]	[884:926]	[884:926]	[884:926]	[884:926]	[884:926]
[926:927]	[926:927]	[926:927]	[926:927]	[926:927]	[926:927]	[926:927]	[926:927]
[927:964]	[927:964]	[927:964]	[927:964]	[927:964]	[927:964]	[927:964]	[927:964]
[964:982]	[964:982]	[964:982]	[964:982]	[964:982]	[964:982]	[964:982]	[964:982]
[982:1024]	[982:1024]	[982:1024]	[982:1024]	[982:1024]	[982:1024]	[982:1024]	[982:1024]
[1024:1029]	[1024:1029]	[1024:1029]	[1024:1029]	[1024:1029]	[1024:1029]	[1024:1029]	[1024:1029]
[1029:1082]	[1029:1082]	[1029:1082]	[1029:1082]	[1029:1082]	[1029:1082]	[1029:1082]	[1029:1082]
[1082:1104]	[1082:1104]	[1082:1104]	[1082:1104]	[1082:1104]	[1082:1104]	[1082:1104]	[1082:1104]
[1104:1141]	[1104:1141]	[1104:1141]	[1104:1141]	[1104:1141]	[1104:1141]	[1104:1141]	[1104:1141]
[1141:1185]	[1141:1185]	[1141:1185]	[1141:1185]	[1141:1185]	[1141:1185]	[1141:1185]	[1141:1185]
[1185:1238]	[1185:1238]	[1185:1238]	[1185:1238]	[1185:1238]	[1185:1238]	[1185:1238]	[1185:1238]
[1238:1240]	[1238:1240]	[1238:1240]	[1238:1240]	[1238:1240]	[1238:1240]	[1238:1240]	[1238:1240]
[1240:1282]	[1240:1282]	[1240:1282]	[1240:1282]	[1240:1282]	[1240:1282]	[1240:1282]	[1240:1282]
[1282:1284]	[1282:1284]	[1282:1284]	[1282:1284]	[1282:1284]	[1282:1284]	[1282:1284]	[1282:1284]
[1284:1321]	[1284:1321]	[1284:1321]	[1284:1321]	[1284:1321]	[1284:1321]	[1284:1321]	[1284:1321]
[1321:1343]	[1321:1343]	[1321:1343]	[1321:1343]	[1321:1343]	[1321:1343]	[1321:1343]	[1321:1343]
[1343:1380]	[1343:1380]	[1343:1380]	[1343:1380]	[1343:1380]	[1343:1380]	[1343:1380]	[1343:1380]
[1380:1383]	[1380:1383]	[1380:1383]	[1380:1383]	[1380:1383]	[1380:1383]	[1380:1383]	[1380:1383]
[1383:1420]	[1383:1420]	[1383:1420]	[1383:1420]	[1383:1420]	[1383:1420]	[1383:1420]	[1383:1420]
[1420:1436]	[1420:1436]	[1420:1436]	[1420:1436]	[1420:1436]	[1420:1436]	[1420:1436]	[1420:1436]
[1436:1448]							
[1448:1530]	[1448:1530]	[1448:1530]	[1448:1530]	[1448:1530]	[1448:1530]	[1448:1530]	[1448:1530]
1530 Caract.	975 Caract.	N/A	N/A	808 Caract.	867 Caract.	808 Caract.	808 Caract.

observar que os casos de testes não exercem 100% de cobertura quando avaliados com relação ao *template* main.jet. Dos 28 casos de testes avaliados, 16 casos de teste atingiram 34,26% de cobertura dos caracteres, os demais não ultrapassaram 30% de cobertura. Efetivamente, os casos de teste avaliados demonstraram baixa complementaridade, fato este que pode ser justificado pela característica dos casos de teste com injeção de uma ocorrência por elemento e/ou pela existência de caminhos não executáveis.

Em contrapartida, o *template* classes.jet com predominância de segmentos estáticos oscilou entre três grupos de casos de teste que produzem resultados distintos. O primeiro grupo com resultados significativos é composto por nove casos de testes, sendo eles: Caso de Teste 5, 6, 8, 11, 13, 14, 24, 25 e 28. Este grupo alcança índices de cobertura superiores a 75% dos caracteres exercitados pela transformação.

O segundo grupo de casos de teste avaliados com relação ao *template* classes.jet é composto por seis casos de teste, sendo eles: Caso de Teste 4, 7, 10, 12, 23 e 27. A avaliação deste grupo alcançou índices de coberturas entre 19% e 25%. Por fim, os terceiro grupo inclui os demais casos de teste, que não estimulam a geração de código considerando-se o *template* classes.jet.

- ***Cobertura do número de marcadores do* template *substituídos por elementos do modelo de entrada***

  Este critério de cobertura é computado exclusivamente por meio das informações obtidas nos relatórios de análise e cômputo apresentados na Figura 5.1. As informações são sumarizadas de acordo com o conjunto de casos de teste para cada *template* avaliado.

  A avaliação do *template* main.jet, dada suas características estruturais, não reporta quaisquer ocorrências que implicassem em cobertura. Assim, a ausência dos marcadores *(tag's)* na estrutura do *template* impossibilita a avaliação do conjunto de casos de teste quanto a este critério de cobertura.

  O *template* classes.jet apresenta resultados com grupos heterogêneos que permitem distintas avaliações. O primeiro grupo é composto por nove casos de testes, sendo eles: Caso de Teste 5, 6, 8, 11, 13, 14, 24, 25 e 28. Este grupo alcançou índices de cobertura superiores a 63% dos marcadores exercitados pelas transformações.

  O segundo grupo é composto por cinco casos de testes, sendo eles: Caso de Teste 7, 10, 12, 23 e 27. Neste grupo os índices de cobertura são inferiores a

30%, dado a presença de elementos predominantemente não utilizáveis pelos marcadores. O último grupo não permitiu avaliação, ou seja, não existem elementos do modelo utilizados pelos marcadores nas transformações.

- ***Cobertura do número de instruções exercitadas pelos* templates *na transformação***

  Ao sumarizar este critério estrutural, ressalta-se que para este preceito em particular, a distinção dos *templates* em segmentos estáticos e dinâmicos não é relevante, tendo em vista que uma instrução pode estar contida em qualquer um dos segmentos. Para determinar a escala percentil utilizada no cômputo da avaliação do *template* main.jet pauta-se em um N de quatro e o *template* classes.jet em um N de oito instruções máximas avaliadas.

  Ainda que o total de instruções avaliadas para o template main.jet seja sucinto, nota-se que todos os casos de testes exercitados pelas transformações obtiveram 100% de cobertura quanto às instruções exercitadas.

  Desta vez, o *template* classes.jet produz resultados com dois grupos distintos. O primeiro grupo composto por quinze casos de testes, sendo eles: Caso de Teste 5, 6, 7, 8, 10, 11, 12, 13, 14, 22, 23, 24, 25, 27 e 28. Este conjunto produziu uma cobertura superior a 58%, sendo que na maioria dos casos de testes a cobertura chegou a 75%. Os 13 casos de testes restantes não produziram resultados no processo de análise e inviabilizaram qualquer cômputo.

IV) ***Avaliação de Cobertura dos Casos de Teste - Três ocorrências por elemento do percurso***

A seguir será apresentada a sumarização dos critérios de cobertura avaliados em casos de teste contendo três ocorrências por elemento do percurso. Cabe ressaltar que a metodologia de apresentação e avaliação dos resultados seguem os mesmos moldes da sumarização apresentada anteriormente. Assim, esta avaliação dará ênfase quanto às vantagens e desvantagens percebíveis no incremento do número de ocorrências por elementos presentes nos casos de teste. Novamente, o processo de cômputo e avaliação é ilustrado por meio das Tabelas 5.4 e 5.5.

- ***Cobertura do número de caracteres exercitado no* template *pela transformação que gera código-fonte***

  Ao gerar casos de teste com três ocorrências e avaliar a cobertura do número de caracteres é possível observar que o *template* main.jet obteve um acréscimo sensível e homogêneo de 27.1% de cobertura. Dos 28 casos de teste avaliados,

Tabela 5.4: Avaliação do Conjunto de Casos de Teste - Main.jet

Arquivo Template Main.jet		Modelo 03	Modelo 07	Modelo 08	Modelo 09	Modelo 38	Modelo 41
Mapa OffSet Completo	Mapa OffSet Estático	Caso Teste 01	Caso Teste 04	Caso Teste 05	Caso Teste 06	Caso Teste 25	Caso Teste 28
[0:58]	[0:58]	[0:58]	[0:58]	[0:58]	[0:58]	[0:58]	[0:58]
[58:60]	[58:60]	[58:60]	[58:60]	[58:60]	[58:60]	[58:60]	[58:60]
[60:124]	[60:124]	[60:124]	[60:124]	[60:124]	[60:124]	[60:124]	[60:124]
[124:126]	[124:126]	[124:126]	[124:126]	[124:126]	[124:126]	[124:126]	[124:126]
[126:178]							
[178:180]	[178:180]	[178:180]	[178:180]	[178:180]	[178:180]	[178:180]	[178:180]
[180:203]							
[203:206]	[203:206]	[203:206]	[203:206]	[203:206]	[203:206]	[203:206]	[203:206]
[206:241]							
[241:244]	[241:244]	[241:244]	[241:244]	[241:244]	[241:244]	[241:244]	[241:244]
[244:330]							
[330:334]	[330:334]	[330:334]	[330:334]	[330:334]	[330:334]	[330:334]	[330:334]
[334:426]							
[426:432]	[426:432]	[426:432]	[426:432]	[426:432]	[426:432]	[426:432]	[426:432]
[432:510]							
[510:514]	[510:514]	[510:514]	[510:514]	[510:514]	[510:514]	[510:514]	[510:514]
[514:526]							
[526:529]	[526:529]	[526:529]	[526:529]	[526:529]	[526:529]	[526:529]	[526:529]
[529:544]							
[544:550]	[544:550]	[544:550]	[544:550]	[544:550]	[544:550]	[544:550]	[544:550]
[550:584]							
[584:587]	[584:587]	[584:587]	[584:587]	[584:587]	[584:587]	[584:587]	[584:587]
[587:672]							
[672:676]	[672:676]	[672:676]	[672:676]	[672:676]	[672:676]	[672:676]	[672:676]
[676:767]							
[767:773]	[767:773]	[767:773]	[767:773]	[767:773]	[767:773]	[767:773]	[767:773]
[773:851]							
[851:855]	[851:855]	[851:855]	[851:855]	[851:855]	[851:855]	[851:855]	[851:855]
[855:867]							
[867:870]	[867:870]	[867:870]	[867:870]	[867:870]	[867:870]	[867:870]	[867:870]
[870:885]							
[885:891]	[885:891]	[885:891]	[885:891]	[885:891]	[885:891]	[885:891]	[885:891]
[891:924]							
[924:927]	[924:927]	[924:927]	[924:927]	[924:927]	[924:927]	[924:927]	[924:927]
[927:1011]							
[1011:1015]	[1011:1015]	[1011:1015]	[1011:1015]	[1011:1015]	[1011:1015]	[1011:1015]	[1011:1015]
[1015:1105]							
[1105:1111]	[1105:1111]	[1105:1111]	[1105:1111]	[1105:1111]	[1105:1111]	[1105:1111]	[1105:1111]
[1111:1189]							
[1189:1193]	[1189:1193]	[1189:1193]	[1189:1193]	[1189:1193]	[1189:1193]	[1189:1193]	[1189:1193]
[1193:1205]							
[1205:1208]	[1205:1208]	[1205:1208]	[1205:1208]	[1205:1208]	[1205:1208]	[1205:1208]	[1205:1208]
[1208:1223]							
[1223:1228]	[1223:1228]	[1223:1228]	[1223:1228]	[1223:1228]	[1223:1228]	[1223:1228]	[1223:1228]
[1228:1240]							
[1240:1241]	[1240:1241]	[1240:1241]	[1240:1241]	[1240:1241]	[1240:1241]	[1240:1241]	[1240:1241]
[1241:1254]							
[1254:1255]	[1254:1255]	[1254:1255]	[1254:1255]	[1254:1255]	[1254:1255]	[1254:1255]	[1254:1255]
[1255:1327]							
[1327:1332]	[1327:1332]	[1327:1332]	[1327:1332]	[1327:1332]	[1327:1332]	[1327:1332]	[1327:1332]
[1332:1429]							
[1429:1430]	[1429:1430]	[1429:1430]	[1429:1430]	[1429:1430]	[1429:1430]	[1429:1430]	[1429:1430]
[1430:1437]							
1437 Caract.	216 Caract.	64 Caract.	94 Caract.	94 Caract.	94 Caract.	94 Caract.	94 Caract.

Tabela 5.5: Avaliação do Conjunto de Casos de Teste - Classes.jet

Arquivo Template Classes.jet		Modelo 03	Modelo 07	Modelo 08	Modelo 09	Modelo 38	Modelo 41
Mapa OffSet Completo	Mapa OffSet Estático	Caso Teste 01	Caso Teste 04	Caso Teste 05	Caso Teste 06	Caso Teste 25	Caso Teste 28
[0:8]	[0:8]	[0:8]	[0:8]	[0:8]	[0:8]	[0:8]	[0:8]
[8:38]	[8:38]	[8:38]	[8:38]	[8:38]	[8:38]	[8:38]	[8:38]
[38:40]	[38:40]	[38:40]	[38:40]	[38:40]	[38:40]	[38:40]	[38:40]
[40:79]							
[79:80]	[79:80]	[79:80]	[79:80]	[79:80]	[79:80]	[79:80]	[79:80]
[80:165]							
[165:217]	[165:217]	[165:217]	[165:217]	[165:217]	[165:217]	[165:217]	[165:217]
[217:229]							
[229:230]	[229:230]	[229:230]	[229:230]	[229:230]	[229:230]	[229:230]	[229:230]
[230:237]							
[237:238]	[237:238]	[237:238]	[237:238]	[237:238]	[237:238]	[237:238]	[237:238]
[238:278]							
[278:279]	[278:279]	[278:279]	[278:279]	[278:279]	[278:279]	[278:279]	[278:279]
[279:365]							
[365:417]	[365:417]	[365:417]	[365:417]	[365:417]	[365:417]	[365:417]	[365:417]
[417:429]							
[429:430]	[429:430]	[429:430]	[429:430]	[429:430]	[429:430]	[429:430]	[429:430]
[430:437]							
[437:438]	[437:438]	[437:438]	[437:438]	[437:438]	[437:438]	[437:438]	[437:438]
[438:479]							
[479:480]	[479:480]	[479:480]	[479:480]	[479:480]	[479:480]	[479:480]	[479:480]
[480:567]							
[567:575]	[567:619]	[567:619]	[567:619]	[567:619]	[567:619]	[567:619]	[567:619]
[575:617]							
[617:619]							
[619:631]							
[631:632]	[631:632]	[631:632]	[631:632]	[631:632]	[631:632]	[631:632]	[631:632]
[632:639]							
[639:653]	[639:690]	[639:690]	[639:690]	[639:690]	[639:690]	[639:690]	[639:690]
[653:688]							
[688:690]							
[690:734]							
[734:793]	[734:793]	[734:793]	[734:793]	[734:793]	[734:793]	[734:793]	[734:793]
[793:800]							
[800:814]	[800:814]	[800:814]	[800:814]	[800:814]	[800:814]	[800:814]	[800:814]
[814:871]							
[871:884]	[871:884]	[871:884]	[871:884]	[871:884]	[871:884]	[871:884]	[871:884]
[884:926]	[884:926]	[884:926]	[884:926]	[884:926]	[884:926]	[884:926]	[884:926]
[926:927]	[926:927]	[926:927]	[926:927]	[926:927]	[926:927]	[926:927]	[926:927]
[927:964]	[927:964]	[927:964]	[927:964]	[927:964]	[927:964]	[927:964]	[927:964]
[964:982]	[964:982]	[964:982]	[964:982]	[964:982]	[964:982]	[964:982]	[964:982]
[982:1024]	[982:1024]	[982:1024]	[982:1024]	[982:1024]	[982:1024]	[982:1024]	[982:1024]
[1024:1029]	[1024:1029]	[1024:1029]	[1024:1029]	[1024:1029]	[1024:1029]	[1024:1029]	[1024:1029]
[1029:1082]	[1029:1082]	[1029:1082]	[1029:1082]	[1029:1082]	[1029:1082]	[1029:1082]	[1029:1082]
[1082:1104]	[1082:1104]	[1082:1104]	[1082:1104]	[1082:1104]	[1082:1104]	[1082:1104]	[1082:1104]
[1104:1141]	[1104:1141]	[1104:1141]	[1104:1141]	[1104:1141]	[1104:1141]	[1104:1141]	[1104:1141]
[1141:1185]	[1141:1185]	[1141:1185]	[1141:1185]	[1141:1185]	[1141:1185]	[1141:1185]	[1141:1185]
[1185:1238]	[1185:1238]	[1185:1238]	[1185:1238]	[1185:1238]	[1185:1238]	[1185:1238]	[1185:1238]
[1238:1240]	[1238:1240]	[1238:1240]	[1238:1240]	[1238:1240]	[1238:1240]	[1238:1240]	[1238:1240]
[1240:1282]	[1240:1282]	[1240:1282]	[1240:1282]	[1240:1282]	[1240:1282]	[1240:1282]	[1240:1282]
[1282:1284]	[1282:1284]	[1282:1284]	[1282:1284]	[1282:1284]	[1282:1284]	[1282:1284]	[1282:1284]
[1284:1321]	[1284:1321]	[1284:1321]	[1284:1321]	[1284:1321]	[1284:1321]	[1284:1321]	[1284:1321]
[1321:1343]	[1321:1343]	[1321:1343]	[1321:1343]	[1321:1343]	[1321:1343]	[1321:1343]	[1321:1343]
[1343:1380]	[1343:1380]	[1343:1380]	[1343:1380]	[1343:1380]	[1343:1380]	[1343:1380]	[1343:1380]
[1380:1383]	[1380:1383]	[1380:1383]	[1380:1383]	[1380:1383]	[1380:1383]	[1380:1383]	[1380:1383]
[1383:1420]	[1383:1420]	[1383:1420]	[1383:1420]	[1383:1420]	[1383:1420]	[1383:1420]	[1383:1420]
[1420:1436]	[1420:1436]	[1420:1436]	[1420:1436]	[1420:1436]	[1420:1436]	[1420:1436]	[1420:1436]
[1436:1448]							
[1448:1530]	[1448:1530]	[1448:1530]	[1448:1530]	[1448:1530]	[1448:1530]	[1448:1530]	[1448:1530]
1530 Caract.	975 Caract.	N/A	N/A	916 Caract.	975 Caract.	916 Caract.	916 Caract.

