Universidade Federal da Bahia
Universidade Salvador
Universidade Estadual de Feira de Santana

TESE DE DOUTORADO

Sistematizando o Desenvolvimento de Transformações Modelo a Modelo em uma Abordagem Dirigida a Modelos

Ana Patrícia Fontes Magalhães Mascarenhas

Programa Multiinstitucional de Pós-Graduação em Ciência da Computação - PMCC

Salvador
Agosto de 2016

ANA PATRÍCIA FONTES MAGALHÃES MASCARENHAS

SISTEMATIZANDO O DESENVOLVIMENTO DE TRANSFORMAÇÕES MODELO A MODELO EM UMA ABORDAGEM DIRIGIDA A MODELOS

Esta Tese de Doutorado foi apresentada ao Programa Multiinstitucional de Pós-Graduação em Ciência da Computação da Universidade Federal da Bahia, Universidade Estadual de Feira de Santana e Universidade Salvador, como requisito parcial para obtenção do grau de Doutor em Ciência da Computação.

Orientadora: Aline Maria Santos Andrade
Co-orientadora: Rita Suzana Pitangueira Maciel

Salvador
Agosto de 2016

Ficha catalográfica.

MAGALHÃES, Ana Patrícia

Sistematizando o Desenvolvimento de Transformações Modelo a Modelo em uma Abordagem Dirigida a Modelos/ Ana Patrícia Fontes Magalhães Mascarenhas– Salvador, 04/08/2016.

188p.: il.

Orientadora: Aline Maria Santos Andrade.
Co-orientadora: Rita Suzana Pitangueira Maciel.
Tese (Doutorado - Doutorado Multiinstitucional em Ciência da Computação)– UNIVERSIDADE FEDERAL DA BAHIA, INSTITUTO DE MATEMÁTICA, 04/08/2016.

1. Transformações de Modelos. 2. Framework para Desenvolvimento de Transformações de Modelos. 3. Processo de Desenvolvimento de Transformação. 4. Perfil para Transformação.
I. ANDRADE, Aline. II. MACIEL, Rita Suzana Pitangueira.
III. UNIVERSIDADE FEDERAL DA BAHIA. INSTITUTO DE MATEMÁTICA. IV. Título.

TERMO DE APROVAÇÃO

ANA PATRÍCIA FONTES MAGALHÃES MASCARENHAS

SISTEMATIZANDO O DESENVOLVIMENTO DE TRANSFORMAÇÕES MODELO A MODELO EM UMA ABORDAGEM DIRIGIDA A MODELOS

Esta Tese de Doutorado foi julgada adequada à obtenção do título de Doutor em Ciência da Computação e aprovada em sua forma final pelo Multiinstitucional de Pós-Graduação em Ciência da Computação - PMCC da Universidade Federal da Bahia, Universidade Estadual de Feira de Santana e Universidade Salvador.

Salvador, 04 de Agosto de 2016

Profa. Dr. Franklin de Souza Ramalho
Universidade Federal de Campina Grande

Prof. Dr. Toacy Cavalcanti de Oliveira
Universidade Federal do Rio de Janeiro

Profa. Dr. Claudio Nogueira Sant'ana
Universidade Federal da Bahia

Prof. Dr. Sérgio Gorender
Universidade Federal da Bahia

Profa. Dra. Aline Maria Santos Andrade
Universidade Federal da Bahia

AGRADECIMENTOS

O desenvolvimento desta tese de doutorado certamente foi uma das experiências mais desafiadoras da minha vida. Não resta dúvida de que o sucesso por mim alcançado se deve em grande parte à dedicação e ao apoio de professores, familiares e amigos que caminharam comigo ao longo desta jornada. Sendo assim, gostaria de agradecer a todos que de alguma forma contribuiram com este trabalho.

A meu marido, Erico, e filhos, Felipe e Juliana, pela paciência e comprensão que tiveram nos momentos em que não pude estar presente em suas vidas, sempre me incentivando com amor a enfrentar as dificuldade e seguir em frente.

A meus pais por estarem sempre presentes e em especial pelo apoio e amor que deram aos meus filhos quando em muitos momentos precisei me ausentar para me dedicar ao trabalho.

A minha orientadora, Aline, que me acompanha desde o mestrado sempre acreditando no meu potencial, agradeço pelas ricas discussões sobre cada tema desta pesquisa, pelas perguntas que me deixavam sem resposta e muitas vezes me levaram a repensar o rumo do trabalho, pelas minunciosas revisões realizadas nos textos que elevaram imensamente a qualidade do trabalho, por estar sempre disponível para me atender até mesmo em sua casa, por ser uma pessoa do bem e um exemplo no qual tento me espelhar para orientar meus alunos, mas principalmente pela amizade que se consolidou durante todos esses anos.

A minha amiga e co-orientadora, Rita Suzana, inicialmente por me incentivar a seguir a carreira acadêmica ainda quando ingressei no mestrado, e agora, no doutorado, pelas contribuições que deu ao meu trabalho, em especial por ter insistido no desenvolvimento de um longo processo de experimentação importante para evidenciar os resultados alcançados e principalmente pelo incentivo nos vários momentos de dúvida.

Aos colegas e alunos que me ajudaram participando do estudo de caso e experimento, por terem dedicado seu precioso tempo ao meu trabalho. Sem a contribuição de vocês não teríamos evidenciado os resultados.

Aos funcionários do programa, em especial a Davilene, por estar sempre disposta a me ajudar quando precisei.

Aos professores que participaram da minha banca, Franklin, Toacy, Sergio e Claudio, pelas valiosas sugestões de melhorias ao trabalho.

A Cristiane, professora de estatistica, pela ajuda na análise dos resultados da avaliação.

A Jorge Amaro, pela ajuda com as normas da ISO.

As minhas amigas que sempre me incentivaram a continuar nos momentos em que pensei que não teria sucesso.

RESUMO

No contexto do Desenvolvimento Dirigido a Modelos (DDM), transformações de modelos são softwares que recebem modelos de entrada e geram modelos de saída de acordo com um conjunto de regras de transformações que especificam como modelos escritos em linguagens fonte são transformados em modelos escritos em linguagens alvo. A especificação de uma transformação é feita entre metamodelos das linguagens de modelagem fonte e alvo, que definem domínios de aplicação, tal que qualquer transformação entre modelos que são instâncias dos metamodelos envolvidos seja gerada.

A especificação, projeto e codificação de transformações detém uma complexidade inerente ao domínio de transformações, adicionalmente à complexidade do desenvolvimento de software em geral. Logo, um processo que trate de todas as fases do desenvolvimento de transformações é de real importância para facilitar este desenvolvimento. No entanto, trabalhos atuais que tratam da construção de transformações abordam aspectos, específicos como o projeto e a implementação, sem se ater em um processo completo de desenvolvimento. Um processo DDM pode ser utilizado neste contexto trazendo as vantagens desta abordagem ao desenvolvimento de transformações de modelos. Neste sentido, uma transformação pode também ser gerada através de transformações de modelos e uma linguagem específica deste domínio é requerida. Muitos dos trabalhos encontrados na literatura seguem nesta direção bem como a nossa proposta. Considerando estes aspectos esta tese propõe um framework chamado MDTD (*Model Driven Transformation Development*), na abordagem dirigida a modelos, com um perfil UML para modelagem de transformações e um processo de desenvolvimento de transformações que considera todo o seu ciclo de vida.

O framework MDTD sistematiza a construção de transformações através de um processo iterativo e incremental que conduz o desenvolvimento da transformação desde a especificação dos requisitos até à codificação da transformação, em que modelos de transformação de modelos são construídos em alto nível de abstração e transformados de forma (semi) automática em modelos menos abstratos até a geração do código da transformação. Com este framework, foi possível (semi) automatizar o processo por uma cadeia de transformações que gera modelos de transformações nos diversos níveis de abstração até o código nas linguagens ATL e QVT, que são específicas para programação de transformações, além de poder ser executado em ambiente Eclipse sem demandar o uso de ferramentas proprietárias.

O framework foi avaliado através de estudo de caso e experimento controlado e os resultados evidenciaram que pessoas com diferentes níveis de conhecimento em DDM e sem experiência em linguagens de transformação desenvolveram transformações através do framework MDTD e tiveram o código executável gerado, evidenciando assim a eficácia da proposta.

Mostramos com esse trabalho que o desenvolvimento de transformações de modelos pode ser facilitado através do desenvolvimento dirigido a modelos e, consequentemente, acreditamos que este é um passo importante para uma possível expansão do uso da DDM na indústria de software.

Palavras-chave: framework de desenvolvimento de transformação, processo de desenvolvimento de transformação, linguagem de modelagem para o domínio de transformação de modelos, perfil UML para transformação.

ABSTRACT

In the context of Model Driven Development (MDD), model transformations are softwares which receive models as input and generate models as output according to a set of transformations that specify how models written in source languages are transformed in models written in target languages. The specification of a transformation is defined between metamodels of source and target languages, which define application domains, such as any transformation between models, instances of the involved metamodels, are generated.

The specification, design and codification of transformations hold a complexity inherent of the transformation domain, additionally to the complexity of software development in general. Therefore, a process that addresses all phases of the transformation development is really important in order to facilitate this development. However, current works dealing with the construction of transformations address specific issues, such as design and implementation, without concerning the complete development process. A MDD process can be used on this context to take advantages of this approach to the development of model transformation. In this sense, a transformation can also be generated through model transformations and specific languages are required. For this many works found on literature follow this direction as well as our proposal. Considering these aspects this thesis proposes a framework, named MDTD (Model Driven Transformation Development), on model driven approach, with a UML profile, for transformation modeling, and a transformation development process that covers the entire development life cycle.

The framework MDTD systematizes the construction of transformations through an iterative and incremental process which guides the transformation development since requirement specification to transformation codification, where model transformation models are constructed in high abstraction level and (semi) automatically transformed into models in low abstraction level until transformation code. With this framework, we could automatize the process by a transformation chain, that generates transformation models on different levels of abstraction until code in ATL and QVT language, which are program languages specific for transformations. In addition it can be executed in Eclipse environment, without requiring the use of proprietary tools.

The framework were evaluated through a case study and a controlled experiment which results showed that people, with different levels of knowledge on MDD or without experience on transformation languages, developed transformations through the Framework MDTD and generated an executable code, emphasizing the effectiveness of the proposal.

We showed with this work that the development of model transformations can be facilitated through the model driven development and, as a consequence, we believe that this is an important step towards the expansion of the use of MDD in software industry.

Keywords: framework for transformation development, transformation development process, modeling languages for model transformation domain, UML profile for transformation.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

Capítulo 1

INTRODUÇÃO

Processos cognitivos são frequentemente utilizados pela mente humana para representação da realidade sob uma perspectiva específica, e dentre estes processos está o de abstração. No campo científico a abstração está associada ao conceito de modelagem, modelos como uma representação simplificada ou parcial da realidade, e é largamente utilizada para generalizar características específicas de objetos reais, classificar objetos de maneira coerente e representar objetos complexos através da agregação de objetos mais simples, facilitando a documentação, a comunicação e a compreensão de problemas reais (BRAMBILLA; CABOT; WIMMER, 2012).

Na ciência da computação é notório o aumento da complexidade dos sistemas seja por fatores técnicos, como a heterogeneidade de plataformas, ou por fatores não técnicos, como a própria complexidade dos domínios hoje automatizados pelos sistemas de informação. Mecanismos de abstração têm sido adotados para lidar com essa complexidade e dentre eles destaca-se as linguagens de modelagem. Modelos são utilizados para enfatizar características relevantes de um domínio, projetar a arquitetura de sistemas, expressar o comportamento, dentre outros aspectos que envolvem o desenvolvimento de aplicações computacionais.

O uso de modelos no desenvolvimento de software fez surgir diversas linguagens de modelagem nas últimas décadas que culminaram com a padronização em 1997, pela *Object Management Group* (OMG), da Linguagem Unificada de Modelagem (*Unified Modeling Language* - UML) (BOOCH; RUMBAUGH; JACOBSON, 2006) e do *Meta Object Facility* (MOF)[1], padrão para a definição de linguagens de modelagem. Contudo, embora os modelos tenham trazido significante contribuição, a tradução manual desses modelos em código ainda é um passo utilizado no desenvolvimento. Como consequência, evoluções para novas plataformas requerem refazer a implementação do software, o que frequentemente torna os modelos obsoletos, pois os sistemas são modificados diretamente no código.

[1]Meta Object Facility. Versão 2.5 disponível em //www.omg.org/spec/MOF/2.5

Dentro do contexto de uso de modelos no desenvolvimento de sistemas está inserido o Desenvolvimento Dirigido a Modelos (DDM), em que os modelos são considerados os principais artefatos do desenvolvimento. Nesta abordagem modelos em alto nível de abstração são convertidos para níveis mais baixos, através de uma cadeia de transformações até gerar o código fonte da aplicação. Desta forma, os modelos deixam de ser vistos apenas como documentação e passam a ser especificados em uma linguagem com sintática e semântica bem definida, que possibilita, dentre outras coisas, realizar validação sintática, simulações, verificações e principalmente processá-los para gerar outros modelos ou código. A realização mais conhecida do DDM, padronizada pela OMG em 2001, é a Arquitetura Dirigida a Modelos (MDA – *Model Driven Architecture*)[2] que estabelece níveis de abstração para o desenvolvimento e padroniza o uso de tecnologias em direção à construção de software interoperáveis e portáveis.

Na abordagem DDM dois elementos são essenciais, os modelos, que representam um sistema em diversos níveis de abstração, e a cadeia de transformação, reponsável pelo processamento dos modelos até a geração do código. (BRAMBILLA; CABOT; WIMMER, 2012), (STAHL; VOLTER, 2010).

1.1 CONTEXTUALIZAÇÃO DO PROBLEMA

A estratégia de adoção da DDM por uma organização pode envolver um ambiente que já ofereça um processo pré definido de desenvolvimento na abordagem DDM ou a customização dos processos internos de desenvolvimento de software da organização. Na primeira opção, engenhos de transformação podem ser utilizados com transformações prontas, mas requerem que a organização se adapte ao processo de desenvolvimento proposto pelo ambiente. A outra opção consiste em adaptar os processos já utilizados pela empresa para a abordagem DDM, estratégia que vem sendo adotada para introduzir a DDM de maneira gradual nas organizações (HUTCHINSON; WHITTLE, 2011). No cerne desta segunda opção está a construção de uma cadeia de transformação para automatizar o processo de desenvolvimento de software da organização até a geração do código.

A construção de transformações tem se mostrado um desafio para a adoção da DDM pelas organizações, pois detém uma complexidade inerente ao domínio de transformações, adicionalmente à complexidade do desenvolvimento de software em geral. Transformações processam modelos escritos de acordo com linguagens de modelagens e são especificadas de acordo com os metamodelos dessas linguagens, que definem domínios de aplicação. O desenvolvimento de transformações, portanto, envolve a utilização (e muitas vezes a criação) de metamodelos, e consequentemente, conhecimento em técnicas de metamodelagem, e o conhecimento do domínio da aplicação para definir os mapeamentos entre os elementos dos metamodelos envolvidos.

Para ilustrar a complexidade que envolve o desenvolvimento de transformações vamos analisar um exemplo. Considere que queremos construir uma transformação para criar um livro a partir de um conjunto de artigos publicados em um congresso, conforme ilustrado na Figura 1.1. Desta forma, dado um modelo que represente um congresso específico (ex. CBSOFT) e seus diversos artigos, queremos gerar outro modelo que represente

[2]MDA Guide. Version 1.0.1 (omg/2003-06-01)

um livro cujos capítulos são os artigos do congresso. Para que o modelo de entrada, que representa um congresso, seja processado pela nossa transformação ele precisa ser implementado em uma linguagem de modelagem com sintaxe e semântica bem definida, geralmente especificada sob a forma de metamodelos. Analogamente, a transformação deverá gerar como saída um modelo que represente um livro que é uma instância de um metamodelo nesse domínio. Esses dois metamodelos, na figura chamados de *Congresso* e *Livro*, caso não existam, precisam ser inicialmente especificados e implementados. A implementação de metamodelos envolve também o uso de metametalinguagens. Com base nos metamodelos, a transformação (*Congresso2Livro*) é então definida e deverá ser capaz de processar qualquer modelo instância do metamodelo *Congresso* para gerar um modelo intância do metamodelo *Livro* como saída.

A construção da transformação envolve analisar detalhadamente cada elemento que compõe os metamodelos envolvidos para definir um conjunto de regras de mapeamento entre eles (em cinza na Figura 1.1), levando em consideração questões, tais como: quais elementos do metamodelo *Congresso* devem ser mapeados em elementos no metamodelo *Livro*? Como esses elementos deverão ser gerados, por exemplo, quando um livro for criado, qual será o nome deste livro e a editora? como manter a integridade do modelo que será gerado em relação às regras de boa formação do seu metamodelo, por exemplo, o metamodelo *Livro* considera que os autores estão associados a um capítulo ou ao livro todo?

Em geral os desenvolvedores de software estão habituados a trabalhar com modelos de aplicações, mas não com metamodelos e metametamodelos, que tratam de definições em níveis de abstração bem mais elevados e podem aumentar a complexidade do desenvolvimento.

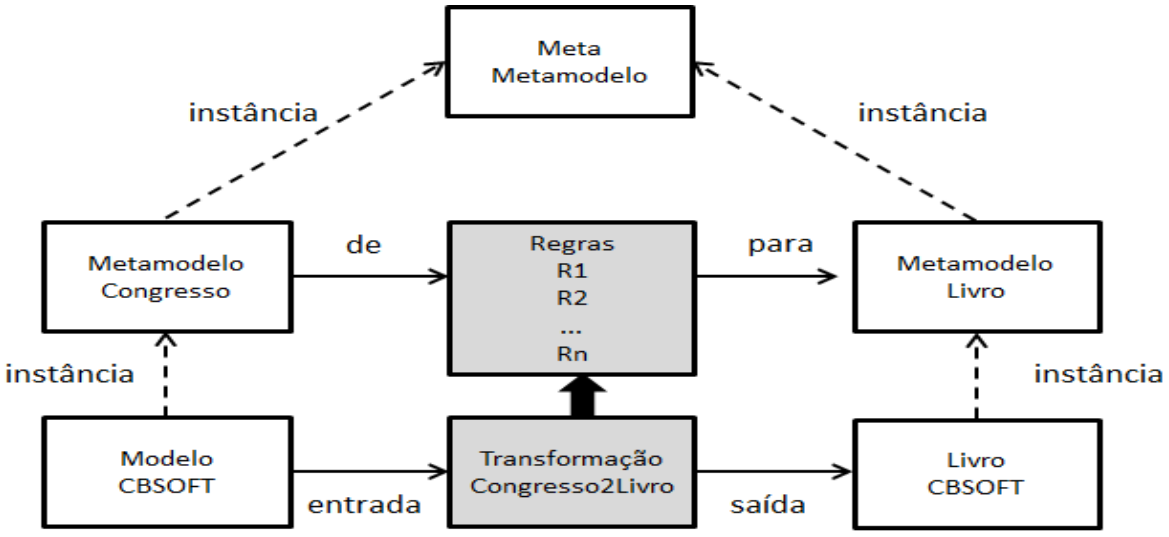


Figura 1.1 Esquema da transformação Congresso2Livro com os modelos e metamodelos envolvidos

Além dessas questões técnicas, o desenvolvimento de transformações envolve tecnologias que nem sempre estão inseridas no contexto de desenvolvimento de software conven-

cional e que em geral não são de domínio da comunidade de desenvolvimento de sistemas. Metametalinguagens, como por exemplo o MOF, são utilizadas na especificação de metamodelos. Já para o desenvolvimento de transformações, embora seja possível o uso de linguagens convencionais como Java, existem linguagens específicas para transformação, a exemplo de QVT (*Query/View/Transformation*)[3] e de ATL (*Atlas Transformation Language*)[4], que já oferecem recursos para leitura de modelos, identificação de elementos no modelo, mapeamento entre metamodelos, dentre outros. O uso destas linguagens também requer ambientes de modelagem específicos e ferramentas para implementação e execução de transformações além de estratégias para integração entre essas ferramentas, que possibilitem, por exemplo, que um modelo criado em um ambiente de modelagem possa ser lido e processado pela ferramenta que executa a transformação.

Esse cenário se agrava quando transformações são especificadas de forma *ad hoc*, usando linguagem natural, e implementadas diretamente em código (GUERRA et al., 2010a), (SANI; POLACK; PAIGE, 2011), ou seja, quando o código fonte da transformação é escrito manualmente pelo programador sem nenhum planejamento prévio. Esta conduta dificulta a adoção de técnicas, a exemplo dos padrões de projeto e reuso, que contribuem para um bom desenvolvimento, além de prover uma documentação deficiente que pode dificultar também a manutenção e a evolução da transformação (BOLLATI et al., 2013).

Adicionalmente aos aspectos técnicos e tecnológicos que envolvem o desenvolvimento da transformação, observa-se também um aumento de complexidade nos sistemas de informação atuais que é refletido diretamente no software de transformação utilizado para construir esses sistemas. Como consequência, as transformações tem se tornado softwares mais robustos e é eminente que seu desenvolvimento contemple níveis mais altos de abstração para tratar não só a implementação, mas também a especificação, projeto e teste desses softwares.

Neste contexto insere-se o desenvolvimento de transformações utilizando a própria abordagem DDM, em que uma transformação é representada por modelos em diferentes níveis de abstrações que são transformados até a geração de código. Esta ideia decorre do conceito de modelo de transformações de modelos (*model transformation model*) proposto por (BEZIVIN et al., 2006). A visão de transformações como modelos implica em um ciclo de vida de desenvolvimento com diferentes fases, relacionadas à especificação, análise, projeto, implementação e testes, correspondentes à construção dos modelos de transformação de modelos em diversos níveis de abstração. Como consequência o desenvolvimento demanda o uso de uma notação apropriada e ferramentas automatizadas. Além disso, assim como no desenvolvimento de outros tipos de software, processos que apoiem as diversas fases do desenvolvimento também se fazem importantes para guiar o desenvolvedor.

Para contribuir com o desenvolvimento de transformações, trabalhos vem sendo propostos relacionados a abordagens para sistematizar o desenvolvimento (KUSTER; RYNDUNA; HAUSER, 2005) (SIIKARLA et al., 2008) (LI; YIN, 2010), linguagens específicas

[3]QVT specification - http://www.omg.org/spec/QVT/1.0/PDF/
[4]ATL Project - http://www.eclipse.org/m2m/atl/

de modelagem (RAHIM; MANSOOR, 2008), (GUERRA et al., 2010a), ferramentas de apoio (BOLLATI et al., 2013), (AVAZPOUR; GRUNDY; GRUNSKE, 2015), entre outros (discutidos no Capítulo 8 desta tese). Esses trabalhos, em sua maioria, abordam fases específicas do desenvolvimento, principalmente o projeto e a construção da transformação, não contemplando todo o ciclo de vida do software. Particularmente, embora organizem as atividades envolvidas no desenvolvimento, até o momento da escrita deste texto não foi encontrado nenhum trabalho que especifique um processo para desenvenvolvimento de transformações de modelos em uma linguagem de modelagem de processos (*Process Modeling Language* - PML), compreendendo elementos, tais como, fases, tarefas, artefatos, papéis, entre outros, importantes para guiar os desenvolvedores na construção de transformações.

O desenvolvimento de transformações, portanto, apresenta uma diversidade de propostas que cobrem aspectos específicos do desenvolvimento, mas carece de abordagens que especifiquem um processo contemplando as diversas fases do ciclo de vida integradas a uma linguagem de modelagem, automação e ambiente de desenvolvimento.

Diversos problemas estão relacionados ao desenvolvimento de transformações:

1. Ausência de processos para desenvolvimento de transformações de modelos especificados em linguagens de modelagem de processos (*Process Modeling Languages* - PML). Processos escritos de maneira *ad-hoc* podem ser incompletos e difíceis de serem seguidos. PMLs fornecem os conceitos necessários para especificar, documentar e apresentar os elementos apropriados a um processo de software de maneira padronizada. Como consequência, facilitam o entendimento das atividades, artefatos e demais elementos envolvidos e a relação entre eles, bem como a execução do processo pelos desenvolvedores.

2. Carência de abordagens que tratem de todo o ciclo de vida de desenvolvimento, pois as abordagens atuais, em sua maioria, tratam de fases específicas. Sendo assim, o processo de integração das abordagens fica a cargo do desenvolvedor, que precisa selecionar abordagens apropriadas a cada fase e integrá-las. Essa tarefa pode ser ainda mais difícil, pois nem sempre as abordagens usam as mesmas linguagens ou trabalham de maneira uniforme;

3. Aumento da complexidade dos sistemas é refletido na complexidade dos metamodelos usados para representá-los. Como consequência, a tarefa de definição das regras que compõem uma transformação também se torna mais complexa, uma vez que requer conhecimento aprofundado dos elementos que compõem o metamodelo e suas relações;

4. Falta de orientação das abordagens quanto à construção dos metamodelos envolvidos na transformação, atividade que, quando necessária, requer alta capacidade de abstração, conhecimento em técnicas de metamodelagem e conhecimento em metametalinguagens;

5. Uso, em muitos casos, de linguagens específicas para a implementação de transformações (ex. ATL, QVT e RubyTL), que não são de domínio comum da comuni-

dade de desenvolvimento de software, e precisam ser adquiridas pelos desenvolvedores;

6. Demanda de utilização de ferramentas, tais como engenhos de transformação, ferramentas de modelagem e linguagens para metamodelagem que não são comuns ao desenvolvimento tradicional de software. A seleção de quais ferramentas utilizar, a manipulação de diversas ferramentas concomitantemente e principalmente o intercâmbio de documentos entre essas ferramentas diminuem a produtividade, gera problemas de compatibilidade entre ferramentas e consequentemente contribuem para aumentar a complexidade do desenvolvimento.

Alguns desses problemas se potencializam quando o código fonte do sistema que representa a transformação é escrito manualmente pelo programador sem usar nenhuma técnica de análise e projeto, pois é difícil planejar e visualizar o que está sendo construído sem ter uma visão estrutural e comportamental da transformação como um todo, isto é, uma visão que separe aspectos relevantes da transformação (ex. mapeamentos entre metamodelos) das definições específicas de plataforma. Por exemplo, um módulo escrito na linguagem ATL contém diversas regras. À medida que cresce a quantidade de regras do módulo aumenta a complexidade para organizá-las de maneira que se possa identificar as dependências entre elas e os mapeamentos que elas representam. Consequentemente, torna-se mais difícil analisar o impacto de evoluções e manutenções das mesmas.

Diante do exposto, o principal problema tratado neste trabalho reside na carência de processos de apoio ao desenvolvimento de transformações, devidamente especificados em PMLs, que tratem de todo o ciclo de vida de desenvolvimento e integre linguagens de modelagem e recursos de automação adequados a cada uma das fases deste ciclo de vida.

1.2 CONTRIBUIÇÕES

Nesta tese propomos uma sistematização do desenvolvimento de transformações modelo a modelo com base na abordagem dirigida a modelos. A proposta está sintetizada em um framework, chamado *MDTD* (*Model Driven Transformation Development*), para desenvolvimento de transformações. O framework é composto de: processo de desenvolvimento, linguagem de modelagem para especificação de transformação, cadeia de transformação e ambiente de desenvolvimento.

O desenvolvimento de tranformações com o framework é guiado por um processo de desenvolvimento iterativo e incremental com base na abordagem DDM, chamado *MDTD-proc*. O processo está formalizado na PML SPEM[5] e cobre diversas fases do desenvolvimento de transformações, desde a especificação de requisitos até a geração do código. Desta forma, há uma mudança no foco do desenvolvimento de transformações, antes voltado para o código, passa a ser realizado através da construção de modelos em alto nível de abstração que são transformados, de maneira (semi-)automática, em modelos menos abstratos até serem utilizados para geração do código fonte da transformação.

A modelagem das transformações ao longo do processo de desenvolvimento é apoiada por uma linguagem de modelagem definida sob a forma de metamodelos e perfis

[5]Software Process Engineering Metamodel Specification, Version 2.0, (formal/08-04-01)

específicos para o domínio de transformações. Desta forma, modelos de transformação são construídos independente de linguagem de programação para depois serem transformados em linguangens específicas, favorecendo a abstração em detrimento de detalhes de implementação relacionados às linguagens de programação.

O desenvolvimento com o framework MDTD é automatizado por uma cadeia de transformações que processam modelos de transformações nos diversos níveis de abstração até a geração do código atualmente nas linguagens ATL e QVT. Desta forma, pode-se dizer que o framework utiliza uma cadeia de transformações para gerar transformações.

Para apoiar o desenvolvimento o framework utiliza um ambiente *open source* de desenvolvimento que integra diversas ferramentas já existentes, tais como, ferramenta de modelagem e engenhos de transformação. Desta forma a utilização da abordagem é independente de plataforma de desenvolvimento.

Para avaliar a eficácia do framework foi realizada uma avaliação do processo com base na norma ISO/IEC-15504 (ISO, 2005), através de um estudo de caso. Em seguida foi avaliada a qualidade da transformação construída com o processo em relação a transformações construídas diretamente no código através de um experimento controlado. Adicionalmente, a expressividade do perfil proposto para a especificação de transformações também foi avaliada através de estudo exploratório.

Cada um dos elementos que compõem o framework representa uma contribuição desta tese, mas consideramos a integração destes como essencial na busca pela diminuição da complexidade do desenvolvimento de transformações.

Em suma, as seguintes contribuições são alcançadas como resultado desta tese:

1. Redução do nível de complexidade do desenvolvimento das transformações através do framework MDTD;

2. Apoio ao desenvolvimento incremental de transformações. A adoção da DDM tem sido realizada de forma gradual pelas organizações através da automação de pequenas partes do processo de desenvolvimento. Nesta direção, o framework possibilita desenvolver transformações de forma iterativa e incremental, onde pequenas versões da transformação são construídas de acordo com as necessidades da organização, colaborando com essa tendência;

3. Um processo (semi) automatizado e formalizado na linguagem SPEM para desenvolver transformações de modelos que compreende desde a fase de levantamento de requisitos até a geração do código;

4. Um metamodelo e um perfil, independentes de plataforma, que possibilitam a especificação de transformações de modelos em diversos níveis de abstração, desde a especificação de requisitos;

5. Uma cadeia de transformação que automatiza o processo proposto com transformações que cobrem desde os requisitos até a geração de código nas linguagens ATL e QVT;

6. Um ambiente integrado, não proprietário, para o desenvolvimento de transformações seguindo o framework proposto;

7. Um guia para construção de metamodelos, artefato essencial no desenvolvimento de transformações;

8. Uma adaptação da norma ISO/IEC-15504 de avaliação de processo de software para o domínio de transformação de modelos;

9. Apoio à utilização de boas práticas de desenvolvimento de software, tais como, composição e reuso, que favorecem não só o aumento de produtividade e qualidade do desenvolvimento, mas também ajuda no planejamento de manutenções e evoluções da transformação.

1.3 ESCOPO DA TESE

O framework proposto neste trabalho se aplica ao desenvolvimento de transformações modelo-a-modelo unidirecionais. Foge ao escopo desta tese o desenvolvimento de transformações modelo-a-texto e /ou bidirecionais.

A linguagem de modelagem proposta para o framework utiliza a abordagem declarativa como mecanismo para especifição de transformações. Desta forma, instruções imperativas não são contempladas para a construção dos modelos de transformação independentes de plataforma, mas podem ser adicionadas quando o código da transformação for gerado pelo framework.

1.4 ESTRUTURA DO DOCUMENTO

O capítulo 2 apresenta uma visão geral sobre a abordagem de desenvolvimento dirigido a modelos, detalhando seus principais elementos, os modelos e as transformações.

O Capítulo 3 introduz o framework MDTD, que sintetiza a nossa proposta para desenvolvimento de transformações de modelos. Inicialmente discorre-se sobre o arcabouço de elementos relevantes ao desenvolvimento de transformações utilizando a abordagem DDM. Em seguida é apresentada uma visão geral sobre o framework e seus elementos. Para melhor entender o seu funcionamento é apresentado também um exemplo de transformação desenvolvida com o framework.

O Capítulo 4 apresenta a linguagem de modelagem proposta para o framework. Discorre-se sobre como interpretamos o conceito de transformações de modelos nos vários níveis de abstração da linguagem de modelagem proposta. É apresentado o metemodelo de transformações nos diversos níveis de abstração e o perfil para transformações de modelos.

O capítulo 5 detalha o processo de desenvolvimento de transformações MDTDproc. Inicialmente é apresentado uma visão geral do ciclo de vida de desenvolvimento proposto no processo e em seguida são detalhadas cada uma das fases que compõe esse ciclo de vida e suas atividades.

O Capítulo 6 apresenta o arcabouço de automação definido para o framework. Inicialmente é apresentado o ambiente definido para a execução do mesmo e em seguida é

detalha a cadeia de transformações desenvolvida para automatizar o processo de desenvolvimento proposto.

O Capítulo 7 apresenta a avaliação do framework realizada através de estudo de caso e experimento controlado.

No Capítulo 8 é apresentada uma revisão bibliográfica em desenvolvimento de transformações de modelos. Considerando que o processo de desenvolvimento proposto é o elemento central do framework, este capítulo apresenta inicialmente uma revisão da literatura sobre as estratégias sistemáticas de desenvolvimento de transformações. Em seguida discorre sobre os trabalhos relacionados às linguagens e notações para especificação de transformações de modelos.

Finalmente, no Capítulo 9 são apresentadas as considerações finais e propostas de trabalhos futuros.

Capítulo 2

DESENVOLVIMENTO DIRIGIDO A MODELOS

Este capítulo discorre sobre a abordagem de Desenvolvimento Dirigido a Modelos (DDM). Inicialmente é apresentada uma visão geral da abordagem. Em seguida seus principais elementos, os modelos e as transformações, são detalhados. Finalmente é apresentado como o desenvolvimento dirigido a modelos pode ser utilizado para a construção de transformações.

2.1 VISÃO GERAL DA ABORDAGEM DDM

O Desenvolvimento Dirigido a Modelos (DDM) é uma abordagem de desenvolvimento de software que usa modelos como os artefatos principais do desenvolvimento. Desta forma, caracteriza-se pela mudança da ênfase do desenvolvimento, antes no código, para os modelos. Na DDM modelos em alto nível de abstração são transformados de maneira automática / semi-automática, através de uma cadeia de transformações, em modelos menos abstratos até chegar ao código fonte da aplicação (MELLOR, 2004).

A Figura 2.1 (A) apresenta uma visão geral do processo de desenvolvimento de software usando a abordagem DDM. Inicialmente é especificado o modelo da aplicação em alto nível de abstração (ex. *Modelo 1*). Esse modelo é transformado em modelos menos abstratos (ex. *Modelo 2*) e assim sucessivamente até o *Modelo n*. A conversão dos modelos nos diversos níveis de abstração é realziada por uma cadeia, composta por um conjunto de softwares chamados de transformações, responsável pela (semi-) automação do desenvolvimento. Cada novo modelo gerado pode ser modificado pelo desenvolvedor antes de ser convertido no próximo modelo. Na Figura 2.1 (A), por exemplo, o *Modelo 1* pode representar o modelo de uma aplicação no nível de requisitos, o *Modelo 2* o modelo da aplicação no nível de projeto e assim por diante. Ao final do processo de desenvolvimento é gerado o código fonte da aplicação em linguagens específicas.

A realização mais conhecida da DDM é o framework especificado pela OMG (Object Management Group) chamado Model Driven Architecture (MDA) (OMG, 2014). A MDA objetiva aumentar a portabilidade, interoperabilidade e produtividade de sistemas de software, partindo do princípio de que conceitos são mais estáveis que tecnologias, ou seja,

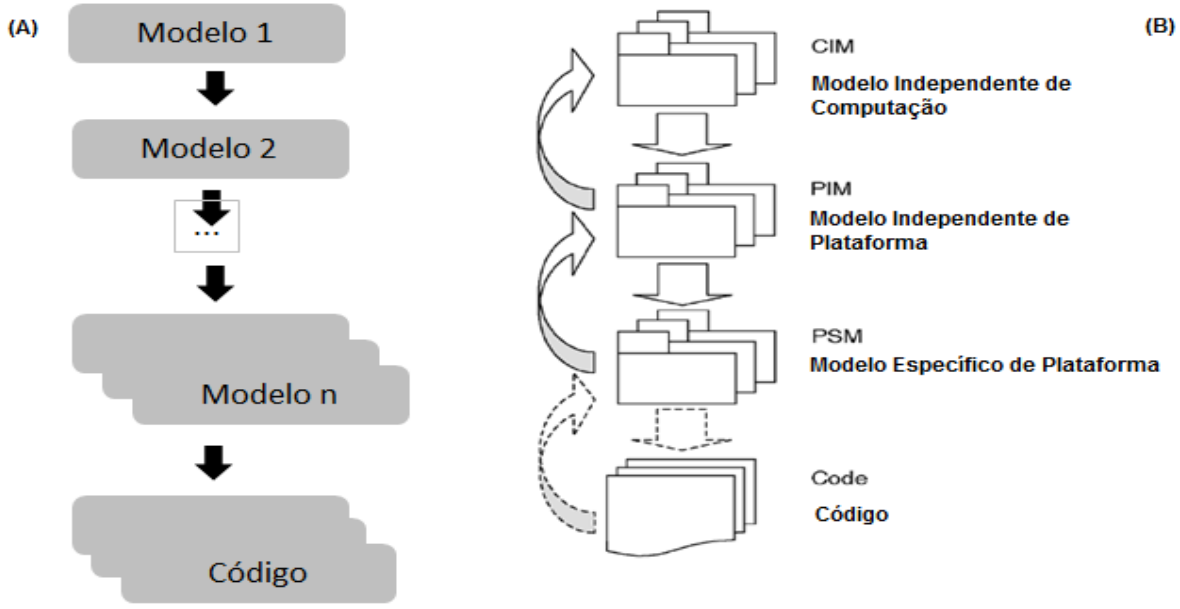


Figura 2.1 Visão geral da abordagem DDM (A), Visão geral da MDA (B)

o negócio do sistema existe independente de tecnologia. Portanto, é possível modelar o negócio e depois definir em quais tecnologias ele será implementado.

A MDA considera uma cadeia de transformações entre três níveis de modelos (Figura 2.1 B), modelo independente de computação (CIM), modelo independente de plataforma (PIM) e modelo específico de plataforma (PSM), até a geração do código fonte do sistema. O modelo independente de computação (CIM) é responsável pelo levantamento das necessidades que envolvem o negócio; o modelo independente de plataforma (PIM) é responsável pelo projeto do sistema independente da plataforma; o modelo específico de plataforma (PSM) representa o projeto do sistema em uma plataforma tecnológica específica; e o código consiste no código fonte do sistema (MELLOR, 2004).

A abordagem DDM/MDA contém dois elementos essenciais, os modelos, artefatos que representam o software nos diversos níveis de abstração; e as transformações, responsáveis pela (semi-) automação da conversão dos modelos até a geração do código. As subseções a seguir detalham cada um desses elementos.

2.2 MODELOS

Modelos são representações abstratas de um sistema e compreendem estrutura e comportamento (OMG, 2014). Na DDM modelos não são vistos como simples documentação de software, mas como matéria prima para a geração de outros modelos e do código fonte da aplicação. Por isso, os modelos precisam ser formalmente escritos utilizando linguagens de modelagem, com sintaxe e semântica bem definidas.

Atualmente existem diversas linguagens de modelagem, algumas delas são específicas para um domínio (uma área de conhecimento), chamadas de DSL (*Domain Specific Language*), outras são de propósito geral, chamadas de GPL (*General Purpose Languages*),

como, por exemplo, a UML (*Unified Modeling Language*). No contexto de DDM as DSLs são mais frequentemente utilizadas, pois encapsulam um domínio de aplicação, o que torna os modelos mais expressivos.

Uma linguagem de modelagem compreende quatro elementos principais: a sintaxe abstrata, onde são definidos os contrutores da linguagem; a semântica estática, com as regras de boa formação e restrições definidas para a linguagem; a sintaxe concreta, que especifica a notação concreta para representação dos construtores da linguagem; e a semântica comportamental, onde é definido como os modelos instanciados são executados possibilitando a simulação destes modelos.

Na DDM a sintaxe abstrata e a semântica estática das linguagens são expressas em termos de metamodelos. Desta forma, os modelos construídos ao longo do desenvolvimento são instâncias de metamodelos. De maneira análoga os metamodelos são descritos a partir de metametalinguagens representadas sob a forma de metametamodelos (STAHL; VOLTER, 2010).

A relação entre modelos, metamodelos e metametamodelos é definida seguindo a arquitetura em camadas proposta pela OMG e ilustrada na Figura 2.2. A camada M0 corresponde às aplicações do mundo real. Na camada M1, essas aplicações são representas como modelos. Estes modelos por sua vez são instâncias de metamodelos (camada M2). Analogamente, os metamodelos são instâncias de metametamodelos (camada M3). Por exemplo, uma aplicação para um restaurante (camada M0) pode ser modelada utilizado um diagrama de classes (camada M1) que é uma instância do metamodelo da linguagem UML (camada M2) que por sua vez é uma instância do metametamodelo MOF (camada M3).

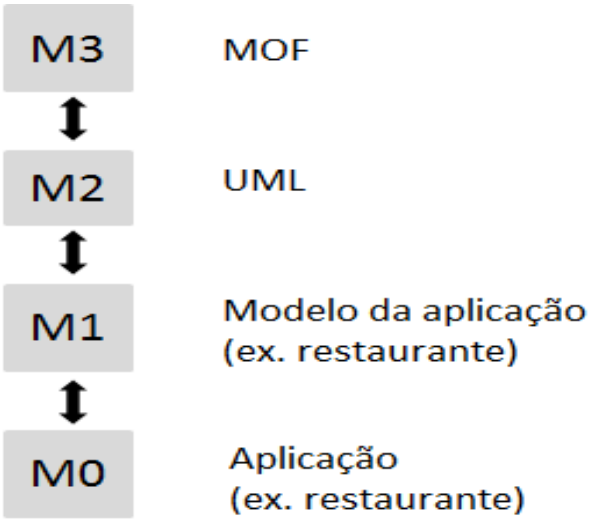


Figura 2.2 Arquitetura em camadas da OMG

Os modelos devem obedecer a uma relação de conformidade com seus metamodelos. Diz-se que um modelo M está em conformidade com um metamodelo MM se M for sintaticamente correto em relação a MM e satisfizer as restrições definidas por MM. Na DDM os modelos representam a especificação de uma aplicação, portanto, eles também

devem satisfazer as propriedades da aplicação, ou seja, devem ser corretos em relação à aplicação. A relação de correção não é garantida pela relação de conformidade, uma vez que os metamodelos definem propriedades gerais de um conjunto de aplicações e não propriedades específicas de cada aplicação. A garantia de correção dos modelos exige a utilização, portanto, de técnicas de validação e verificação como em outros processos de desenvolvimento de software (BRAGA; SANTOS; DA SILVA, 2014).

A linguagem UML, embora seja de propósito geral, pode ser customizada para domínios específicos, a partir da especificação de perfis, para ser utilizada na abordagem DDM. Um perfil UML é um mecanismo de extensão da própria UML para definir uma linguagem específica de domínio (OMG, 2005).

Um perfil não altera a semântica original dos elementos da UML e sim especializa esses elementos vinculando-os a um domínio. A definição de um perfil compreende a especificação de estereótipos, metaclasses e valores etiquetados. Os estereótipos são rótulos que podem ser aplicados aos diversos elementos da UML. Esses rótulos estendem metaclasses, que representam os conceitos da UML aos quais os estereótipos poderão ser aplicados. Finalmente, os valores etiquetados possibilitam a atribuição de valores aos conceitos UML estendidos. O uso de perfil possibilita aproveitar o suporte ferramental já existe na UML para a especificação de modelos em um domínio específico.

2.3 TRANSFORMAÇÕES

Transformações mapeiam modelos entre os diversos níveis de abstração ao longo do desenvolvimento (STAHL; VOLTER, 2010). As transformações são classificados de acordo com os artefatos que estão sendo gerados: quando o artefato gerado é um texto (ex. código fonte) são chamadas de transformações de programa; quando o artefato gerado é um modelo são chamadas de transformações de modelo. (MENS; CZARNECKI; GORP, 2006)

Uma transformação de modelos consiste na geração automática de um modelo de saída a partir de um modelo de entrada de acordo com um conjunto de regras de transformação que juntas descrevem como um modelo escrito em uma linguagem fonte pode ser transformado em um modelo escrito em uma linguagem alvo. Cada regra de transformação representa uma definição de como um ou mais construtores em uma linguagem fonte podem ser transformados em um ou mais construtores de uma linguagem alvo (KLEPPE; WARMER; BAST, 2003). Transformações de modelos podem envolver também múltiplos modelos de entrada e múltiplos modelos de saída (MENS; CZARNECKI; GORP, 2006).

Transformações processam modelos, mas são definidas no nível de metamodelos. A Figura 2.3 mostra um exemplo de como é definida e processada uma transformação. Conforme ilustrado na figura, uma transformação T mapeia elementos do metamodelo fonte *MMf* em elementos do metamodelo alvo *MMa*. Esta transformação é implementada utilizando linguagens específicas de transformação. Uma vez implementada ela pode ser executada. A execução do software da transformação é realizada por um engenho de transformação que recebe como entrada um modelo *me*, em conformidade com o metamodelo fonte *MMf*, e gera como saída outro modelo *ms*, em conformidade com um metamodelo alvo *MMa*. Ambos os metamodelos *MMf* e *MMa* estão também em

conformidade com o metametamodelo *MMM*.

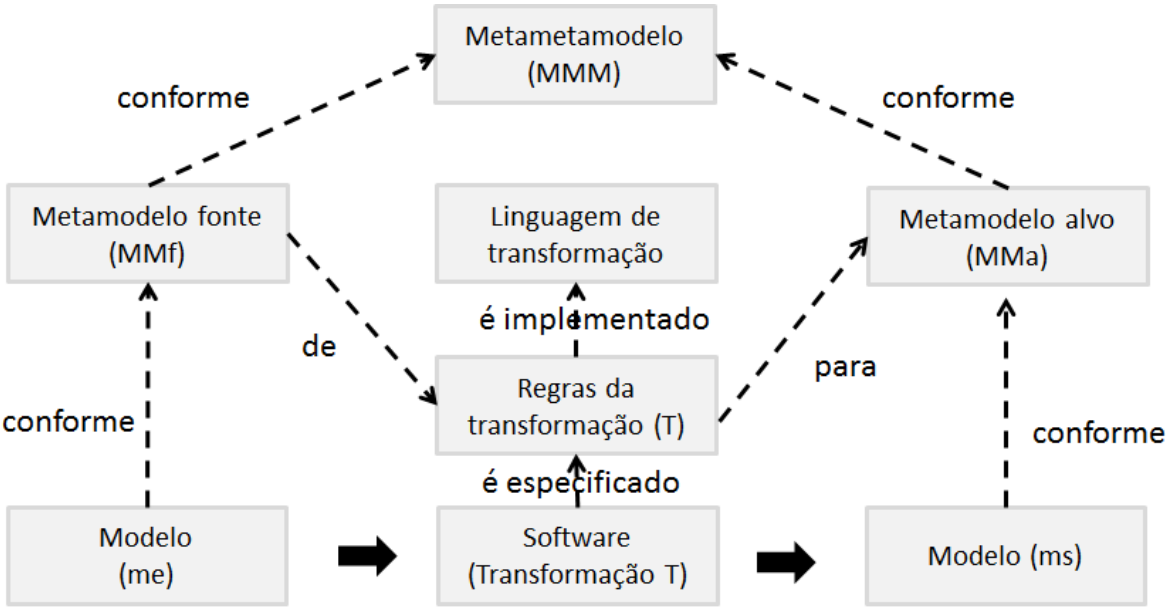


Figura 2.3 Definição de transformação

A execução de uma transformação pode acontecer em duas direções: unidirecional e bidirecional. A transformação unidirecional converte modelos de entrada em modelos de saída. A transformação bidirecional possibilita também a conversão dos modelos de saída em modelos de entrada. Independente do sentido de execução, os modelos processados são especificados em linguagens de modelagem. As transformações cujos modelos de entrada e saída são definidos na mesma linguagem de modelagem são chamadas de endógenas e as transformações cujos modelos são escritos em linguagens de modelagem diferentes são chamadas de exógenas. Adicionalmente, transformações de modelos também podem ser classificadas, quanto ao nível de abstração, em horizontais e verticais. Na transformação horizontal, os modelos de entrada e saída estão no mesmo nível de abstração. Já na transformação vertical, há uma mudança de nível de abstração entre o modelo de entrada e o modelo de saída (MENS; CZARNECKI; GORP, 2006).

Conforme definido na seção anterior, os modelos que participam de uma transformação devem preservar uma relação de conformidade com seus respectivos metamodelos. A relação de conformidade entre modelos e metamodelos está diretamente ligada à correção sintática da transformação. Uma transformação é dita sintaticamente correta se e somente se dado um modelo de entrada, em conformidade com o metamodelo fonte, a transformação gera um modelo de saída em conformidade com o metamodelo alvo. A correção sintática de transformações é requisito elementar para a abordagem DDM, pois é imprescindível garantir que transformações que recebem modelos de entrada em conformidade com metamodelos fonte gerem modelos de saída em conformidade com metamodelos alvo. Contudo, outras propriedades também são relevantes para o desenvolvimento de transformações, como, por exemplo, a correção semântica e a completude (LANO; CLARK, 2008).

A correção semântica de uma transformação está relacionada à garantia de preservação de propriedades do modelo fonte no modelo alvo gerado após a execução da transformação. Desta forma, dado um modelo de entrada que atenda a um conjunto P de propriedades, uma transformação é dita semanticamente correta se e somente se a transformação gerar um modelo de saída que preserve todas as propriedades p pertencentes a P (VARRO; PATARICZA, 2003) (LANO; CLARK, 2008).

A propriedade de completude está relacionada ao nível de cobertura da transformação sobre os elementos definidos no metamodelo fonte. Desta forma, uma transformação é dita completa se e somente se para cada elemento do metamodelo fonte existir um elemento correspondente no metamodelo alvo mapeado pela transformação (VARRO; PATARICZA, 2003).

2.4 MODELOS DE TRANSFORMAÇÕES DE MODELOS

Transformações de modelos podem tambem ser especificadas em alto nível de abstração como modelos de transformações de modelos (BEZIVIN et al., 2006). Nesta direção, a definição de uma transformação passa a ser representada por um modelo de transformação que por sua vez está em conformidade com um metamodelo que representa o domínio de transformações (Figura Figura 2.4)

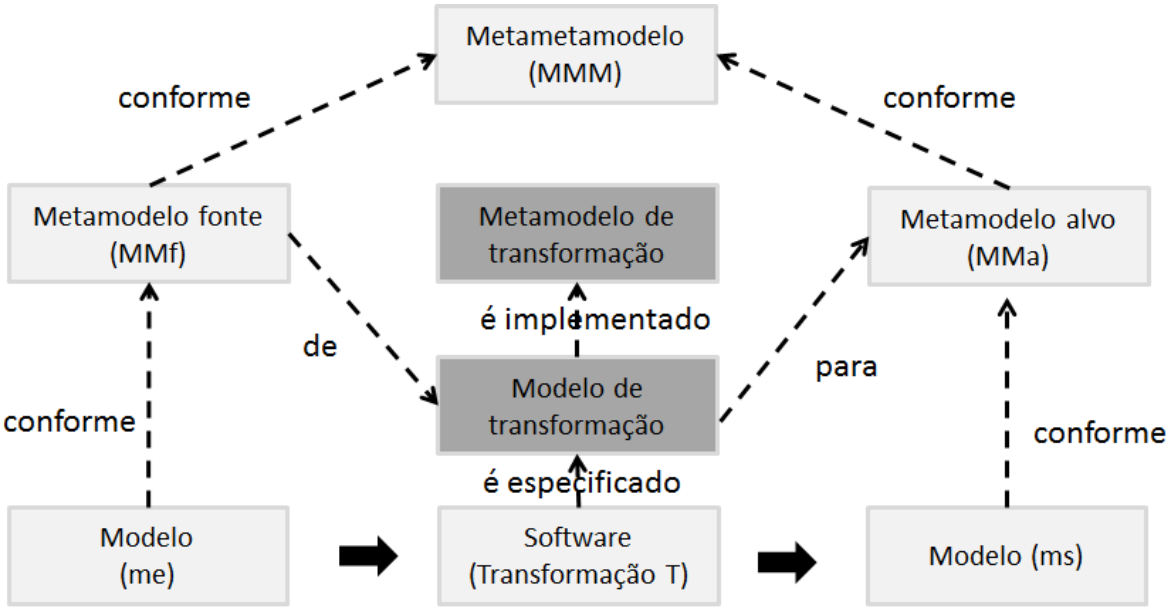


Figura 2.4 Modelo de transformação de modelo

A especificação da transformação através de modelos possibilita que a própria abordagem DDM seja utilizada para desenvolver transformações. Modelos de transformação de modelos são especificados em alto nível de abstração e transformados em modelos de transformação de modelos menos abstratos até a geração do código em uma linguagem de transformação específica (Figura 2.5).

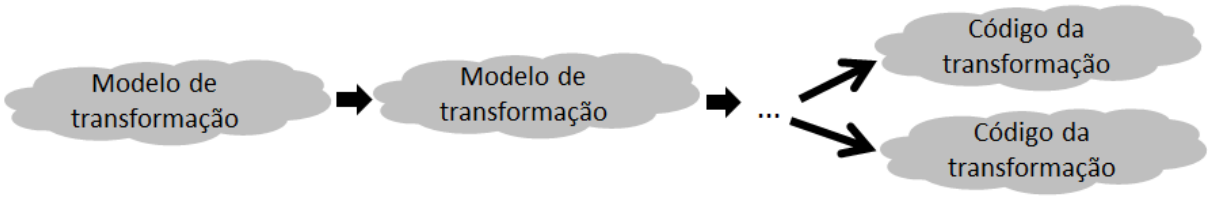


Figura 2.5 DDM para desenvolver transformações de modelos

2.5 CONSIDERAÇÕES SOBRE O CAPÍTULO

Neste capítulo mostramos uma visão geral da abordagem de desenvolvimento dirigido a modelos detalhando a importância dos modelos e das transformações dentro desta abordagem. Mostramos também que a própria abordagem DDM tem sido utilizada para construir transformações, possibilitantdo que o código da transformação seja gerado ao final da cadeia. Esta tese segue nesta direção e busca diminuir a complexidade que permeia o desenvolvimento de transformações elevando o nível de abstração do seu desenvolvimento do código para os modelos.

Capítulo

3

FRAMEWORK PARA DESENVOLVIMENTO DE TRANSFORMAÇÕES DE MODELO (MDTD)

Este capítulo apresenta o framework MDTD (*Model Driven Transformation Development*) que sintetiza a nossa proposta de desenvolvimento de transformações de modelos. Inicialmente é definido um arcabouço com os elementos relevantes à definição do framework. Em seguida é apresentada uma visão geral do framework MDTD proposto e de seus principais elementos. Finalmente é mostrado um exemplo de transformação desenvolvida com o framework.

3.1 ARCABOUÇO PARA CONSTRUIR TRANSFORMAÇOES COM DDM

No desenvolvimento de transformações usando a abordagem DDM, modelos de transformações de modelos são construídos em alto nível de abstração e transformados em modelos menos abstratos, até que seja gerado o código fonte em uma linguagem de transformação específica. Para apoiar esse desenvolvimento, assim como no desenvolvimento de outros softwares usando a abordagem DDM, é pertinente adotar uma estrutura que possibilite a construção dos modelos de transformação e o processamento destes até a geração do código. Nesta direção, definimos um arcabouço para o desenvolvimento de transformações, ilustrado na Figura 3.1, com os elementos relevantes ao desenvolvimento de transformações na abordagem DDM.

De acordo com a Figura 3.1 modelos de transformação de modelos podem ser inicialmente construídos independentes de plataforma (por exemplo os modelos de transformação 1 e 2 na figura) e em seguida serem mapeados em modelos de plataformas específicas (modelos de transformação n na figura). A construção dos modelos independentes de plataforma demanda a definição de linguagens de modelagem específicas de domínio (DSMLs) para os diversos níveis de abstração desejados. Na figura são ilustrados dois níveis de abstração, assim como na MDA, mas podem existir mais níveis. Analogamente, a construção dos modelos específicos de plataforma demanda a definição de DSMLs para as plataformas desejadas. Em ambos os casos, é necessário o uso de ferramentas automatizadas e customizadas para essas DSMLs.

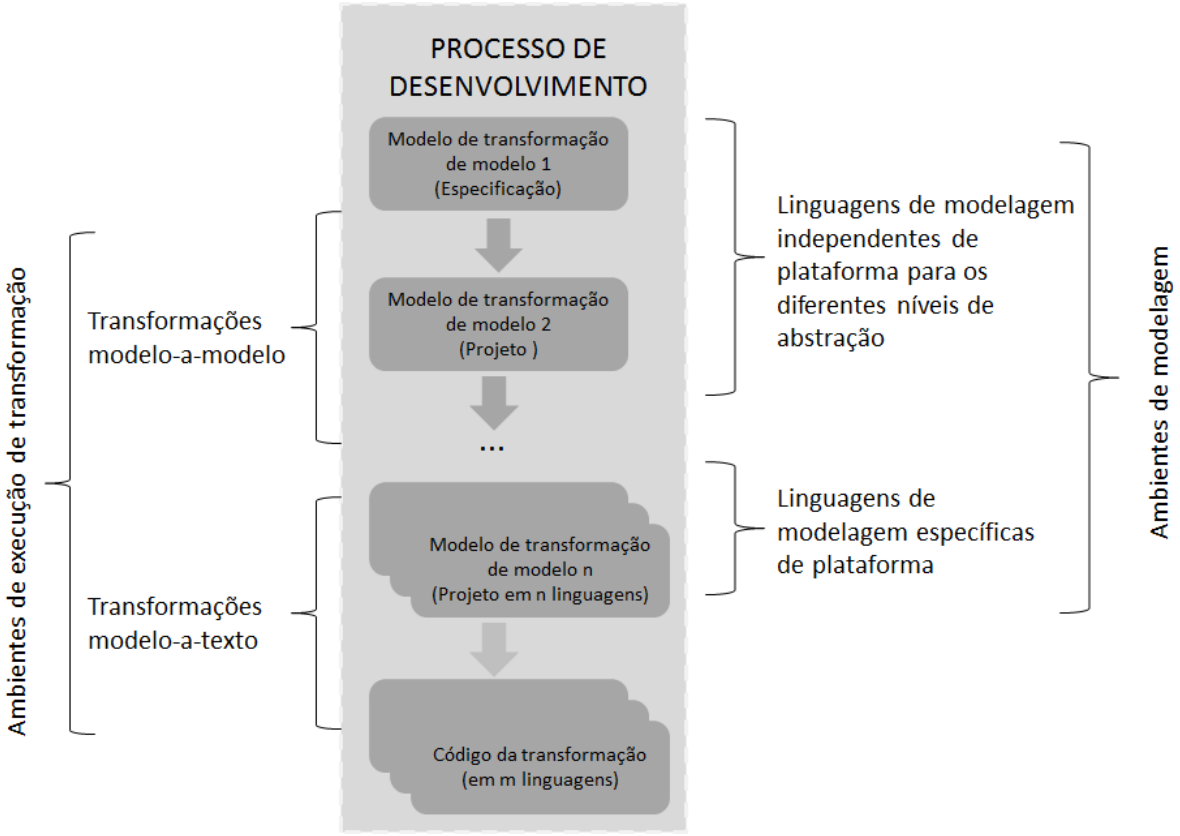


Figura 3.1 Arcabouço para desenvolvimento de transformações na abordagem DDM

Para que os modelos construídos possam ser processados é necessário também implementar uma cadeia com transformações específicas para as DSMLs utilizadas. Esta cadeia deve contemplar tanto transformações de modelos (modelo-a-modelo), para converter os modelos nos diversos níveis de abstração, quanto transformações de programas (modelo-a-texto), para geração do código nas linguagens desejadas, e devem ser apoiadas por engenhos que permitam executá-las.

Finalmente, assim como no desenvolvimento de outros tipos de software, é relevante definir também um processo de desenvolvimento (destacado no centro da figura) que auxilie o desenvolvedor ao longo de todo o ciclo de vida de desenvolvimento da transformação. O processo é importante para conduzir o desenvolvedor não só nas atividades relacionadas ao desenvolvimento, mas também no uso dos elementos envolvidos neste desenvolvimento, tais como as linguagens, ambientes, e cadeia de transformações.

3.2 VISÃO GERAL DO FRAMEWORK MDTD

Nesta tese propomos uma sistematização do desenvolvimento de transformações modelo a modelo com base na abordagem dirigida a modelos. A proposta está sintetizada em um framework, chamado *MDTD*, uma solução integrada para desenvolvimento de transformações unidirecionais usando a abordagem de desenvolvimento dirigido a modelos que

compreende diversos elementos, relacionados ao arcabouço definido na seção anterior: (i) linguagem de modelagem independente de plataforma; (ii) linguagens de modelagem para plataformas específicas; (iii) processo de desenvolvimento de transformações; (iv) cadeia de transformações; e (v) ambiente de desenvolvimento. A Figura 3.2 apresenta uma visão geral destes elementos e suas relações.