16 casos de teste atingiram 43,52% de cobertura dos caracteres, os demais não ultrapassaram 30% de cobertura. Assim, é conclusivo associar a baixa cobertura alcançada com a característica predominantemente dinâmica do *template* avaliado. Outro fator preponderante e digno de ressalva quanto ao *template* main.jet é que por ser um arquivo com conteúdo estritamente populado por configurações para a transformação, como por exemplo, a criação de arquivos e estrutura de diretórios, existe uma tendência de saturação da cobertura, tendo em vista que o conjunto de elementos avaliados não produz novas alternativas para as configurações.

Um problema relacionado ao teste estrutural é a impossibilidade, em geral, de se determinar automaticamente se um caminho é ou não executável, ou seja, não existe um algoritmo que, dado um caminho completo qualquer, decida se o caminho é executável e forneça o conjunto de valores que causam a execução desse caminho (VERGILIO; MALDONADO; JINO, 1993). Assim, é preciso a intervenção do testador para determinar quais são os caminhos não executáveis para o *template* testado.

Nota-se também que os casos de teste avaliados quanto ao template classes.jet apresentam acréscimos heterogêneos que variam entre 5% a 13.37% de complementaridade da cobertura. Este ganho de índice fez com que o conjunto de casos de teste avaliados alcançassem 100% de cobertura. Anteriormente, o conjunto avaliado com uma ocorrência por elemento obteve uma relação de pertinência e inclusão máxima de 88.92% de cobertura em relação ao conjunto de 28 casos de teste.

Outro fator de destaque e que certamente está relacionado com as restrições semânticas aplicadas aos modelos de entrada das transformações M2T é o conjunto de 13 casos de teses já identificados anteriormente que mantiveram-se inertes quanto ao critério de cobertura avaliado. Como o propósito do modelo de entrada é gerar Classes do tipo Java, só há efetiva geração de código quando o elemento ***ENTY*** está presente no percurso, caso contrário apenas as instruções são exercitadas. Entretanto, não há como garantir a exclusão deste conjunto de 13 casos de teste, pois é possível que haja segmentos específicos que tratem de suas ocorrências e outras implementações melhoradas ou aprimoradas dos *templates*.

- ***Cobertura do número de marcadores do* template *substituídos por elementos do modelo de entrada***

Novamente, para este critério de cobertura, o *template* main.jet não produz informações com significância, o que implica na impossibilidade de apresentação dos resultados.

Anteriormente, quando este critério foi analisado com uma ocorrência por elemento para o *template* classes.jet, foi constatada uma relação de pertinência e inclusão para o conjunto de casos de teste com índice máximo de cobertura de 63%. Um grupo composto por nove casos de teste, sendo eles: Caso de Teste 5, 6, 8, 11, 13, 14, 24, 25 e 28, cobre 100% dos marcadores avaliados para o template classes.jet. Assim, é possível afirmar que a complementaridade para este conjunto é dada pela cardinalidade incrementada quanto ao número de ocorrências por elemento em cada caso de teste.

- ***Cobertura do número de instruções exercitadas pelos* templates *na transformação***

  É necessário destacar que, dentre o conjunto de casos de teste avaliado é possível diagnosticar que um grupo apresenta uma característica reincidente de não proporcionar estímulo ao transformador quanto à geração de código nas análises elaboradas sobre o *template* classes.jet. O grupo é composto por 13 caso de teste, sendo eles: Caso de Teste 1, 2, 3, 9, 15, 16, 17, 18, 19, 20, 21, 22 e 26. Entretanto, este mesmo grupo produz resultados relevantes e consideráveis quanto ao *template* main.jet, o que de certo modo inviabiliza o seu descarte.

  A avaliação deste último critério obteve 100% de cobertura com o casos de teste que implementam três ocorrências por elemento. Os ganhos de cobertura são evidenciados pela análise uniforme dos *templates* quanto aos casos de teste, ou seja, não há diferença discricionária entre os segmentos dinâmicos e estáticos do *template*. Outro fator determinante está implicitamente relacionado ao número de ocorrências, quando avalia-se o *template* classes.jet com uma ocorrência por elemento, a cobertura máxima dada à relação de pertinência e inclusão computada não ultrapassa 75% das instruções. Assim, a particularidade deste critério de cobertura permite inferir que quanto maior o número de elementos pertinentes ao domínio, mais efetivos tornam-se os casos de teste.

  Em particular, este critério de cobertura configura-se como o de menor complexidade de cômputo. Assim, a avaliação realizada quanto a este critério é superficial e só demonstra as possibilidades de refinamento deste método de cobertura, como, por exemplo a cobertura de decisões de condições que envolvam elementos do modelo de entrada.

5.2.6 Resultados

Neste estudo piloto foram adotadas duas estratégias quanto à cardinalidade dos elementos presentes em cada caso de teste, sendo elas: a primeira com uma ocorrência por elemento presente no percurso e a segunda com três ocorrências por elemento.

- **Quanto à Avaliação com uma ocorrência por elemento**

  Uma tarefa muito citada na condução e avaliação da atividade de teste é a análise de cobertura, que consiste basicamente em determinar o percentual de elementos requeridos por um dado critério de teste que foram exercitados pelo conjunto de casos de teste utilizado (VERGILIO; MALDONADO; JINO, 1993; DELAMARO; MALDONADO; JINO, 2007). A partir dessa informação o conjunto de casos de teste pode ser aprimorado, acrescentando a ele novos casos para exercitar os elementos ainda não cobertos. Nessa perspectiva, é fundamental o conhecimento sobre as limitações teóricas inerentes à atividade de teste, pois os elementos requeridos podem ser não executáveis, e em geral, determinar a não executabilidade de um dado requisito de teste envolve a participação do testador.

  Observou-se que a primeira estratégia obteve índices baixos de cobertura quando computada a relação de pertinência, inclusão e complementaridade conforme o conjunto de casos de teste avaliado. Ressalta-se que a premissa ideal para um critério de teste é atingir um índice de 100% de cobertura.

  No processo de cômputo e avaliação dos critérios com uma ocorrência por elemento, constatou-se a baixa complementaridade do conjunto de casos de teste. Entretanto, como já explicitado, essa característica já torna-se evidente em conjuntos com cardinalidade monotônica.

  Outro aspecto relevante da avaliação é pautado na dependência que alguns *templates* possuem com relação a determinados elementos que compõe os casos de teste. Um grupo de 13 casos de teste apresentou um comportamento sistemático de não produzir resultados quando avaliados com relação ao *template* classes.jet.

  As constatações observadas basicamente sustentam a necessidade da segunda estratégia. Adicionalmente, é possível afirmar que na geração de casos de teste é preciso atentar não só para os dados relacionados e organizados semanticamente, mas também para especificidade dos dados do domínio da aplicação e para a estrutura dos *templates* avaliados.

- **Quanto à Avaliação com três ocorrências por elemento**

  Dado os perceptíveis ganhos já demonstrados quanto a cada critério avaliado e ao concreto fato de que alguns casos de teste atingem de modo singular o índice de 100% de cobertura com o incremento do número de ocorrências, os ganhos de cobertura quanto a avaliação do *template* main.jet só alcançam um índice de 100% quanto ao critério de teste que versa sobre o número de instruções exercitadas pela transformação. Este fato é intrinsecamente relacionado a característica do critério de cobertura que torna irrelevante a estrutura do arquivo de *template*. O critério de cobertura que versa sobre o número de marcadores não é avaliado para o *template* main.jet, pois não existe em sua estrutura interna qualquer marcador que permita cômputo.

  Com relação ao critério de cobertura do número de caracteres exercitado, a estratégia de fazer uso de três ocorrências por elemento para os casos de teste produz uma cobertura de 43,52% do *template* main.jet por meio das transformações realizadas. Houve um ganho na cobertura, entretanto, dadas as peculiaridades já discutidas quanto ao *template* avaliado, mesmo que o número de ocorrências por elementos seja novamente aumentado, estes ganhos serão sensíveis e porventura não atingirão índices superiores a 50% de cobertura.

  Em contrapartida, foi possível observar que a estratégia proporciona ganhos consideráveis quanto a cobertura do *template* classes.jet. O caso de teste seis em especial atinge um índice de 100% de cobertura quanto aos três critérios avaliados. Entretanto, mesmo desconsiderando este caso de teste, a complementaridade dos demais casos de teste avaliados também atingem 100% de cobertura, o que de certo modo confirma a relevância da estratégia adotada.

5.3 Estudo Exploratório com uma Aplicação Web Completa - Autoria de Conteúdo para Web

Neste estudo será aplicada a abordagem apresentada no Capítulo 4 com suas fases e estratégias. Cabe ressaltar que fora utilizada a mesma estrutura aplicada ao estudo exploratório piloto. Assim, o método descritivo deste estudo exploratório mantém o padrão metodológico aplicado anteriormente, entretanto, dada a maior complexidade do domínio e da aplicação utilizada e visto que detalhes comuns já foram anteriormente explicitados, os tópicos foram sintetizados dando ênfase às discussões e resultados alcançados.

Na Seção 5.3.1 será apresentada uma descrição do domínio e aplicação utilizados neste estudo exploratório.

5.3.1 Contexto e Definição do Estudo

Este estudo exploratório faz uso de um domínio de aplicações de autoria de conteúdo para Web. São aplicações nas quais um administrador publica o conteúdo a ser visualizado por muitos usuários como, por exemplo, portais de notícias, páginas pessoais, fóruns e *blogs*. Trata-se de um domínio técnico, envolvendo os requisitos de persistência e navegação web.

O objeto de estudo foi implementado por Lucrédio (2009) e faz parte de sua tese de doutoramento, na qual foi desenvolvida uma infra-estrutura composta de artefatos de software gerados e não-gerados, arquitetados em três camadas. Três modelos de entrada são utilizados pelo transformador: um modelo de dados construído por meio do GMF (ISTRIA et al., 2013), um modelo de navegação usando o Xtext (ECLIPSE.ORG, 2013b) e um modelo de *features* simples, baseado em EMF (STEINBERG et al., 2009).

A abordagem proposta nesta dissertação será aplicada no modelo de navegação que gera um conjunto de dados e relacionamentos, o que, consequentemente, justifica a maior necessidade quanto à aplicação de técnicas de testes. Em consonância com o estudo piloto, segue-se o modelo de condução de estudo exploratório proposto por Wohlin et al. (2000).

5.3.2 Planejamento

O estudo em questão está inserido no contexto de testes de transformações M2T, o qual envolve a testabilidade dos artefatos gerados por meio da caracterização dos dados complexos de teste e a avaliação dos critérios de cobertura sob os *templates* utilizados nas transformações. O objeto de estudo é uma aplicação web completa denominada "Autoria de Contéudo para Web". A escolha desse objeto de estudo deu-se por sua relevância, tendo em vista que a aplicação faz uso do método *templates* juntamente com implementação de referência e todo arcabouço de soluções como, por exemplo, GMF, EMF e Xtext, já discutidos neste trabalho.

O intuito do procedimento descrito a seguir é verificar a viabilidade de aplicação da abordagem proposta quanto a caracterização dos dados complexos de testes em transformações M2T e verificar o grau de adequação dos casos de teste gerados com base nos percursos de hipergrafos direcionados em relação aos critérios estruturais. Para isso,

segue-se cada uma das fases especificadas pela abordagem proposta. Os materiais e soluções utilizadas para este estudo também estão em consonância com a relação apresentada na Seção 5.2.2.

5.3.3 Operação

Nesta Seção apresenta-se a condução do estudo. Com base na gramática/metamodelo, os dados complexos (elementos do modelo) são representados por meio de um hipergrafo. Posteriormente, esta representação é implementada pelo Algoritmo 1 *SB-Visita*, adaptado de (GUEDES; MARKENZON; FARIA, 2011) gerando os percursos que são base para construção dos casos de teste. Para a criação dos casos de teste, é utilizada uma estratégia de injeção de dados com três ocorrências por elemento presentes no percurso designado. Ressalta-se que os resultados obtidos no estudo piloto por meio da injeção de dados com três ocorrências por elemento obtiveram maior relevância quanto aos critérios de teste empregados.

Na Figura 5.2 é possível observar a representação do metamodelo analisado e na Figura 5.3 a sua correspondente representação por meio do hipergrafo.

Tendo todos os casos de teste elaborados, cada um deles é aplicado como modelo de entrada para o transformador *JET*. Cada transformação realizada gera os arquivos de mapeamento e rastreabilidade. Posteriormente, utiliza-se um algoritmo de *parsing* que interpreta os arquivos de mapeamento e rastreabilidade juntamente com a estrutura dos *templates*. Para garantir a validade dos dados, a execução dos testes é monitorada certificando que todos os casos de teste que compõem os conjuntos são executados.

5.3.4 Coleta de Dados

Os dados coletados nesse estudo foram:

- Os percursos gerados por meio do Algoritmo 1 *SB-Visita* e as devidas adaptações para detecção de ciclo e validação das restrições semânticas. É preciso destacar que a restrições semânticas utilizadas para este estudo exploratório são apresentadas na Figura 4.1 da Seção 4.5.

  Na Tabela 5.6 são apresentados os percursos gerados pelo Algoritmo 1 *SB-Visita*. Cada percurso (hipercaminho) é identificado por um número de ordem apresentado na tabela. As colunas identificadas como A, B e C na tabela são correspondentes às restrições implementadas e validadas pelo algoritmo proposto. Cabe ressaltar

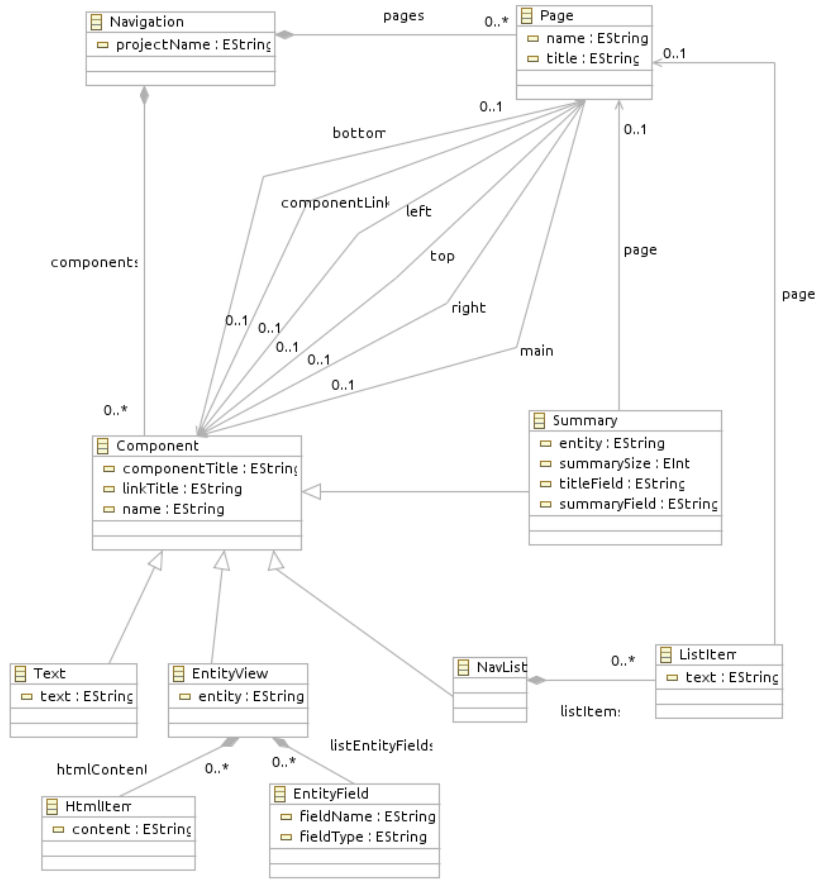


Figura 5.2: Metamodelo - Navigation - (LUCRÉDIO, 2009)

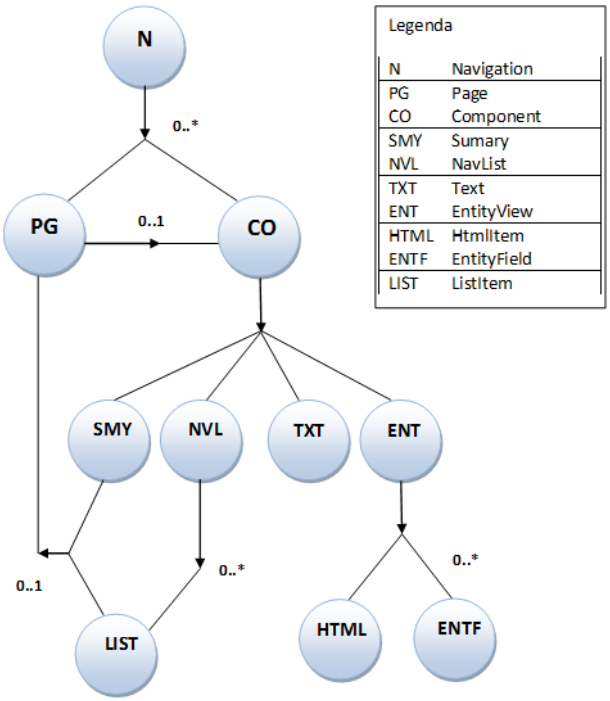


Figura 5.3: Representação Hipergrafo - Metamodelo Navigation

que somente foram incluídos na tabela os percursos validados que serão base para a geração dos casos de teste.

Tabela 5.6: Conjunto de Hipercaminhos

Nº ID	Restrições			Hipercaminhos
	A	B	C	
1	true	true	true	[N, CO, NVL, LIST, PG, SMY, TXT, ENT, HTML]
2	true	true	true	[N, CO, TXT]
3	true	true	true	[N, CO, TXT, SMY]
4	true	true	true	[N, CO, TXT, SMY, PG]
5	true	true	true	[N, CO, TXT, SMY, PG, NVL, LIST]
6	true	true	true	[N, CO, TXT, SMY, PG, ENT, HTML]
7	true	true	true	[N, CO, TXT, SMY, PG, ENT, HTML, ENTF]
8	true	true	true	[N, CO, TXT, SMY, PG, ENT, ENTF]
9	true	true	true	[N, CO, TXT, SMY, PG, NVL, LIST, ENT, HTML]
10	true	true	true	[N, CO, TXT, SMY, PG, NVL, LIST, ENT, HTML, ENTF]
11	true	true	true	[N, CO, TXT, SMY, PG, NVL, LIST, ENT, ENTF]
12	true	true	true	[N, CO, TXT, SMY, NVL, LIST]
13	true	true	true	[N, CO, TXT, SMY, NVL, LIST, PG]
14	true	true	true	[N, CO, TXT, SMY, NVL, LIST, PG, ENT, HTML]
15	true	true	true	[N, CO, TXT, SMY, NVL, LIST, PG, ENT, HTML, ENTF]
16	true	true	true	[N, CO, TXT, SMY, NVL, LIST, PG, ENT, ENTF]
17	true	true	true	[N, CO, SMY]
18	true	true	true	[N, CO, SMY, PG]
19	true	true	true	[N, CO, SMY, PG, TXT]
20	true	true	true	[N, CO, SMY, PG, TXT, NVL, LIST]
21	true	true	true	[N, CO, SMY, PG, TXT, ENT, HTML]
22	true	true	true	[N, CO, SMY, PG, TXT, ENT, HTML, ENTF]
23	true	true	true	[N, CO, SMY, PG, TXT, ENT, ENTF]
24	true	true	true	[N, CO, SMY, PG, TXT, NVL, LIST, ENT, HTML]
25	true	true	true	[N, CO, SMY, PG, TXT, NVL, LIST, ENT, HTML, ENTF]
26	true	true	true	[N, CO, SMY, PG, TXT, NVL, LIST, ENT, ENTF]
27	true	true	true	[N, CO, SMY, PG, NVL, LIST]
28	true	true	true	[N, CO, SMY, PG, ENT, HTML]
29	true	true	true	[N, CO, SMY, PG, ENT, HTML, ENTF]
30	true	true	true	[N, CO, SMY, PG, ENT, ENTF]
31	true	true	true	[N, CO, SMY, PG, NVL, LIST, TXT]
32	true	true	true	[N, CO, SMY, PG, NVL, LIST, TXT, ENT, HTML]
33	true	true	true	[N, CO, SMY, PG, NVL, LIST, TXT, ENT, HTML, ENTF]
34	true	true	true	[N, CO, SMY, PG, NVL, LIST, TXT, ENT, ENTF]
35	true	true	true	[N, CO, SMY, PG, NVL, LIST, ENT, HTML]
36	true	true	true	[N, CO, SMY, PG, NVL, LIST, ENT, HTML, ENTF]
37	true	true	true	[N, CO, SMY, PG, NVL, LIST, ENT, ENTF]
38	true	true	true	[N, CO, NVL, LIST]
39	true	true	true	[N, CO, NVL, LIST, PG]
40	true	true	true	[N, CO, NVL, LIST, PG, TXT]
41	true	true	true	[N, CO, NVL, LIST, PG, TXT, SMY]
42	true	true	true	[N, CO, NVL, LIST, PG, TXT, SMY, ENT, HTML]

Continua na próxima página...

Tabela 5.6 – continuação da página anterior

Nº ID	Restrições			Hipercaminhos
	A	B	C	
43	true	true	true	[N, CO, NVL, LIST, PG, TXT, SMY, ENT, HTML, ENTF]
44	true	true	true	[N, CO, NVL, LIST, PG, TXT, SMY, ENT, ENTF]
45	true	true	true	[N, CO, NVL, LIST, PG, SMY]
46	true	true	true	[N, CO, NVL, LIST, PG, SMY, TXT]
47	true	true	true	[N, CO, NVL, LIST, PG, SMY, TXT, ENT, HTML]
48	true	true	true	[N, CO, NVL, LIST, PG, SMY, TXT, ENT, HTML, ENTF]
49	true	true	true	[N, CO, NVL, LIST, PG, SMY, TXT, ENT, ENTF]
50	true	true	true	[N, CO, NVL, LIST, PG, SMY, ENT, HTML]
51	true	true	true	[N, CO, NVL, LIST, PG, SMY, ENT, HTML, ENTF]
52	true	true	true	[N, CO, NVL, LIST, PG, SMY, ENT, ENTF]
53	true	true	true	[N, CO, NVL, LIST, PG, ENT, HTML]
54	true	true	true	[N, CO, NVL, LIST, PG, ENT, HTML, ENTF]
55	true	true	true	[N, CO, NVL, LIST, PG, ENT, ENTF]
56	true	true	true	[N, CO, ENT, HTML]
57	true	true	true	[N, CO, ENT, HTML, ENTF]
58	true	true	true	[N, CO, ENT, ENTF]
59	true	true	true	[N, PG]
60	true	true	true	[N, PG, CO]
61	true	true	true	[N, PG, CO, TXT]
62	true	true	true	[N, PG, CO, TXT, SMY]
63	true	true	true	[N, PG, CO, TXT, SMY, NVL, LIST]
64	true	true	true	[N, PG, CO, TXT, SMY, ENT, HTML]
65	true	true	true	[N, PG, CO, TXT, SMY, ENT, HTML, ENTF]
66	true	true	true	[N, PG, CO, TXT, SMY, ENT, ENTF]
67	true	true	true	[N, PG, CO, TXT, SMY, NVL, LIST, ENT, HTML]
68	true	true	true	[N, PG, CO, TXT, SMY, NVL, LIST, ENT, HTML, ENTF]
69	true	true	true	[N, PG, CO, TXT, SMY, NVL, LIST, ENT, ENTF]
70	true	true	true	[N, PG, CO, SMY]
71	true	true	true	[N, PG, CO, SMY, TXT]
72	true	true	true	[N, PG, CO, SMY, TXT, NVL, LIST]
73	true	true	true	[N, PG, CO, SMY, NVL, LIST]
74	true	true	true	[N, PG, CO, SMY, ENT, HTML]
75	true	true	true	[N, PG, CO, SMY, ENT, HTML, ENTF]
76	true	true	true	[N, PG, CO, SMY, ENT, ENTF]
77	true	true	true	[N, PG, CO, SMY, TXT, NVL, LIST, ENT, HTML]
78	true	true	true	[N, PG, CO, SMY, TXT, NVL, LIST, ENT, HTML, ENTF]
79	true	true	true	[N, PG, CO, SMY, TXT, NVL, LIST, ENT, ENTF]
80	true	true	true	[N, PG, CO, SMY, NVL, LIST, TXT]
81	true	true	true	[N, PG, CO, SMY, NVL, LIST, TXT, ENT, HTML]
82	true	true	true	[N, PG, CO, SMY, NVL, LIST, TXT, ENT, HTML, ENTF]
83	true	true	true	[N, PG, CO, SMY, NVL, LIST, TXT, ENT, ENTF]
84	true	true	true	[N, PG, CO, SMY, NVL, LIST, ENT, HTML]
85	true	true	true	[N, PG, CO, SMY, NVL, LIST, ENT, HTML, ENTF]
86	true	true	true	[N, PG, CO, SMY, NVL, LIST, ENT, ENTF]
87	true	true	true	[N, PG, CO, SMY, TXT, ENT, HTML]
88	true	true	true	[N, PG, CO, SMY, TXT, ENT, HTML, ENTF]

Continua na próxima página...

Tabela 5.6 – continuação da página anterior

Nº ID	Restrições			Hipercaminhos
	A	B	C	
89	true	true	true	[N, PG, CO, SMY, TXT, ENT, ENTF]
90	true	true	true	[N, PG, CO, NVL, LIST]
91	true	true	true	[N, PG, CO, NVL, LIST, TXT]
92	true	true	true	[N, PG, CO, NVL, LIST, TXT, SMY]
93	true	true	true	[N, PG, CO, NVL, LIST, TXT, SMY, ENT, HTML]
94	true	true	true	[N, PG, CO, NVL, LIST, TXT, SMY, ENT, HTML, ENTF]
95	true	true	true	[N, PG, CO, NVL, LIST, TXT, SMY, ENT, ENTF]
96	true	true	true	[N, PG, CO, NVL, LIST, SMY]
97	true	true	true	[N, PG, CO, NVL, LIST, ENT, HTML]
98	true	true	true	[N, PG, CO, NVL, LIST, ENT, HTML, ENTF]
99	true	true	true	[N, PG, CO, NVL, LIST, ENT, ENTF]
100	true	true	true	[N, PG, CO, NVL, LIST, SMY, TXT]
101	true	true	true	[N, PG, CO, NVL, LIST, SMY, TXT, ENT, HTML]
102	true	true	true	[N, PG, CO, NVL, LIST, SMY, TXT, ENT, HTML, ENTF]
103	true	true	true	[N, PG, CO, NVL, LIST, SMY, TXT, ENT, ENTF]
104	true	true	true	[N, PG, CO, NVL, LIST, SMY, ENT, HTML]
105	true	true	true	[N, PG, CO, NVL, LIST, SMY, ENT, HTML, ENTF]
106	true	true	true	[N, PG, CO, NVL, LIST, SMY, ENT, ENTF]
107	true	true	true	[N, PG, CO, ENT, HTML]
108	true	true	true	[N, PG, CO, ENT, HTML, ENTF]
109	true	true	true	[N, PG, CO, ENT, ENTF]

O Algoritmo 1 *SB-Visita* processou 189 percursos para o hipergrafo que caracterizava a gramática/metamodelo estudada. Desse conjunto, 80 percursos não atenderam as validações impostas pelas restrições semânticas, restando 109 percursos processados e validados. Somente os 109 percursos validados são apresentados na Tabela 5.6.

- Casos de teste gerados de acordo com os percursos selecionados como base para injeção de dados com três ocorrências por elemento. Novamente destaca-se que a estratégia com injeção de dados com três ocorrências por elemento atingiu índices maiores de cobertura quanto aos critérios avaliados no estudo piloto. Em virtude dos resultados empíricos quanto ao estudo piloto realizado, adotou-se para esta pesquisa exploratória casos de uso com três ocorrências por elemento.

- A cobertura dos *templates* pelos critérios estruturais definidos pela estratégia da abordagem já explicitados na Seção 4.8.

5.3.5 Análise

Esta Seção tem por objetivo elaborar uma análise dos dados resultantes das fases executadas pela abordagem proposta e apresentar algumas conclusões possíveis em conformidade com os objetivos definidos neste trabalho. Para isso, foi realizada uma análise quantitativa desses dados de acordo com cada um dos tópicos de dados coletados para este estudo.

I) ***Os percursos e casos de teste gerados***

A representação da gramática/metamodelo, por meio do hipergrafo direcionado, gerou 189 (100%) percursos seguindo o algoritmo implementado e tratando as redundâncias de elementos e detecção de ciclos. Após as devidas validações de acordo com as restrições implementadas, obteve-se 109 (58%) percursos selecionados e 80 (42%) percursos não selecionados. Cabe ressaltar que a Tabela 5.6 apresenta apenas os percursos selecionados que serão utilizados como base para criação dos casos de teste.

Compara-se os resultados da estratégia utilizada pela abordagem para geração dos casos de teste com um método simplista para a formação de subconjuntos, como por exemplo, a Permutação Circular, com um número de combinações dado pela fórmula: $P_c(m) = (m-1)!$, onde $(10-1)! = 362.880$ combinações possíveis. É possível perceber um ganho considerável na otimização do conjunto de casos de tese gerados. Adicionalmente, destaca-se os cuidados com relação à quebra dos estados hierárquicos dos modelos e o alivio das dificuldades de compreensão dos testadores, uma vez que a geração dos casos de teste otimizados correspondem intuitivamente ao significado semântico validado por meio das restrições e do hipercaminho traçado.

Diferente do estudo piloto, no qual utilizou-se duas estratégias para geração dos casos de teste apresentadas por meio de uma tabela que demonstrava as correspondências entre o percurso, caso de teste e modelo de entrada, para esta avaliação devido à complexidade da aplicação analisada e o volume expressivo de casos de teste, estes dados não serão apresentados neste tópico. Entretanto, ressalta-se que não houve alteração no protocolo de condução e que a geração seguiu os trâmites já descritos e formalizados anteriormente.

É importante destacar que para este estudo foram utilizadas três ocorrências por elemento para geração dos casos de teste, tendo em vista os melhores índices de cobertura alcançados relatados pelo estudo piloto.

II) ***A cobertura dos* templates *pelos critérios estruturais definidos***

Neste estudo exploratório o processo de avaliação e cômputo da cobertura dos *templates*, por meio das transformações, utilizou 109 casos de teste. Cada transformação exercitada gerou 218 arquivos de rastreabilidade e mapeamentos que viabilizam a extração das informações utilizadas no processo de análise. Na Seção 4.8 é apresentada a estrutura e detalhes destes arquivos por meio das Figuras 4.8 e 4.9.

Para este estudo a abordagem proposta analisou 36 arquivos de *templates* que fazem parte da estrutura da aplicação. Na Tabela 5.7 são apresentados os *templates* avaliados agrupados de acordo com a estrutura de diretório da aplicação. Cada transformação realizada dá origem a um relatório de cobertura já apresentado na Figura 5.1. Novamente, a estrutura de cada *template* é analisada e separada em dois segmentos, sendo eles: a estrutura dinâmica e a estrutura estática dos *templates*.