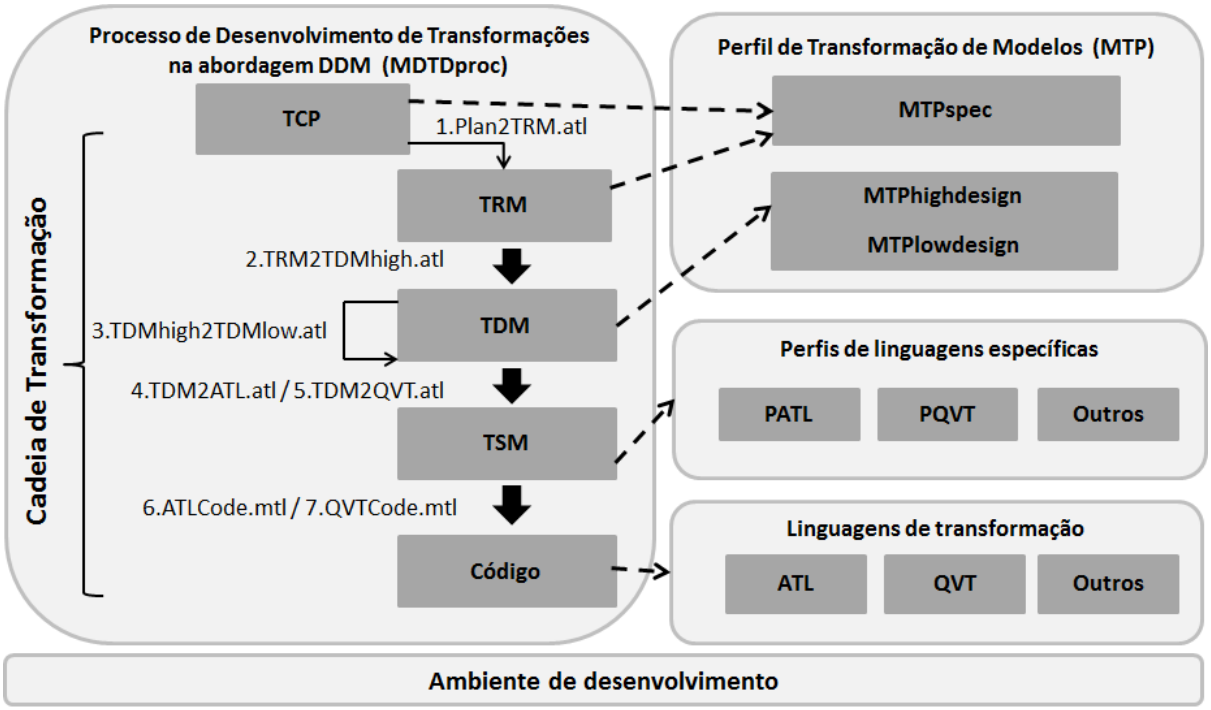


Figura 3.2 Elementos do framework MDTD

O processo MDTDproc (lado esquerdo da Figura 3.2) é responsável por guiar todo o desenvolvimento e contém um ciclo de vida que compreende cinco fases: Planejamento (*Transformation Chain Planning - TCP*), relacionada ao planejamento da cadeia e suas transformações; Especificação (*Transformation Requirements Modeling - TRM*), relacionado ao detalhamento dos requisitos da transformação e a definição dos metamodelos envolvidos; Projeto independente (*Transformation Design Modeling - TDM*), relacionado ao projeto da transformação independente de plataforma; Projeto específico (*Transformation Specific Modeling - TSM*), relacionado ao projeto da transformação em uma plataforma específica; e Código, relacionado ao código da transformação em uma linguagem específica. O detalhamento do processo MDTDproc é apresentado na Seção 5.1.

A linguagem de modelagem independente de plataforma, chamada Perfil de Transformação de Modelos (MTP - *Model Transformation Profile*), compreende um conjunto de metamodelos e perfis que representam os conceitos do domínio de transformação de modelos. O MTP (no topo à direita da Figura 3.2), apresentado no Capítulo 4, apoia as atividades executadas nas três primeiras fases do processo de desenvolvimento de transformações (*TCP*, *TRM* e *TDM*) e permite a modelagem de transformações em diferentes níveis de abstração, independente de linguagem de programação, utilizando diagramas da

UML. Desta forma, possibilita o projeto de soluções disassociadas de detalhes específicos de implementação, portáveis para linguagens específicas de transformação.

As linguagens de modelagem específicas de plataforma também são definidas sob a forma de perfis UML. Dois perfis específicos compõem o framework, o perfil *PATL* (para a linguagem ATL) e o perfil *PQVT* (para a linguagem PQVT), do lado direito da Figura 3.2. Estes perfis apoiam as atividades do processo de desenvolvimento de transformações na fase de projeto específico de plataforma (*TSM*), e possibilitam a geração e representação de modelos de transformação nas linguagem ATL e QVT (na base direita da Figura 3.2) usando o diagrama de classes da UML. Os perfis PATL e PQVT estão disponíveis nos apêndices G e H.

A cadeia de transformação (detalhada no Capítulo 6), lado direito da figura, (semi) automatiza o processo de desenvolvimento através de um conjunto de sete transformações que geram modelos de transformações de modelos em diversos níveis de abstração, desde os requisitos até a geração do código fonte.

O ambiente de desenvolvimento (detalhado no Capítulo 6), apresentado na base da Figura 3.2, integra ferramentas de modelagem, ferramentas de metamodelagem e engenhos de transformações, no ambiente Eclipse. Este ambiente utiliza apenas ferramentas livres que podem ser substituídas por outras de mesmo fim. Por exemplo, o uso de um perfil UML para modelagem da transformação possibilita trocar o ambiente de modelagem atual por outra ferramenta de modelagem que utilize a linguagem UML.

A integração do framework a novas linguagens de transformação pode ser realizada através da construção de transformações que convertam o modelo independente de plataforma em modelos específicos de outras linguagens.

Para melhor ilustrar como acontece o desenvolvimento de transformações aqui proposto, será apresentado um exemplo de transformação desenvolvida com o framework, a transformação de um modelo de classes da UML para um modelo lógico de banco de dados. Trata-se de um exemplo clássico utilizado em muitos trabalhos ((RAHIM; MANSOOR, 2008),(GUERRA et al., 2010b), (SANI; POLACK; PAIGE, 2011), (BOLLATI et al., 2013)) para ilustrar o desenvolvimento de transformações. Este exemplo contempla as principais atividades do processo de desenvolvimento e possibilita o uso de vários dos conceitos definidos na linguagem de modelagem proposta, possibilitando uma compreensão geral do framework. Outro exemplo de transformação desenvolvida com o framework pode ser encontrada no apêndice E. Para introduzir o exemplo, inicialmente é descrito o cenário que será utilizado para em seguida ser apresentado o desenvolvimento de uma transformação neste cenário.

3.2.1 Cenário da transformação do exemplo

Considere um processo de desenvolvimento de software na abordagem orientada a objetos. Neste processo um dos diagramas construído é o diagrama de classes. A partir deste diagrama de classes o analista então modela o banco de dados que será criado para o sistema. Neste exemplo desejamos automatizar a geração do banco de dados a partir do diagrama de classes definido pelo analista de sistemas. Essa automação deve compreender a geração do modelo lógico do banco de dados e em seguida a geração do script que será

utilizado para a criação do banco propriamente dito.

Para compreender melhor o exemplo, consideremos o desenvolvimento de um sistema para controle acadêmico de uma universidade. Suponhamos que o analista responsável por desenvolver esse sistema construiu um diagrama de classes com várias classes, tais como, *Aluno, Curso, Disciplina*, entre outras. Agora ele precisa criar o modelo lógico e em seguida o script do banco de dados para esse sistema identificando quais as tabelas, campos, chaves primárias, chaves estrangeiras e outros elementos que deverão ser criados.

3.2.2 Desenvolvimento do exemplo de transformação

Nesta seção é detalhado o desenvolvimento da transformação especificada no cenário anterior. Serão apresentados os principais documentos produzidos, organizados de acordo com as fases do processo MDTDproc, até a geração do código da transformação.

3.2.2.1 Fase TCP A fase de planejamento objetiva definir a cadeia de transformações e os requisitos de cada uma das transformações que a compõe. Neste exemplo vamos automatizar a geração do banco de dados através da definição de uma cadeia composta por duas transformações: uma transformação model-a-modelo que transforma um diagrama de classes em um modelo lógico de banco de dados; e uma transformação modelo-a-texto que gera o script do banco de dados a partir do modelo lógico. Mais especificamente será detalhado nesta seção o desenvolvimento da primeira transformação, aqui chamada de *OO2RDBMS*.

Para identificar os requisitos da transformação *OO2RDBMS* o processo MDTDproc indica a utilização de exemplos de modelos já construídos para aplicações anteriores no mesmo domínio da transformação. Sendo assim, a Figura 3.3 (A) mostra um exemplo de diagrama de classes construído para um sistema acadêmico e a Figura 3.3 (B) mostra o modelo lógico do banco de dados correspondente. Baseado neste e em outros exemplos similares o desenvolvedor identifica os requisitos da transformação.

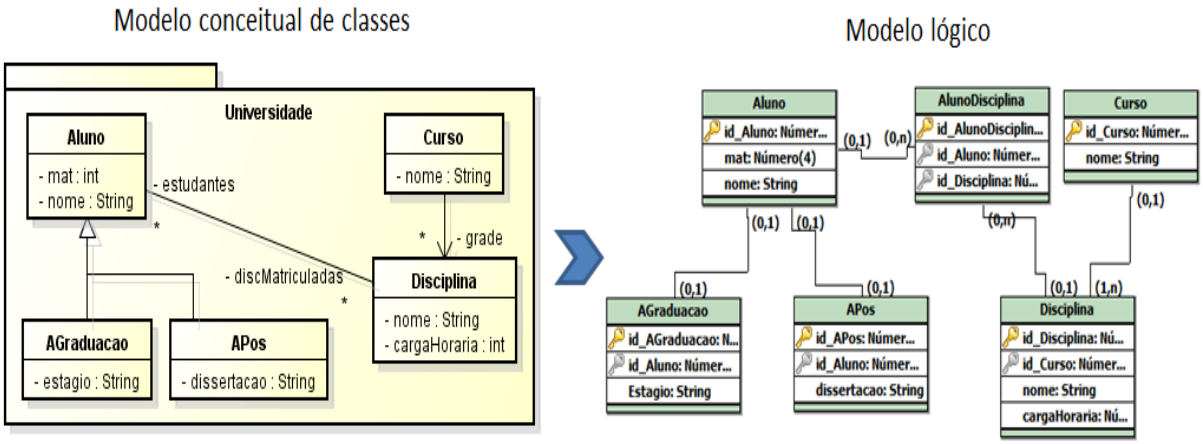


Figura 3.3 Exemplo de diagrama de classes (A) e do modelo lógico correspondente (B)

O framework MDTD modela os requisitos de uma transformação utilizando um dia-

grama de casos de uso. A Figura 3.4 mostra alguns dos requisitos que podem ser extraídos do sistema acadêmico utilizado como exemplo. Observa-se que três requisitos principais foram identificados: *MapearModelo*, para mapear o sistema como um todo e representá-lo em um esquema do banco de dados; *MapearClasse*, para mapear as classes do sistema; e *MapearAtributo*, para mapear os atributos de cada classe. Alguns desses requisitos foram também detalhados. Por exemplo, o requisito *MapearClasse* compreende dois outros requisitos (associação *include*), *CriarID* e *CriarPK* para criar, respectivamente, um campo identificado e uma chave primária para a tabela que será gerada. Analogamente, o requisito *MapearAtributo* é especializado em dois outros requisitos, *MapearAtributoMultiVal*, para tratar os atributos multivalorados, e *MapearAtributoMonoVal*, para tratar os atributos monovalorados. Adicionalmente, também foi definida uma restrição *Constraint* associada ao requisito *MapearClasse*.

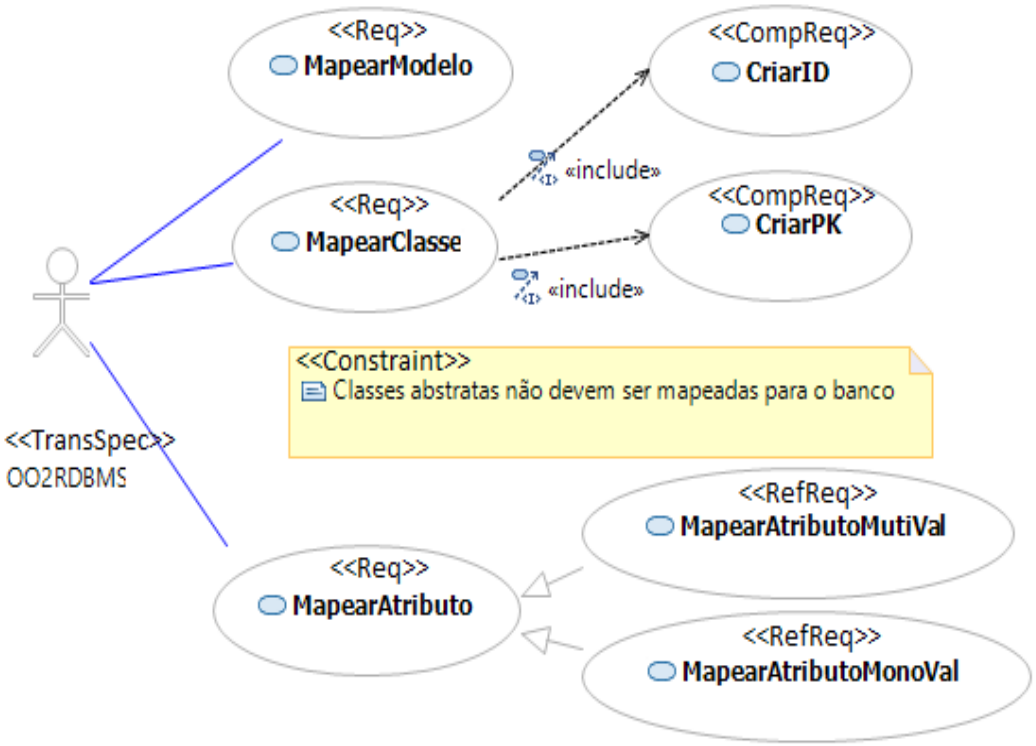


Figura 3.4 Diagrama de casos de uso com os requisitos da transformação OO2RDBMS

A partir destes requisitos iniciais o framework gera automaticamente um diagrama de classes, também com os requisitos da transforamção, para que outros detalhes possam ser adicionados à especificação inicial (Figura 3.5), como, por exemplo, o nível de prioridade e a versão em que o requisito será desenvolvido.

3.2.2.2 Fase TRM A fase TRM compreende diversas atividades, dentre elas documentar o objetivo tanto da transformação quanto dos requisitos e identificar ou definir os metamodelos envolvidos na transformação. Para o nosso exemplo, foi acrescentada a descrição do objetivo de cada um dos requisitos (ver atributo *description* em cada uma

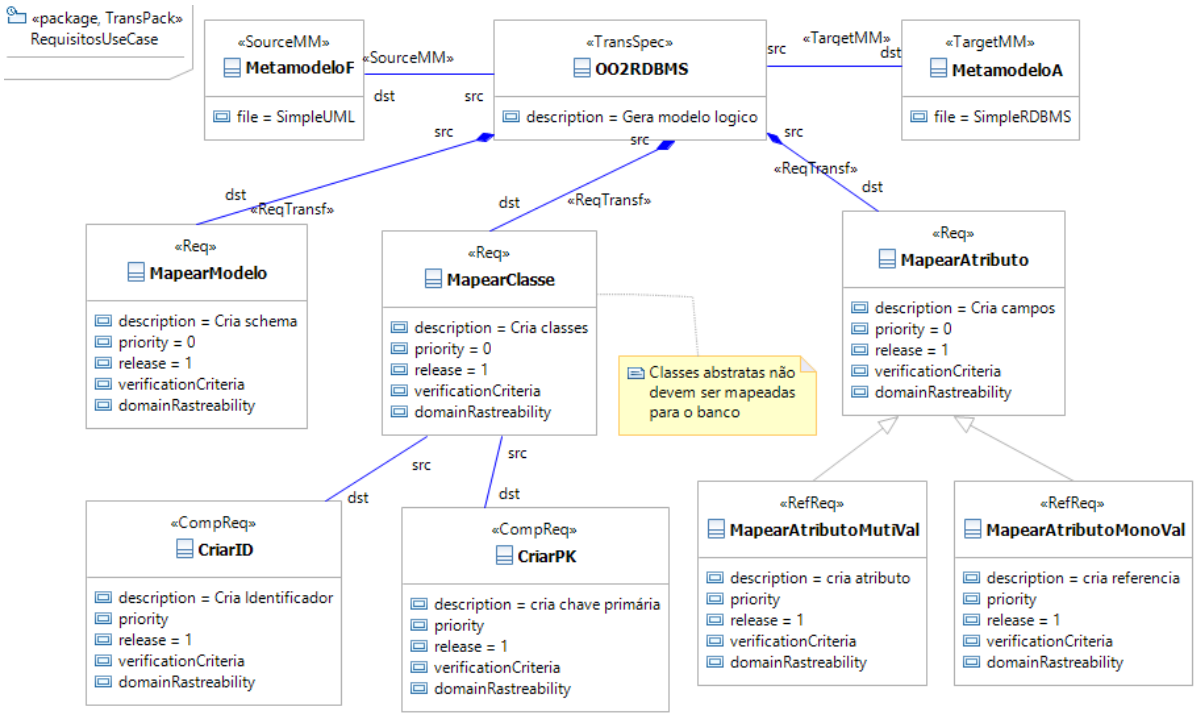


Figura 3.5 Diagrama de classes com os requisitos da transformação

das classes da Figura 3.5). Adicionalmente, foram definidos também quais os metamodelos fonte e alvo que estão envolvidos na transformação (ver as classes estereotipadas como *SourceMM* e *TargetMM* também na Figura 3.5). Para o exemplo da transforamção *OO2RDBMS* foi criado o metamodelo *SimpleUML* e o metamodelo *SimpleRDBMS*, conforme ilustrado nas Figuras 3.6 e 3.7. O metamodelo *SimpleUML* representa o domínio de diagrama de classes da UML. Neste, um modelo é representado por um pacote (*UmlPackage*) que contém classes (*Umlclass*) que por sua vez compreendem atributos (*UmlAttributes*) e se relacionam através de associações (*UmlAssociation*). O metamodelo *SimpleRDBMS* representa o domínio de banco de dados. Neste, um esquema de banco (*RdbmsSchema*) é composto de tabelas (*RdbmsTable*) que por sua vez contém campos (*RdbmsFields*), chave primária (*RdbmsKey*) e chaves estrangeiras (*RdbmsForeignKey*).

3.2.2.3 Fase TDM Na fase TDM é construído o projeto da transformação independente de plataforma em dois níveis de abstração, o projeto de alto nível e o projeto de baixo nível.

O projeto de alto nível se inicia com a especificação da arquitetura da transformação através de um diagrama de componentes (ver Figura 3.8). Esta arquitetura modela a estrutura da transformação como uma composição de uma ou mais transformações menores representadas por componentes que se relacionam. Para a transformação *OO2RDBMS* foi definida uma arquitetura com somente um componente, que provê uma interface, chamada *IExecucao*, com as operações disponibilizadas pelo componente (essas operações serão as futuras regras implementadas para a tranformação), e requer uma interface, cha-

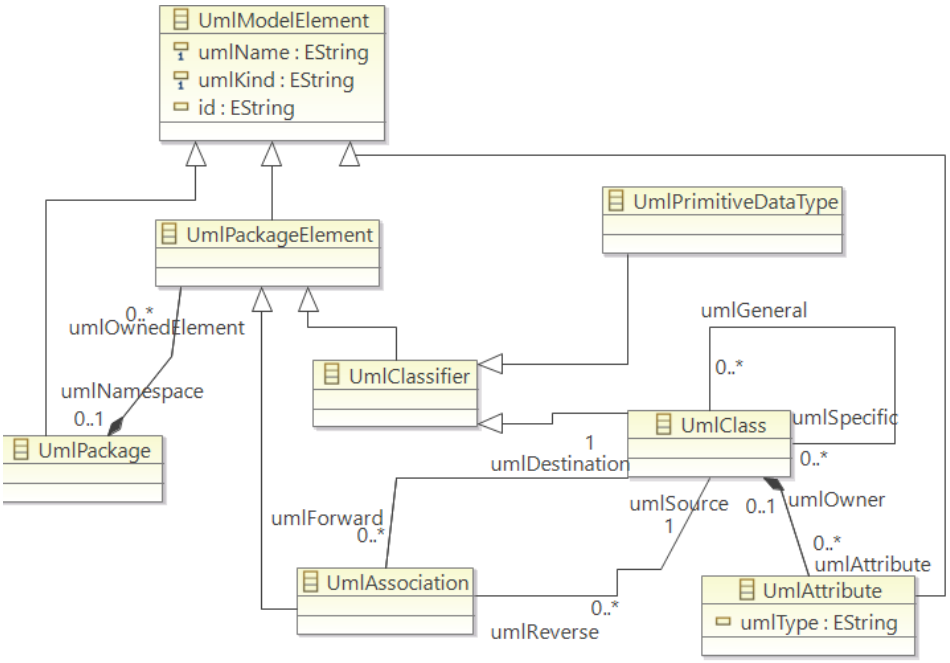


Figura 3.6 Metamodelo SimpleUML

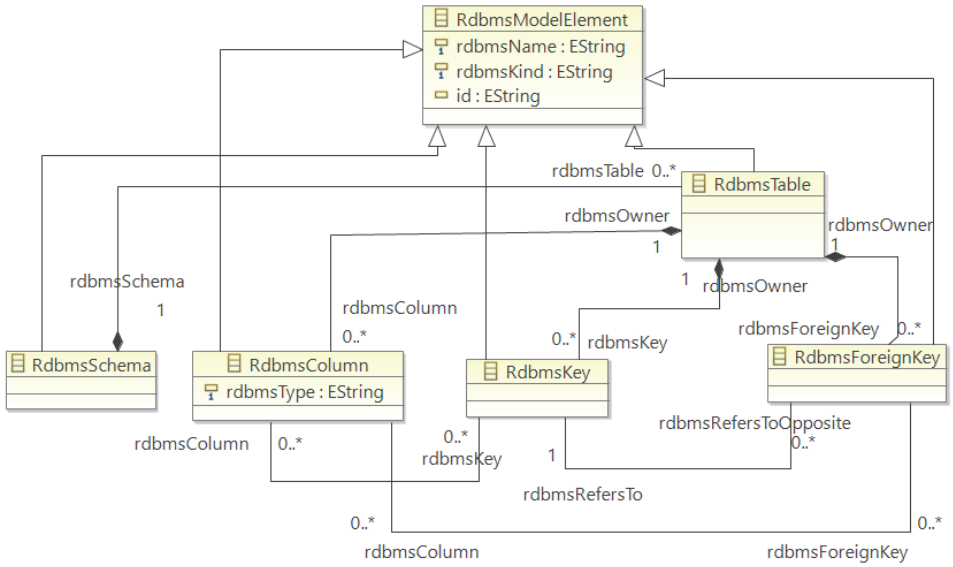


Figura 3.7 Metamodelo SimpleRDBMS

mada de *IModelos*, que contém operações para carregar os modelos e metamodelos de entrada e saída da transformação, e devem ser implementadas por quem deseje utilizar o componente. As operações não estão ilustradas na figura.

No projeto de alto nível são especificados também os relacionamentos existentes entre os elementos do metamodelo fonte e os elementos do metamodelo alvo participantes da transformação. Para esta especificação o framework gera automaticamente um modelo

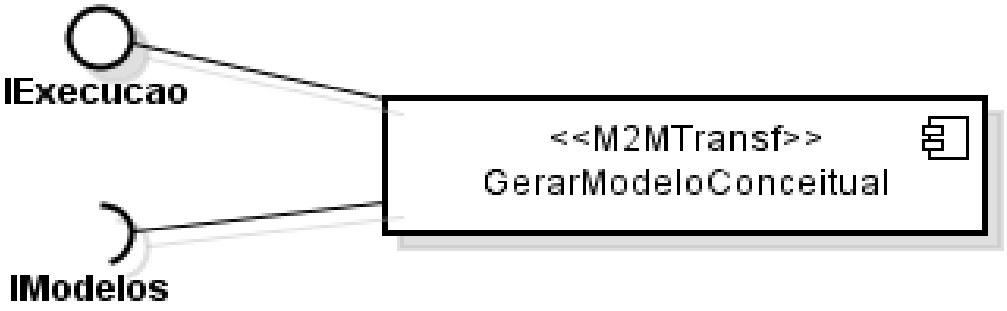


Figura 3.8 Arquitetura da transformação OO2RDBMS

inicial, com base na especificação de requisitos, que é complementado pelo desenvolvedor. A Figura 3.9 ilustra parte do modelo de projeto de alto nível com os relacionamentos definidos para a transformação *OO2RDBMS*. O modelo contém três pacotes: o primeiro deles representa o metamodelo fonte, no exemplo *SimpleUML*, e seus elementos; o segundo representa a transformação, no exemplo *OO2RDBMS*, e um conjunto de relacionamentos (estereotipados como *Relation*) entre os metamodelos; e o terceiro representa o metamodelo alvo, no exemplo *SimpleRDBMS*, e seus elementos. Os relacionamentos contidos no pacote *OO2RDBMS* são sugeridos pelo framework com base nos requisitos especificados. O desenvolvedor então deve mapear os elementos do metamodelo fonte aos elementos do metamodelo alvo ligando-os através de associações. Por exemplo, o relacionamento *MapearClasse* mapeia o elemento *UmlClass* aos elementos *RdbmsTable*, *RdbmsColumn* e *RdbsmKey*.

Durante o projeto de alto nível também são especificados casos testes e propriedades que poderão ser utilizadas na validação e verificação da transformação. Após o projeto de alto nível, o framework MDTD dispõe de uma transformação que automaticamente gera o modelo inicial de projeto de baixo nível para que o desenvolvedor continue sua especificação.

No projeto de baixo nível é definido como serão inicializadas as propriedades dos elementos que serão criados no modelo de saída gerado pela transformação. A Figura 3.10 mostra parte da especificação do projeto de baixo nível da regra *MapearClasse*. Esta regra recebe como entrada o elemento *UmlClass* e gera como saída os elementos *RdbmsTable*, *RdbmsColumn* e *RdbmsKey*. Portanto, é preciso especificar como as propriedades desses três elementos serão inicializadas quando eles forem criados. Para o elemento *RdbmsTable*, por exemplo, quatro propriedades são configuradas (ver figura): a propriedade *rdbmsName* que deve ser inicializada utilizando o mesmo nome da classe que a gerou; a propriedade *rdbmsColumn* que deve ser inicializada com uma referência à coluna criada como identificador da tabela, a propriedade *rdbmsKey*, inicializada com uma referência à chave primária criada para a tabela e, finalmente, a propriedade *rdbmsSchema* que deve refereciar o esquema criado para o banco de dados.

No final desta fase o desenvolvedor escolhe em que linguagem deseja implementar a transformação e transforma o modelo de projeto em um modelo específico para a plata-

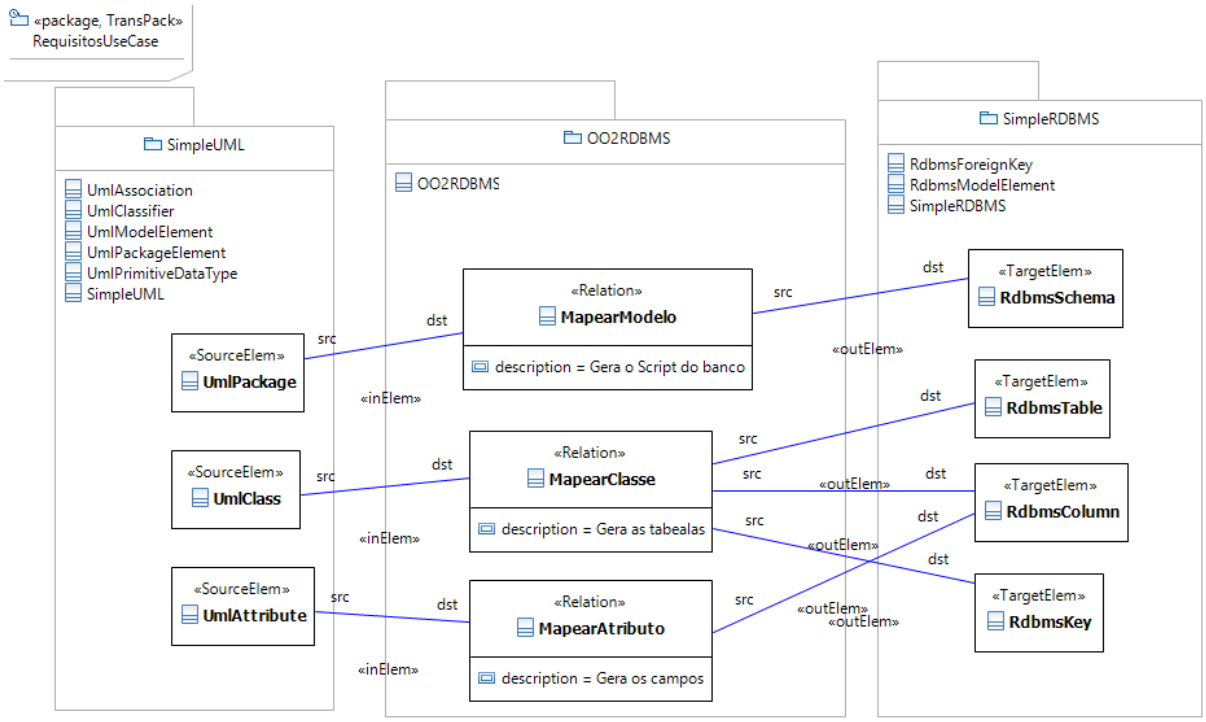


Figura 3.9 Parte do projeto de alto nível sa transformação OO2RDBMS

forma desejada. Este modelo será então utilizado na próxima fase.

3.2.2.4 Fase TSM Nesta fase é realizado o projeto da transformação em uma linguagem específica. Para o nosso exemplo foi gerado o projeto específico em linguagem ATL. Trata-se de um diagrama de classes estereotipado com os conceitos da linguagem ATL. A Figura 3.11 mostra parte do modelo gerado pelo framework. Neste, a transformação *OO2RDBMS* é representada como um *Module* que está associado a modelos e metamodelos (*OclModel*) e compreende regras, no nosso exemplo do tipo *MatchedRule*. Esta regra possui padrões de entrada e saída (*inPattern* e *outPattern*), onde o padrão de saída especifica um conjunto de inicializações (*biding*) para os atributos dos elementos que serão criados no modelo de saída. O modelo gerado pelo framework pode ser alterado pelo desenvolvedor, caso deseje acrescentar alguma definição que seja específica da linguagem escolhida. Para o nosso exemplo nada foi acrescentado. No final da fase TSM é gerado o código da transformação na linguagem escolhida.

3.2.2.5 Fase Código Nesta fase o código gerado para a transformação pode ser complementado e testado. Para a transformação *OO2RDBMS* não foi necessário acrescentar nenhuma linha de código, apenas realizar os testes desejados. A Figura 3.12 ilustra parte do código gerado automaticamente pelo framework para a transformação desenvolvida. O código gerado contém todo o cabeçalho da transformação (com os metamodelos envolvidos) e as regras definidas. Também são acrescentados comentários de acordo com o detalhamento especificado durante a fase de requisitos.

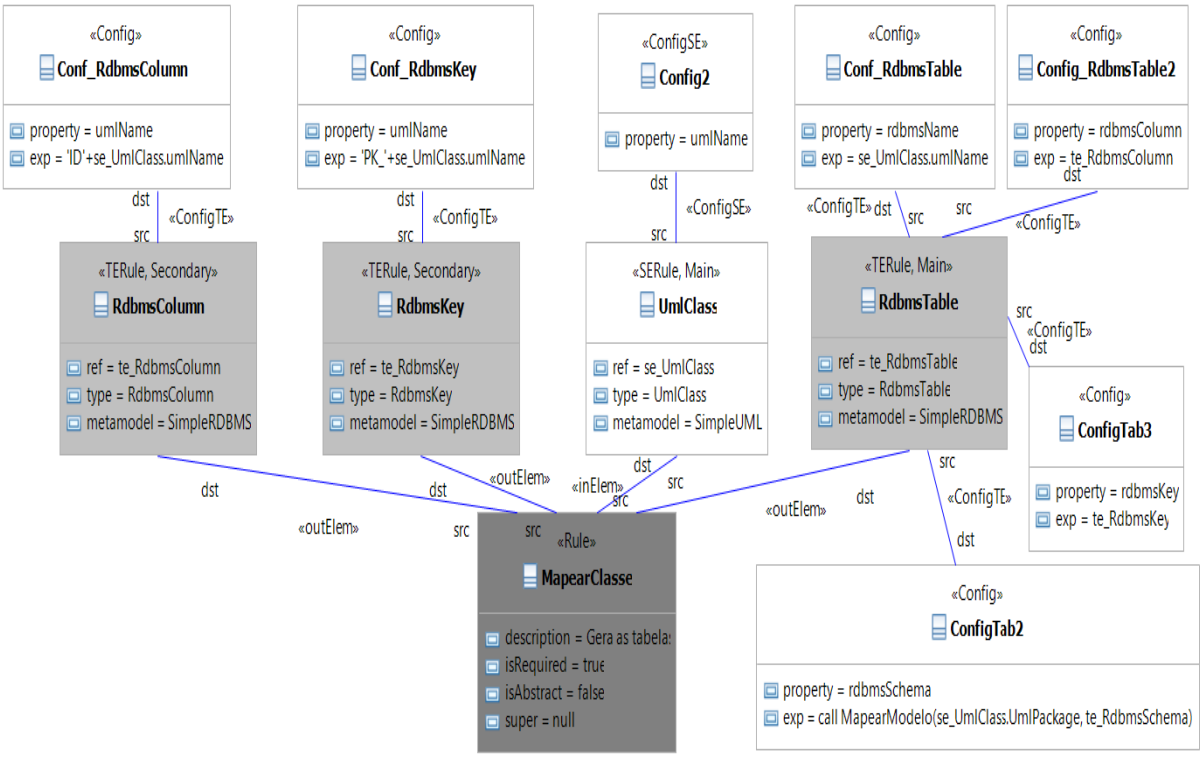


Figura 3.10 Parte do projeto de baixo nível da regra MapearClasse

3.3 CONSIDERAÇÕES SOBRE O CAPÍTULO

Este capítulo apresentou o framework MDTD para desenvolvimento de transformações de modelos unidirecionais na abordagem DDM. O framework compreende diversos elementos considerados relevantes para viabilizar o desenvolvimento de transformações na abordagem DDM. Mais que isso, integra esses elementos em uma solução única que compreende linguagens, processo e recursos de automação.

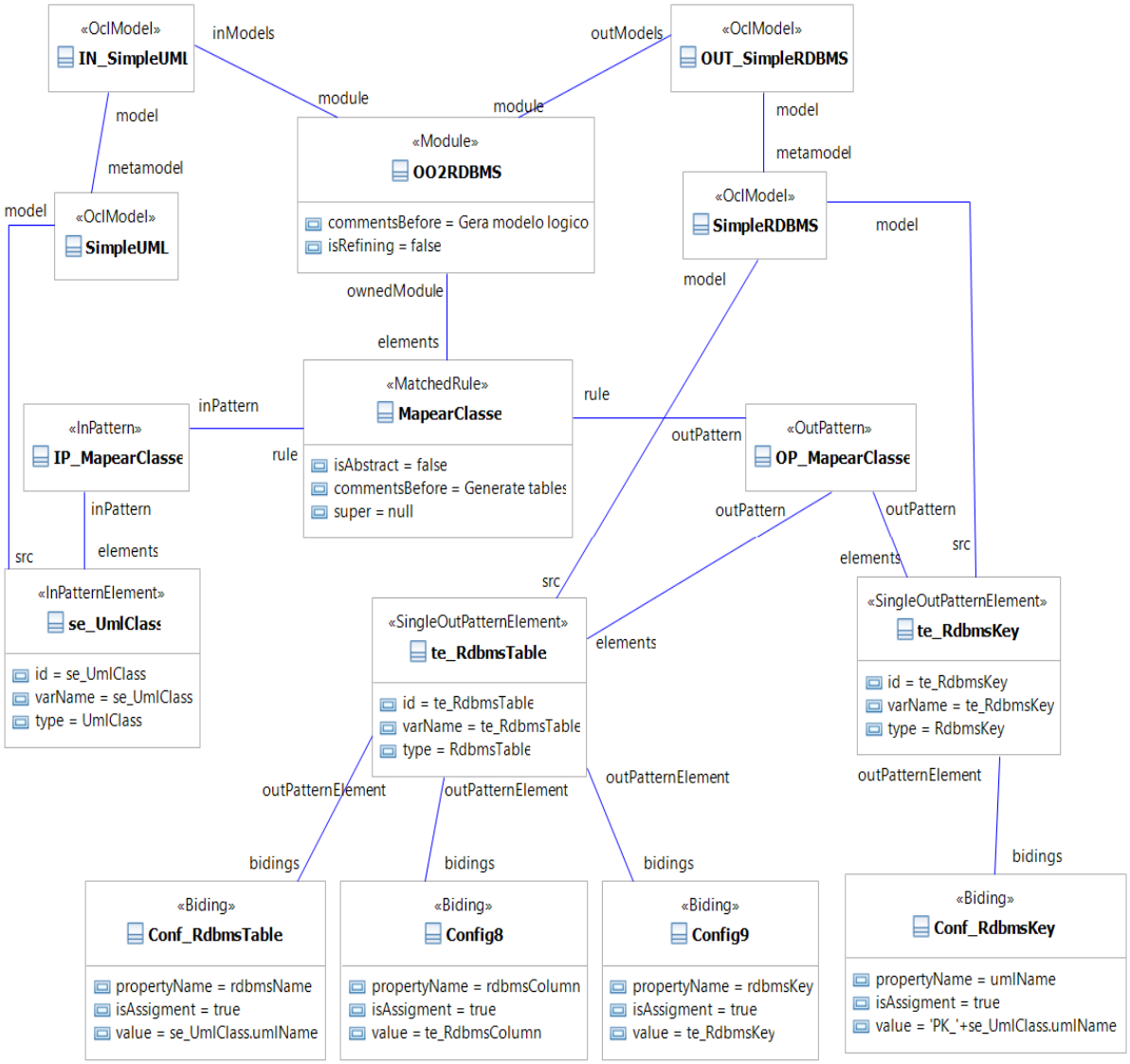


Figura 3.11 Parte do modelo de projeto específico em ATL

```
1.  -- *** Gera modelo logico ***
2.  module OO2RDBMS;
3.  --*** modelos de saida e de entrada ***
4.    create OUT_SimpleRDBMS:SimpleRDBMS   from IN_SimpleUML:SimpleUML;
5.  -- *** regras da transformação ***
6.    -- *** Cria tabelas ***
7.   rule MapearClasse{
8.   from
9.         se_UmlClass:SimpleUML!UmlClass
10.   to
11.       te_RdbmsColumn:SimpleRDBMS!RdbmsColumn(
12.       rdbmsKey<-te_RdbmsKey,
13.       rdbmsOwner<-te_RdbmsTable,
14.       rdbmsName<-'ID_'+se_UmlClass.umlName,
15.       rdbmsType<-'INTEGER'
16.       ),
17.       te_RdbmsTable:SimpleRDBMS!RdbmsTable(
18.       rdbmsName<-se_UmlClass.umlName,
19.       rdbmsSchema<-thisModule.resolveTemp(se_UmlClass.umlNamespace,'te_RdbmsSchema')
20.       ),
21.       te_RdbmsKey:SimpleRDBMS!RdbmsKey(
22.       rdbmsOwner<-te_RdbmsTable,
23.       rdbmsName<-'PK_'+se_UmlClass.umlName
24.       )
25.  } ....
```

Figura 3.12 Parte do código da transformação OO2RDBMS em ATL gerada pelo framework

Capítulo

4

LINGUAGEM DE MODELAGEM PARA O DOMÍNIO DE TRANSFORMAÇÕES DE MODELOS

O uso da DDM para desenvolver transformações requer a definição de uma linguagem de modelagem na qual os modelos de transformação possam ser instanciados. Este capítulo apresenta a linguagem de modelagem proposta nesta tese cuja definição compreende a sintaxe abstrata e a semantica estática da linguagem, especificadas sob a forma de metamodelo e regras OCL (Seção 4.1), e a sintaxe concreta da linguagem, especificada sob a forma de perfil UML (Seção 4.2). Esta linguagem possibilita a modelagem de transformações independentes de plataforma utilizando os diagramas da UML. Adicionalmente foram especificados também perfis para as linguagens ATL e QVT, que apoiam a construção de modelos específicos de plataforma, e podem ser encontrados nos apêndices G e H.

4.1 METAMODELO DE TRANSFORMAÇÃO (MMT)

O Metamodelo de Transformação (*MMT*), definido na camada M2 do modelo de camadas da OMG, está organizado em três níveis de abstração conforme ilustrado na Figura 4.1: *MMTspec*, para instanciar modelos de requisitos; *MMThighdesign*, para instanciar modelos de projeto de alto nível independente de plataforma; e *MMTlowdesign*, para instanciar modelos de projeto de baixo nível independente de plataforma. Esses metamodelos estão especificados em conformidade com a metametalinguagem MOF (camada M3 da OMG) e serão detalhados nas subseções a seguir.

O MMT foi especificado utilizando-se como base os metamodelos das liguagens ATL, amplamente utilizada pela comunidade de desenvolvimento de transformações e QVT-Relation, padão OMG para especificação de transformações, e em consonância com a taxonomia de transformações proposta por (MENS; CZARNECKI; GORP, 2006). Os construtores das linguagens ATL e QVT foram analisados e abstraídos em uma linguagem de modelagem independente de plataforma. Esta estratégia visa atingir um nível de cobertura aceitável para a linguagem aqui proposta. A validação da linguagem pode ser encontrada na Seção 7.3.

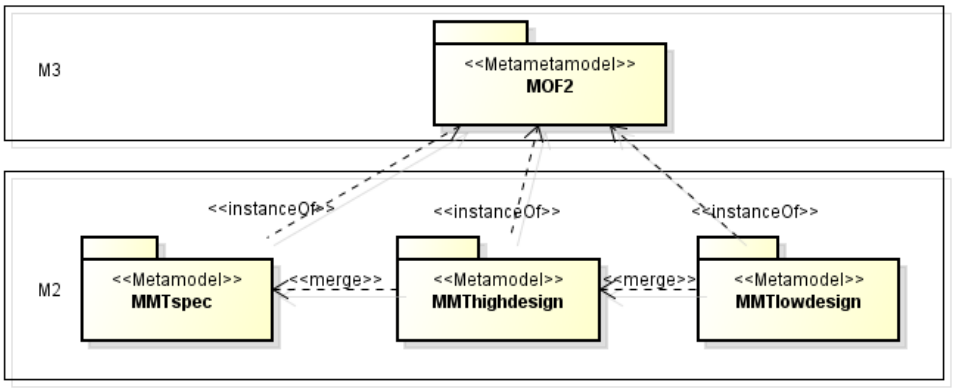


Figura 4.1 Esquema de representação dos metamodelos utilizados pela abordagem proposta nesta tese, nos diversos níveis de abstração

4.1.1 Metamodelo para Especificação de Transformação (MMTspec)

O objetivo do metamodelo MMTspec é definir a gramática da linguagem de modelagem no nível de especificação de requisitos da transformação. Para isso, procuramos especificar neste metamodelo as seguintes características do domínio de transformações: (i) os requisitos funcionais da transformação, ou seja, o comportamento que a transformação deverá contemplar para alcançar seu objetivo; (ii) seleção dos metamodelos envolvidos na transformação; e (iii) propriedades que a transformação deve atender, que são utilizadas para a verificação semântica da transformação.

Definimos, portanto, uma *Especificação de Transformação (ET)* como uma tupla *ET* = <*Req, MMf, MMa, P*> em que *Req* é o conjunto de requisitos que representam as necessidades de automação que a transformação deve contemplar, *MMf* e *MMa* representam respectivamente o conjunto de metamodelos fonte e alvo que participam da transformação e *P* é o conjunto de propriedades da transformação.

O conceito de Especificação de Trasformação que adotamos pode ser visto como uma abstração do conceito de *contrato* proposto por Braga (2014), onde uma transformação compreende relações entre metamodelos fonte e alvo e propriedades da transformação. No *MMTspec* não é possível definir relações entre metamodelos porque os metamodelos ainda não foram identificados. É necessário primeiramente entender o que precisa ser transformado, isto é identificar os requisitos, para então selecionar e/ou definir os metamodelos apropriados. Consideramos, portanto, os requisitos como abstrações de relações, pois eles são fortes candidatos a serem refinados como tal. Adicionalmente, há também uma preocupação já neste nível de abstração em definir propriedades que possam ser depois refinadas para uma notação formal e utilizadas na verificação da correção semântica da transformação.

Os construtores necessários para representar uma *Especificação de Transformação* estão definidos no metamodelo *MMTspec* conforme ilustrado na Figura 4.2 em formato visual e no apêndice F em formato textual (implementado na metametalinguagem Ecore).

No metamodelo uma *TransformationSpecification* compreende o conjunto dos requisitos (*Requirements*) que a transformação deve implementar e está associado a metamodelos

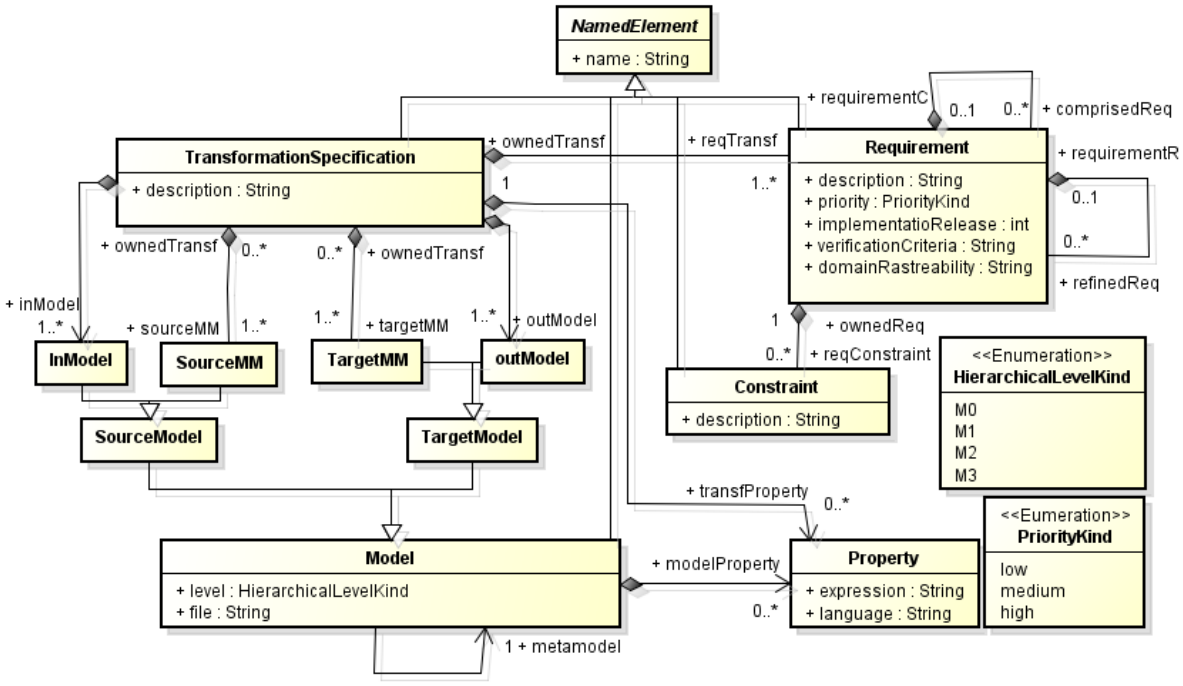


Figura 4.2 Metamodelo *MMTspec* em formato visual

fonte (*SourceMM*) e alvo (*TargetMM*) e a modelos de entrada (*InModel*) e a modelos de saída (*OutModel*), podendo conter zero ou muitas propriedades (*Property*). Toda *TransformationSpecification* possui um nome (herdado de um *NamedElement*) e contém uma descrição (atributo *description*) que indica qual o propósito da transformação que será desenvolvida. O conceito *NamedElement* representa um elemento qualquer de um modelo que possui um nome (atributo *name*). Ele é definido em uma classe abstrata e foi criado para ser reutilizado pelos demais conceitos.

Os requisitos (*Requirement*) de uma transformação representam as funcionalidades que a transformação deve implementar para automatizar um processo de desenvolvimento. Todo *Requirement* possui um nome (herdado de *NamedElement*) e uma descrição (*description*) que define o objetivo do requisito. Requisitos podem ser desenvolvidos em versões de forma incremental. O atributo *implementationRelease* registra a versão em que cada requisito será desenvolvido. Essas versões são definidas de acordo com o nível de prioridade estabelecido para o requisito. O nível de prioridade (*priority*) pode ser estabelecido como *alto*, *médio* ou *baixo* de acordo com a *Enumeration PriorityKind (high, midium, low)*. Para cada requisito é também definido o critério de verificação que será utilizado para validação do mesmo ao longo do desenvolvimento (atributo *verificationCriteria*). Finalmente, o atributo *domainRastreability* guarda a rastreabilidade entre os requisitos do domínio (ver processo descrito no Capítulo 5) e os requisitos da transformação aqui especificados. Requisitos podem ser refinados em outros requisitos (relacionamento *refinedReq*) e também podem compreender outros requisitos (relacionamento *comprisedReq*).

Restrições (*Constraints*) também podem ser relacionadas aos requisitos indicando

condições (booleanas) que devem ser aceitas pela transformação. Uma *Constraint* contém um nome (herdado de *NamedElement*) e uma descrição (atributo *description*) que neste nível de abstração é registrada em linguagem natural.

A especificação de uma transformação também requer a identificação dos metamodelos fonte (*sourceMM*) e os metamodelos alvo (*targetMM*) envolvidos na transformação. Estes metamodelos são identificados de acordo com as recomendações definidas no processo proposto (descrito no capitulo 5). É possível que o *sourceMM* e o *targetMM* representem o mesmo metamodelo (transformações endógenas) ou metamodelos diferentes (transformações exógenas).

Um *Model* representa um modelo que participa da transformação. Cada *Model* possui um nome (herdado de *NamedElement*) e está localizado em um nível hierárquico (atributo *level*), segundo o modelo de camadas da OMG. São exemplos de *Model* um metametamodelo (ex. *name MOF, level M3*), um metamodelo (ex. *name UML, level M2*), o modelo de uma transformação qualquer (que estará no *level M1*). No contexto de uma *TransformationSpecification*, um *Model* pode ser de dois tipos *SourceModel* ou *TargetModel*. *SourceModel* são modelos de entrada da transformação (ex. um metamodelo fonte). *TargetModel* são modelos de saída da transformação (ex. um metmodelo alvo).

Tanto um *Model* quanto uma *TransformationSpecification* pode ter um conjunto de propriedades (*Property*) associadas. As propriedades são definidas neste nível em linguagem natural usando o atributo *expression* de *Property*. Em fases posteriores do processo de desenvolvimento elas poderão ser reescritas em linguagens formais para serem utilizadas para verificação dos modelos construídos.

A especificação do metamodelo *MMTspec* contem regra OCL para garantir a conformidade entre os modelos de diferentes níveis hierárquicos (Figura 4.3). A regra é aplicada ao contexto *TransformationSpecification* e define duas invariantes, *ModelMetaMConform1* e *ModelMetaMConform2*, que garantem que os metamodelos fonte (*sourceMM*) e alvo (*targetMM*) estão no nível M2 do modelo de camadas da OMG.

```
context TransformationSpecification
  inv ModelMetaMConform1: self.sourceMM.level = HierarchicalLevelKind::M2
  inv ModelMetaMConform2: self.targetMM.level = HierarchicalLevelKind::M2
```

Figura 4.3 Regras OCL para garantir conformidade entre níveis hierárquicos de modelos no contexto de *TransformationSpecification*

4.1.2 Metamodelo de Projeto de Alto Nível da Transformação (MMThighdesign)

O conceito de *Especificação de Transformação* inicialmente introduzido no *MMTspec* é agora refinado, sob a perspectiva de projeto de alto nível, no conceito de *Transformação de Modelos*.

Uma *Transformação de Modelo* (*TM*) é uma tupla $TM = <MMf, MMa, R, P>$ em que *MMf* e *MMa* representam os metamodelos fonte e alvo respectivamente, R é um conjunto de relações e P um conjunto de propriedades. Cada relação R representa um mapeamento unidirecional entre elementos dos metamodelos fonte (*Ef*) e elementos dos

metamodelos alvo (*Ea*). Vários tipos de relações podem ser especificadas em *R*: um para um; um para muitos; muitos para um; muitos para muitos; zero para um; zero para muitos; um para zero e muitos para zero.

Os construtores necessários para representar uma transformação de modelos no projeto de alto nível estão representados no metamodelo *MMThighdesign* na Figura 4.4 em formato visual e no apêndice F no formato textual implementado na metametalinguagem Ecore.

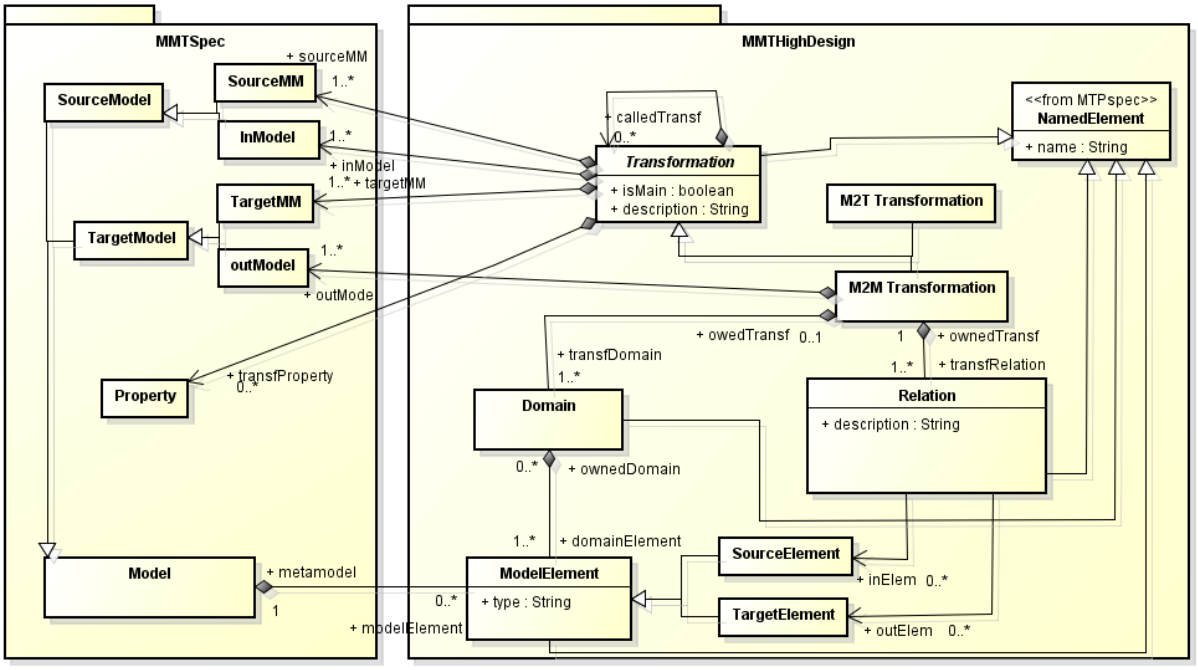


Figura 4.4 Metamodelo *MMThighdesign* em formato visual

O metamodelo *MMThighdesign* reusa definições do metamodelo *MMTspec*, tais como, *NamedElement*, *Model*, *SourceMM*, *TargetMM*, *ModelElement* e *Property*. Por exemplo, *Transformation* é uma subclasse de *NamedElement*, conceito reusado do metamodelo *MMTspec*.

No metamodelo *MMThighdesign* uma transformação de modelos (*M2MTransformation*) é um tipo de transformação (*Transformation*) composta por metamodelos fonte (*SourceMM*), metamodelos alvo (*TargetMM*), relações (*Relation*) e propriedades (*transfProperty*.

Uma transformação de modelo recebe um ou mais modelos de entrada (*inModel*) e gera um ou mais modelos de saída (*outModel*). Uma *Relation* relaciona zero ou mais elementos do metamodelo fonte (*SourceElement*) a zero ou mais elementos do metamodelo alvo (*TargetElement*). *SourceElement* e *TargetElement* são tipos de *ModelElement* que representa os elementos de um modelo. Todo *ModelElement* tem um nome (herdado de *NamedElement*) e um tipo (atributo *type*).

Uma *M2MTransformation* contem domínios (*Domain*) que definem quais elementos dos metamodelos são contemplados pela transformação. A definição do domínio da

transformação está relacionada à propriedade de completude de uma transformação. No contexto do *MMThighdesign*, uma transformação é dita completa se para cada elemento definido no *Domain* existir uma *Relation* ao qual o elemento participa.

Transformações (*Transformation*) podem ser compostas por outras transformações (relacionamento *calledTransf*) possibilitando a composição de transformações e consequentemente o reuso das mesmas. Desta forma é possível utilizar transformações já existentes para compor novas transformações. Este trabalho utiliza o conceito de composição externa proposto por (WAGELSAR, 2008). Nesta, a composição é representada pelo encadeamento de transformações independentes ligadas através dos modelos produzidos como saída. Assim, o modelo de saída de uma transformação é utilizado como entrada na transformação seguinte.

As regras de boa formação que não puderam ser representadas no metamodelo *high-design* estão especificadas em linguagem OCL sob a forma de invariantes e ilustradas na Figura 4.5 e 4.6.

Para o contexto *Model* (Figura 4.5) foram definidas duas invariantes. A primeira, *M2M3ConformanceElement*, visa garantir que metamodelos são instâncias de metametamodelos, ou seja, para todo metamodelo (*hirarchicalLevelKind M2*) os elementos nele usados são instâncias dos elementos do metametamodelo a ele associado. Analogamente, a segunda invariante, *M1M2ConformanceElement*, define que modelos são instâncias de metamodelos.

```
context Model
inv M2M3ConformanceElement:
  if(self.level = hierarchicalLevelKind::M2) then
     self.modelElement -> forAll (x : ModelElement | self.metamodel.modelElement -> exists (y : ModelElement | y.name=x.type))
  else true
inv M1M2ConformanceElement:
  if(self.level = hierarquicalLevel::M1) then
     self.modelElement -> forAll (x : ModelElement | self.metamodel.modelElement -> exists(y : ModelElement | y.name = x.type) )
  else true
```

Figura 4.5 Regras OCL para garantir conformidade entre níveis hierárquicos de modelos no contexto de *Model*

Para o conexto *Relation* foram definidas duas invariantes. A primeira, (*inv SEConformanceSMM*), tem como objetivo garantir que os elementos utilizados como entrada da relação (*sourceElem*) pertencem aos metamodelos fonte definidos para a transformação (*sourceMM*). Analogamente, a segunda invariante, (*inv TEConformanceTMM*), garante que os elementos de saída da relação (*targetElem*) pertencem aos metamodelos alvo da transformação (*targetMM*).

4.1.3 Metamodelo de Projeto de Baixo Nível da Transformação (MMT-lowdesign)

Sob a perspectiva do projeto de baixo nível, esta tese define transformação de modelos de maneira análoga a (KLEPPE; WARMER; BAST, 2003) como um conjunto de regras, que compõem a transformação. Cada regra é uma tripla $REG = <Ef,\ Ea,\ Cond>$ onde *Ef* são os elementos de entrada da regra, *Ea* os elementos de saída da regra e *Cond* um

```
context Relation
inv SEConformanceSMM:
 self.inElem->exists ( x : ModelElement | self.ownedTransf.sourceMM->exists (y: Model | x.metamodel = y) )
inv TEConformanceTMM:
 self.outElem->exists ( x : ModelElement | self.ownedTransf.targetMM->exists (y: Model | x.metamodel = y) )
```

Figura 4.6 Regras OCL para garantir que os elementos selecionados pertencem aos metamodelos envolvidos na transformação

conjunto de condições booleanas que devem ser avaliadas como verdadeiras para que a regra seja executada.

Conforme conceituado por Kleppe (2003), o objetivo de cada regra é definir como um ou mais construtores em uma linguagem fonte podem ser transformados em um ou mais construtores em uma linguagem alvo. No *MMTlowdesign* isto é definido através da configuração das propriedades que compõem os elementos de saída da regra, isto é, cada elemento de saída da regra tem suas propriedades (atributos) configuradas indicando como elas devem ser inicializadas no modelo de saída. Neste ponto vale ressaltar que quando no projeto de alto nível uma *Relation* é definida sem nenhum elemento de saída (*Relation* do tipo 1-0, ou N-0), a definição da *Rule* não é necessária. Esses casos são modelados no projeto de alto nível somente para possibilitar a verificação da propriedade de completude da transformação[1] , mas não demandam o detalhamento da *Relation* em uma *Rule*, pois de fato nenhum código será gerado para a *Relation*.

Os construtores necessários para representar uma *Transformação de Modelos* no projeto de baixo nível estão representados no metamodelo *MMTlowdesign* na Figura 4.7 em formato visual e no apêndice F no formato textual implementado na metametalinguagem Ecore.

Conforme ilustrado no metamodelo, uma regra (*Rule*) possui nome (atributo *name* herdado de *NamedElement*) e descrição (atributo *description*), com um comentário sobre o propósito da regra, e pode ser abstrata (atributo *isAbstract* com valor *true*) ou concreta (atributo *isAbstract* com valor *false*). *Rule* abstrata é utilizada como padrão para criação de outras regras e possibilita o reuso de definições comuns através de herança. O relacionamento *Super* indica, para o caso de regras que herdem características de outras regras, qual a *Rule* superior (pai). Uma *Rule* também pode ser requerida ou não requerida (atributo *isRequired*). Quando *isRequired* assume o valor *true*, a *Rule* será sempre executada se no modelo de entrada da transformação existir alguma ocorrência do(s) elemento(s) de entrada da regra (ex. *matched rule da ATL* ou *Top Relation da QVT*). Caso contrário, se o atributo *isRequired* assumir o valor *false*, a *Rule* será executada apenas quando for chamada por outra *Rule* (ex. *called rule* da ATL ou cláusula *where* na QVT).

A execução de uma *Rule* pode estar condicionada por um conjunto de *Condition*. Uma *Condition* representa uma expressão booleana que deve ser avaliada como *true* para que a *Rule* seja executada. *Condition* é definida através do atributo *exp* utilizando a

[1] A definição de Relations dos tipos 1-0 ou N-0 é importante para possibilitar a verificação da propriedade de completude, pois indica que os elementos de entrada foram considerados na especificação da transformação e não geram nenhum elemento como saída.