É preciso destacar que no estudo piloto, devido à menor complexidade da solução analisada, foi possível elaborar uma avaliação singular para cada *template*. Entretanto, a realidade desta aplicação é outra, pois o maior número de *templates* (36 arquivos) juntamente com o maior conjunto de casos de teste (109, no total) dificultam a elaboração de uma avaliação discricionária. Assim, os dados deste estudo são sumarizados já considerando a complementaridade do conjunto de casos de teste com relação a cada *template* avaliado. Mesmo assim, tabelas analíticas que subsidiam a geração dos gráficos e resultados apresentados a seguir podem ser encontradas na Apêndice A deste trabalho.

O processo de avaliação passa então a sumarizar estes dados que serão apresentados a seguir conforme cada critério de cobertura.

III) ***Cobertura do número de caracteres exercitado no* template *pela transformação***

A diversidade de critérios de teste existente gera algumas questões relevantes. Saber qual desses critérios deve ser utilizado ou como utilizá-lo de forma complementar a fim de obter o melhor resultado constitui desafios à testabilidade de software (MALDONADO, 1991). A realização deste estudo exploratório possibilitou, por meio da análise dos dados obtidos por esse critério de cobertura, demonstrar que o conjunto de casos de teste, como um todo, ou seja, sem considerar a capacidade individual de cada caso de teste, proporcionou índices de cobertura superiores a 80% na maioria dos *templates* avaliados.

Tabela 5.7: Relação de Arquivos de Templates da Aplicação Avaliados

ID	Templates
1	main.jet
Diretório Navigation	
2	generatePages.jet
3	/pages/userContent.jet
4	/pages/uploadUserCommentForm.jsp.jet
5	/pages/page.jsp.jet
6	/pages/components.jet
Diretório UserContent	
7	generateUserContent.jet
8	/files/UserContentDAOAbstractFactory.java.jet
9	/files/UserContentDAO.java.jet
10	/files/UserContentActions.java.jet
11	/files/UserContent.java.jet
12	/files/moderation.jsp.jet
13	/files/DerbyUserContentDAOFactory.java.jet
14	/files/DerbyUserContentDAO.java.jet
15	/files/DefaultUserContentDAOFactoryProvider.java.jet
Diretório Persistence	
16	/actions/generateActions.jet
17	/actions/classes/ActionClass.java.jet
18	/beans/generateBeans.jet
19	/beans/classes/BeanClass.java.jet
20	/common/common.jet
21	/daos/generateDAOs.jet
22	/daos/classes/DefaultDAOFactoryProvider.java.jet
23	/daos/classes/DAOAbstractFactory.java.jet
24	/daos/classes/DAOAbstractClass.java.jet
25	/daos/derby/generateDerbyDAOs.jet
26	/daos/derby/classes/DerbyDAOFactory.java.jet
27	/daos/derby/classes/DerbyDAOClass.java.jet
28	/daos/derby/sql/generateDerbyScript.jet
29	/daos/derby/sql/createDerby.sql.jet
30	/forms/generateForms.jet
31	/forms/pages/listpage.jsp.jet
32	/forms/pages/editpage.jsp.jet
33	/forms/pages/jspf/adminSuffix.jspf.jet
34	/forms/pages/jspf/adminPrefix.jspf.jet
35	/resources/generateResources.jet
36	/resources/files/messages.properties.jet

É preciso destacar também que a estratégia adotada pela abordagem proposta prima pelos estados hierárquicos dos modelos de entrada. Assim, cada percurso do hipergrafo é complementar quanto a inclusão de seus elementos. Entretanto, dada a cardinalidade destes elementos com seus diferentes tipos que compõe o caso de teste é possível associar a cardinalidade dos tipos ao aumento do índice de cobertura.

Na Figura 5.4 é apresentado um gráfico de barras que permite constatar que dos 36 *templates* avaliados quanto a esse critério de cobertura, 25 *templates* identificados na Tabela 5.7 pela coluna ID obtiveram coberturas superiores a 80%. Os *templates* que obtiveram índices inferiores de cobertura ou possuem uma característica predominantemente dinâmica ou o conjunto de casos de teste não possui um tipo de elemento específico que possa determinar 100% de cobertura.

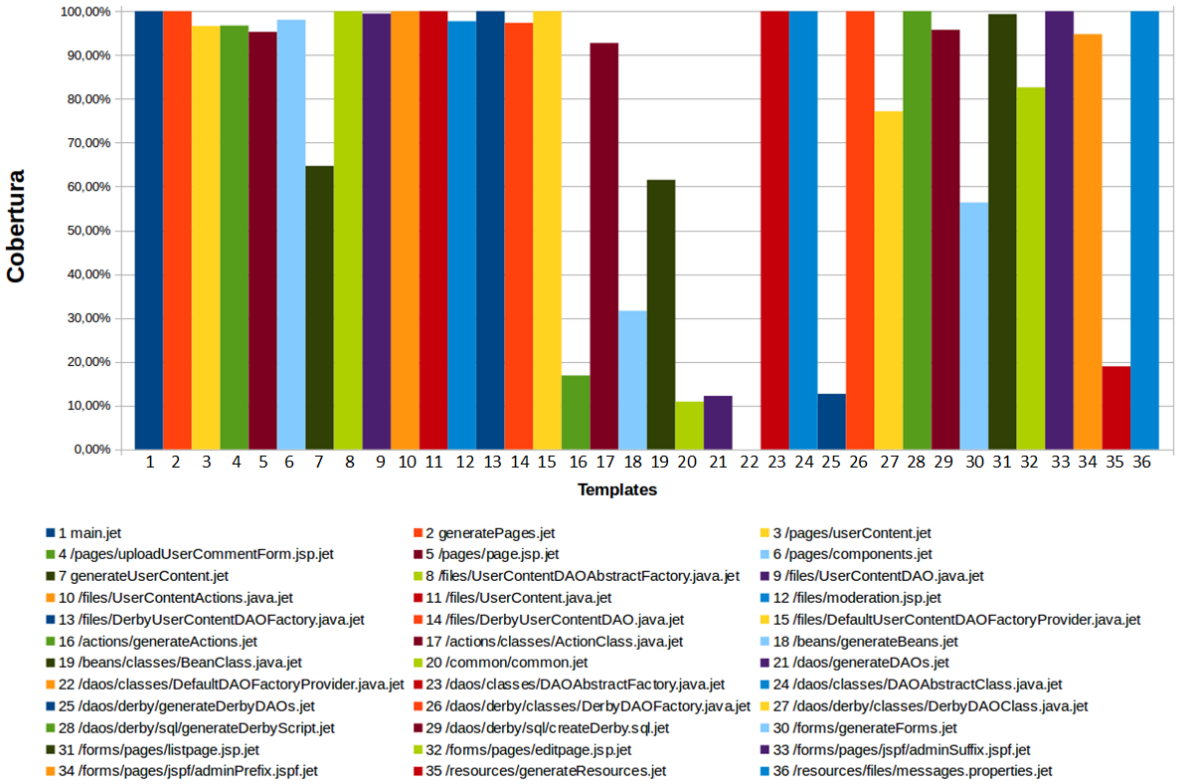


Figura 5.4: Gráfico de barras - Visualização Cobertura dos Casos de Teste - Caracteres exercitados pela Transformação

IV) ***Cobertura dos marcadores exercitados no* template *pela transformação***

Neste critério, apenas 18 *templates* (50% do total) obtiveram índices superiores a 75% de cobertura. Após uma averiguação mais detalhada, constatou-se que os 18 *templates* restantes não possuem em sua estrutura interna nenhum marcador, inviabilizando assim qualquer tipo de cômputo.

Novamente a intervenção do testador é necessária quanto à composição dos casos de teste, dado o número de ocorrências utilizadas. Esta afirmação está pautada nos tipos de dados inseridos para cada ocorrência por elemento do percurso, pois estes possuem relação direta com o domínio da aplicação avaliada e impactam os critérios de coberturas definidos. É possível observar a cobertura de cada *template* no gráfico apresentado por meio da Figura 5.5.

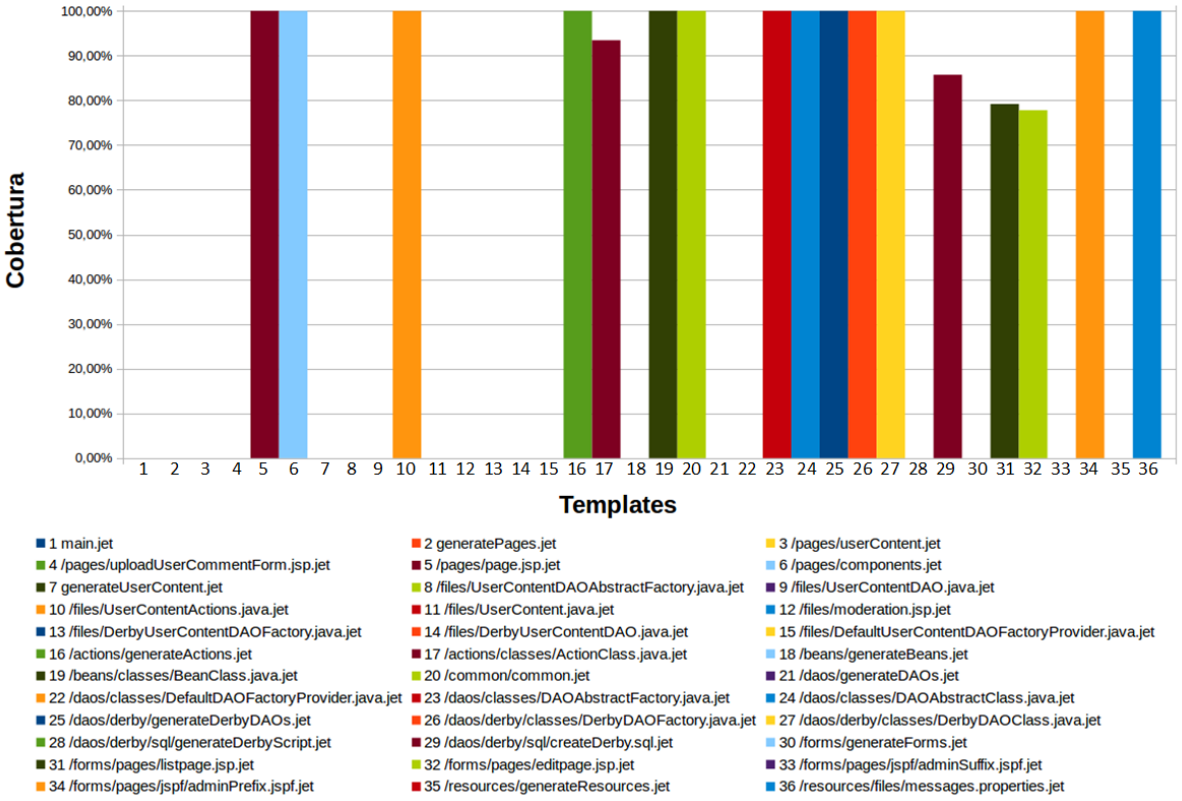


Figura 5.5: Gráfico de barras - Visualização Cobertura dos Casos de Teste - Marcadores exercitados pela Transformação

V) ***Cobertura do número de instruções exercitadas pela transformação***

Na avaliação deste último critério de cobertura, 24 *templates* obtiveram cobertura com índices superiores a 75%, sendo que em grande parte dos *templates* avaliados o índice de cobertura chegou a 100%. Na Figura 5.6 pode-se observar, por meio do gráfico de barras, os resultados relevantes obtidos. Ressalta-se também que os *templates* sem resultado expressivo no gráfico são constituídos de estruturas com características estáticas, ou seja, não possuem qualquer instrução ao transformador, tendo em seu conteúdo apenas trechos que serão transformados diretamente em código-fonte.

A análise discricionária entre os segmentos dinâmicos e estáticos do *template* relacionada com o número de ocorrências por elemento nos casos de teste evidenciam um aumento significativo nos índices de cobertura. Quando avalia-se o *template* classes.jet com uma ocorrência por elemento, a cobertura máxima dada à relação de pertinência e inclusão computada não ultrapassa 75% das instruções. Assim, a par-

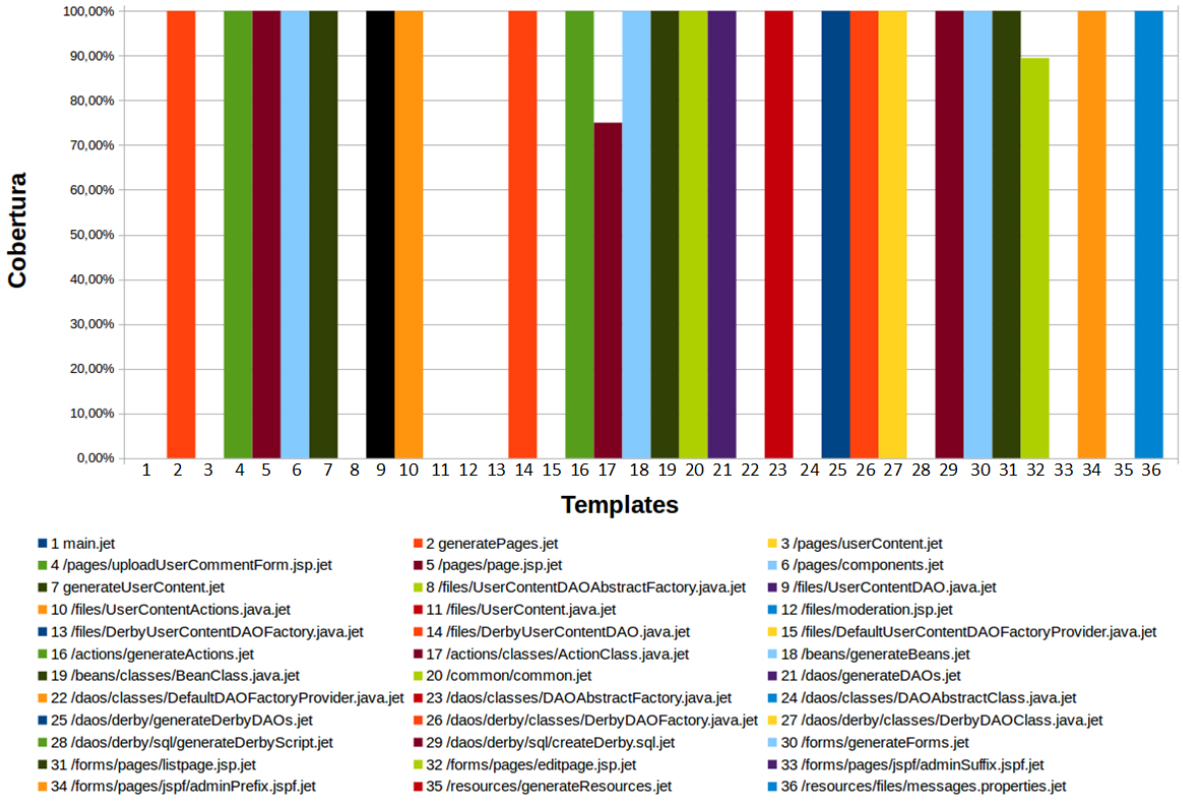


Figura 5.6: Gráfico de área - Visualização da Complementaridade dos Casos de Teste - Comandos do Template exercitadas pela Transformação

ticularidade deste critério de cobertura permite inferir que quanto maior o número de elementos pertinentes ao domínio, mais efetivos tornam-se os casos de teste.

5.3.6 Resultados

O objeto de estudo utilizado no segundo estudo exploratório pertence ao contexto de aplicações web. É uma domínio técnico, envolvendo os requisitos de persistência e navegação web, desenvolvido e utilizado por Lucrédio (2009). Em linhas gerais, o estudo foi aplicado no modelo de navegação que gera um conjunto de dados e relacionamentos utilizados na aplicação. Com base na gramática/metamodelo foram criados os percursos que foram utilizados como base para geração do casos de teste. Ressalta-se que para este estudo exploratório somente foi utilizada a estratégia de três ocorrências por elemento do percurso, tendo em vista os resultados conclusivos do estudo piloto. Outro fator de ressalva é a necessidade de adequação dos casos de teste gerados, dada a especificidade de dados do domínio avaliado.

O número de casos de teste gerados e o número de *templates* para este estudo ex-

ploratório são consideravelmente maiores e mais complexos que os avaliados no estudo piloto. Consequentemente, para esse caso, o processo de análise e cômputo demandou mais esforço quanto a geração e adequação manual dos casos de teste e o cômputo da complementaridade do conjunto de casos de teste gerados. Entretanto, a cobertura dos critérios estruturais alcançados foi muito semelhante ao resultados obtidos na estratégia com três elementos do estudo piloto.

Dada a natureza e complexidade da aplicação avaliada e de modo complementar a estrutura dos arquivos de *template* exercitados pelas transformações, é possível atribuir que, independentemente do critério de teste, em algumas situações o cômputo do índice de cobertura não atinge 100% em detrimento da existência de caminhos não executáveis.