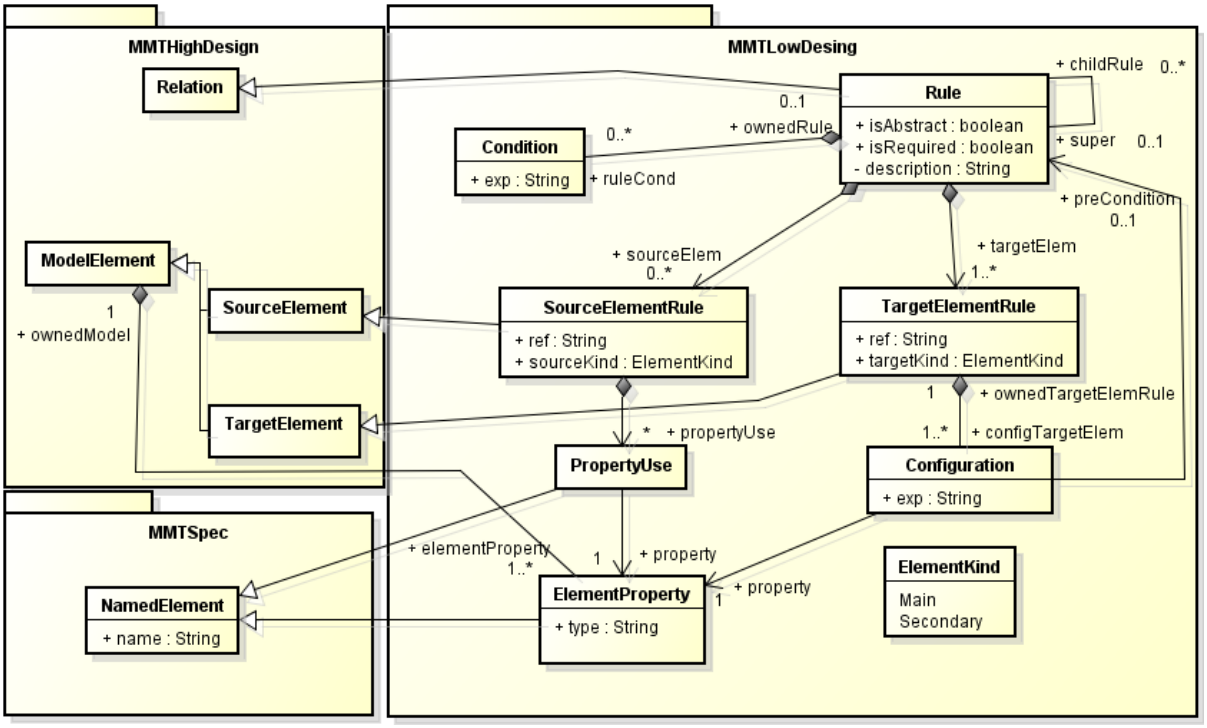


Figura 4.7 MMTlowdesign com a sintaxe abstrata do projeto de baixo nível

linguagem OCL.

Voltando ao metamodelo da Figura 4.7, o *SourceElementRule* e o *TargetElementRule* representam respectivamente os elementos de entrada e saída da regra. *SourceElementRule* e *TargetElementRule* são subclasses de *SourceElement* e *TargetElemet* respectivamente, portanto, possuem os atributos *name*, *metamodel* e *type* herdados de suas superclasses. Possui também o atributo *ref*, uma variável que é utilizada para fazer referência a esse elemento e será útil quando o desenvolvedor for definir a configuração dos elementos de saída da regra. Considerando que podem existir n elementos de entrada e m elementos de saída em uma *Rule*, é possível definir qual o elemento principal (de entrada e de saída) da regra e quais os elementos secundários (associados ao elemento principal), relevantes caso o código seja gerado em linguagens bidirecionais. Essa definição é feita na propriedade *sourceKind* e *targetKind*, do *SourceElementRule* e *TargetElementRule* respectivamente, com base no *Ennumeration* <<*ElementKind* >> que define os valores *Main* para o elemento principal e *Secondary* para os elementos secundários.

Cada elemento de saída da regra *TargetElementRule* deve ser configurado indicando como suas propriedades (*ElementProperty*) serão inicializadas quando o elemento for criado. Se o desenvolvedor não definir a configuração quando o modelo de saída for criado, as propriedades do elemento serão inicializadas como nulas. Desta forma, para cada *TargetElementRule* podem ser definidas várias configurações (*Configuration*). Pelo menos uma configuração deve obrigatoriamente ser definida (ex. configurar a propriedade *name* do *TargetElementRule* a ser criado). Configurações possuem nome (herdado de *NamedElement*), fazem referência a uma propriedade do elemento de saída (*ElementProperty*)

e possuem uma expressão (atributo *exp*). O atributo *exp* define a inicialização da propriedade que está sendo configurada e pode assumir os seguintes valores: uma expressão na linguagem OCL; uma propriedade de um elemento de entrada da regra (*SourceElementRule.ElementProperty.name*); ou uma referência a um elemento gerado na própria regra (*TargetElementRule.ref*). Em casos especiais pode ser preciso inicializar uma propriedade utilizando um elemento gerado em outra regra na mesma transformação. Nestes casos, a inicialização da propriedade depende da execução de outra regra. O MMTlowdesign provê neste caso uma chamada a outras regras a partir da instrução *call RegraChamada(param1,param2)*, onde *RegraChamada* corresponde ao nome da regra que se deseja chamar, *param1* corresponde ao elemento de entrada da regra chamada e *param2* corresponde ao elemento de saída gerado pela regra chamada.

Ainda em relação à Figura 4.7 , o conceito *PropertyUse* é utilizado para definir quais as propriedades dos elementos de entrada da regra (*SourceElementRule*) que podem ser utilizadas para definir a inicialização dos elementos de saída da regra.

4.2 PERFIL PARA TRANSFORMAÇÃO DE MODELOS (MTP)

Um perfil UML é um mecanismo de extensão para definição de linguagens específicas de domínio usando os conceitos da UML[2] . O perfil para transformação de modelos (*Model Transformation Profile - MTP*) definido neste trabalho é uma extensão da UML para o domínio de transformação de modelos.

O *MTP* contém estereótipos e valores rotulados definidos com base nos construtores dos metamodelos *MMTspec*, *MMThighdesign* e *MMTlowDesign* apresentados nas seções anteriores. Portanto, analogamente à definição destes metamodelos, o perfil está dividido em três pacotes, o *MTPspec*, o *MTPhighdesign* e o *MTPlowdesign* referentes a cada um dos níveis de abstração dos metamodelos, conforme apresentado nas subseções a seguir.

Para ilustrar o uso dos perfis aqui propostos será utilizado o mesmo exemplo apresentado na Seção 3.2.1, a transformação de um modelo de classes da UML para um modelo lógico de banco de dados, chamada de transformação *OO2RDBMS*.

4.2.1 Perfil MTPspec

O perfil MTPspec está definido no pacote *MTPspec*, ilustrado na Figura 4.8.

O pacote *MTPspec* contém estereótipos, correspondentes aos conceitos do metamodelo *MMTspec*, e metaclasses estendidas da UML (em cinza no centro). O nome dos estereótipos foi abreviado em relação ao metamodelo por restrição da ferramenta utilizada para implementação. A tabela 4.1 apresenta a correspondência entre os elementos do metamodelo e os estereótipos criados, além da metaclasse e do diagrama em que o estereótipo pode ser aplicado durante o desenvolvimento. Por exemplo, o conceito *TransformationSpecification* do metamodelo *MMTspec* corresponde ao estereótipo <<*TransSpec*>>do perfil *MTPspec*, aplicado à metaclasse *Actor* do diagrama de casos de uso da UML. São utilizados também as associações de <<*include*>>e <<*extend*>>, para representar requisitos que compreendem outros requisitos e a associação de generalização,

[2]www,omg.org/spec/UML/2.5/

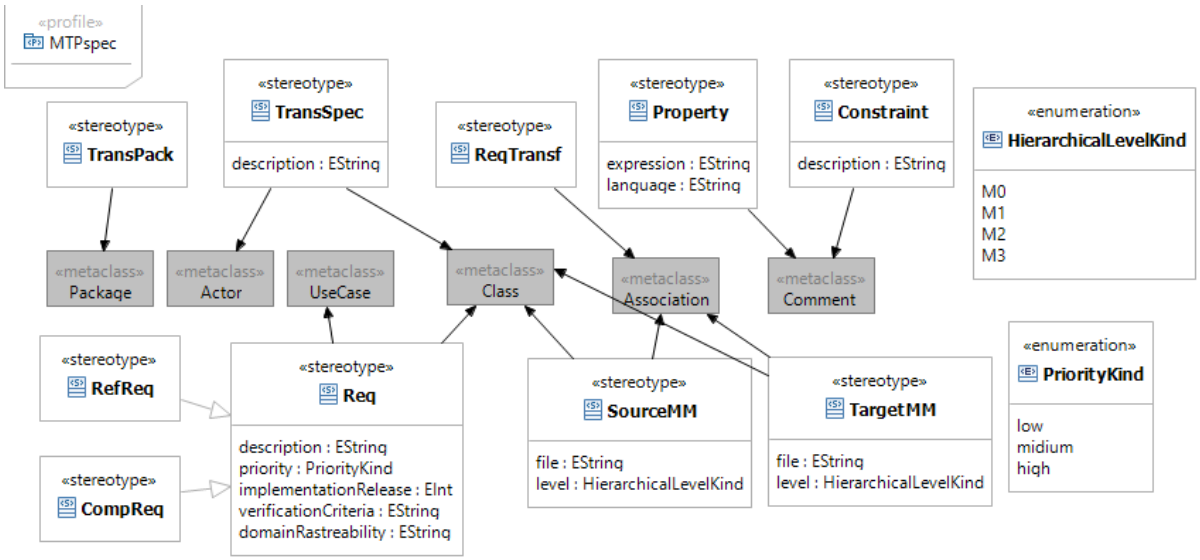


Figura 4.8 Pacote *MTPspec* com estereótipos e metaclasses

para refinar requisitos.

Tabela 4.1 Elementos do metamodelo *MMT* e estereótipos do perfil *MTPspec* com suas respectivas metaclasses

Metamodelo MMTspec	Estereótipo MTPspec	Metaclasse	Diagrama
TransformationSpecification	TransSpec	Actor	Caso de uso
Requirement	Req	UseCase	Caso de uso
Requirement	Req	Class	Classes
ReqTransf	ReqTransf	Association	Caso de uso Classes
Property	Property	Comment	Caso de uso Classes
Constraint	Constraint	Comment	Caso de uso Classes
RefinedReq	RefReq	UseCase	Caso de uso
RefinedReq	RefReq	Class	Classes
ComprisedReq	CompReq	UseCase	Caso de uso
ComprisedReq	CompReq	Class	Classes
TargetMM	TargetMM	Class	Classes
SourceMM	SourceMM	Class	Classes

4.2.1.1 Exemplo de Uso do Perfil MTPspec

Para a transformação *OO2RBDMS*, utilizada como exemplo, é preciso transformar um modelo de classes em um modelo lódigo de banco de dados. Para isso precisamos converter as classes em tabelas e seus atributos em campos. Adicionalmente, as tabelas devem conter um identificador que represente a

sua chave primária. Além disso, é preciso considerar que a depender do tipo do atributo, monovalorado ou multivalorado, a conversão deste para o modelo lógico acontecerá de maneira diferente. Finalmente, classes abstratas não devem ser convertidas em tabelas.

A Figura 4.9 apresenta os requisitos da transformação OO2RDBMS especificados em um diagrama de casos de uso estereotipado de acordo com o perfil MTPspec.

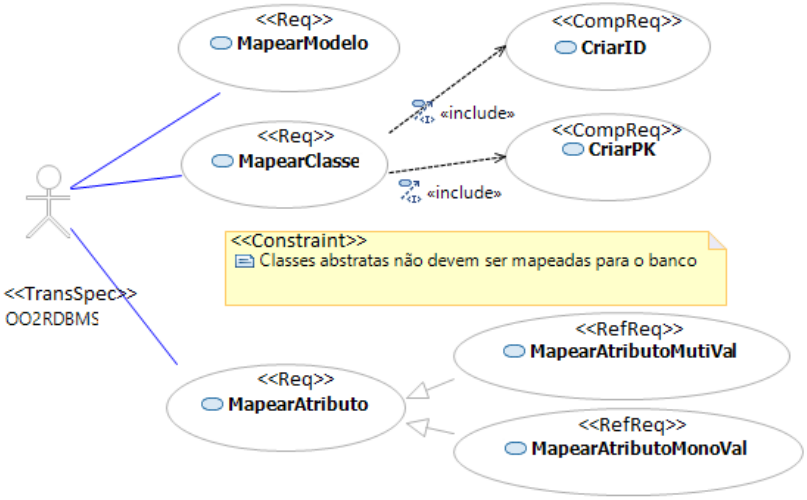


Figura 4.9 Exemplo de diagrama de casos de uso de requisitos

A transformação, representada pelo ator, contém três requisitos, representados pelos casos de uso, *MapearModelo*, para criar o modelo lógico, *MapearClasse*, para criar as tabelas, e *MapearAtributo*, para criar os campos. O requisito *MapearClasse* compreende dois outros requisitos *CriarID* e *CriarPK* (casos de uso incluídos) assim como o requisito *MapearAtributo* é refinado nos requisitos *MapearAtributoMultiVal* e *MapearAtributoMonoVal* (casos de uso especializados). Há um restrição definida em uma nota associada ao requisito *MapearClasse* que indica que classes abstratas não serão transformadas. Todos os elementos do diagrama de casos de uso estão estereotipados de acordo com o perfil *MTPspec*.

A Figura 4.10 mostra um exemplo do diagrama de classes que também pode ser utilizado para especificação dos requisitos da transformação.

O diagrama apresenta os mesmos requisitos do diagrama de casos de uso, porém agora representados como classes estereotipadas. Esse diagrama é gerado automaticamente pela cadeia de transformações (Seção **??**) que automatiza o processo de desenvolvimento MDTDproc. Para cada classe alguns atributos foram adicionados conforme a definição do metamodelo da linguagem (*MMTspec*), como por exemplo o atributo *description*. Classes para representar o metamodelo fonte e para o metamodelo alvo foram criadas. Novas classes podem ser adicionadas quando necessário. Este diagrama contém um nível mais detalhado de informação, por exemplo, possibilita definir prioridade, versão e critério de verificação dos requisitos.

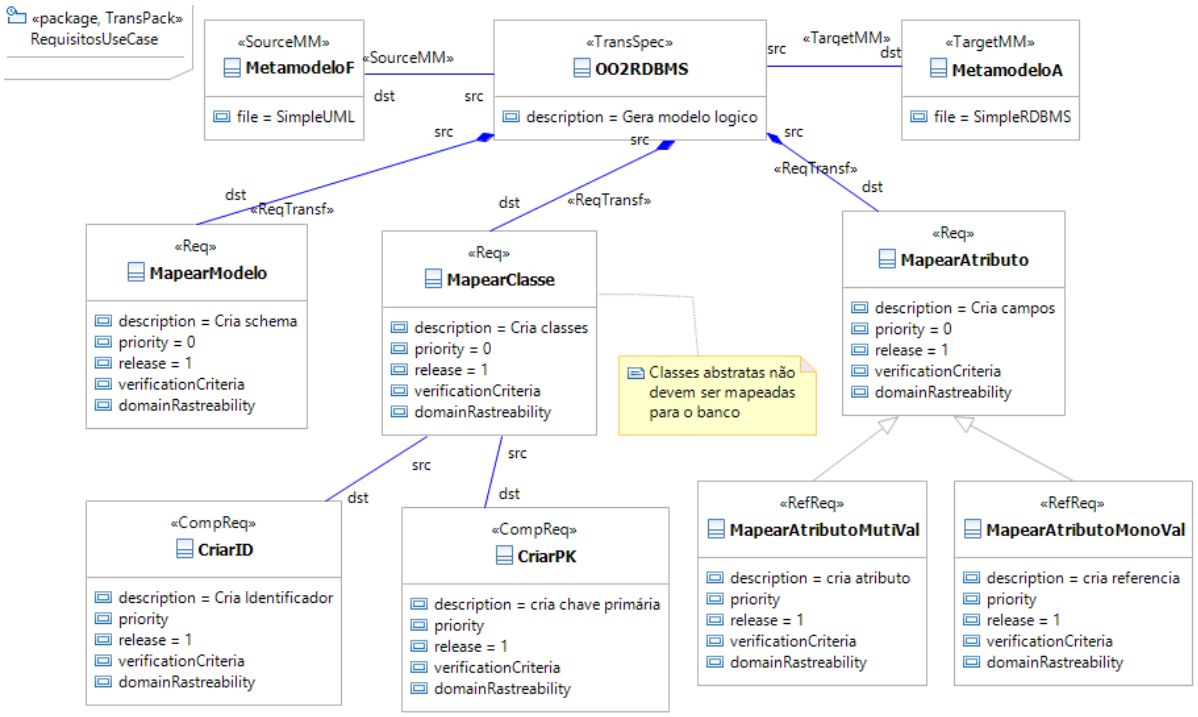


Figura 4.10 Exemplo de diagrama de classes de requisitos

4.2.2 Perfil MTPhighd

A Figura 4.11 ilustra a o pacote que especifica o perfil *MTPhighd*.

O pacote *MTPhighd* contém estereótipos, correspondentes aos conceitos do metamodelo *MMThighdesign*, e metaclasses estendidas da UML (em cinza no centro). Valores rotulados foram criados de acordo com o metamodelo *MMThighdesign*. Analogamente ao *MTPspec*, o nome de alguns estereótipos foi abreviado em relação ao metamodelo por imposição da ferramenta utilizada para implementação.

A tabela 4.2 apresenta a correspondência entre os elementos do metamodelo e os estereótipos criados, além das metaclasses estendidas e do diagrama em que o estereótipo pode ser aplicado durante o desenvolvimento. Por exemplo, o conceito *M2MTransformation* do metamodelo *MMThighdesign* corresponde ao estereótipo *<<M2MTransf>>*do perfil *MTPhighd* e pode ser aplicado a um componente, quando for construído um diagrama de componentes, ou a classes e pacotes quando for construído um diagrama de classes.

Como ilustrado na tabela, o perfil *MTPhighd* pode ser aplicado aos diagramas de componentes, para definir a arquitetura da transformação, a diagramas de atividades, para orquestrar as transformações e a diagramas de classes, para definir o relacionamentos entre elementos dos metamodelos fonte e alvo.

4.2.2.1 Exemplo de Uso do Perfil MTPhighd

A Figura 4.12 mostra um exemplo do diagrama de componentes com a arquitetura da transformação *OO2RDMBS*. No exemplo, a transformação é representada um componente composto por dois outros componen-

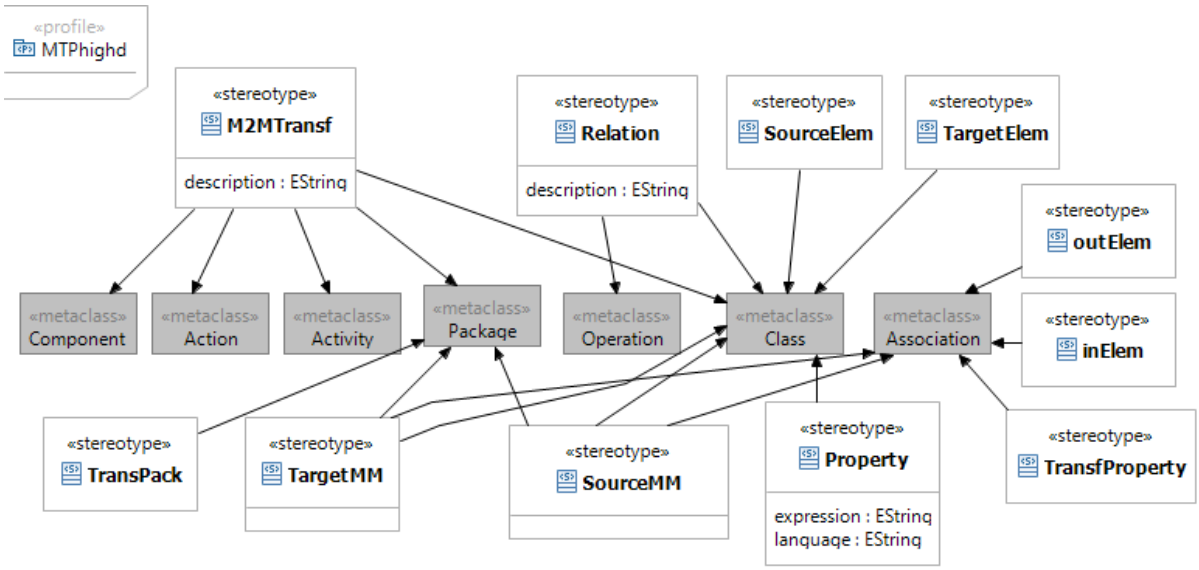


Figura 4.11 Pacote *MTPhighd* com estereótipos e metaclasses

Tabela 4.2 Conceitos do metamodelo MMT e estereótipos do perfil *MTPhighd*

Metamodelo MMThighdesign	**Estereótipo MTPhighd**	**Metaclasse**	**Diagrama**
M2MTransformation	M2MTransf	Component	Componente
M2MTransformation	M2MTransf	Package Class	Classes
M2MTransformation	M2MTransf	Activity Action	Atividades
Relation	Relation	Class	Classes
Relation	Relation	Operation	Componentes
Property	Property	Class	Classes
SourceElement	SourceElem	Class	Classes
TargetElement	TargetElem	Class	Classes
InElement	InElem	Association	Classes
OutElement	OutElem	Association	Classes
TransfProperty	TransfProperty	Association	Classes
SourceMM	SourceMM	Package Class Association	Classes
TargetMM	TargetMM	Package Class Association	Classes
	TransPack	Package	Todos os diagramas

tes, *GerarModeloConceitual* e *GerarModeloLogico*. O componente *OO2RDMBS* fornece uma interface, chamada no exemplo de *IExecucao*, com os serviços por ele prestados. Estes serviços são representados por um conjunto de operações, que no MTPhighd correspodem às relações definidas para a transformação. Para o nosso exemplo, temos as operações *MapearModelo*, *MapearClasse* e *MapearAtributo*. O componente *OO2RDBMS* também requer uma interface, no exemplo chamada de *IModelos*, que contem uma operação de leitura, responsável por carregar o modelo de entrada, e os metamodelos fonte e alvo, e uma operação de escrita, responsável por gerar o modelo de saída. Os componentes internos, *GerarModeloConceitual* e *GerarModeloLogico* estão conectados diretamente através de portas, indicando que o modelo gerado como saída da transformação *GerarModeloConceitual* será utilizado como modelo de entrada da transformação *GerarModeloLogico*. Neste caso essa comunicação é totalmente automática. Para os casos em que a comunicação entre os componentes internos é semi-automática, interfaces fornecidas e providas também devem ser definidas.

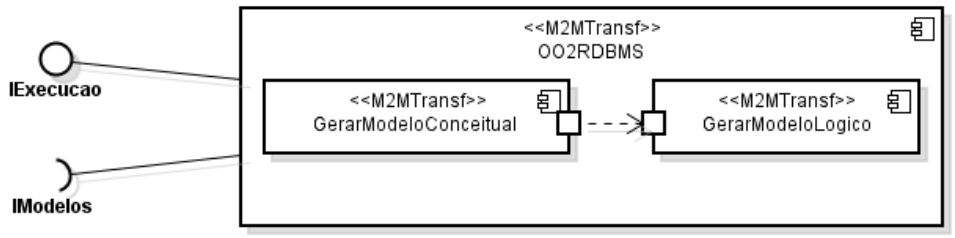


Figura 4.12 Exemplo de diagrama de componentes com a arquitetura da transformação OO2RDBMS

A Figura 4.13 mostra um exemplo do diagrama de atividades com a orquestração dos componentes definidos na arquitetura da transformação OO2RDMBS. No exemplo, o fluxo de controle entre os dois componentes é sequencial .

A Figura 4.14 mostra um exemplo do diagrama de classes com o mapeamento das relações existentes entre os metamodelos envolvidos na transformação OO2RDBMS.

O diagrama (Figura 4.14) contém três pacotes: o pacote *SimpleUML*, que contem os elementos do metamodelo fonte; o pacote *SimpleRDBMS*, que contém os elementos do metamodelo alvo; e o pacote *OO2RDBMS*, que representa a transformação e contém as relações (*Relation*) definidas. Para cada *Relation* é indicado quais os elementos de entrada e saída respectivamente. No exemplo, três *Relation* foram definidas. A primeira *MapearModelo*, recebe como entrada o elemento *UmlPackage* e gera como saída o elemento *RdbmsSchema*, ou seja, é uma relação do tipo 1-1. A segunda, *MapearClasse*, recebe como entrada uma *UmlClass* e gera como saída uma *RdbmsTable*, um *RdbmsColumn* e um *RdbmsKey*, ou seja, é uma relação do tipo 1-N. Finalmente a terceira, *MapearAtributo*, recebe um *UmlAttribute* como entrada e gera um *RdbmsKey* como saída, ou seja, é uma relação do tipo 1-1. Os elementos contidos nos pacotes que representam os metamodelos fonte e alvo, *SimpleUML* e *SimpleRDBMS* respectivamente, definem o domínio (*Domain*) da transformação.

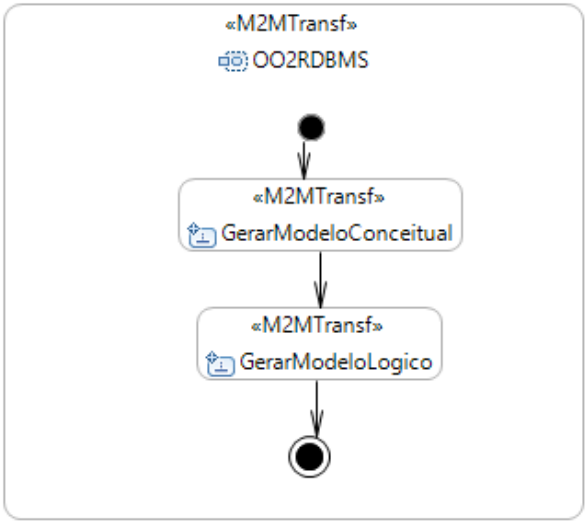


Figura 4.13 Exemplo de diagrama de atividades com a orquestração dos módulos que fazem parte da transformação OO2RDBMS

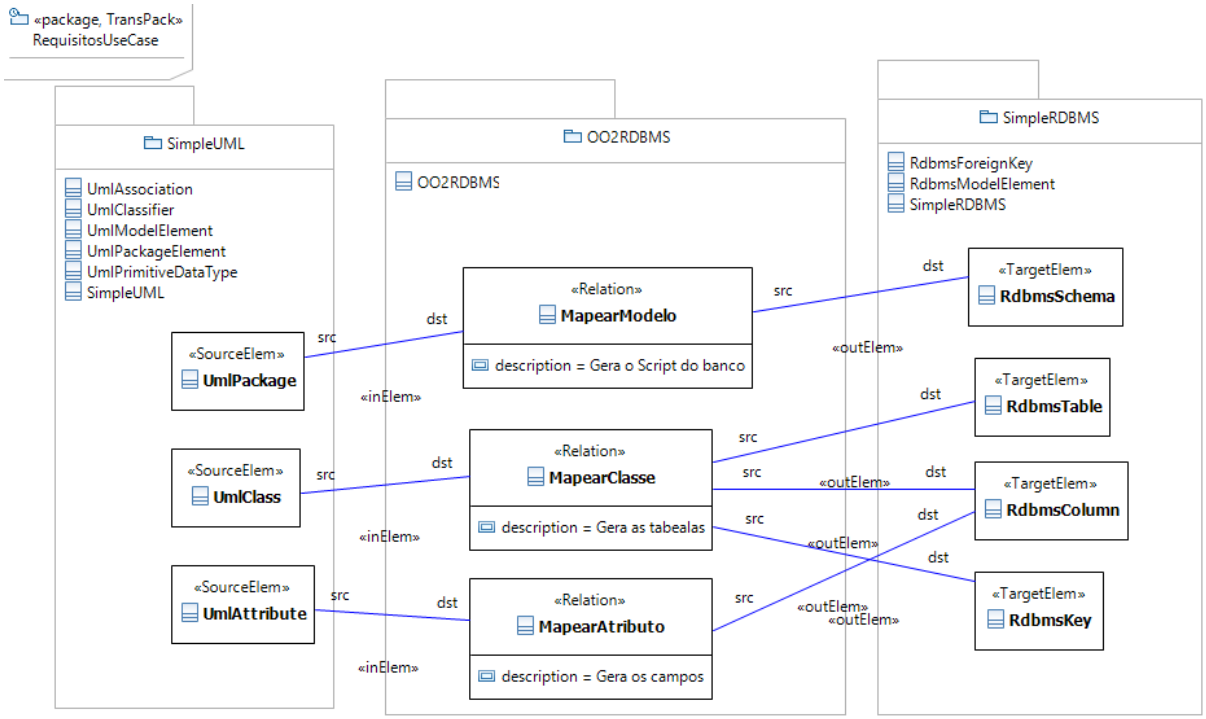


Figura 4.14 Exemplo de diagrama que define o relacionamento entre elementos do metamodelo fonte e alvo para a transformação OO2RDBMS

4.2.3 Perfil MTPlowd

A Figura 4.15 apresenta o pacote que especifica o perfil *MTPlowd*.

O pacote *MTPlowd* contém estereótipos que correspondem aos conceitos do metamo-

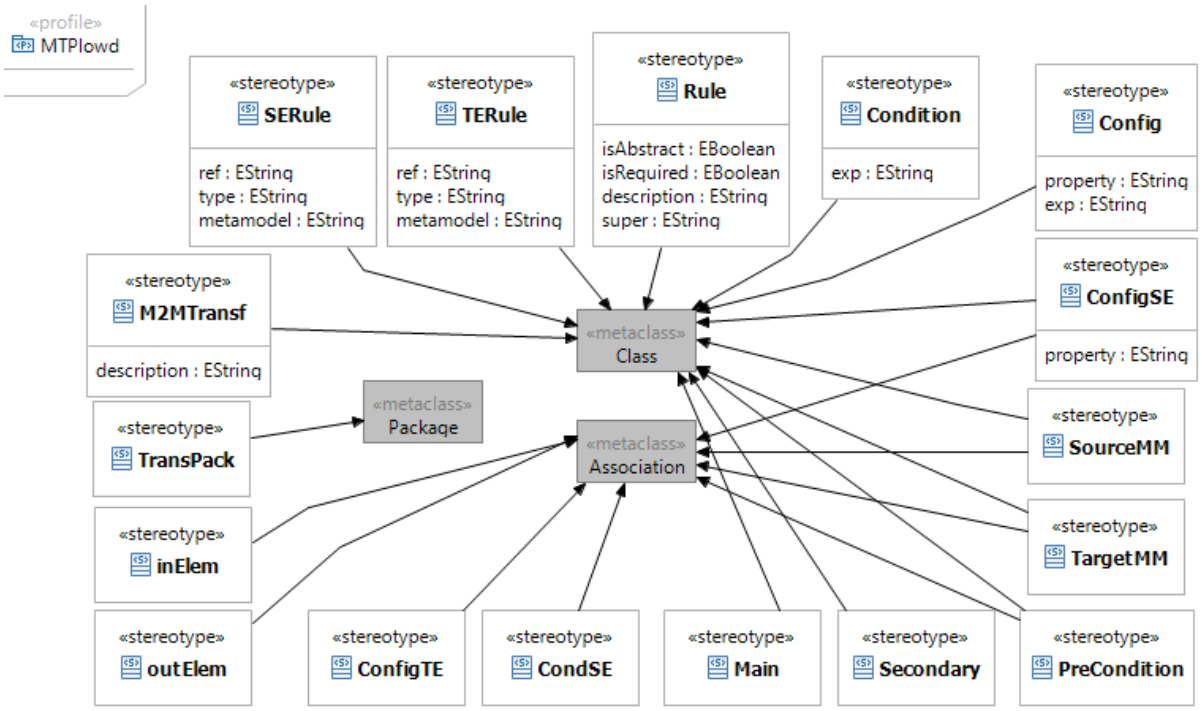


Figura 4.15 Pacote MTPlowd com estereótipos e metaclasses

delo *MMTlowdesign* e as metaclasses estendidas da UML (em cinza). O nome de alguns estereótipos foi abreviado em relação ao metamodelo por restrição da ferramenta utilizada para implementação. A tabela 4.3 apresenta a correspondência entre os elementos do metamodelo e os estereótipos criados, além das metaclasses estendidas e do diagrama em que o estereótipo pode ser aplicado durante o desenvolvimento. Por exemplo, o conceito *M2MTransformation* do *MMTlowdesign* corresponde ao estereótipo *M2MTransf* no perfil *MTPlowd* e pode ser aplicado a uma classe no diagrama de classes.

4.2.3.1 Exemplo de Uso do Perfil MTPlowd O perfil *MTPlowd* é aplicado ao diagrama de classes. A Figura 4.16 mostra um exemplo deste diagrama de classe que detalha a regra *MapearClasse* da transformação OO2RDBMS. Nesta regra existe um elemento de entrada, o elemento *umlClass* (estereotipado como $<<SERule>>$), e três elementos de saída, *RdbmsTable*, *RdbmsColumn* e *RdbmsKey* (estereotipados como $<<TERule>>$). Portanto cada classe deve gerar como saída uma tabela com um campo identificador e uma chave associada a este campo.

Na Figura 4.16 a regra *MapearClasse* (estereotipada como $<<Rule>>$no centro) está associada aos elementos de entrada (estereotipados como $<<SERule>>$) e saída (estereotipados como $<<TERule>>$). Cada elemento de saída tem um conjunto de configurações (estereotipados como $<<Config>>$). As configurações são definidas indicando a propriedade que está sendo configurada (atributo *property*) e a expressão de inicialização (atributo *exp*). Por exemplo, a configuração do nome da tabela (atributo *property=rdbmsName*) recebe o nome do elemento de entrada (*exp=seUmlClass.UmlName*)

Tabela 4.3 Conceitos do metamodelo MMTlowdesign e estereótipos do perfil MTPlowd

Metamodelo MMTlowdesign	**Estereótipo MTPlowd**	**Metaclasse**	**Diagrama**
M2MTransformation	M2MTransf	Class	Classes
Rule	Rule	Class	Classes
SourceElementRule	SERule	Class	Classes
TargetElementRule	TERule	Class	Classes
Condition	Condition	Class	Classes
Configuration	Config	Class	Classes
ConfigSourceElem	ConfigSe	Association	Classe
ConfigTargetElem	ConfigTE	Association	Classes
SourceMM	SourceMM	Association	Class
TargetMM	TargetMM	Association	Class
ruleCont	Cond	Association	Class
sourceElem	inElem	Association	Class
targetElem	outElem	Association	Class
	transPack	Package	Class

da regra. Para configurar o schema (atributo *property=RdbmsSchema*) em que a tabela será criada chamamos outra regra (atributo *exp=call MapearModelo*) através da instrução *call*, pois é preciso criar o schema primeiro para depois associar à tabela. Como a transformação é unidirecional, a configuração do elemento de entrada (estereotipado como *<<SERule >>*) indica somente qual a propriedade que está sendo utilizada pela regra, não contem inicializações.

4.3 CONSIDERAÇÕES SOBRE O CAPÍTULO

A definição clássica de transformação de modelos se baseia na identificação de mapeamentos entre elementos dos metamodelos participantes da transformação. Por exemplo, no nível de especificação, (BRAGA et al., 2011) define transformação como um contrato que contém um conjunto de relações entre metamodelos. Já no nível de projeto, (KLEPPE; WARMER; BAST, 2003) define transformação como um conjunto de regras entre elementos dos metamodelos. Neste capítulo abstraímos estas definições e conceituamos uma transformação de modelos no nível de requisitos, chamada de Especificação de Transformação. Em seguida refinamos este conceito em mais dois níveis de abstração, projeto de alto nível e projeto de baixo nível e os representamos sob a forma de metamodelos. Adicionalmente definimos um perfil UML para cada um desses níveis de abstração. Contribuimos assim com a especificação de modelos de transformação de modelos em diferentes níveis de abstração através de diagramas UML e possibilitamos o uso da DDM no desenvolvimeno de transformações desde a fase de requisitos até a fase de projeto de baixo nível sem se ater a tecnologias específicas. Esses modelos independentes podem ser utilizados para gerar modelos em diferentes plataformas. Adicionalmente, os metamodelos propostos também compreendem definições que possibilitam a verificação

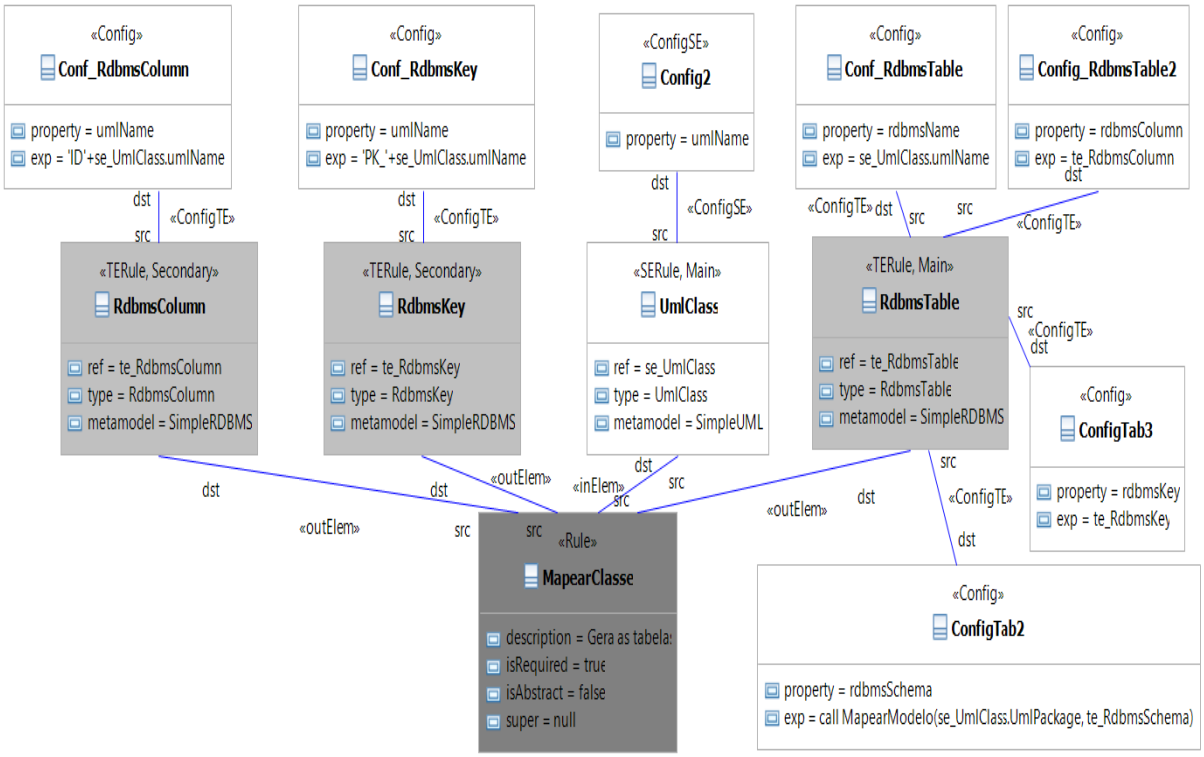


Figura 4.16 Diagrama de classes que detalha o comportamento da regra MapearClasse

de propriedades importantes ao desenvolvimento de transformações, como, por exemplo, a verificação da propriedade de completude da transformação.

Capítulo

5

PROCESSO PARA DESENVOLVIMENTO DE TRANSFORMAÇÕES DE MODELOS

Este capítulo apresenta o processo para desenvolvimento de transformações, MDTDproc proposto nesta tese. Inicialmente é apresentada uma visão geral do processo, destacando o ciclo de vida de desenvolviemnto. Em seguida são detalhadas cada uma das fases que compõe o processo com seus respectivos elementos.

5.1 PROCESSO DE DESENVOLVIMENTO DE TRANSFORMAÇÕES DE MODELOS

O MDTDproc é um processo iterativo e incremental para desenvolvimento de transformações de modelos unidirecionais com base na abordagem DDM. O ciclo de vida de desenvolvimento do MDTDproc é composto de cinco fases que compreendem desde o planejamento de uma cadeia com suas respectivas transformações até a geração do código fonte destas transformações. Conforme ilustrado na Figura 5.1 uma transformação compreende um conjunto de requisitos e pode ser desenvolvida em um ou mais incrementos. Utilizando a DDM modelos de transformação de modelos são construídos e transformados, a partir de uma cadeia (semi) automática de transformações, em modelos menos abstratos até a geração do código da transformação. Assim como na MDA (MELLOR, 2004), no MDTDproc são considerados três níveis de abstração: especificação, projeto independente de plataforma e projeto específico de plataforma. Cada um desses níveis pode ser desenvolvido de maneira iterativa (*iterac* na Figura 5.1), dependendo da complexidade da transformação, e tem como resultado um modelo da transformação no nível de abstração específico (modelo de requisitos, modelo de projeto independente e modelo de projeto específico). A passagem de um nível de abstração para o outro é apoiada por transformações definidas para o processo MDTDproc. Por exemplo, ao final da fase *TRM* tem-se um *modelo de requisitos* que é utilizado para gerar o modelo inicial da fase de *TDM*. O resultado de cada incremento é o código fonte da transformação desenvolvida.

Para adotar a DDM como abordagem de desenvolvimento de software em uma organização algumas estratégias podem ser utilizadas, dentre elas a customização do processo

de desenvolvimento já utilizado pela empresa. Nesta direção, por utilizar um modelo iterativo e incremental o MDTDproc possibilita a customização gradual do processo de desenvolvimento de uma organização à abordagem DDM, pois etapas do processo de desenvolvimento da empresa podem ser incrementalmente automatizadas. A estratégia de desenvolvimento do MDTDproc pode ser considerada também aderente ao desenvolvimento ágil de software uma vez que transformações podem ser desenvolvidas em pequenos incrementos a depender das demandas e prioridades da organização.

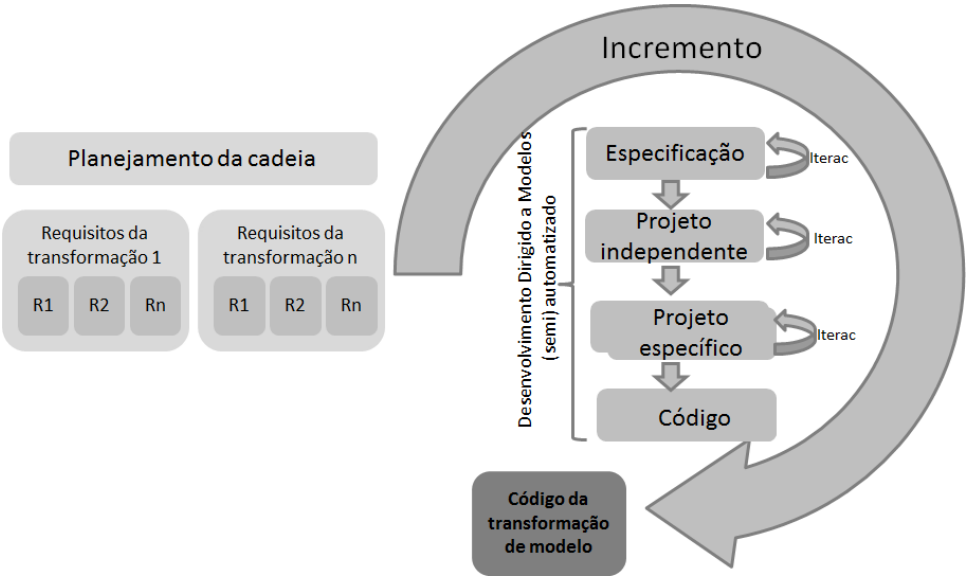


Figura 5.1 Ciclo de vida de desenvolvimento da transformação utilizado pelo framework

O processo MDTDproc está formalizado na linguagem SPEM - *Software Process Engineering Metamodelo* (OMG, 2008) e estruturado em dois pacotes: o pacote *Method Content*, que representa uma biblioteca de conteúdos contendo, dentre outros elementos, a definição de *o que* (artefatos), *quem* (papéis) e *como* (tarefas e passos) uma transformação será construída; e o pacote *Process*, que define *quando* (ciclo de vida organizado em fases e iterações) as definições do *Method Content* devem ser realizadas. Todos esses elementos estão implementados na ferramenta *Eclipse Process Framework*[1] e publicados no site do projeto (MAGALHAES, 2016).

O ciclo de vida do MDTDproc está organizado em cinco fases, conforme ilustrado na Figura 5.2. A primeira fase, *Planejamento* (*TCP*), é a fase aonde o desenvolvedor planeja a cadeia de transformações que vai automatizar o seu processo de desenvolvimento e as fases seguintes *Especificação* (*TRM*), *Projeto independente* (*TDM*), *Projeto específico* (*TSM*) e *código* (*Code*) são responsáveis pelo desenvolvimento das transformações que compõe a cadeia especificada na primeira fase.

Considerando as diversas especialidades envolvidas, sete papéis, que colaboram na construção de transformações, foram definidos para o processo: *Especialista de domínio*, que detém o conhecimento do negócio a ser automatizado; *Especificador de transformações*,

[1]EFP Project - https://eclipse.org/epf/

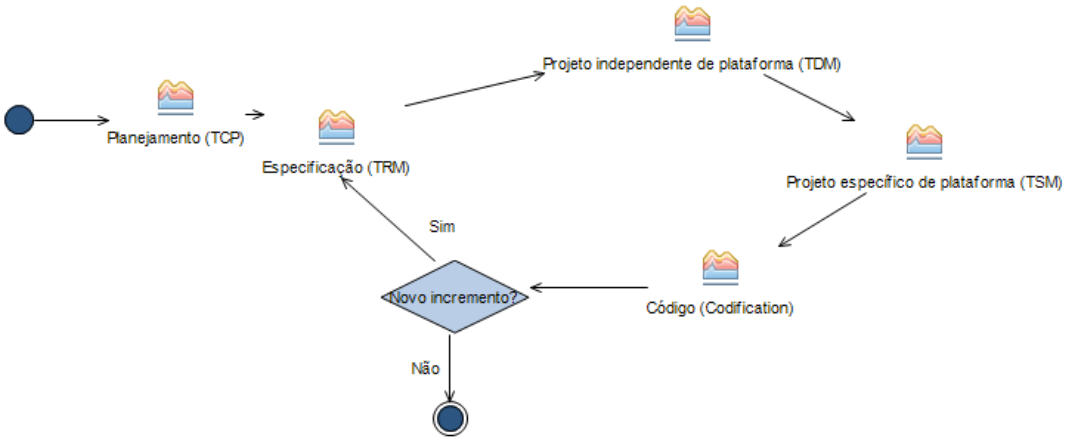


Figura 5.2 Fases do processo MDTDproc

para realizar a especificação da transformação com o apoio do especialista de domínio; *Desenvolvedor de transformações*, para conceber o projeto da transformação; *Arquiteto de transformações*, para definir a composição das transformações; *Especialista em linguagem formal*, para realizar as tarefas de verificação formal da transformação; *Especialista em linguagem de transformações*, para trabalhar com o projeto específico e modificações em nível de código; e *Testador*, para validar a transformação. Estes papéis estão definidos na especificação do processo e podem ser acessados em (MAGALHAES, 2016).

As Seções 5.1.1 a 5.1.5 detalham cada uma das fases que compreendem o processo MDTDproc. A especificação completa do processo foi implementada na ferramenta *Eclipse Process Framework* (EPF) e está disponível em (MAGALHAES, 2016).

5.1.1 Planejamento (Transformation Chain Planning – TCP)

A customização do processo de desenvolvimento de software de uma organização para a abordagem DDM pode envolver o desenvolvimento de uma única transformação ou o desenvolvimento de uma cadeia com várias transformações. Assim, o objetivo da fase *Planejamento* é analisar as necessidades de uma organização e planejar a sua cadeia de transformação. Esse planejamento envolve identificar quais partes do processo de desenvolvimento da empresa podem ser automatizadas, ou seja, quais são os requisitos relacionados à automação do processo de desenvolvimento dentro do domínio, aqui chamado de *requisitos do domínio*, para então definir quais as transformações que precisarão ser construídas e os requisitos de cada uma dessas transformações.

A fase de planejamento utiliza como base exemplos de modelos de aplicações já construídas pela empresa no domínio trabalhado. Esses exemplos de modelos são analisados para identificar o que pode ser automatizado dentro do processo de desenvolvimento da organização. Por exemplo, para definir a cadeia de transformações analisa-se em que pontos do processo de desenvolvimento da organização são construídos modelos e se a passagem de um modelo para outros pode conter algum nível de automação.

A Figura 5.3 ilustra o fluxo de execução das tarefas pertencentes à fase de planeja-

mento. Para esta fase todas as tarefas são executadas pelo mesmo papel, o *Especificador de transformações*, conforme ilustrado no topo da figura. A primeira tarefa executada consiste na identificação dos requisitos de domínio, com base em exemplos. A partir destes é planejada a cadeia e mapeados os requisitos funcionais de cada transformação que a compõe. Diversos exemplos podem ser utilizados iterativamente até que os requisitos sejam definidos. Em casos de transformações mais simples, a cadeia poderá ser formada por uma única transformação, e é possível que os requisitos do domínio e os requisitos funcionais sejam os mesmos.

Transformações de modelos também possuem requisitos não funcionais. Estes requisitos estão diretamente relacionados à qualidade da transformação, como por exemplo, modularidade, padronização de código, performance, reuso, modificabilidade, entendibilidade (AMSTEL; MARCEL; BRAND, 2010). O uso do processo MDTDproc pode conduzir o desenvolvedor ao alcance de vários desses requisitos. Por exemplo, modularidade pode ser alcançada através do planejamento da cadeia e de cada uma das transformações, e a padronização é favorecida pela geração automática de código.

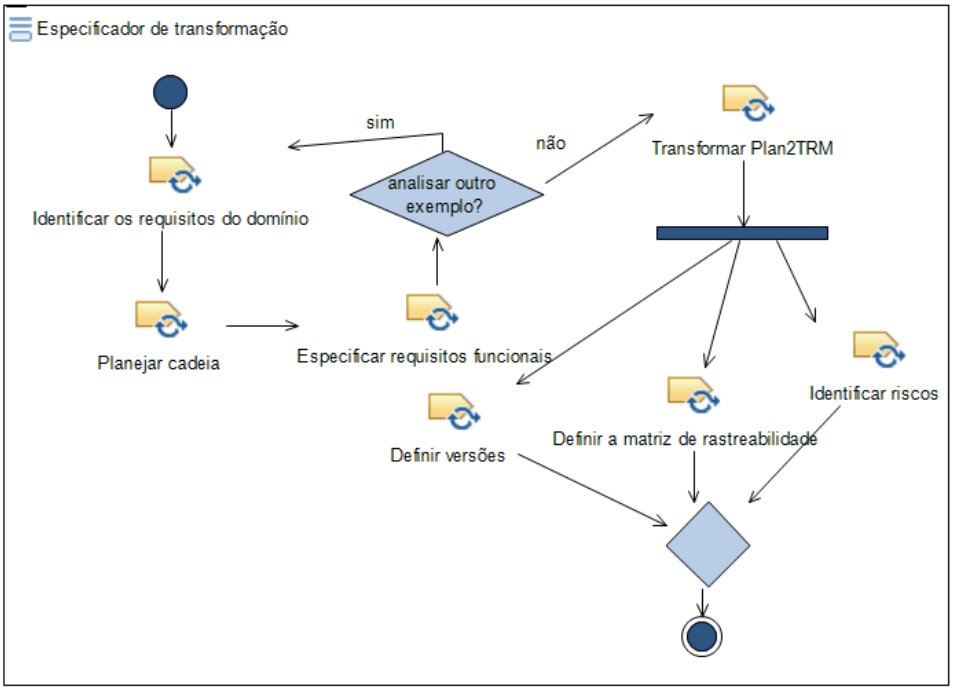


Figura 5.3 Fluxo das tarefas executadas na fase *TCP*

A definição detalhada de cada uma das tarefas que compõe o processo está especificada conforme a linguagem SPEM e publicada em (MAGALHAES, 2016). Por exemplo, a Figura 5.4 mostra a definição da tarefa *Especificar requisitos do domínio*. Esta tarefa é de responsabilidade do papel *Especificador de transformação* definido como *Primary Performer* apoiado pelo *Especialista de domínio*, que é o *Additional Performers*. Para executá-la é necessário analisar exemplos de aplicações reais (*Input Mandatory*) produzindo como resultado (*outputs*) os requisitos do domínio da aplicação. A tarefa contém

também uma descrição geral do seu objetivo (*Main descripion*) e quatro passos (*steps*) detalhadamente descritos para orientar a equipe de desenvolvimento. Adicionalmente, oferece alguns exemplos de requisitos de domínio como material de apoio (*Supporting Materials*) que podem ser consultados pelos desenvolvedores.

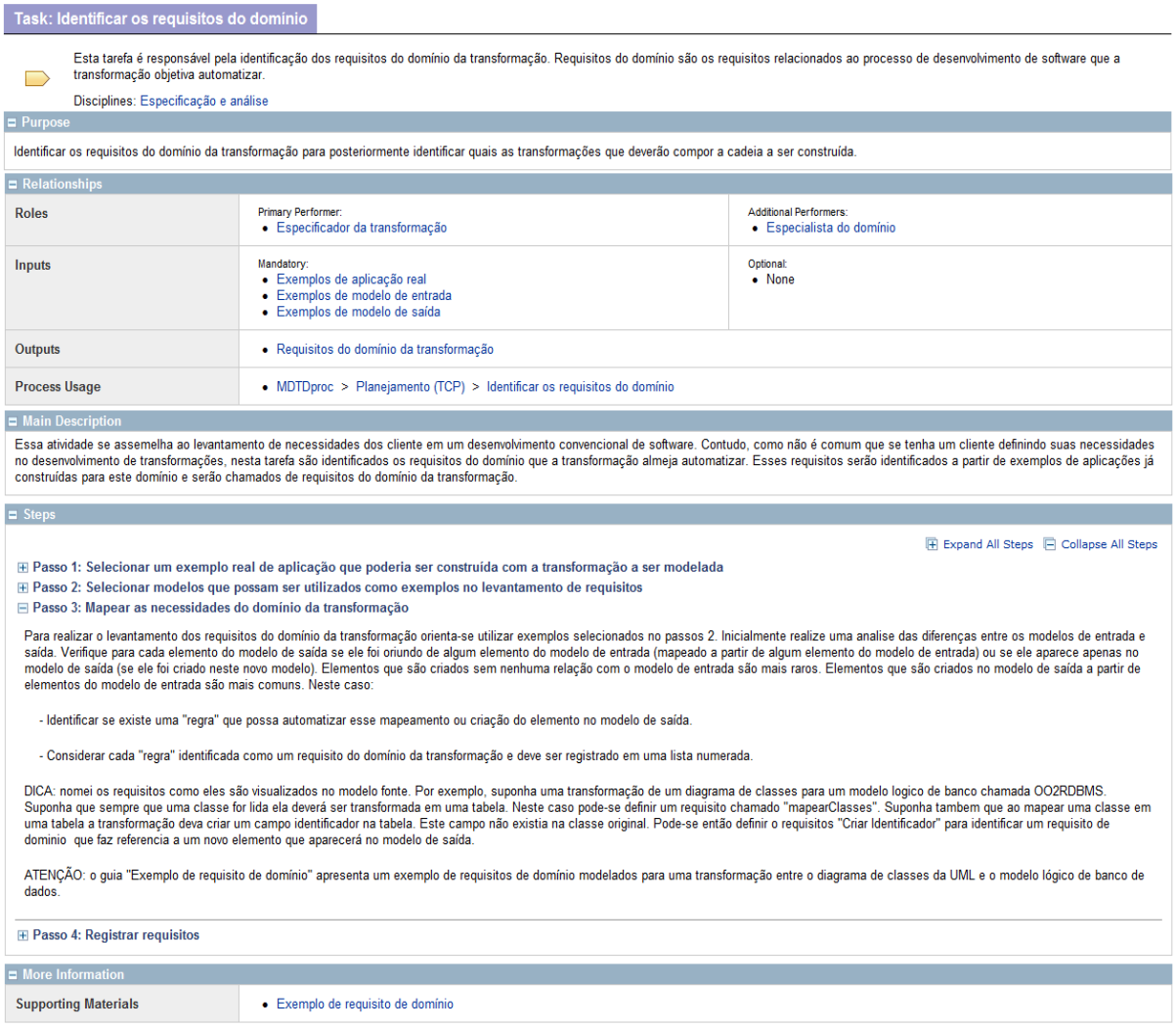


Figura 5.4 Definição da tarefa *Especificar requisitos do domínio* na ferramenta EPF

Os requisitos de domínio são especificados em linguagem natural em um *template* provido pelo processo MDTDproc. Para a especificação dos requisitos funcionais o MDTDproc recomenta inicialmente a utilização do diagrama de casos de uso da UML. O diagrama de casos de uso foi selecionado por ser amplamente utilizado pela comunidade para especificar requisitos, contudo não é expressivo o suficiente para registrar todas as informações requeridas pelo processo MDTDproc. Portanto, uma vez concluída a identificação desses requisitos, a tarefa *Transformar Plan2TRM* é executada e consiste em converter automaticamente o diagrama de casos de uso em um diagrama de classes com os mesmos requisitos, para que novas informações possam ser inseridas, tais como, o ob-

jetivo de cada requisitos, a versão em que o requisito será desenvolvido e o critério de verificação que será utilizado para validar o requisito. Exemplos destes diagramas podem ser encontrados na Seção 3.2.2.1.

As últimas tarefas da fase de planejamento são *Definir versões*, *Definir matriz de rastreabilidade* e *Identificar riscos*. A definição de versões envolve determinar os incrementos que compreenderão o desenvolvimento da transformação. Para isso os requisitos são priorizados e a depender da prioridade estabelecida as versões são definidas. Já o registro de rastreabilidade tem como objetivo relacionar os requisitos do domínio aos respectivos requisitos funcionais da transformação para possibilitar que futuramente toda a cadeia de artefatos produzidos possa ser rastreada desde o requisito do domínio até o código. A tarefa *Identificar riscos* se destina a identificar os riscos relacionados ao sucesso do desenvolvimento. O processo MDTDproc já disponibiliza uma lista de possíveis riscos, tais como, falta de conhecimento do domínio, ausencia de metamodelo apropriado e falta de conhecimento das ferramentas. O objetivo desta tarefa é analisar esses riscos em relação à transformação em desenvolvimento (e identificar novos, caso necessário) visando identificar o que precisa ser monitorado para evitar problemas durante o desenvolvimento. Essa atividade é apoiada por um *template* que facilita a identificação e o registro desses riscos.

Ao final desta fase a cadeia de transformação está definida. As fases seguintes correspondem então ao desenvolvimento individual de cada transformação que compõe a cadeia.

5.1.2 Especificação (Transformation Requirement Modeling -TRM)

A fase *TRM* tem como objetivo detalhar os requisitos de um novo incremento no desenvolvimento de uma transformação. A Figura 5.5 mostra as tarefas que a compõem organizadas de acordo com os papeis responsáveis pela sua execução.

Três tarefas são realizadas inicialmente: *Detalhar requisitos*, *Selecionar ou definir metamodelos* e *Definir restrições*, todas pelo papel *Especificador de transformações*. A tarefa *Detalhar requisitos* é puramente documental e consiste em definir o objetivo da transformação e de cada requisito a ela associado. Esta tarefa é importante, pois com base nessa documentação serão gerados comentários no código da transformação. A tarefa *Selecionar ou definir metamodelos* é fundamental para o processo, pois todo o projeto da transformação será realizado com base nos metamodelos selecionados. A seleção dos metamodelos se baseia nos requisitos (do domínio e da transformação) definidos no início do processo e consiste em identificar um metamodelo que atenda às necessidades da transformação. Em casos específicos pode ser necessário inclusive criar novos metamodelos. Para essas situações o framework contém um guia para ajudar a definição de metamodelos (MAGALHãES; MACIEL; ANDRADE, 2015), disponível no *site* do projeto (MAGALHAES, 2016). Finalmente, a tarefa *Definir restrições* possibilita identificar restrições aplicadas aos requisitos das transformações. Essas restrições serão importantes na fase de projeto para a especificação de condições booleanas associadas às regras. Adicionalmente o processo contém também a tarefa *Definir propriedades* (opcional), relacionada à verificação da correção semântica da transformação. O processo MDTDproc

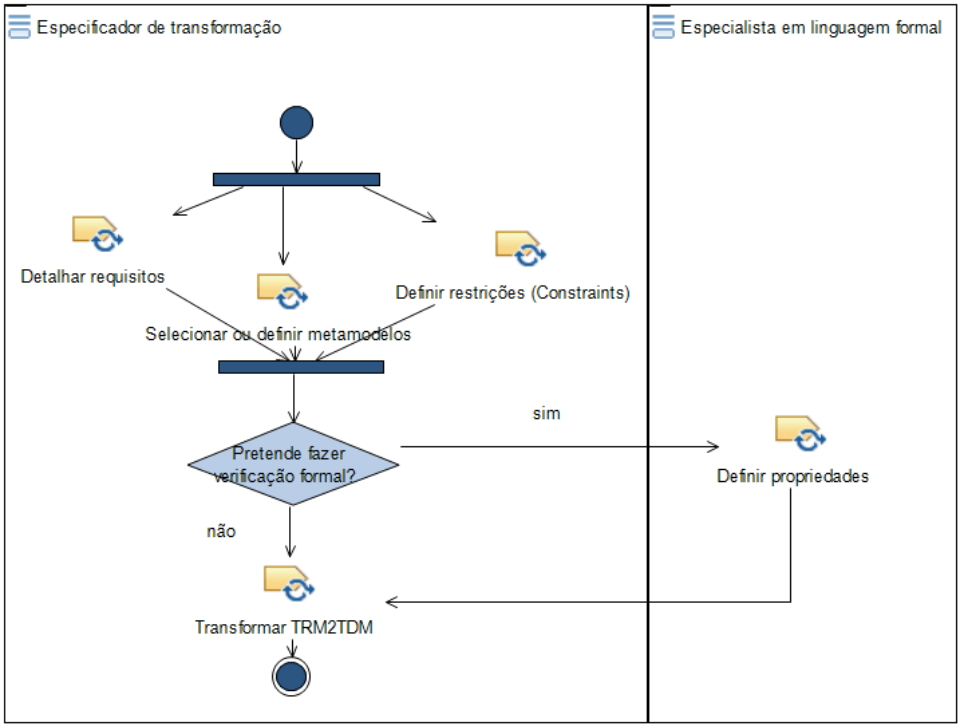


Figura 5.5 Fluxo das tarefas da fase *TRM*

prevê a especificação formal de propriedades relacionadas tanto a transformação quanto aos modelos de entrada e saída da transformação. Contudo, nenhuma abordagem específica para verificação de transformação está integrada ao framework. Faz parte da tarefa a escolha de uma abordagem para ser utilizada.