Satisfazer um critério significa exercitar todos os elementos requeridos por esse critério. Neste caso, diz-se que o conjunto de casos de teste $\boldsymbol{T}$ utilizado é adequado em relação ao critério em consideração. A Abordagem proposta, entre outras características, fornece ao testador o mapa de *offset* estático não cobertos, conforme apresentado na Figura 5.1. Esta informação permite que sejam determinados os tipos de ocorrências necessários para os elementos que não foram cobertos pelo critério definido. Assim, é possível, por meio da abordagem, consolidar a adequação de um conjunto de casos de teste, $\boldsymbol{T}$, em relação aos critérios utilizados. Cabe ressaltar que a adequação dos critérios de cobertura não foram tratadas no escopo deste estudo.

Uma constatação importante versa sobre a complementaridade do conjunto de casos de teste. Os casos de testes são gerados por meio de um algoritmo usando combinatória de percursos em hipergrafos, assim é possível preservar o estado hierárquico dos modelos. Entretanto, a condição de complementaridade somente é percebida quando existe o impacto da cardinalidade dos elementos. Quando não há esta situação, o conjunto de casos de teste gerados possui somente uma relação de pertinência ou inclusão de acordo com a teoria de conjuntos.

Outro fator de destaque deste estudo exploratório em particular é quanto à estrutura dos modelos. A abordagem de teste proposta só avaliou o modelo de navegação por meio do conjunto de casos de teste gerados, sendo que algumas especificidades da aplicação causaram falhas no processo da transformação M2T, inviabilizando assim a execução da aplicação. Adicionalmente, constatou-se também que existiam casos de testes que não geram código em situação alguma. Esses casos de teste estão semanticamente corretos e seguem o estado hierárquico do modelo, entretanto não estão em consonância com a sintaxe da linguagem alvo para a transformação M2T. Estas ocorrências podem ser

amenizadas com a implementação de regras em *OCL* que validem os casos de teste em eventos de pré ou pós-condições.

5.4 Considerações Finais

Neste capítulo foram apresentados os dois estudos exploratórios desenvolvidos durante este trabalho de mestrado. Foram exploradas cada uma das fases da abordagem proposta no Capítulo 4, a fim de constatar a viabilidade da proposta, avaliando os critérios de teste estrutural considerados. O objetivo da avaliação foi analisar os resultados obtidos inferidos pelos índices de cobertura alcançados na aplicação de conjuntos de teste, derivados da caracterização de dados complexos por meio de hipergrafos em gramáticas/metamodelos.

As duas avaliações permitiram observar que existe uma deficiência em alguns casos de teste que não conseguem cobrir, de maneira satisfatória, os requisitos definidos para os critérios estruturais. Tal insuficiência torna-se irrelevante quando o aspecto de complementaridade é computado entre o conjunto de casos de teste. Entretanto, fica evidente a possibilidade de otimização a fim de reduzir o esforço na geração dos casos de teste.

Percebeu-se também que devido ao alto nível de abstração do modelo utilizado no segundo estudo exploratório, o esforço para geração dos casos de teste e cômputo dos resultados foi consideravelmente maior. Sendo assim, ressalta-se a necessidade de um ambiente integrado de ferramentas de teste com modelos incorporados, a partir dos quais seriam gerados os casos de teste automaticamente e elaboradas otimizações parametrizadas quanto à especifidade de dados de cada domínio avaliado.

No próximo capítulo são apresentadas as conclusões deste trabalho, bem como a identificação de trabalhos a serem realizados futuramente.

Capítulo 6

Conclusão

Neste trabalho foi investigada a caracterização de dados complexos para testes em transformações de modelo para texto (M2T) e foi proposta uma abordagem de representação, por meio de hipergrafos, com objetivo de compor uma estratégia de teste viável e alinhada com os critérios de teste estruturais. A análise foi elaborada em termos de aplicabilidade prática, por meio de dois estudos exploratórios e a avaliação foi pautada nos resultados computados segundo os critérios de cobertura definidos. Os estudos, além da caracterização dos dados complexos, permitiram a avaliação da própria problemática, ou seja, instaurar (implantar) uma estratégia de teste baseadas em transformações M2T sob o ponto de vista das técnicas e desafios reportados pela literatura.

A abordagem proposta neste trabalho sugere um roteiro de aplicação definido por meio de fases, com intuito de proporcionar sua replicabilidade em diferentes contextos de domínios de aplicação. Entretanto, quanto maior o nível de abstração do modelo de entrada avaliado, maior será o esforço empregado quanto à geração e adequação manual dos casos de teste e o cômputo da complementaridade do conjunto de casos de teste gerados.

Pela avaliação dos resultados obtidos, por meio da análise de cobertura, que consiste basicamente em determinar o percentual de elementos requeridos por um dado critério de teste exercitado pelo conjunto de casos de teste utilizado, pôde-se observar que a abordagem proposta apresentou índices de cobertura superiores a 85%, quanto aos estudos exploratórios realizados. A partir dessa informação o conjunto de casos de teste pode ser aprimorado, acrescentando-se novos casos de teste para exercitar os elementos ainda não cobertos. Nessa perspectiva, é fundamental o conhecimento sobre as limitações teóricas inerentes à atividade de teste, pois os elementos requeridos podem ser não executáveis, e

em geral, determinar a não executabilidade de um dado requisito de teste que envolve a participação do testador.

Conclui-se ainda, que no processo de caracterização de dados complexos é preciso atentar não somente para os dados relacionados e organizados semanticamente, mas também para a especificidade dos dados do domínio da aplicação e para a estrutura dos *templates* avaliados. Essa constatação certamente constitui um desafio a mais para a geração automática dos casos de testes. Adicionalmente, ressalta-se a necessidade de um ambiente integrado de ferramentas de teste, que caracterizem dados complexos por meio dos hipergráfos abstraídos automaticamente da gramática ou metamodelo avaliado. Torna-se necessário ainda, a geração automática dos casos de testes com a possibilidade de ajustes parametrizados em relação à cardinalidade dos elementos e os tipos de dados pertinentes ao domínio.

Sabe-se que a investigação e a abordagem proposta ainda não são suficientes, sendo necessária a realização de mais estudos sobre a otimização dos percursos, a implementação das técnicas de *SAT Solver* ao conjunto de restrições e, principalmente, a geração automática dos casos de testes alinhadas aos critérios de cobertura estrutural.

6.1 Contribuições

As principais contribuições deste trabalho são:

- A revisão sistemática da literatura que constatou a necessidade de abordagens para caracterização de dados complexos para testes estruturais em transformações M2T;
- A abordagem proposta que vislumbrou caracterizar o modelo de entrada de uma transformação M2T, de modo a gerar casos de testes que exercitem os *templates* envolvidos nas transformações, de modo a garantir níveis satisfatórios quanto aos critérios de cobertura definidos para os testes;
- O uso de hipergrafos como uma estratégia, a qual não incorre em um processo minimalista que quebra os estados hierárquicos dos modelos para os casos de teste gerados;
- A realização de 02 (dois) estudos exploratórios que proporcionaram dados para o processo de avaliação da caracterização dos dados complexos até o teste das transformações.

6.2 Limitações

O estudo realizado neste trabalho apresentou as seguintes limitações:

Quanto à abordagem proposta:

- A interpretação manual da gramática/metamodelo pelo testador para o processo de caracterização por meio de hipergrafos.
- A otimização do conjunto de percursos por meio das restrições semânticas.
- A geração manual do conjunto dos casos de testes.
- A falta de uma análise metódica quanto ao custo computacional dos algoritmos utilizados.
- A falta de um estudo para comparar a adequação dos critérios de teste sob os fatores de custo, eficácia e dificuldade de satisfação.

Quanto aos estudos exploratórios:

- O reduzido número de amostras com apenas 02 (duas) aplicações avaliadas.
- A utilização de apenas um transformador para os estudos exploratórios.
- Apesar da abordagem possuir uma proposta replicável e não dependente de tecnologias, os estudos exploratórios são pautados apenas na tecnologia *EMF Ecore*.
- O alto nível de abstração de um modelo de entrada pode inviabilizar a utilização da abordagem devido ao relativo aumento do esforço mecânico empregado.
- A falta de análises sistemáticas quanto à questão da cardinalidade, principalmente quanto ao número de ocorrências por elemento que compõe os casos de teste.

6.3 Trabalhos Futuros

Dentre as atividades que podem ser realizadas para dar continuidade ao trabalho e contribuir para a melhoria do mesmo, destacam-se:

- Considerar a aplicação da abordagem a outros transformadores.

- Otimizar o conjunto de percursos por meio das técnicas de SAT Solver aplicadas às restrições (*constraints*).
- Formalizar computacionalmente o processo de caracterização dos modelos de entrada (Gramáticas ou Metamodelos) para a representação baseada em hipergrafos.
- Validar a abordagem quanto ao custo e sua efetividade comparando seus resultados por meio de métricas e quantificadores encontrados na literatura.
- Implementar uma ferramenta de teste que possibilite a integração entre modelos de entrada e a caracterização dos dados complexos que permita ajustes parametrizados com relação a cardinalidade dos elementos e os tipos de dados pertinentes ao domínio e consequentemente aos *templates* avaliados.

Referências

ABAD, P.; AGUIRRE, N.; BENGOLEA, V.; CIOLEK, D.; FRIAS, M. F.; GALEOTTI, J. P.; MAIBAUM, T.; MOSCATO, M.; ROSNER, N.; VISSANI, I. Improving Test Generation under Rich Contracts by Tight Bounds and Incremental SAT Solving. In: *Proceedings of the 6th International Conference on Software Testing, Verification and Validation (ICST'13)*. Luxemburgo, Luxemburgo: IEEE PRESS, 2013. p. 21–30.

ABADE, A.; FERRARI, F.; LUCRÉDIO, D. Testing M2T Transformations - A Systematic Literature Review. In: *Proceedings of the 17th International Conference on Enterprise Information Systems (ICEIS'15)*. Barcelona, Spain: ScitePress Digital Library, 2015. p. 177–187.

AHO, A. V.; LAM, M. S.; SETHI, R.; ULLMAN, J. D. *Compilers: Principles, Techniques, and Tools*. 2nd. ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2006. ISBN 0321486811.

ALVAREZ, C.; CASALLAS, R. MTC Flow: A Tool to Design, Develop and Deploy Model Transformation Chains. In: *Proceedings of the Workshop on ACadeMics Tooling with Eclipse (ACME'13)*. New York, NY, USA: ACM, 2013. p. 7:1–7:9. ISBN 978-1-4503-2036-8.

AMRANI, M.; LUCIO, L.; SELIM, G.; COMBEMALE, B.; DINGEL, J.; VANGHELUWE, H.; TRAON, Y. L.; CORDY, J. R. A Tridimensional Approach for Studying the Formal Verification of Model Transformations. In: *Proceedings of the 5th International Conference on Software Testing, Verification and Validation (ICST'12)*. Washington, DC, USA: IEEE Computer Society, 2012. p. 921–928. ISBN 978-0-7695-4670-4.

ARCAINI, P.; GARGANTINI, A.; RICCOBENE, E. Optimizing the Automatic Test Generation by SAT and SMT Solving for Boolean Expressions. In: *Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering (ASE'11)*. Washington, DC, USA: IEEE Computer Society, 2011. p. 388–391. ISBN 978-1-4577-1638-6.

ARCURI, A.; IQBAL, M.; BRIAND, L. Random Testing: Theoretical Results and Practical Implications. *Software Engineering, IEEE Transactions on*, v. 38, n. 2, p. 258–277, March 2012. ISSN 0098-5589.

ATKINSON, C.; KUHNE, T. Model-Driven Development: A Metamodeling Foundation. *IEEE Software*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 20, n. 5, p. 36–41, September 2003. ISSN 0740-7459.

AUSIELLO, G.; D'ATRI, A.; SACCà, D. Graph Algorithms for Functional Dependency Manipulation. *Journal of the ACM (JACM)*, ACM, New York, NY, USA, v. 30, n. 4, p. 752–766, octuber 1983. ISSN 0004-5411.

AUSIELLO, G.; FRANCIOSA, P.; FRIGIONI, D. Directed Hypergraphs: Problems, Algorithmic Results, and a Novel Decremental Approach. In: RESTIVO, A.; ROCCA, S. R. D.; ROVERSI, L. (Ed.). *Theoretical Computer Science*: Proceedings of the 7^{th} italian conference ICTCS'01. Torino, Italy: Springer Berlin Heidelberg, 2001. v. 2202, p. 312–328. ISBN 978-3-540-42672-1.

AUSIELLO, G.; NANNI, U.; ITALIANO, G. F. Dynamic Maintenance of Directed Hypergraphs. *Theoretical Computer Science*, Elsevier Science Publishers Ltd., Essex, UK, v. 72, n. 2-3, p. 97–117, maio 1990. ISSN 0304-3975.

BABIC, D.; BINGHAM, J.; HU, A. J. B-Cubing: New Possibilities for Efficient SAT-Solving. *IEEE Transactions on Computers*, IEEE Computer Society, Los Alamitos, CA, USA, v. 55, n. 11, p. 1315–1324, Nov 2006. ISSN 0018-9340.

BAILLIEZ, B.; KEN, N. B.; BERGERON, J.; BODEWIG, S. *Apache ANT Documentation - 1.9.6 Manual*. Dezembro 2013. Disponível Online em: `http://ant.apache.org/manual/index.html`. Acessado em Setembro de 2015.

BASTOS, A.; CRISTALLI, R.; MOREIRA, T.; RIOS, E. *Base de Conhecimento Em Teste de Software*. 3^{rd}. ed. São Paulo, SP, Brasil: Martins Fontes Editora, 2007. ISBN 9788580630534.

BATORY, D. Feature Models, Grammars, and Propositional Formulas. In: *Proceedings of the 9^{th} International Conference on Software Product Lines (SPLC'05)*. Berlin, Heidelberg, Germany: Springer-Verlag, 2005. p. 7–20. ISBN 3-540-28936-4, 978-3-540-28936-4.

BAUDRY, B.; DINH-TRONG, T.; MOTTU, J. M.; SIMMONDS, D.; FRANCE, R.; GHOSH, S.; FLEUREY, F.; TRAON, Y. L. Model Transformation Testing Challenges. In: *Proceedings of the IMDT workshop on Integration of Model Driven Development and Model Driven Testing in conjunction with ECMDA'06*. Bilbao, Spain: Inria, 2006.

BAUDRY, B.; GHOSH, S.; FLEUREY, F.; FRANCE, R.; TRAON, Y. L.; MOTTU, J.-M. Barriers to Systematic Model Transformation Testing. *Magazine Communications of the ACM*, ACM, New York, NY, USA, v. 53, n. 6, p. 139–143, jun. 2010. ISSN 0001-0782.

BELL, M. G. Hyperstar: A multi-path Astar algorithm for risk averse vehicle navigation. *Transportation Research Part B: Methodological*, Elsevier, Amsterdam, Netherlands, v. 43, n. 1, p. 97 – 107, 2009. ISSN 0191-2615.

BERGSTRA, J.; TUCKER, J. The completeness of the algebraic specification methods for computable data types. *Information and Control*, v. 54, n. 3, p. 186 – 200, 1982. ISSN 0019-9958.

BERTOLINO, A. Software Testing Research: Achievements, Challenges, Dreams. In: *Proceedings of the Future of Software Engineering (FOSE '07)*. Washington, DC, USA: IEEE Computer Society, 2007. p. 85–103. ISBN 0-7695-2829-5.

BETTINI, L. *Implementing Domain-Specific Languages with Xtext and Xtend*. Birmingham, UK: Packt Publishing Ltda, 2013. ISBN 9781782160304.

BEYDEDA, S.; BOOK, M.; GRUHN, V. *Model-Driven Software Development*. Berlin, Heidelberg, Germany: Springer-Verlag, 2005. ISBN 354025613X.

BIEHL, M. *Literature Study on Model Transformations*. Stockholm, Sweden, July 2010. Royal Institute of Technology, Technical Report - RT-ES 679/05.

BOOCH, G.; RUMBAUCH, J.; JACOBSON, I. *Unified Modeling Language User Guide*. 2nd edition. ed. Addison-Wesley Object Technology Series: Addison-Wesley Professional, 2005. ISBN 0321267974.

BOYER, R. S.; ELSPAS, B.; LEVITT, K. N. SELECT a Formal System for Testing and Debugging Programs by Symbolic Execution. *ACM SIGPLAN Notices - International Conference on Reliable Software*, ACM, New York, NY, USA, v. 10, n. 6, p. 234–245, abr. 1975. ISSN 0362-1340.

BRAMBILLA, M.; CABOT, J.; WIMMER, M. *Model-Driven Software Engineering in Practice*. 1^{st}. ed. California, USA: Morgan & Claypool Publishers, 2012. (Synthesis Lectures on Software Engineering). ISBN 1608458822.