O produto final da fase *TRM* é o *modelo de requisitos da transformação* contendo os diversos dados levantados na especificação, tais como, requisitos, metamodelos envolvidos, restrições e propriedades. Este modelo é utilizado como entrada da última tarefa desta fase, *Transformar TRM2TDM*, onde é transformado de maneira automática pelo framework no modelo inicial da fase de projeto da transformação (*TDM*).

5.1.3 Projeto Independente (Transformation Design Modeling - TDM)

A fase *TDM* tem como objetivo definir o projeto da transformação e está dividida em dois níveis de abstração: o primeiro nível corresponde à definição de *o que* será transformado; o segundo nível tem como foco a definição de *como* o modelo de saída será gerado pela transformação. Desta forma, as tarefas que compõem a fase *TDM* estão divididas em duas iterações chamadas respectivamente de projeto de alto nível (*HighDesign*) e projeto de baixo nível (*LowDesign*).

5.1.3.1 Projeto de Alto Nível A Figura 5.6 ilustra o fluxo de atividades do projeto de alto nível.

A tarefa inicial, *Definir arquitetura da transformação*, é realizada decompondo-se uma

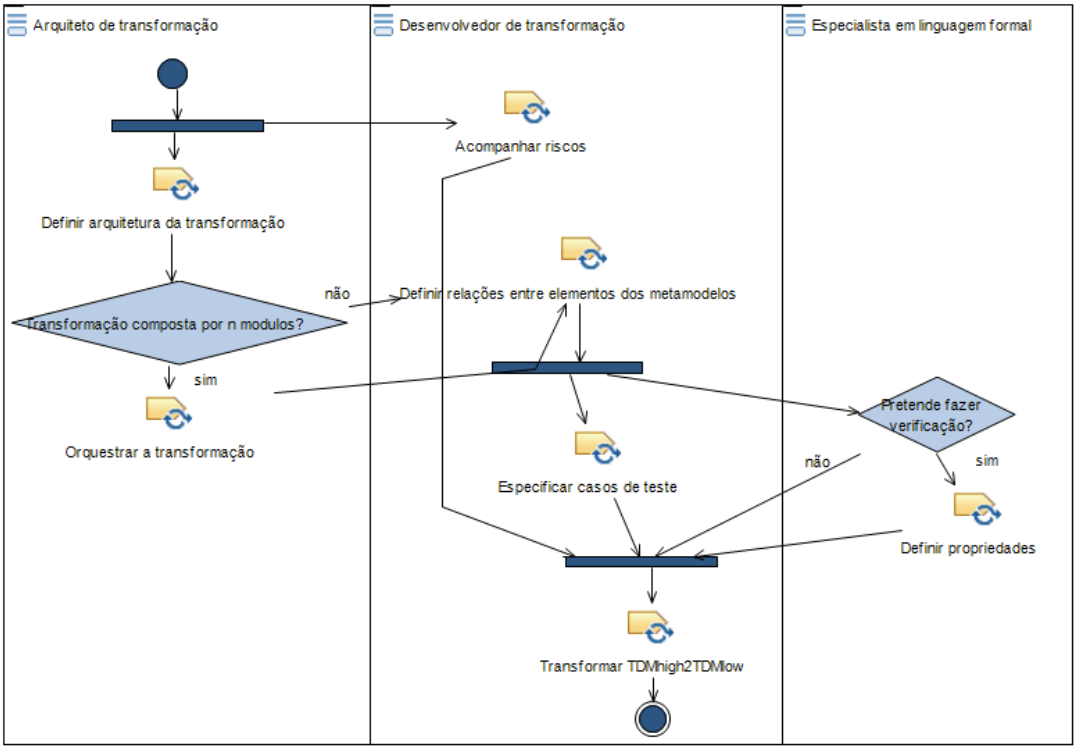


Figura 5.6 Fluxo das tarefas da fase *TDM* no projeto de alto nível

transformação em transformações menores (chamadas de componentes). Toda transformação é formada por pelo menos um componente que contém uma interface fornecida, com os serviços providos pelo componente (futuramente implementados como regras da transformação), e uma interface requerida, com serviços para leitura dos modelos e metamodelos de entrada e para geração do modelo de saída, que devem ser implementados por quem deseja utilizar o componente. Caso a transformação seja composta por mais de um componente, estes podem se conectar de duas maneiras: através de portas, quando a sua execução é automática, ou seja, o modelo de saída de um componente é utilizado como modelo de entrada do outro componente; ou através da definição de interfaces, quando a sua execução é semi-automática e precisa da intervençao do desenvolvedor.

Após a definição da arquitetura é executada a tarefa *Orquestrar a transformação*. A orquestração é uma tarefa necessária nos casos em que a transformação é composta por vários componentes, pois seu objetivo é definir o fluxo de execução destes componentes. Essas duas tarefas são de responsabilidade do *Arquiteto de transformações*.

A tarefa seguinte, *Definir relações entre os elementos dos metamodelos*, é de responsabilidade do *Desenvolvedor de transformações* e consiste em definir os relacionamentos entre elementos dos metamodelos fonte e elementos dos metamodelos alvo, ou seja, o que será transformado em o que.

Prosseguindo com o projeto de alto nível, duas tarefas são executadas, sem ordem de prioridade: *Especificar casos de teste*, pelo *Desenvolvedor de transformações*, que visa provê uma documentação de teste, de acordo com um *template* pré-definido pelo

processo, para ser utilizada na validação da transformação quando esta estiver pronta; e *Definir propriedades*, pelo *Especialista em linguagem formal*, que visa à especificação formal de propriedades a serem utilizadas no final do desenvolvimento para verificar os modelos gerados pela transformação. Adicionalmente, a tarefa *Acompanhar riscos*, também executada pelo *Desenvolvedor de transformações*, é realizada ao longo de todo o processo e visa acompanhar o desenvolvimento para prevenir a ocorrência dos riscos inicialmente identificados.

O produto final do projeto de alto nível é um modelo que contém as relações entre elementos dos metamodelos fonte e alvo envolvidos na transformação. Este modelo é transformado automaticamente, através da execução da tarefa *Transformar MTPhigh2MTPlow*, em um modelo de projeto de baixo nível para ser utilizado nas tarefas subsequentes do processo.

5.1.3.2 Projeto de Baixo Nível A Figura 5.7 apresenta o fluxo de atividades do projeto de baixo nível da transformação, todas elas de responsabilidade do *Desenvolvedor de transformações*. Neste nível de abstração uma transformação é vista como um conjunto de regras que juntas transformam um modelo de entrada em um modelo de saída. É preciso então especificar *como* esse modelo será gerado.

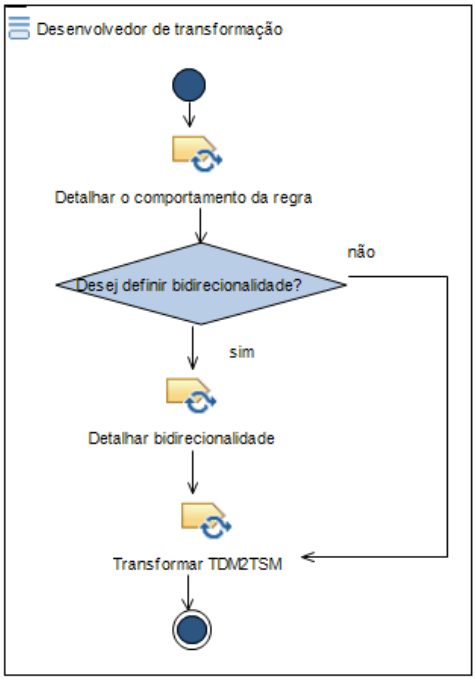


Figura 5.7 Fluxo das tarefas da fase *TDM* no projeto de baixo nível

Na primeira tarefa,*Detalhar o comportamento da regra* é definido como as propriedades (atributos) dos elementos gerados no modelo de saída serão inicializados. O framework MDTD é aplicado para desenvolver transformações unidirecionais. Contudo, possibilita a especificação de transformações unidirecionais para serem codificadas em linguagens

bidirecionais, a exemplo da QVT-R. Este tipo de linguagem requer também a definição de uma configuração para o modelo de entrada da transformação. Desta forma, caso o desenvolvedor deseje gerar código em uma linguagem bidirecional, deverá também executar a tarefa *Detalhar bidirecionalidade* para definir a configuração dos elementos de entrada de cada regra.

O produto final do projeto de baixo nível é um modelo, independente de linguagem de transformação, que detalha a transformação com suas respectivas regras. Este modelo é transformado a partir da tarefa *Transformar TDM2TSM* em um modelo específico de plataforma na linguagem escolhida pelo desenvolvedor. Atualmente o processo oferece automação para a geração de modelo específico de plataforma em duas linguagens ATL e QVT-R. O modelo específico de plataforma gerado após a execução desta tarefa é utilizado como entrada na próxima fase do processo.

Exemplos dos documentos produzidos nesta fase podem ser encontrados na Seção 3.2.2.3.

5.1.4 Projeto Específico (Transformation Specific Modeling - TSM)

A fase *TSM* tem como objetivo modelar o projeto da transformação em uma linguagem específica. Como mencionado na seção anterior, o MDTDproc gera modelos específicos em duas linguagens: ATL e QVT-R de forma automática. Automações para novas linguagens podem futuramente ser implementadas. O modelo de projeto específico gerado para estas linguagens é representado em um diagrama de classes estereotipado com os perfis *PATL* ou *PQVT* respectivamente, definidos pelo framework e detalhados nos apêndices G e H.

A Figura 5.8 apresenta o fluxo das tarefas que compõem a fase *TSM* e os papeis responsáveis pela execução destas tarefas.

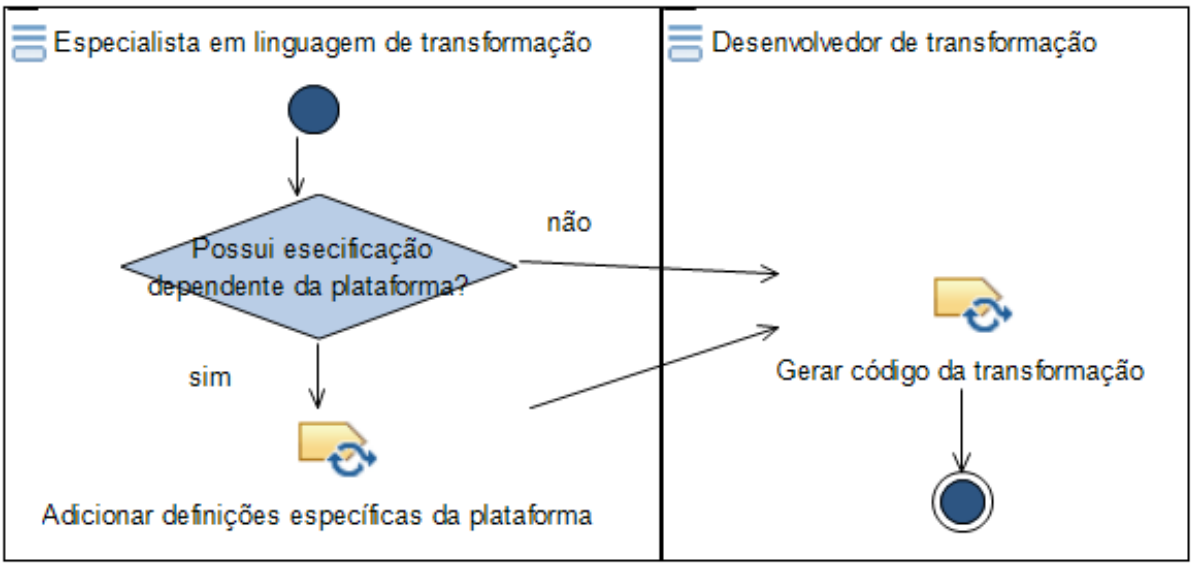


Figura 5.8 Fluxo das tarefas da fase *TSM*

A primeira tarefa executada, *Adicionar definições específicas da plataforma* é de res-

ponsabilidade do *Especialista em linguagem de transformação* e consiste, quando necessário, na complementação do modelo gerado com especificidades da linguagem escolhida. Como dito anteriormente, trata-se de um diagrama de classe estereotipado de acordo com os construtores da linguagem específica. Considerando que estas linguagens foram essencialmente criadas para programação e não para modelagem, esses modelos em geral são bastante complexos, pois possuem um grande número de classes. Sendo assim, esta tarefa não é obrigatória, ela será executada somente se o desenvolvedor perceber a necessidade de realizar alterações em nível de plataforma.

Em seguida a tarefa *Gerar código da transformação* é responsável pela geração do código fonte da transformação na linguagem escolhida.

5.1.5 Codigo (Code)

A fase *Codigo* tem como objetivo finalizar a implementação e realizar as validações necessárias para concluir uma versão (incremento) da transformação.

A Figura 5.9 apresenta o fluxo das tarefas que compõem a fase *Codigo*. Observa-se que diversos papeis colaboram para que a fase seja concluída, pois existem atividades relacionadas à questões, tais como, programação, verificação, validação e gerenciamento de versões.

A fase se inicia com a tarefa *Complementar o código* executada pelo *Especialista em linguagem de programação*. Esta é uma tarefa opcional que será realizada somente quando o código gerado não estiver completo, ou seja, se existir alguma definição que o desenvolvedor não conseguiu expressar no nível de modelagem e que, portanto, precise ser acrescentada ao código.

Uma vez o código pronto, inicia-se as tarefas relacionadas à verificação e validação da transformação, executadas pelo *Especialista em linguagem formal* e pelo *Testador*, respectivamente. A tarefa *Executar os testes* visa testar a transformação com base nos casos de teste previamente definidos. A ocorrência de algum defeito pode significar o retorno a fases anteriores do processo para que o defeito seja corrigido e o código novamente gerado. A tarefa *Verificar o modelo de saída* consiste na verificação do modelo produzido como saída de acordo com as propriedades previamente definidas. Esta verificação está relacionada à correção semântica da transformação e não está automatizada pelo processo. Desta forma, abordagens complementares de verificação podem ser utilizadas para executar esta tarefa. A última tarefa desta fase consiste é de responsabilidade do *Desenvolvedor de transformações* e consiste em *Analisar a versão* construída para decidir se será necessário iniciar o desenvolvimento de um novo incremento (uma nova versão) da transformação ou se o desenvolvimento pode ser considerado finalizado.

5.2 CONSIDERAÇÕES SOBRE O CAPÍTULO

Este capítulo apresentou o processo MDTDproc para desenvolvimento de transformações de modelos unidirecionais na abordagem DDM. Diferentemente das propostas atuais o processo abrange tanto o planejamento da cadeia quanto o desenvolvimento das transformações que a compõe. Ao considerar uma transformação como parte de uma cadeia

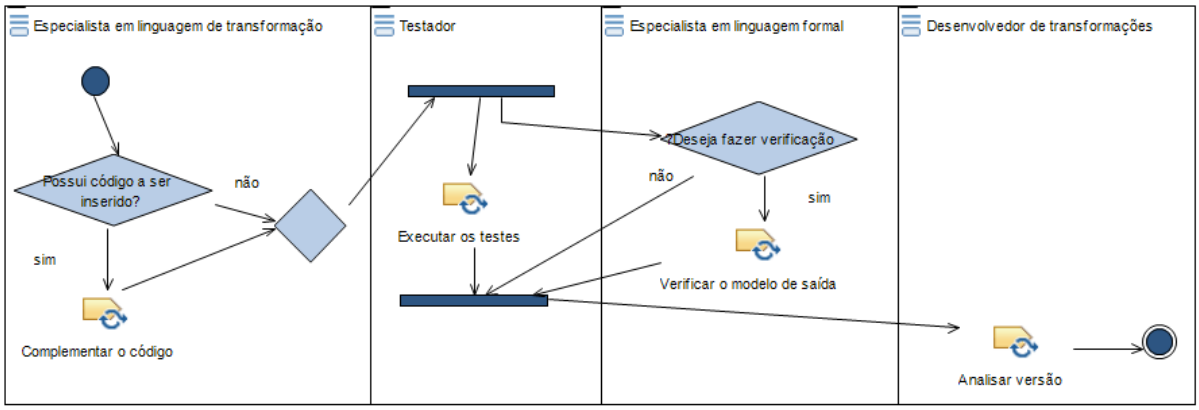


Figura 5.9 Fluxo das tarefas da fase de codigo

que automatiza um processo de software, o MDTDproc apresenta uma visão mais abrangente que as propostas atuais: a partir do levantamento das necessidades de automação de um domínio, uma cadeia é planejada definindo as transformações que serão construídas e os requisitos que serão contemplados em cada transformação. Com base nos requisitos os metamodelos apropriados a cada transformação são selecionados e/ou construídos e então cada transformação é efetivamente desenvolvida. O desenvolvimento é guiado por um processo especialmente definido para o domínio de transformações, formalizado na linguagem SPEM e apoiado por um perfil UML. A integração do processo ao perfil visa conduzir o desenvolvedor passo a passo na especificação de modelos de transformação em diversos níveis de abstração através de diagramas UML e com isso contribuir para reduzir a complexidade inerente a esse tipo de desenvolvimento.

Capítulo 6

AUTOMAÇÃO DO FRAMEWORK MDTD

Este capítulo apresenta os recursos de automação definidos para apoiar o framework proposto nesta tese: um ambiente de desenvolvimento; e uma cadeia de transformações que (semi) automatiza o processo de desenvolvimento. As seções a seguir detalham cada um desses recursos.

6.1 AMBIENTE DE DESENVOLVIMENTO

O desenvolvimento de transformações com o framework MDTD requer o uso de ferramentas automatizadas que possibilitem: (i) a implementação dos metamodelos envolvidos na transformação; (ii) a criação dos modelos de transformação de modelos; (iii) a execução de transformações modelo-a-modelo, tanto para transformar os *modelos de transformação de modelos* ao longo do desenvolvimento quanto para testar a transformação de modelo construída com o framework; e (iv) a execução de transformações modelo-a-texto para a geração do código da transformação. Nesta tese utilizamos o ambiente *Eclipse* e alguns *plug-ins* para realizar estas atividades conforme ilustrado na Figura 6.1.

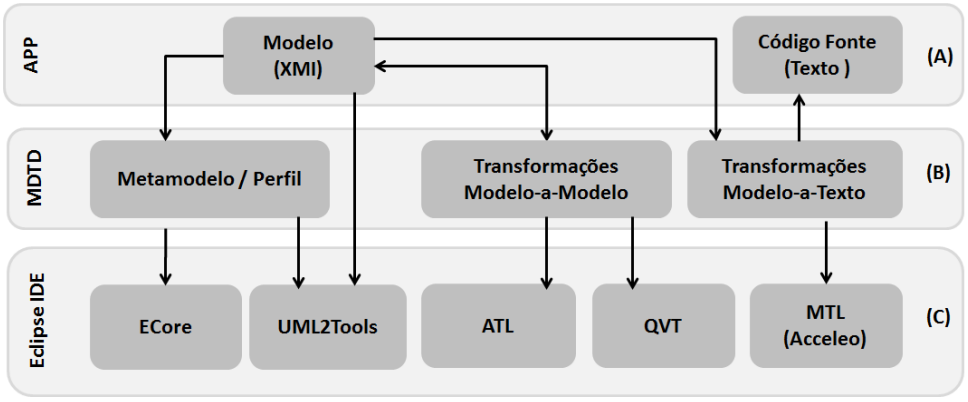


Figura 6.1 Ambiente de desenvolvimento usado pelo framework MDTD

A base da Figura 6.1(C) apresenta os *plug-ins* adicionados ao Eclipse para configuração do ambiente de desenvolvimento: Ecore, metametalinguagem utilizada na implementação dos metamodelos e perfis; UML2Tools, ambiente UML de modelagem; plugin ATL, ambiente utilizado para a construção e execução das transformações que automatizam o processo MDTDproc bem como para o teste das transformações construídas no framework e geradas em código ATL; plugin QVT, ambiente utilizado para testar as transformações construídas no framework e geradas em código QVT; plugin Acceleo, ambiente utilizado para executar as transformações modelo-a-texto que automatizam o processo MDTDproc.

Ao centro da Figura 6.1(B) estão os artefatos implementados para apoiar o processo MDTDproc, os metamodelos (descritos no Capítulo 4) e perfis (descritos no Capítulo 4 e nos Apêndices G e H) e as transformações (descritas na seção a seguir).

O topo da Figura 6.1(A) apresenta os artefatos que podem ser construídos e ou gerados ao longo do desenvolvimento de uma transformação. Os modelos são construídos em conformidade com os metamodelos e perfis utilizando o plugin UML2Tools. Uma vez construídos os modelos, o arquivo XMI correspondente é utilizado como entrada das transformações *model2model* e *model2text* que uma vez executadas geram como saída, respectivamente, outro arquivo XMI ou o código fonte da transformação em construção.

6.2 CADEIA DE TRANSFORMAÇÃO

Esta seção apresenta a cadeia de transformação implementada para (semi) automatizar o processo MDTDproc. A cadeia contempla a automação do desenvolvimento desde as fases iniciais relacionadas à especificação de requisitos até à geração do código da transformação nas linguagens ATL e QVT e é composta de sete transformações, conforme ilustrado na Figura 6.2. As cinco primeiras transformações são do tipo *modelo-a-modelo* e foram implementadas na linguagem ATL. As duas últimas transformações são do tipo *modelo-a-texto* e foram implementadas na linguagem *Model to Text Language* (MTL)[1]. Transformações para outras linguagens podem ser construídas utilizando como entrada o modelo de projeto da transformação independente de plataforma.

Inicialmente, na fase *TCP* do processo, a transformação *Plan2TRM.atl* possibilita gerar um modelo de classes com os requisitos da transformação a partir do modelo de casos de uso previamente construído. Em seguida, na fase *TRM* a transformação *TRM2TDMhigh.atl* transforma o modelo de requisitos no modelo inicial de projeto de alto nível para que seja dado início a fase de projeto *TDM*. Em seguida, o modelo de alto nível é transformado no modelo de baixo nível pela transformação *TDMhigh2TDMlow.atl*. Até este ponto o projeto é independente de plataforma. A partir deste momento duas transformações possibilitam gerar o modelo em uma plataforma específica: a transformação *TDM2ATL.atl*, gera o modelo ATL da transformação e a *TDM2QVT.atl* gera o modelo QVT da transformação. O último passo consiste na geração do código através das transformações *CodeATL.mtl* e *CodeQVT.mtl*.

As seções a seguir detalham cada uma das sete transformações construídas. O código completo das transformações encontra-se no *site* do projeto (MAGALHAES, 2016).

[1]MTL - http://www.omg.org/spec/MOFM2T/1.0/

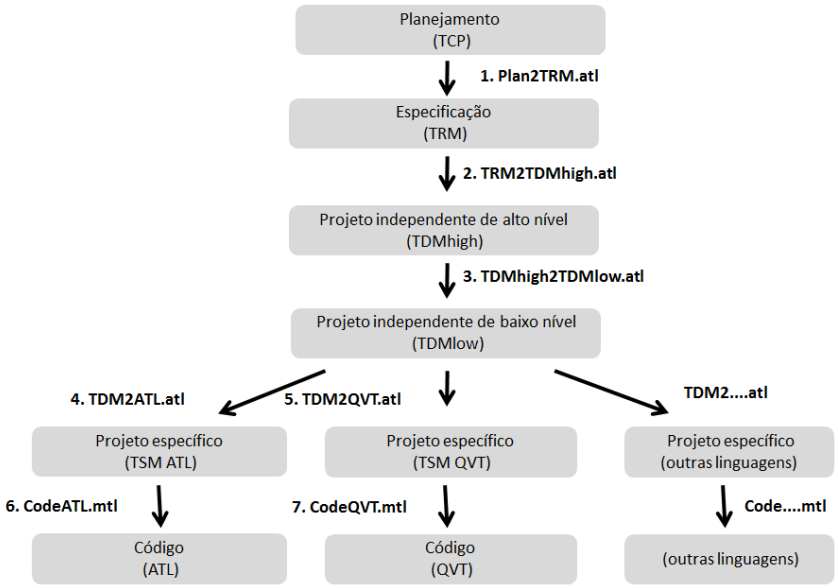


Figura 6.2 Cadeia de transformações do MDTDproc

6.2.1 Transformação Plan2TRM.atl

A transformação *Plan2TRM.atl* tem como objetivo transformar o diagrama de caso de uso, com os requisitos da transformação, em um diagrama de classes, também de requisitos.

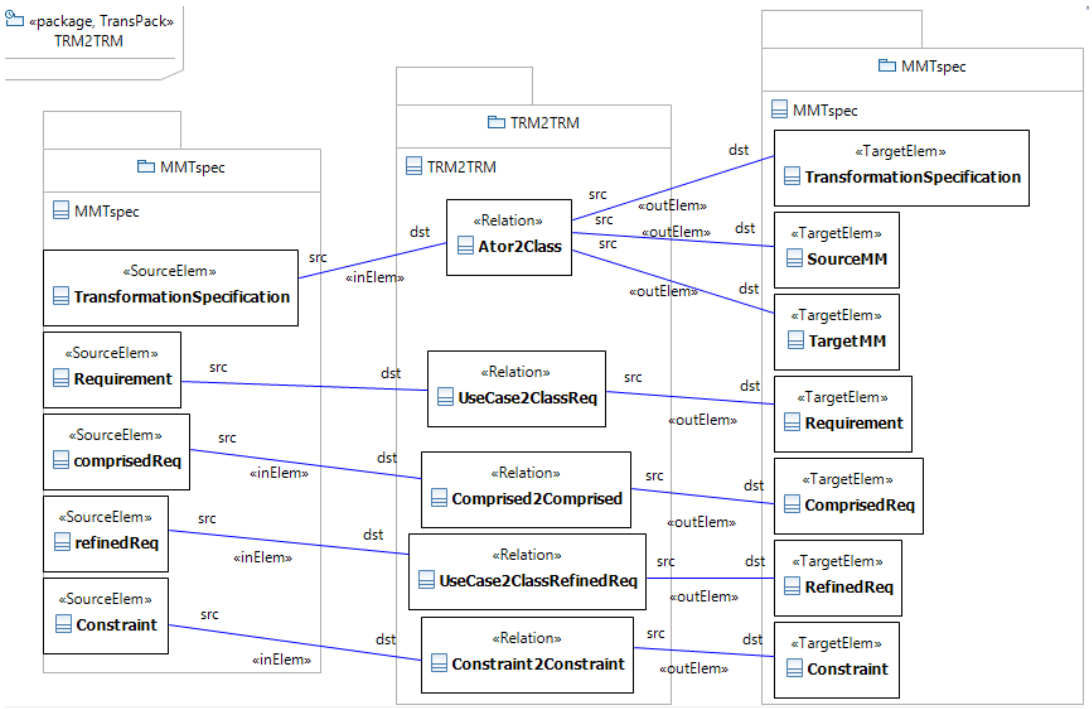


Figura 6.3 Especificação das relações da transformação Plan2TRM

Plan2TRM.atl é uma transformação endógena, pois o metamodelo fonte e o metamodelo alvo é o mesmo, o MMTspec. A especificação das relações que compõem *Plan2TRM.atl* é apresentada na Figura 6.3. Cinco relações foram definidas, ilustradas no pacote central em classes estereotipadas como <<*Relation*>>. Por exemplo, a relaçao *Ator2Classe*, que é do tipo 1-N, mapeia elementos do tipo *TransformationSpecification* em elementos dos seguintes tipos: *TransformationSpecification*, *SourceMM* e *TargetMM*.

A Figura 6.4 mostra um exemplo de execução da transformação *Plan2TRM.atl*. Do lado esquerdo da figura é apresentado um exemplo de diagrama de casos de uso, como modelo de entrada da transformação, e do lado direito da figura é apresentado um exemplo de diagrama de classes, gerado como modelo de saída da transformação. As setas pontilhadas indicam quais elementos do modelo de entrada geram elementos no modelo de saída. Por exemplo, a execução da regra que representa a relação *Ator2Classe* (da Figura 6.3) recebeu como entrada o ator *OO2RDBMS*, que é um elemento do tipo *TransformationSpecificaton*, e gerou como saída três classes: a classe *OO2RDBMS* estereotipada como <<*TransSpec*>>, que é um elemento do tipo *TransformationSpecification*; e duas classes que representam os metamodelos fonte e alvo estereotipadas como <<*SourceMM*>>e <<*TargetMM*>>respectivamente, que são elementos do tipo *SourceMM* e *TargetMM*. De maneira análoga, a execução da regra que representa a relação *UseCase2ClassReq* (da Figura 6.3) recebeu como entrada o caso de uso *MapearClasse*, que é uma elemento do tipo *Requirement*, e gerou como saída a classe *MapearClasse* estereotipada como <<*Req*>>, que também é um elemento do tipo *Requirement*.

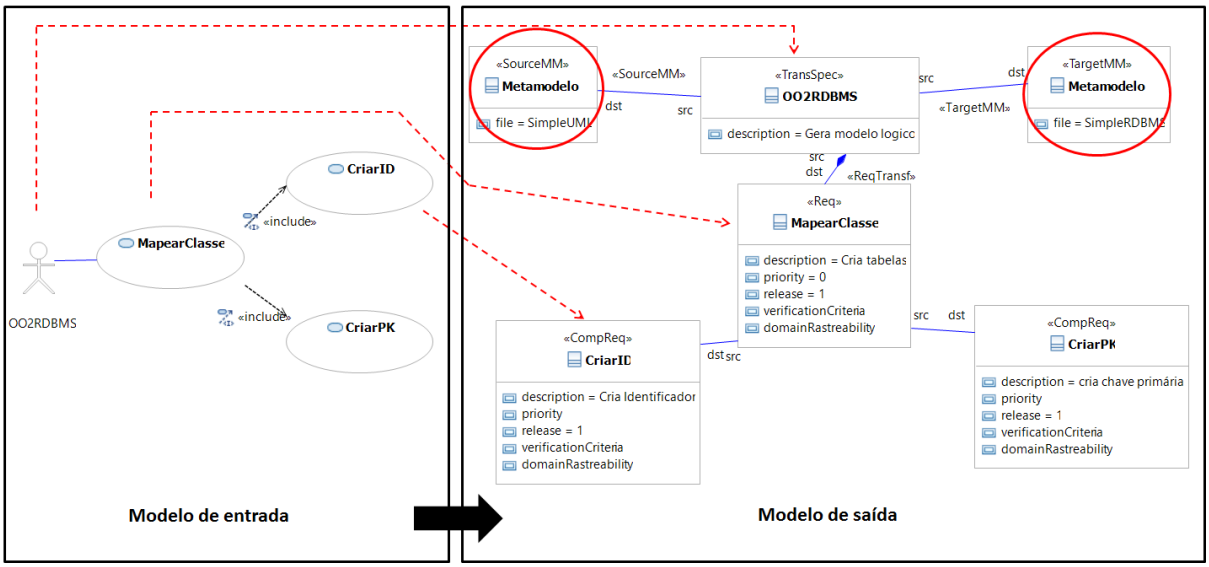


Figura 6.4 Exemplo do modelo de entrada e saída da transformação Plan2TRM

6.2.2 Transformação TRM2TDMhigh.atl

A transformação *TRM2TDMhigh.atl* tem como objetivo gerar um diagrama de classes, modelo inicial de projeto de alto nível, a partir do diagrama de classes com a especificação

dos requisitos da transformação.