BROS, N.; PIERS, W. *ATL Concepts*. Junho 2013. Disponível Online em: `http://wiki.eclipse.org/ATL/Concepts`. Acessado em Setembro de 2015.

BROWN, A.; CONALLEN, J.; TROPEANO, D. Introduction: Models, Modeling, and Model-Driven Architecture - MDA. In: BEYDEDA, S.; BOOK, M.; GRUHN, V. (Ed.). *Model-Driven Software Development*. Berlin, Germany: Springer Berlin Heidelberg, 2005. p. 1–16. ISBN 978-3-540-25613-7.

BROY, M.; JONSSON, B.; KATOEN, J.; LEUCKER, M.; PRETSCHNER, A. *Model-Based Testing of Reactive Systems, Advanced Lectures*. Berlin, Heidelberg, Germany: Springer-Verlag, 2005. (Lecture Notes in Computer Science, v. 3472).

CADAR, C.; SEN, K. Symbolic Execution for Software Testing: Three Decades Later. *Commun. ACM*, ACM, New York, NY, USA, v. 56, n. 2, p. 82–90, fev. 2013. ISSN 0001-0782.

CAMARDI, G. Computational models and information theory. *Journal of Experimental and Theoretical Artificial Intelligence*, v. 24, n. 3, p. 401–417, 2012.

CAPLAT, G.; SOURROUILLE, J.-L. Model Mapping Using Formalism Extensions. *IEEE SOFTWARE*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 22, n. 2, p. 44–51, mar. 2005. ISSN 0740-7459.

CARIOU, E.; MARVIE, R.; SEINTURIER, L.; DUCHIEN, L. *Model Transformation Contracts and their Definition in UML and OCL*. Université des Sciences et Technologies de Lille, Villeneuve d'Ascq Cedex, France, April 2004. Technical Report - UMR CNRS 8022.

CHAVEZ, H. M.; SHEN, W.; FRANCE, R. B.; MECHLING, B. A. An approach to testing java implementation against its uml class model. In: *16th International Conference on Model Driven Engineering Languages and Systems (MODELS'13)*. Berlin, Heidelberg, Germany: Springer-Verlag, 2013. p. 220–236 (LNCS 8107).

COMINETTI, R.; CORREA, J. Common-Lines and Passenger Assignment in Congested Transit Networks. *Transportation Science*, Institute for Operations Research and the Management Sciences (INFORMS), Linthicum, Maryland, USA, v. 35, n. 3, p. 250–267, jun. 2001. ISSN 1526-5447.

COURCELLE, B. Monadic Second-Order Logic and Hypergraph Orientation. In: *Proceedings of the 8th Annual IEEE Symposium on Logic in Computer Science (LICS '93)*. Montreal, Canadá: IEEE Computer Society PRESS, 1993. p. 179–190.

CZARNECKI, K. Overview of Generative Software Development. In: *Proceedings of the International Conference on Unconventional Programming Paradigms (UPP'04)*. Berlin, Heidelberg, Germany: Springer-Verlag, 2005. p. 326–341. ISBN 3-540-27884-2, 978-3-540-27884-9.

CZARNECKI, K.; HELSEN, S. Feature-based Survey of Model Transformation Approaches. *IBM Systems Journal*, IBM Corporate, Riverton, NJ, USA, v. 45, n. 3, p. 621–645, jul. 2006. ISSN 0018-8670.

DARABOS, A.; PATARICZA, A.; VARRÓ, D. Towards Testing the Implementation of Graph Transformations. *Electronic Notes in Theoretical Computer Science*, Elsevier, v. 211, p. 75 – 85, 2006. ISSN 1571-0661. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1571066108002466>.

DARMONT, J.; BOUSSAID, O.; RALAIVAO, J.; AOUICHE, K. An Architecture Framework for Complex Data Warehouses. In: CHEN, C.; FILIPE, J.; SERUCA, I.; CORDEIRO, J. (Ed.). *Proceedings of the 7th International Conference on Enterprise Information Systems (ICEIS'05)*. Miami, FL, USA: INSTICC, 2005. p. 370–373. ISBN 972-8865-19-8.

DELAMARO, M. E.; MALDONADO, J. C.; JINO, M. *Introdução ao Teste de Software*. Rio de Janeiro, RJ, Brasil: Editora Campus, 2007. ISBN 9788535226348.

DEMILLO, R. A.; LIPTON, R. J.; SAYWARD, F. G. Hints on Test Data Selection: Help for the Practicing Programmer. *IEEE Computer Society*, v. 11, n. 4, p. 34–43, 1978.

DEMILLO, R. A.; OFFUTT, A. J. Constraint Based Automatic Test Data Generation. *IEEE Transactions on Software Engineering*, SE-17, n. 9, p. 900–910, September 1991.

DING, Z.; SHEN, H.; CAO, J. Hypergraph Partitioning for the Parallel Computation of Continuous Petri Nets. In: MALYSHKIN, V. (Ed.). *Parallel Computing Technologies*. Berlin Heidelberg: Springer, 2011, (Lecture Notes in Computer Science, v. 6873). p. 257–271. ISBN 978-3-642-23177-3.

DURAN, J. W.; NTAFOS, S. C. An Evaluation of Random Testing. *IEEE Transactions on Software Engineering*, IEEE Computer Society, v. 10, n. 4, p. 438–444, July 1984. ISSN 0098-5589.

ECLIPSE.ORG. *Xtend User Guide*. December 2013. Disponível Online em: `http://www.eclipse.org/xtend/documentation/2.5.0/XtendUserGuide.pdf`. Acessado em Setembro de 2015.

ECLIPSE.ORG. *Xtext 2.5 Documentation*. December 2013. Disponível Online em: `http://www.eclipse.org/Xtext/documentation/2.5.0/Xtext20Documentation.pdf`. Acessado em Setembro de 2015.

EDVARDSSON, J. A Survey on Automatic Test Data Generation. In: *Proceedings of the 2nd Conference on Computer Science and Engineering (CCSSE'99)*. Linköping, Sweden: IEEE Computer Society, 1999. p. 21–28.

ERIKSSON, A.; LINDSTRöM, B.; OFFUTT, J. Transformation Rules for Platform Independent Testing: An Empirical Study. In: *Proceedings of the 6th International Conference on Software Testing, Verification and Validation (ICST'13)*. Washington, DC, USA: IEEE Computer Society, 2013. p. 202–211. ISBN 978-0-7695-4968-2.

FABBRI, S. C. P. F.; FELIZARDO, K. R.; FERRARI, F. C.; HERNANDES, E. C. M.; OCTAVIANO, F. R.; NAKAGAWA, E. Y.; MALDONADO, J. C. Externalising Tacit Knowledge of the Systematic Review Process. *IET Software*, The Institution of Engineering and Technology, v. 7, n. 6, p. 298–307, 2013. ISSN 1751-8806.

FAMELIS, M.; SALAY, R.; CHECHIK, M. The semantics of partial model transformations. In: *Proceedings of the 4th International Workshop on Modeling in Software Engineering (MiSE'12)*. Zurich, Switzerland: IEEE Computer Society, 2012. p. 64–69. ISSN 2156-788.

FERRARI, F. C. Apoio ao Teste Estrutural e de Mutação de Software Orientado a Objetos e a Aspectos. São Carlos, SP, Brasil: Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo (ICMC/USP), 2007. Exame Geral de Qualificação para Doutorado.

FLEUREY, F.; BAUDRY, B.; MULLER, P.-A.; TRAON, Y. L. Towards Dependable Model Transformations: Qualifying Input Test Data. *Journal of Software and Systems Modeling (SoSyM)*, v. 8, n. 2, p. 185–203, 2009.

FLEUREY, F.; STEEL, J.; BAUDRY, B. Validation in Model-Driven Engineering: testing model transformations. In: *Proceedings of the 1st International Workshop on Model, Design and Validation (MODEVA'04)*. Rennes, France: IEEE Computer Society, 2004. p. 29–40.

FRANKL, P. G.; WEYUKER, E. J. Testing Software to Detect and Reduce Risk. *Journal of Systems and Software*, Elsevier Science Inc., New York, NY, USA, v. 53, n. 3, p. 275–286, 2000. ISSN 0164-1212.

FRATERNALI, P.; TISI, M. Multi-level Tests for Model Driven Web Applications. In: BENATALLAH, B.; CASATI, F.; KAPPEL, G.; ROSSI, G. (Ed.). *Proceedings of the 10st International Conference on Web Engineering (ICWE'10)*. Berlin, Heidelberg, Germany: Springer-Verlag, 2010. (Lecture Notes in Computer Science, v. 6189), p. 158–172. ISBN 978-3-642-13910-9.

GALLO, G.; GENTILE, C.; PRETOLANI, D.; RAGO, G. Max horn sat and the minimum cut problem in directed hypergraphs. *Mathematical Programming*, Elsevier Science Publishers B. V., Amsterdam, Netherlands, v. 80, n. 2, p. 213–237, 1998. ISSN 0025-5610.

GALLO, G.; LONGO, G.; PALLOTTINO, S.; NGUYEN, S. Directed Hypergraphs and Applications. *Discrete Applied Mathematics*, Elsevier Science Publishers B. V., Amsterdam, Netherlands, v. 42, n. 2-3, p. 177–201, abr. 1993. ISSN 0166-218X.

GALLO, G.; SCUTELLÀ, M. Directed hypergraphs as a modelling paradigm. *Rivista di matematica per le scienze economiche e sociali*, Springer-Verlag, v. 21, n. 1-2, p. 97–123, 1998. ISSN 1593-8883.

GALLO, G.; SCUTELLÀ, M. G. A note on minimum makespan assembly plans. *European Journal of Operational Research*, Elsevier Science Publishers B. V., Amsterdam, Netherlands, v. 142, n. 2, p. 309–320, 2002. ISSN 0377-2217.

GOGOLLA, M.; VALLECILLO, A. Tractable Model Transformation Testing. In: FRANCE, R.; KUESTER, J.; BORDBAR, B.; PAIGE, R. (Ed.). *Modelling Foundations and Applications*. Berlin Heidelberg: Springer, 2011, (Lecture Notes in Computer Science, v. 6698). p. 221–235. ISBN 978-3-642-21469-1.

GOMES, C. P.; KAUTZ, H.; SABHARWAL, A.; SELMAN, B. Satisfiability Solvers. In: HARMELEN, V. L. Frank van; PORTER, B. (Ed.). *Handbook of Knowledge Representation*. Amsterdam, Netherlands: Elsevier Science Publishers B. V., 2008, (Foundations of Artificial Intelligence, v. 3). p. 89 – 134.

GROSS, H.; KRUSE, P.; WEGENER, J.; VOS, T. Evolutionary White-Box Software Test with the EvoTest Framework: A Progress Report. In: *Proceedings of the International Conference on Software Testing, Verification and Validation Workshops (ICSTW'09)*. Denver, Colorado, USA: IEEE Computer Society, 2009. p. 111–120.

GUEDES, A.; MARKENZON, L.; FARIA, L. Flow hypergraph reducibility. *Discrete Applied Mathematics*, v. 159, n. 16, p. 1775 – 1785, 2011. ISSN 0166-218X.

GUEDES, A. L. P. *Hipergrafos Direcionados*. Tese (Doutorado) — Instituto Alberto Luiz Coimbra de Pós-Graduação e Pesquisa de Engenharia, Universidade Federal do Rio de Janeiro, 2001. Tese em Engenharia de Sistemas e Computação - COPPE/UFRJ.

GUERRA, E.; LARA, J. D.; KOLOVOS, D.; PAIGE, R. A Visual Specification Language for Model-to-Model Transformations. In: HUNDHAUSEN, C.; PIETRIGA, E.; DÃAZ, P.; ROSSON, M. B. (Ed.). *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'10)*. Leganés, Madrid, Spain: IEEE Computer Society, 2010. p. 119–126. ISSN 1943-6092.

GUERRA, E.; SOEKEN, M. Specification-driven model transformation testing. *Software and Systems Modeling*, Springer Berlin Heidelberg, p. 1–22, 2013. ISSN 1619-1366.

HAMLET, D.; TAYLOR, R. Partition Testing Does Not Inspire Confidence. *IEEE Transactions on Software Engineering*, IEEE Press, Piscataway, NJ, USA, v. 16, n. 12, p. 1402–1411, dez. 1990. ISSN 0098-5589.

HAREL, D.; RUMPE, B. *Modeling Languages: Syntax, Semantics and All That Stuff*. Jerusalem, Israel: Weizmann Science Press, 2004. Technical Report Part I: The Basic Stuff.

HARMAN, M. Open Problems in Testability Transformation. In: *Proceedings of the IEEE International Conference on Software Testing Verification and Validation Workshop (ICSTW'08)*. Lillehammer, Norway: IEEE Computer Society, 2008. p. 196–209.

HARROLD, M. J. Testing: A Roadmap. In: *Proceedings of the International Conference on the future of Software Engineering - held in conjunction with ICSE'00*. Limerick, Ireland: ACM Press, 2000. p. 61–72.

HAXAIRE, A.; MARKUS, A.; LONG, B.; BAKER, W. B.; CARROL, C.; BROWN, C.; LEEUW, d. C.; DUFFY, D.; BLEVINS, J. *Fortran Wiki Home Page*. Junho 2013. Disponível Online em: `http://fortranwiki.org/`. Acessado em Setembro de 2015.

HECHT, M. S.; ULLMAN, J. D. Flow Graph Reducibility. In: *Proceedings of the 4^{th} Annual ACM Symposium on Theory of Computing (STOC'72)*. New York, NY, USA: ACM Press, 1972. p. 238–250.

HECHT, M. S.; ULLMAN, J. D. Characterizations of Reducible Flow Graphs. *J. ACM*, ACM Press, New York, NY, USA, v. 21, n. 3, p. 367–375, jul. 1974. ISSN 0004-5411.

HOWDEN, W. E. Symbolic Testing and the DISSECT Symbolic Evaluation System. *IEEE Transactions on Software Engineering*, v. 3, n. 4, p. 266–278, 1977.

IEEE, I. Standard, *IEEE Standard Glossary of Software Engineering Terminology*. New York, NY, USA: IEEE Computer Society, 1990. Technical Report.

IORDANOV, B. Hypergraphdb: A generalized graph database. In: *Proceedings of the International Conference on Web-age Information Management (WAIM'10)*. Berlin, Heidelberg, Germany: Springer-Verlag, 2010. p. 25–36. ISBN 3-642-16719-5, 978-3-642-16719-5.

ISTRIA, M.; PUPIER, A.; CHAUVIN, M.; WALKER, U. *Graphical Modeling Framework/Documentation*. Junho 2013. Disponível Online em: `http://wiki.eclipse.org/Graphical-Modeling-Framework/Documentation`. Acessado em Setembro de 2015.

JALOTE, P. Functional refinement and nested objects for object-oriented design. *IEEE Transactions on Software Engineering*, v. 15, n. 3, p. 264–270, Mar 1989. ISSN 0098-5589.

JIH, H.; REEVES, T. Mental models: A research focus for interactive learning systems. *Educational Technology Research and Development*, Kluwer Academic Publishers, v. 40, n. 3, p. 39–53, 1992. ISSN 1042-1629.

KIRNER, R. Towards preserving model coverage and structural code coverage. *Journal on Embedded Systems*, Hindawi Publishing Corporate, New York, NY, USA, v. 2009, p. 6:1–6:16, jan. 2009. ISSN 1687-3955.

KITCHENHAM, B. *Procedures for Performing Systematic Reviews*. Keele, Staffordshire, UK, 2004. Keele University Technical Report TR/SE-0401.

KLEPPE, A. G. A Language Description is More than a Metamodel. In: *Proceedings of the 4th International Workshop on Software Language Engineering*. Grenoble, France: megaplanet.org, 2007.

KLEPPE, A. G.; WARMER, J.; BAST, W. *MDA Explained: The Model Driven Architecture: Practice and Promise*. Boston, MA, USA: Addison-Wesley Longman Publishing Co. Inc., 2003. ISBN 032119442X.

KOLOVOS, D. S.; PAIGE, R. F.; POLACK, F. A. Model Comparison: A Foundation for Model Composition and Model Transformation Testing. In: *Proceedings of the International Workshop on Global Integrated Model Management (GaMMa'06)*. New York, NY, USA: ACM, 2006. p. 13–20. ISBN 1-59593-410-3.

KOREL, B. Automated Software Test Data Generation. *IEEE Transactions on Software Engineering*, IEEE Press, Piscataway, NJ, USA, v. 16, n. 8, p. 870–879, 1990.

KÜSTER, J.; ABD-EL-RAZIK, M. Validation of Model Transformations? First Experiences Using a White Box Approach. In: KÜHNE, T. (Ed.). *Models in Software Engineering*. Berlin Heidelberg: Springer-Verlag, 2007, (Lecture Notes in Computer Science, v. 4364). p. 193–204. ISBN 978-3-540-69488-5.

LAGUNA, M.; MARQUES, J. Feature Diagrams and their Transformations: An Extensible Meta-model. In: *Proceedings of the 35th Euromicro Conference on Software Engineering and Advanced Applications (SEAA'09)*. Novi Sad, Serbia: ComSISI, 2009. p. 97–104. ISSN 1089-6503.

LAGUNA, M. A.; MARQUÉS, J. M.; RODRÍGUEZ-CANO, G. Feature diagram formalization based on directed hypergraphs. *Computer Science and Information Systems*, v. 8, n. 3, p. 611–633, 2011.

LAKATOS, E.; MARCONI, M. de A. *Fundamentos de Metodologia Científica*. 7st. ed. São Paulo, Brasil: Editora Atlas, 2010. 234 p. ISBN 9788522457588.

LAMANCHA, B. P.; MATEO, P. R.; POLO, M.; CAIVANO, D. Model-driven Testing - Transformations from Test Models to Test Code. In: *Proceedings of the 6th International Conference on Evaluation of Novel Approaches to Software Engineering - (ENASE'11)*. Beijing, China: SciTePress, 2011. p. 121–130.

LAMARI, M. Towards an Automated Test Generation for the Verification of Model Transformations. In: *Proceedings of the ACM Symposium on Applied Computing (SAC'07)*. New York, NY, USA: ACM, 2007. p. 998–1005. ISBN 1-59593-480-4.

LANO, K.; CLARK, D. Model Transformation Specification and Verification. In: *Proceedings of the 8th International Conference on Quality Software (QSIC'08)*. Oxford, UK: IEEE Computer Society, 2008. p. 45–54. ISSN 1550-6002.

LARA, J. de; GUERRA, E. Formal Support for Model Driven Development with Graph Transformation Techniques. In: ESTÉVEZ, A.; PELECHANO, V.; VALLECILLO, A. (Ed.). *Actas del II Taller sobre Desarrollo de Software Dirigido por Modelos, MDA y Aplicaciones (DSDM'05)*. Madrid, Spain: CEUR-WS, 2005. v. 15, p. 30–39. ISSN ISSN 1613-0073.

LEDO, A.; RAMALHO, F.; MELO, N. MetaTT - A Metamodel Based Approach for Writing Textual Transformations. In: *Proceedings of the 6th Brazilian Symposium on Software Components, Architectures and Reuse*. Natal, RN, Brazil: IEEE Computer Society, 2012. v. 0, p. 61–70. ISBN 978-1-4673-4783-9.

LI, J. Prioritize code for testing to improve code coverage of complex software. In: *Proceedings of the 16th IEEE International Symposium on Software Reliability Engineering (ISSRE'05)*. Chicago, IL, USA: IEEE Computer Society, 2005. p. 10 pp.–84. ISSN 1071-9458.

LIN, Y.; ZHANG, J.; GRAY, J. Model comparison: A key challenge for transformation testing and version control in model driven software development. In: *Proceedings of the OOPSLA/GPCE: Best Practices for Model-Driven Software Development Control in Model-Driven Software Development*. Berlin Heidelberg: Springer, 2004. p. 219–236.

LUCRÉDIO, D. *Uma abordagem orientada a modelos para reutilização de software*. Tese (Doutorado) — Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2009. Disponível Online em: `http://www.teses.usp.br/teses/disponiveis/55/55134/tde-02092009-140533`. Acessado em Setembro de 2015.

MALDONADO, J.; VINCENZI, A.; BARBOSA, E.; SOUZA, S. do Rocio Senger de; DELAMARO, M. *Aspectos teóricos e empíricos de teste de cobertura de software*. SÃ£o Carlos, Brasil: ICMSC-USP, 1998. (Notas Ditáticas do ICMSC-USP).

MALDONADO, J. C. *Critérios Potenciais Usos: Uma Contribuição ao Teste Estrutural de Software*. Tese (Doutorado) — Faculdade de Engenharia Elétrica e de Computação (FEEC), Universidade Estadual de Campinas (UNICAMP), Campinas, SP, Brasil, 1991. Disponível Online em: `http://www.bibliotecadigital.unicamp.br/document/?code=vtls000031945`. Acessado em Setembro de 2015.

MARCOTTE, P.; NGUYEN, S. Hyperpath Formulations of Traffic Assignment Problems. In: MARCOTTE, P.; NGUYEN, S. (Ed.). *Equilibrium and Advanced Transportation Modelling*. New York, NY, USA: Springer-Verlag New York Inc., 1998, (Centre for Research on Transportation). p. 175–200. ISBN 978-1-4613-7638-5.

MELLOR, S.; CLARK, A.; FUTAGAMI, T. Model-driven development. *IEEE Software*, EEE Computer Society, Los Alamitos, CA, USA, v. 20, n. 5, p. 14–18, 2003. ISSN 0740-7459.

MENS, T.; CZARNECKI, K.; GORP, P. V. A Taxonomy of Model Transformation. *Journal Electronic Notes in Theoretical Computer Science (ENTCS)*, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 152, p. 125–142, mar. 2006. ISSN 1571-0661.

MINAS, M.; VIEHSTAEDT, G. DiaGen: a generator for diagram editors providing direct manipulation and execution of diagrams. In: *Proceedings of the 11th IEEE International Symposium on Visual Languages*. Darmstadt, Germany: EEE Computer Society, 1995. p. 203–210. ISSN 1049-2615.

MOTTU, J.-M.; BAUDRY, B.; TRAON, Y. L. Mutation Analysis Testing for Model Transformations. In: *Proceedings of the 2nd European Conference on Model Driven Architecture: Foundations and Applications (ECMDA-FA'06)*. Berlin, Heidelberg, Germany: Springer-Verlag, 2006. p. 376–390. ISBN 3-540-35909-5, 978-3-540-35909-8.

MOTTU, J.-M.; BAUDRY, B.; TRAON, Y. L. Reusable MDA Components: A Testing-for-Trust Approach. In: NIERSTRASZ, O.; WHITTLE, J.; HAREL, D.; REGGIO, G. (Ed.). *Model Driven Engineering Languages and Systems*. Berlin, Heidelberg, Germany: Springer-Verlag, 2006, (Lecture Notes in Computer Science, v. 4199). p. 589–603. ISBN 978-3-540-45772-5.

MOTTU, J.-M.; BAUDRY, B.; TRAON, Y. L. Model transformation testing: Oracle issue. In: *Proceedings of the IEEE International Conference on Software Testing Verification and Validation Workshop (ICSTW'08)*. Lillehammer, Norway: IEEE Computer Society, 2008. p. 105–112.

MYERS, G. J.; SANDLER, C.; BADGETT, T.; THOMAS, T. M. *The Art of Software Testing*. 2nd. ed. Hoboken, NJ, USA: John Wiley & Sons, 2004.

NASLAVSKY, L.; ZIV, H.; RICHARDSON, D. J. Using Model Transformation to Support Model-based Test Coverage Measurement. In: *Proceedings of the 3rd International Workshop on Automation of Software Test (AST'08)*. New York, NY, USA: ACM, 2008. p. 1–6. ISBN 978-1-60558-030-2.

NGUYEN, S.; PALLOTTINO, S. Hyperpaths and Shortest Hyperpaths. In: SIMEONE, B. (Ed.). *Combinatorial Optimization*. Berlin, Heidelberg, Germany: Springer-Verlag, 1989, (Lecture Notes in Mathematics, v. 1403). p. 258–271. ISBN 978-3-540-51797-9.

NIELSEN, L. R.; ANDERSEN, K. A.; PRETOLANI, D. Finding the K Shortest Hyperpaths. *Computers & Operations Research*, Elsevier Science Publishers B. V., Oxford, UK, UK, v. 32, n. 6, p. 1477–1497, jun. 2005. ISSN 0305-0548.

NORTJE, R.; BRITZ, K.; MEYER, T. A normal form for hypergraph-based module extraction for sroiq. In: *Proceedings of the 8th Australasian Ontology Workshop (AOW'12)*. Sydney, Australia: Centre for Artificial Intelligence Research (CAIR), 2012. v. 969, p. 40–51.

OMG, O. M. G. *MDA Guide Version 1.0.1*. Junho 2003. Disponível Online em: `http://www.omg.org/cgi-bin/doc?omg/03-06-01.pdf`. Acessado em Setembro de 2015.

OMG, O. M. G. *MOF - Common Warehouse Metamodel, v1.1*. Junho 2003. Disponível Online em: `http://http://www.omg.org/spec/CWM/1.1/PDF/`. Acessado em Setembro de 2015.

OMG, O. M. G. *MOF Model to Text Transformation Language, v1.0*. Janeiro 2008. Disponível Online em: `http://www.omg.org/spec/MOFM2T/1.0/PDF`. Acessado em Setembro de 2015.

OMG, O. M. G. *MOF Model to Text Transformation Language, v1.0*. Janeiro 2008. Disponível Online em: `http://www.omg.org/spec/MOFM2T/1.0/PDF`. Acessado em Setembro de 2015.

OMG, O. M. G. *Unified Modeling Language^{TM} (OMG UML), Infrastructure, 2.4.1*. Agosto 2011. Disponível Online em: `http://www.omg.org/spec/UML/2.4/Infrastructure/Beta2/PDF`. Acessado em Setembro de 2015.

OMG, O. M. G. *Object Constraint Language (OCL)*. Junho 2012. Disponível Online em: `http://www.omg.org/spec/OCL/2.3.1/PDF/`. Acessado em Setembro de 2015.

OMG, O. M. G. *Meta Object Facility (MOF) Core Specification*. Junho 2013. Disponível Online em: `http://www.omg.org/spec/MOF/2.4.1/PDF/`. Acessado em Setembro de 2015.

Oracle. *The Java EE 5 Tutorial For Sun Java System Application Server*. Junho 2010. Disponível Online em: `http://docs.oracle.com/javaee/5/tutorial/doc/javaeetutorial5.pdf`. Acessado em Setembro de 2015.

OREJAS, F.; WIRSING, M. On the specification and verification of model transformations. In: PALSBERG, J. (Ed.). *Semantics and Algebraic Specification*. Berlin, Heidelberg, Germany: Springer-Verlag, 2009, (Lecture Notes in Computer Science, v. 5700). p. 140–161. ISBN 978-3-642-04163-1.

OSHEROVE, R. *The Art of Unit Testing*: With examples in .net. 1st. ed. Greenwich, CT, USA: Manning Publications Co., 2009. 296 p. ISBN 1933988274.

PANDY, G.; CHAWLA, S. *A Simple and Efficient Algorithm for Hypercycle Detection*. Sidney, Australia: [s.n.], 2003. School of Information Technologies, University of Sydney, Technical Report - RT 538. ISBN 1-86487-583-6.

POLACK, D.; PAIGE, R.; ROSE, L.; POLACK, F. Unit testing model management operations. In: *Proceedings of the International Conference on Software Testing Verification and Validation (ICSTW'08)*. Lillehammer, Norway: IEEE Computer Society, 2008. p. 97–104.

POSSATTO, A. M. *Uma abordagem para Migração Automática de ódigo no Contexto do Desenvolvimento Orientado a Modelos*. Tese (Doutorado) — DC/UFSCAR - Departamento de Computação da Universidade Federal de São Carlos, São Carlos, SP, Brasil, 2013.

PRESSMAN, R. S. *Software engineering - A Practitioner's Approach*. 6th.. ed. New York, NY, USA: McGraw-Hill, 2005. ISBN 978-0-07-722780-7.

PRESSMAN, R. S. *Software engineering: A Practitioner's Approach*. 7th. ed. New York, NY, USA: McGraw-Hill, 2010. ISBN 978-0-07-337597-7.

PRETOLANI, D. A directed hypergraph model for random time dependent shortest paths. *European Journal of Operational Research*, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 123, n. 2, p. 315 – 324, 2000. ISSN 0377-2217.

PRIETO-DIAZ, R.; ARANGO, G. *Domain Analysis and Software Systems Modeling*. Los Alamitos, CA, USA: IEEE Computer Society Press, 1991. ISBN 081868996X.

RAMAMOORTHY, C. V.; HO, S.-B. F.; CHEN, W. T. On the Automated Generation of Program Test Data. *IEEE Transactions on Software Engineering*, IEEE Computer Society, Los Alamitos, CA, USA, v. 2, n. 4, p. 293–300, 1976.

RAPPS, S.; J.WEYUKER, E. Selecting Software Test Data Using Data Flow Information. *IEEE Transactions on Software Engineering*, IEEE Computer Society, v. 11, n. 4, p. 367–375, 1985.

RAPPS, S.; WEYUKER, E. J. Data Flow Analysis Techniques for Program Test Data Selection. In: *Proceedings of the 6th International Conference on Software Engineering (ICSE'82)*. Tokio, Japan: IEEE Computer Society, 1982. p. 272–278.

ROSE, L. M.; MATRAGKAS, N.; KOLOVOS, D. S.; PAIGE, R. F. A Feature Model for Model-to-text Transformation Languages. In: *Proceedings of the 4th International Workshop on Modeling in Software Engineering (MiSE'12)*. Zurich, Switzerland: IEEE Computer Society Press, 2012. p. 57–63. ISBN 978-1-4673-1757-3.

RUMBAUGH, J.; BLAHA, M.; PREMERLANI, W.; EDDY, F.; LORENSEN, W. *Object-oriented Modeling and Design*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1991. ISBN 0-13-629841-9.

SELIM, G. M. K.; CORDY, J. R.; DINGEL, J. Model Transformation Testing: The State of the Art. In: *Proceedings of the 1st Workshop on the Analysis of Model Transformations (AMT'12)*. Innsbruck, Austria: ACM Press, 2012. p. 21–26. ISBN 978-1-4503-1803-7.

SEN, S.; BAUDRY, B.; MOTTU, J.-M. On Combining Multi-formalism Knowledge to Select Models for Model Transformation Testing. In: *Proceedings of the 1st International Conference on Software Testing, Verification, and Validation (ICST'08)*. Lillehammer, Norway: IEEE Computer Society, 2008. p. 328–337. ISBN 978-0-7695-3127-4.

SEN, S.; BAUDRY, B.; MOTTU, J.-M. Automatic Model Generation Strategies for Model Transformation Testing. In: PAIGE, R. F. (Ed.). *Theory and Practice of Model Transformations*. Berlin, Heidelberg, Germany: Springer-Verlag, 2009, (Lecture Notes in Computer Science, v. 5563). p. 148–164. ISBN 978-3-642-02407-8.

SENDALL, S.; KOZACZYNSKI, W. Model transformation: the heart and soul of model-driven software development. *IEEE Software*, IEEE Computer Society, v. 20, n. 5, p. 42–45, 2003. ISSN 0740-7459.

SHAMSHIRI, S.; ROJAS, J. M.; FRASER, G.; MCMINN, P. Random or genetic algorithm search for object-oriented test suite generation? In: *Proceedings of the 2015 on Genetic and Evolutionary Computation Conference*. New York, NY, USA: ACM, 2015. (GECCO '15), p. 1367–1374. ISBN 978-1-4503-3472-3.

SHLAER, S.; MELLOR, S. J. *Object-oriented Systems Analysis*: Modeling the World in Data. Upper Saddle River, NJ, USA: Yourdon Press, 1988. ISBN 0-13-629023-X.

STEEN, M. v. *Graph theory and complex networksan introduction*: An Introduction. Lexington, KY, USA: Maarten van Steen, 2010. Paperback. ISBN 978-90-815406-1-2.

STEINBERG, D.; BUDINSKY, F.; PATERNOSTRO, M.; MERKS, E. *EMF: Eclipse Modeling Framework*. 2. ed. Boston, MA, USA: Addison-Wesley, 2009. ISBN 978-0-321-33188-5.

STUERMER, I.; CONRAD, M.; DOERR, H.; PEPPER, P. Systematic testing of model-based code generators. *IEEE Transactions on Software Engineering*, IEEE Press, Piscataway, NJ, USA, v. 33, n. 9, p. 622–634, set. 2007. ISSN 0098-5589.

SWITHINBANK, P.; CHESSELL, M.; GARDNER, T.; GRIFFIN, C.; MAN, J.; WYLIE, H.; YUSUF, L. *Patterns: Model-Driven Development Using IBM Rational Software Architect*. Online in ibm.com/redbooks: International Business Machines Corporation (IBM), 2005.

THAKUR, M.; TRIPATHI, R. Linear Connectivity Problems in Directed Hypergraphs. *Theoretical Computer Science*, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 410, n. 27 - 29, p. 2592 – 2618, 2009. ISSN 0304-3975.

TIAN, T.; GONG, D. Test data generation for path coverage of message-passing parallel programs based on co-evolutionary genetic algorithms. *Automated Software Engineering*, Springer Science Business, New York, NY, USA, p. 1–32, 2014. ISSN 0928-8910.

TISO, A.; REGGIO, G.; LEOTTA, M. Early Experiences on Model Transformation Testing. In: *Proceedings of the 1st Workshop on the Analysis of Model Transformations (AMT'12)*. New York, NY, USA: ACM, 2012. p. 15–20. ISBN 978-1-4503-1803-7.

TISO, A.; REGGIO, G.; LEOTTA, M. A Method for Testing Model to Text Transformations. In: BAUDRY, B.; DINGEL, J.; LUCIO, L.; VANGHELUWE, H. (Ed.). *Proceedings of the 2nd Workshop on the Analysis of Model Transformations (AMT'13) with the ACM/IEEE 16th International Conference on Model Driven Engineering Languages and Systems (MODELS'13)*. Miami, Florida, USA: CEUR-WS, 2013. v. 1077.

TVEIT, M. S. Specification of Graphical Representations- using hypergraphs or metamodels? In: *Proceedings of the Norsk Informatik konferanse*. Kristiansand, Norway: Universitetet i Agder, 2008. (NIK' 2008).

VALLECILLO, A.; GOGOLLA, M.; NO, L. B.; WIMMER, M.; HAMANN, L. Formal Specification and Testing of Model Transformations. In: BERNARDO, M.; CORTELLESSA, V.; PIERANTONI, A. (Ed.). *Formal Methods for Model-Driven Engineering*: 12th International School on Formal Methods for the Design of Computer, Communication, and Software Systems (SFM'12). Berlin, Heidelberg, Germany: Springer-Verlag, 2012. v. 7320, p. 399–437.

VERGILIO, S. R.; MALDONADO, J. C.; JINO, M. Uma estratégia para geraÃ§Ã£o de dados de teste. In: *VII Simpósio Brasileiro de Engenharia de Software*. Parana - PR: PUC, 1993. (SBES '93), p. 306–319. ISBN 2175-9677.

WANG, J.; KIM, S.-K.; CARRINGTON, D. Verifying metamodel coverage of model transformations. In: *Proceedings of the 17th Australian Conference on Software Engineering (ASWEC'06)*. Sydney, Australia: IEEE Computer Society, 2006. p. 10 pp. 1–10. ISSN 1530-0803.

WANG, J.; KIM, S.-K.; CARRINGTON, D. Automatic Generation of Test Models for Model Transformations. In: *Proceedings of the 19th Australian Conference on Software Engineering (ASWEC'08)*. Perth, Australia: IEEE Computer Society, 2008. p. 432–440. ISSN 1530-0803.

WEGENER, J.; BARESEL, A.; STHAMER, H. Evolutionary test environment for automatic structural testing. *Information and Software Technology*, Elsevier B.V., Berlin, Germany, v. 43, n. 14, p. 841–854, 2001.

WEI, H.; RUILIAN, Z.; QUNXIONG, Z. Integrating Evolutionary Testing with Reinforcement Learning for Automated Test Generation of Object-Oriented Software. *Chinese Journal of Electronics*, House of Electronics Industry, IET Digital Library, v. 24, n. CJE-1, p. 38–45, 2015. ISSN 1022-4653.

WEYUKER, E. J. Using Failure Cost Information for Testing and Reliability Assessment. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, ACM Press, New York, NY, USA, v. 5, n. 2, p. 87–98, 1996.

WHITE, L. J.; COHEN, E. I. A Domain Strategy for Computer Program Testing. *IEEE Transactions on Software Engineering*, IEEE Computer Society, Washington, DC, USA, v. 6, n. 3, p. 247–257, 1980.

WHITTLE, J.; GAJANOVIC, B. Model transformations should be more than just model generators. In: BRUEL, J.-M. (Ed.). *Proceedings of the 8th International Conference on Model Driven Engineering Languages and Systems (MoDELS'05)*. Montego Bay, Jamaica: Springer-Verlag, 2005, (Lecture Notes in Computer Science, v. 3844).

WIMMER, M.; BURGUEO, L. Testing M2T/T2M Transformations. In: MOREIRA, A.; SCHUTZ, B.; GRAY, J.; VALLECILLO, A.; CLARKE, P. (Ed.). *Model-Driven Engineering Languages and Systems*. Berlin, Heidelberg, Germany: Springer-Verlag, 2013, (Lecture Notes in Computer Science, v. 8107). p. 203–219. ISBN 978-3-642-41532-6.

WOHLIN, C.; RUNESON, P.; HöST, M.; OHLSSON, M. C.; REGNELL, B.; WESSLÉN, A. *Experimentation in Software Engineering: An Introduction*. Norwell, MA, USA: Kluwer Academic Publishers, 2000. ISBN 0-7923-8682-5.

ZHAO, C.; ZHANG, K. Transformational Approaches to Model Driven Architecture - A Review. In: *Proceedings of the 31st IEEE Software Engineering Workshop (SEW'07)*. Washington, DC, USA: IEEE Computer Society, 2007. p. 67–74. ISSN 1550-6215.

Apendice A

Estudo de Caso 01: Aplicação Web Completa - Autoria de Contéudo para Web

A.1 Análise e Interpretações

A cobertura dos templates pelos critérios estruturais definidos

Tabela A.1: Sumarização da Complementaridade dos Casos de Testes pelo Critério de Cobertura por Caracteres

ID	Template	Critérios de Cobertura		
		Caracteres		
		Exercitado	Total	%
1	main.jet	15	15	100,00
Grupo 01 - Diretório Navigation				
2	generatePages.jet	49	49	100,00
3	/pages/userContent.jet	1024	1060	96,60
4	/pages/uploadUserCommentForm.jsp.jet	1068	1104	96,74
5	/pages/page.jsp.jet	1170	1228	95,28
6	/pages/components.jet	5126	5228	98,05
Grupo 02 - Diretório UserContent				
7	generateUserContent.jet	66	102	64,71
8	/files/UserContentDAOAbstractFactory.java.jet	145	145	100,00
9	/files/UserContentDAO.java.jet	572	575	99,48
10	/files/UserContentActions.java.jet	2452	2452	100,00
11	/files/UserContent.java.jet	1252	1252	100,00
12	/files/moderation.jsp.jet	1537	1573	97,71
13	/files/DerbyUserContentDAOFactory.java.jet	646	646	100,00
14	/files/DerbyUserContentDAO.java.jet	5671	5825	97,36
15	/files/DefaultUserContentDAOFactoryProvider.java.jet	521	521	100,00
Grupo 03 - Diretório Persistence				
16	/actions/generateActions.jet	59	350	16,86
17	/actions/classes/ActionClass.java.jet	9890	10664	92,74
18	/beans/generateBeans.jet	31	98	31,63
19	/beans/classes/BeanClass.java.jet	2090	3397	61,52
20	/common/common.jet	345	3166	10,90
21	/daos/generateDAOs.jet	58	475	12,21
22	/daos/classes/DefaultDAOFactoryProvider.java.jet	N/A	N/A	N/A
23	/daos/classes/DAOAbstractFactory.java.jet	312	312	100,00
24	/daos/classes/DAOAbstractClass.java.jet	1141	1141	100,00
25	/daos/derby/generateDerbyDAOs.jet	68	536	12,69
26	/daos/derby/classes/DerbyDAOFactory.java.jet	903	903	100,00
27	/daos/derby/classes/DerbyDAOClass.java.jet	7508	9730	77,16
28	/daos/derby/sql/generateDerbyScript.jet	8	8	100,00
29	/daos/derby/sql/createDerby.sql.jet	677	707	95,76
30	/forms/generateForms.jet	102	181	56,35
31	/forms/pages/listpage.jsp.jet	3001	3021	99,34
32	/forms/pages/editpage.jsp.jet	4477	5418	82,63
33	/forms/pages/jspf/adminSuffix.jspf.jet	20	20	100,00
34	/forms/pages/jspf/adminPrefix.jspf.jet	653	689	94,78
35	/resources/generateResources.jet	18	95	18,95
36	/resources/files/messages.properties.jet	1558	1558	100,00

Tabela A.2: Sumarização da Complementaridade dos Casos de Testes pelo Critério de Cobertura por Marker´s Exercitados pela Transformação

ID	Template	Critérios de Cobertura		
		Marker's		
		Exercitado	Total	%
1	main.jet	N/A	N/A	N/A
Grupo 01 - Diretório Navigation				
2	generatePages.jet	N/A	N/A	100,00
3	/pages/userContent.jet	N/A	N/A	N/A
4	/pages/uploadUserCommentForm.jsp.jet	N/A	N/A	N/A
5	/pages/page.jsp.jet	01	01	100,00
6	/pages/components.jet	21	21	100,00
Grupo 02 - Diretório UserContent				
7	generateUserContent.jet	N/A	N/A	N/A
8	/files/UserContentDAOAbstractFactory.java.jet	N/A	N/A	N/A
9	/files/UserContentDAO.java.jet	N/A	N/A	N/A
10	/files/UserContentActions.java.jet	01	01	100,00
11	/files/UserContent.java.jet	N/A	N/A	N/A
12	/files/moderation.jsp.jet	N/A	N/A	N/A
13	/files/DerbyUserContentDAOFactory.java.jet	N/A	N/A	N/A
14	/files/DerbyUserContentDAO.java.jet	N/A	N/A	N/A
15	/files/DefaultUserContentDAOFactoryProvider.java.jet	N/A	N/A	N/A
Grupo 03 - Diretório Persistence				
16	/actions/generateActions.jet	03	03	100,00
17	/actions/classes/ActionClass.java.jet	71	76	93,42
18	/beans/generateBeans.jet	N/A	N/A	N/A
19	/beans/classes/BeanClass.java.jet	13	13	100,00
20	/common/common.jet	29	29	100,00
21	/daos/generateDAOs.jet	N/A	N/A	N/A
22	/daos/classes/DefaultDAOFactoryProvider.java.jet	N/A	N/A	N/A
23	/daos/classes/DAOAbstractFactory.java.jet	04	04	100,00
24	/daos/classes/DAOAbstractClass.java.jet	09	09	100,00
25	/daos/derby/generateDerbyDAOs.jet	03	03	100,00
26	/daos/derby/classes/DerbyDAOFactory.java.jet	07	07	100,00
27	/daos/derby/classes/DerbyDAOClass.java.jet	51	51	100,00
28	/daos/derby/sql/generateDerbyScript.jet	N/A	N/A	N/A
29	/daos/derby/sql/createDerby.sql.jet	06	07	85,71
30	/forms/generateForms.jet	N/A	N/A	N/A
31	/forms/pages/listpage.jsp.jet	19	24	79,17
32	/forms/pages/editpage.jsp.jet	28	36	77,78
33	/forms/pages/jspf/adminSuffix.jspf.jet	N/A	N/A	N/A
34	/forms/pages/jspf/adminPrefix.jspf.jet	01	01	100,00
35	/resources/generateResources.jet	N/A	N/A	N/A
36	/resources/files/messages.properties.jet	11	11	100,00

Tabela A.3: Sumarização da Complementaridade dos Casos de Testes pelo Critério de Cobertura por Linhas Exercitadas pela Transformação

ID	Template	Critérios de Cobertura		
		Linhas		
		Exercitado	Total	%
1	main.jet	10	13	76,92
Grupo 01 - Diretório Navigation				
2	generatePages.jet	06	10	60,00
3	/pages/userContent.jet	01	20	5,00
4	/pages/uploadUserCommentForm.jsp.jet	01	28	3,57
5	/pages/page.jsp.jet	22	72	30,56
6	/pages/components.jet	44	137	32,12
Grupo 02 - Diretório UserContent				
7	generateUserContent.jet	16	27	59,26
8	/files/UserContentDAOAbstractFactory.java.jet	01	06	16,67
9	/files/UserContentDAO.java.jet	01	15	6,67
10	/files/UserContentActions.java.jet	04	71	5,63
11	/files/UserContent.java.jet	01	65	1,54
12	/files/moderation.jsp.jet	01	35	2,86
13	/files/DerbyUserContentDAOFactory.java.jet	01	20	5,00
14	/files/DerbyUserContentDAO.java.jet	04	181	2,21
15	/files/DefaultUserContentDAOFactoryProvider.java.jet	01	14	7,14
Grupo 03 - Diretório Persistence				
16	/actions/generateActions.jet	08	16	50,00
17	/actions/classes/ActionClass.java.jet	91	172	52,91
18	/beans/generateBeans.jet	05	12	41,67
19	/beans/classes/BeanClass.java.jet	17	75	22,67
20	/common/common.jet	65	110	59,09
21	/daos/generateDAOs.jet	10	41	24,39
22	/daos/classes/DefaultDAOFactoryProvider.java.jet	N/A	N/A	N/A
23	/daos/classes/DAOAbstractFactory.java.jet	06	11	54,55
24	/daos/classes/DAOAbstractClass.java.jet	09	15	60,00
25	/daos/derby/generateDerbyDAOs.jet	16	34	47,06
26	/daos/derby/classes/DerbyDAOFactory.java.jet	09	24	37,50
27	/daos/derby/classes/DerbyDAOClass.java.jet	71	207	34,30
28	/daos/derby/sql/generateDerbyScript.jet	03	05	60,00
29	/daos/derby/sql/createDerby.sql.jet	16	37	43,24
30	/forms/generateForms.jet	08	15	53,33
31	/forms/pages/listpage.jsp.jet	26	57	45,61
32	/forms/pages/editpage.jsp.jet	44	116	37,93
33	/forms/pages/jspf/adminSuffix.jspf.jet	01	03	33,33
34	/forms/pages/jspf/adminPrefix.jspf.jet	04	21	19,05
35	/resources/generateResources.jet	04	10	40,00
36	/resources/files/messages.properties.jet	20	35	57,14

Tabela A.4: Sumarização da Complementaridade dos Casos de Testes pelo Critério de Cobertura por Comandos Exercitadas pela Transformação

ID	Template	Critérios de Cobertura		
		Comandos		
		Exercitado	Total	%
1	main.jet	N/A	N/A	N/A
Grupo 01 - Diretório Navigation				
2	generatePages.jet	02	02	100,00
3	/pages/userContent.jet	N/A	N/A	N/A
4	/pages/uploadUserCommentForm.jsp.jet	01	01	100,00
5	/pages/page.jsp.jet	06	06	100,00
6	/pages/components.jet	22	22	100,00
Grupo 02 - Diretório UserContent				
7	generateUserContent.jet	02	02	100,00
8	/files/UserContentDAOAbstractFactory.java.jet	N/A	N/A	N/A
9	/files/UserContentDAO.java.jet	01	01	100,00
10	/files/UserContentActions.java.jet	03	03	100,00
11	/files/UserContent.java.jet	N/A	N/A	N/A
12	/files/moderation.jsp.jet	N/A	N/A	N/A
13	/files/DerbyUserContentDAOFactory.java.jet	N/A	N/A	N/A
14	/files/DerbyUserContentDAO.java.jet	04	04	100,00
15	/files/DefaultUserContentDAOFactoryProvider.java.jet	N/A	N/A	N/A
Grupo 03 - Diretório Persistence				
16	/actions/generateActions.jet	01	01	100,00
17	/actions/classes/ActionClass.java.jet	18	24	75,00
18	/beans/generateBeans.jet	01	01	100,00
19	/beans/classes/BeanClass.java.jet	04	04	100,00
20	/common/common.jet	16	16	100,00
21	/daos/generateDAOs.jet	01	01	100,00
22	/daos/classes/DefaultDAOFactoryProvider.java.jet	N/A	N/A	N/A
23	/daos/classes/DAOAbstractFactory.java.jet	02	02	100,00
24	/daos/classes/DAOAbstractClass.java.jet	N/A	N/A	N/A
25	/daos/derby/generateDerbyDAOs.jet	02	02	100,00
26	/daos/derby/classes/DerbyDAOFactory.java.jet	02	02	100,00
27	/daos/derby/classes/DerbyDAOClass.java.jet	21	21	100,00
28	/daos/derby/sql/generateDerbyScript.jet	N/A	N/A	N/A
29	/daos/derby/sql/createDerby.sql.jet	13	13	100,00
30	/forms/generateForms.jet	01	01	100,00
31	/forms/pages/listpage.jsp.jet	09	09	100,00
32	/forms/pages/editpage.jsp.jet	17	19	89,47
33	/forms/pages/jspf/adminSuffix.jspf.jet	N/A	N/A	N/A
34	/forms/pages/jspf/adminPrefix.jspf.jet	03	03	100,00
35	/resources/generateResources.jet	N/A	N/A	N/A
36	/resources/files/messages.properties.jet	03	03	100,00