TRM2TDMhigh é uma transformação exógena, pois o metamodelo fonte MMTspec é diferente do metamodelo alvo MMThighdesign. A Figura 6.5 mostra a especificação da transformação *TRM2TDMhigh.atl* composta por seis relações, ilustradas no pacote central em classes estereotipadas como <<*Relation*>>. Por exemplo, a relação *class2packageTransf* mapeia elementos do tipo *TransformationSpecification* em elementos do tipo *M2MTransformation*.

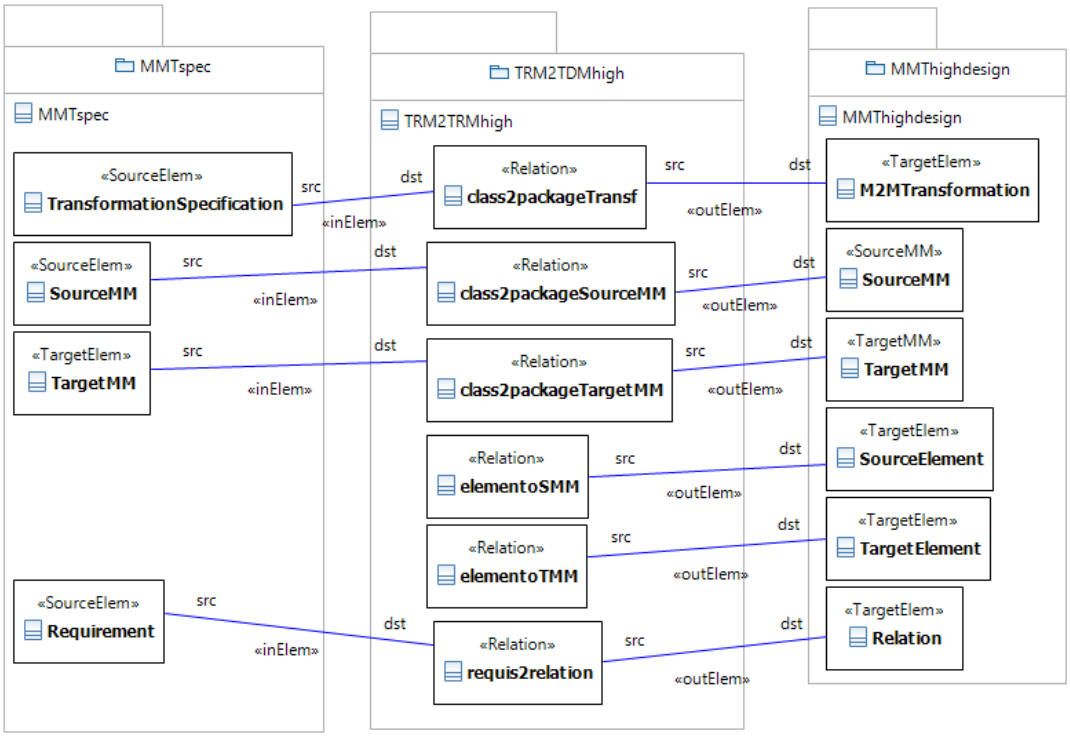


Figura 6.5 Especificação da transformação *TRM2TDMhigh*

A Figura 6.6 ilustra um exemplo de execução da transformação *TRM2TDMhigh.alt*. Do lado esquerdo da figura é apresentado um exemplo de diagrama de classes (de requisitos), modelo de entrada da transformação, e do lado direito da figura é apresentado um exemplo de diagrama de classes (de projeto de alto nível), modelo de saída da transformação. As setas pontilhadas indicam quais elementos do modelo de entrada geram elementos no modelo de saída. A execução da regra que representa a relação *class2packageTransf* da Figura 6.5 recebe como entrada a classe *OO2RDBMS* estereotipada como <<*TransSpec*>>, que é um elemento do tipo *TransformationSpecification*, e gera como saída o pacote *OO2RDBMS* estereotipado como <<*M2MTransf*>>, que é um elemento do tipo *M2MTransformation*. Analogamente, a execução da regra que representa a relação *class2packageSourceMM* recebe como entrada a classe *Metamodelo*, estereotipada como <<*SourceMM*>>e gera como saída pacote *SimpleUML* estereotipado como <<*SourceMM*>>.

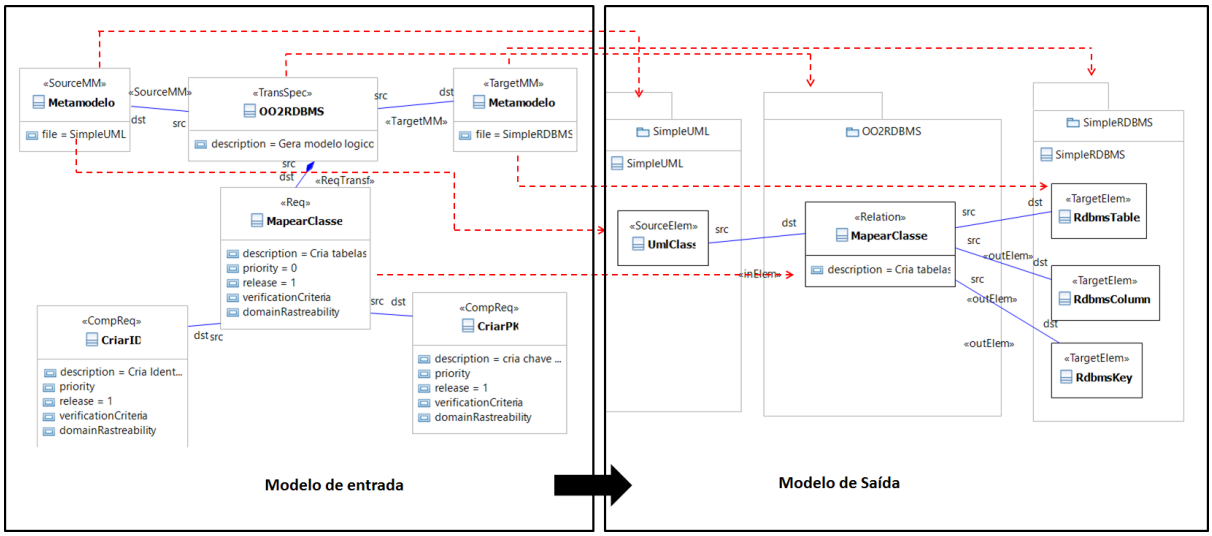


Figura 6.6 Exemplo do modelo de entrada e saída da transformação *TRM2TDMhigh*

6.2.3 Transformação TDMhigh2TDMlow.atl

A transformação *TDMhigh2TDMlow* Figura 6.7 tem o objetivo transformar um diagrama de classes que representa o modelo de projeto de alto nível em um diagrama de classes que representa o modelo de projeto de baixo nível da transformação.

A transformação *TDMhigh2TDMlow* é uma transformação exógena, pois o metamodelo fonte MMThighdesign é diferente do metamodelo alvo MMTlowdesign. A Figura 6.7 mostra a especificação da transformação *TDMhigh2TDMlow.atl* composta por seis relações, ilustradas no pacote central em classes estereotipadas como <<*Relation*>>. Por exemplo, a relação *Relation2Rule* mapeia elementos do tipo *Relation* em elementos do tipo *Rule*. Analogamente, as relações *association2SR* e *association2TER* mapeiam os elementos de entrada e saída da relation (do tipo *SourceElement* e *TargetElement*) em elementos de entrada e saída da regra (do tipo *SourceElementRule* e *TargetElementRule*). A relação *association2TER* gera também para cada elemento de saída da regra (do tipo *TargetElementRule*) uma classe de configuração (do tipo *Configuration*), pois assume que pelo menos uma propriedade deverá ser configurada para cada elemento de saída. Caso o desenvolvedor deseje configurar mais de uma propriedade para o mesmo elemento, ele deve criar novas configurações manualmente.

Um exemplo de execução da transformação *TDMhigh2TDMlow.atl* é apresentado na Figura 6.8.

No topo da Figura 6.8 é apresentado um exemplo de diagrama de classes (de projeto de alto nível) como modelo de entrada da transformação e na base da figura é apresentado um exemplo de diagrama de classes (de projeto de baixo nível) gerado como modelo de saída da transformação. As setas pontilhadas indicam quais elementos do modelo de entrada geram elementos no modelo de saída. Por exemplo, a execução da regra que representa a relação *Relation2Rule* recebe como entrada a classe *MapearClasse* estereotipada como <<*Relation*>>e gera como saída a classe *MapearClasse* estereotipada como <<*Rule*>>.

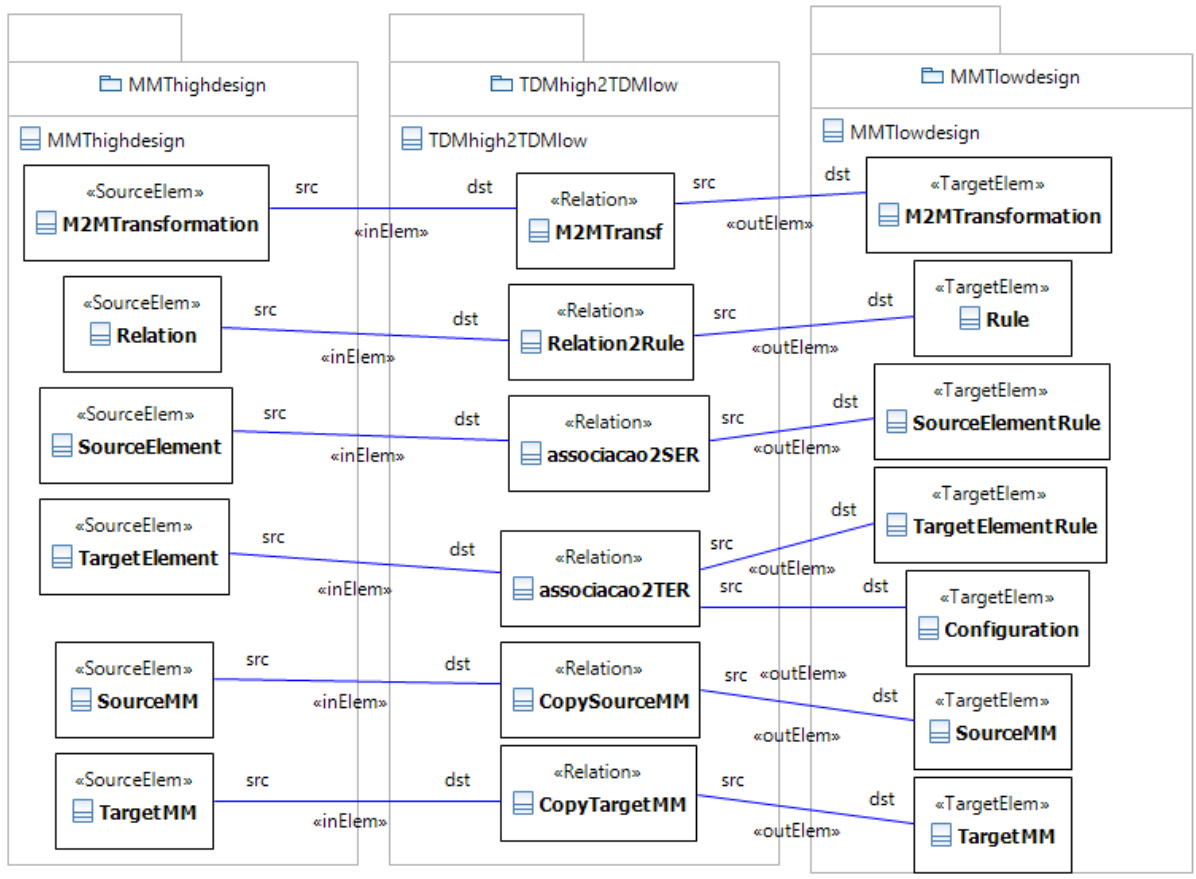


Figura 6.7 Especificação da transformação *TDMhigh2TDMlow*

6.2.4 Transformação TDM2ATL.atl

A transformação *TDM2ATL* tem o objetivo de transformar o diagrama de classes que representa o modelo de projeto de baixo nível no diagrama de classes que representa o modelo de projeto específico na linguagem ATL.

A ATL é uma linguagem híbrida, possui conceitos imperativos e declarativos. O framework MDTD trabalha com especificação de transformação em alto nível de abstração e, portanto, enfatiza o uso de instruções declarativas. Por este motivo, nem todos os conceitos da linguagem ATL foram considerados neste trabalho. Por exemplo, instruções imperativas como o *ActionBlock* da ATL não são utilizadas. Se o desenvolvedor desejar poderá adicioná-las manualmente quando o código da transformação for gerado.

TDM2ATL é uma transformação exógena, pois o metamodelo fonte MMTlowdesign é diferente do metamodelo alvo metamodelo da linguagem ATL. A Figura 6.9 mostra a especificação da transformação *TDM2ATL.alt*, composta por nove relações, ilustradas no pacote central em classes estereotipadas como *<<Relation>>*. A relação *transf2module* mapeia elementos do tipo *M2MTransformation* em elementos do tipo *Module* da ATL. As relações *sMM2sMM* e *tMM2tMM* mapeiam respectivamente cada elemento do tipo *SourceMM* e *TargetMM* em dois elementos do tipo *OCLModel*, um que representa o me-

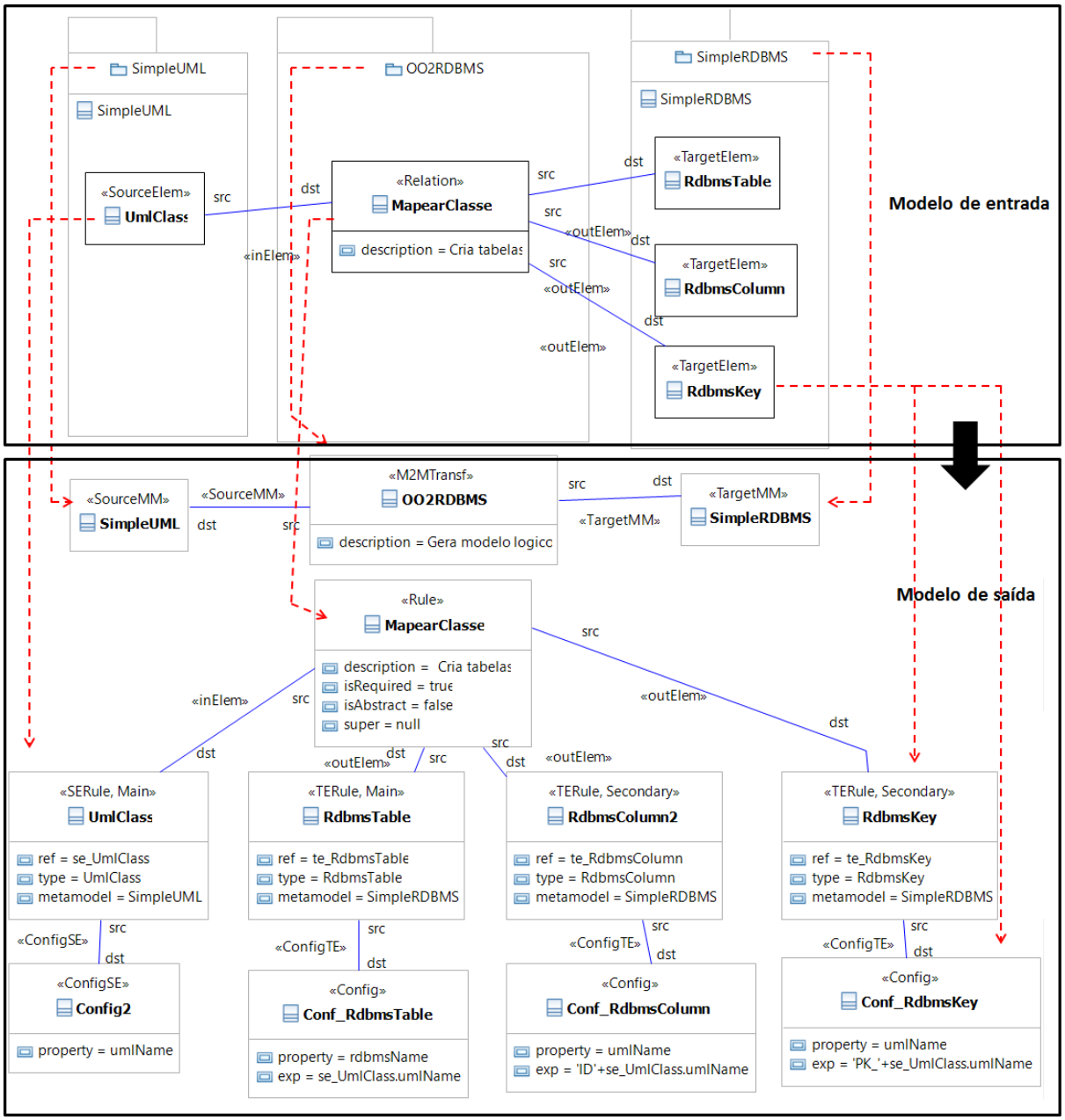


Figura 6.8 Exemplo do modelo de entrada e saída da transformação *TDMhigh2TDMlow*

tamodelo e outro que representa o modelo fonte ou alvo da transformação. Já a relação *Rule2MatchedRule* mapeia elementos do tipo *Rule* em elemento do tipo *MatchedRule*, se o atributo *isRequered = true* (não ilustrado na figura), caso contrário, em elementos do tipo *LazyMatchedRule*. Para cada elemento do tipo *matchedRule* é também mapeado um elemento do tipo *inPattern* e um elemento do tipo *outPattern*. Quando a regra que está sendo modelada não possui elemento de entrada (regras do tipo 0-1 ou 0-N), é executada a relação *Rule2CalledRule* que mapeia elementos do tipo *Rule* em elementos do tipo *CalledRule*. Neste caso a *CalledRule* criada será do tipo *EntryPoint* e somente poderá existir

uma regra deste tipo na transformação. Elementos do tipo *SourceElementRule* e *TargetElementRule* são mapeados respectivamente em elemento do tipo *inPatternElement* e *outPatternElement*. Condições (elementos do tipo *Condition*) são mapeadas pela relação *Condition2Filter* em filtros (elementos do tipo *Filter*). Finalmente, as configurações de inicialização de propriedades (elementos do tipo *Config*) são mapeadas pela relação *Config2Biding* em elementos do tipo *Biding* da linguagem ATL. Conforme mencionado no início desta seção, instruções imperativas, como, por exemplo, *helper*, o bloco *do* e o bloco *using* não são contempladas e podem ser inseridas manualmente pelo desenvolvedor após a geração do código da transformação.

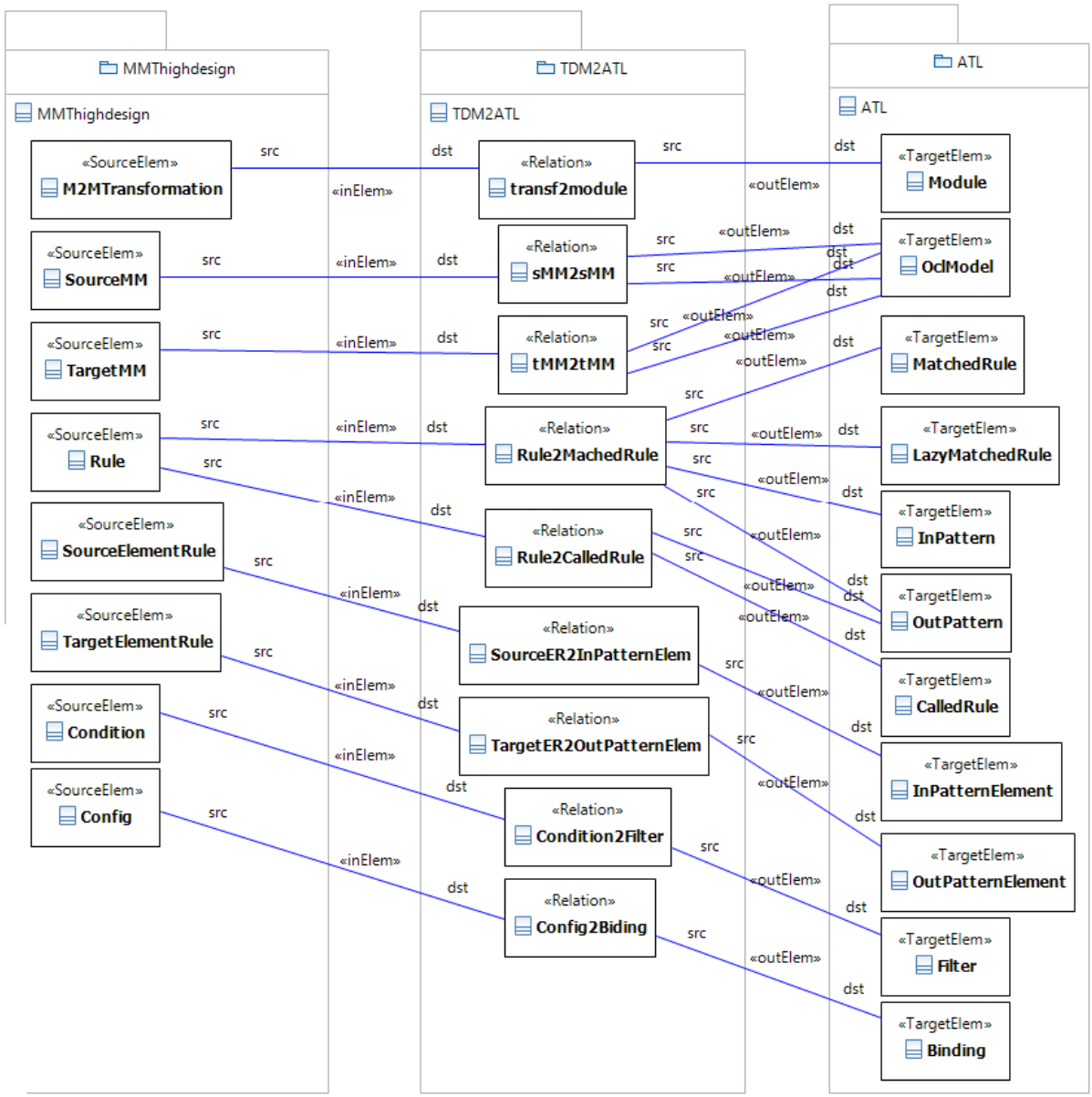


Figura 6.9 Especificação da transformação *TDM2ATL*

A ATL utiliza diversas expressões escritas na linguagem OCL. A representação visual dessas expressões de acordo com o metamodelo da OCL pode tornar o modelo da transformação visualmente complexo devido ao grande número de classes necessárias para representar a expressão. Por este motivo, *TDM2ATL.atl* guarda expressões OCL em campos do tipo *String*, conforme o metamodelo da ATL, tornando assim o modelo da transformação mais simples. Por exemplo, no metamodelo ATL o conceito *Binding* possui um atributo chamado value definido com o tipo *OclExpression*. Para representar essa expressão no modelo da transformação é criado na classe *Binding* um atributo *value* do tipo *String* onde é guardada a expressão OCL.

A Figura 6.10 ilustra um exemplo de execução da transformação *TDM2ATL.alt*.No topo da figura é apresentado um exemplo de diagrama de classes (de projeto de baixo nível) como modelo de entrada da transformação e na base da figura é apresentado um exemplo de diagrama de classes (de projeto específico em ATL) gerado como modelo de saída da transformação. As setas pontilhadas indicam quais elementos do modelo de entrada geram elementos no modelo de saída. Por exemplo, a execução da regra que representa a relação *Rule2MatchedRule* recebe como entrada a classe *MapearClasse* estereotipada como <<*Rule*>>e gera como saída três classes: *MapearClasse* estereotipada como <<*MatchedRule*>>, *IP_MapearClasse* estereotipada como <<*InPattern*>>e *OP_MapearClasse* estereotipada como <<*OutPattern*>>.

6.2.5 Transformação TDM2QVT.atl

A transformação *TDM2QVT.atl* tem o objetivo de transformar o diagrama de classes que representa o modelo de projeto de baixo nível no diagrama de classes que representa o modelo de projeto específico na linguagem QVT-Relation.

TDM2QVT é uma transformação exógena, pois o metamodelo fonte MMTlowdesign é diferente do metamodelo alvo, o metamodelo da linguagem QVT. A Figura 6.11 mostra a especificação da transformação *TDM2QVT.alt* composta por sete relações, ilustradas no pacote central em classes estereotipadas como <<*Relation*>>. Por exemplo, a relação *sMM2TypedModel* mapeia cada elemento do tipo *SouceMM* em dois elementos do tipo *TypedModel*, um para representar o metamodelo e outro para representar o modelo de entrada da transformação. A relação *Rule2TopRelation* mapeia elementos to tipo *Rule* em elementos do tipo *TopRelation*, quando o elemento *Rule* tem a propriedade *isRequired=true* (a especificação da propriedade não está ilustrada na figura).

A Figura 6.12 ilustra um exemplo de execução da transformação *TDM2QVT.alt*.No topo da figura é apresentado um exemplo de diagrama de classes (de projeto de baixo nível) como modelo de entrada da transformação e na base da figura é apresentado um exemplo de diagrama de classes (de projeto específico em QVT) gerado como modelo de saída da transformação. As setas pontilhadas indicam quais elementos do modelo de entrada geram elementos no modelo de saída. Por exemplo, a execução da regra que representa a relação *sMM2TypedModel* recebe como entrada a classe *SimpleUML* estereotipada como <<*SourceMM* >>e gera como saída as classes *SimpleUML* e *source* ambas estereotipadas como *TypedModel*, que representam o metamodelo e o modelo de entrada da regra, respectivamente. A execução da regra que representa a relação *Rule2TopRelation*

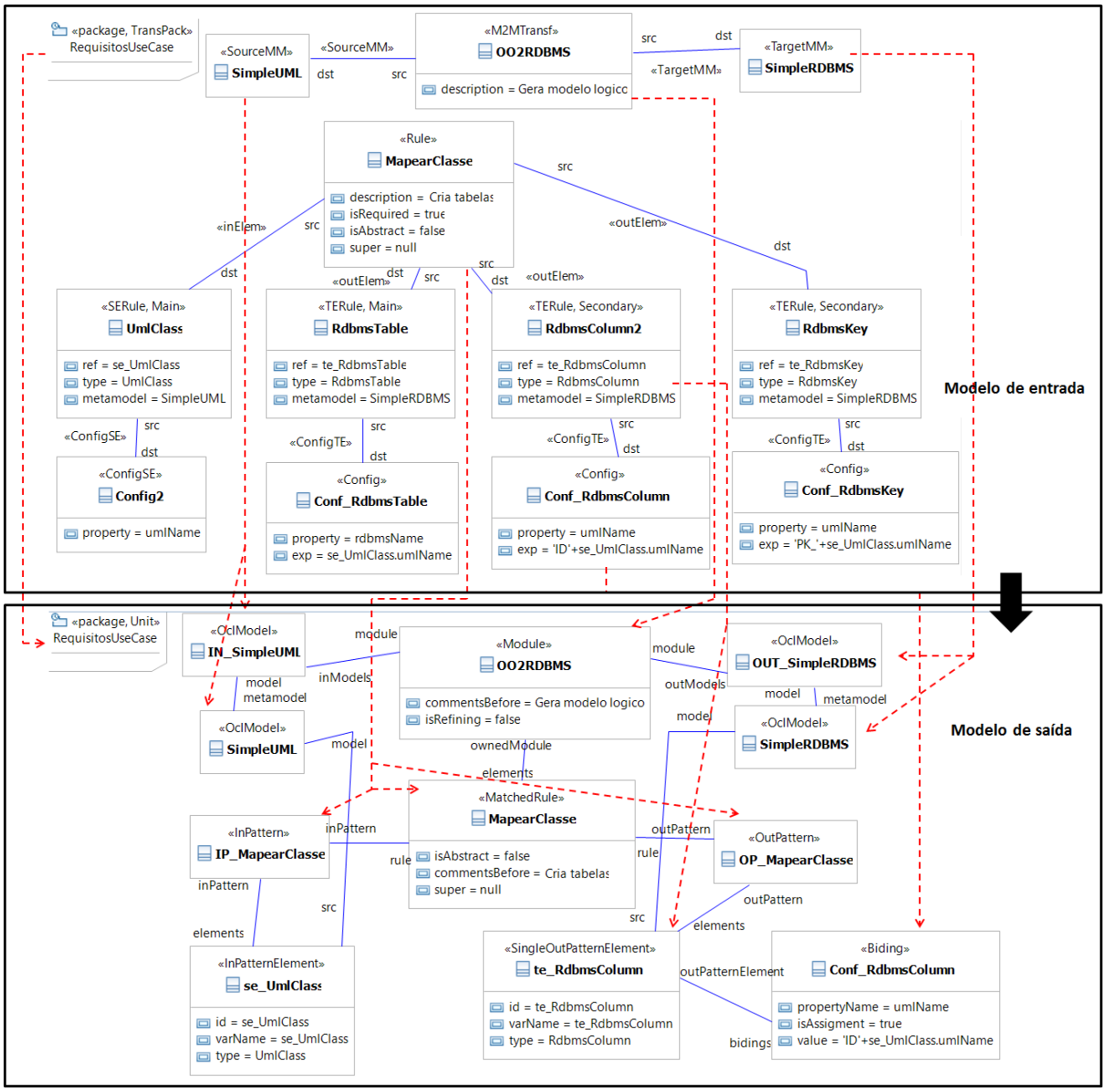


Figura 6.10 Exemplo do modelo de entrada e saída da transformação *TDM2TATL*

recebe como entrada a classe *MapearClasse* estereotipada como <<*Rule*>>e gera como saída a classe *MapearClasse* estereotipada como *Relation*.

6.2.6 Transformação CodeATL.mtl

A transformação *CodeATL.mtl* tem o objetivo de gerar o código da transformação na linguagem ATL a partir de um modelo de entrada representado por um diagrama de classes em conformidade com o perfil *PATL* (descrito no apêndice G). Esta transformação está implementada na linguagem *Model To Text Transformation Language (MTL)*.

A Figura 6.13 apresenta parte do código gerado pela transformação *CodeATL.mtl*

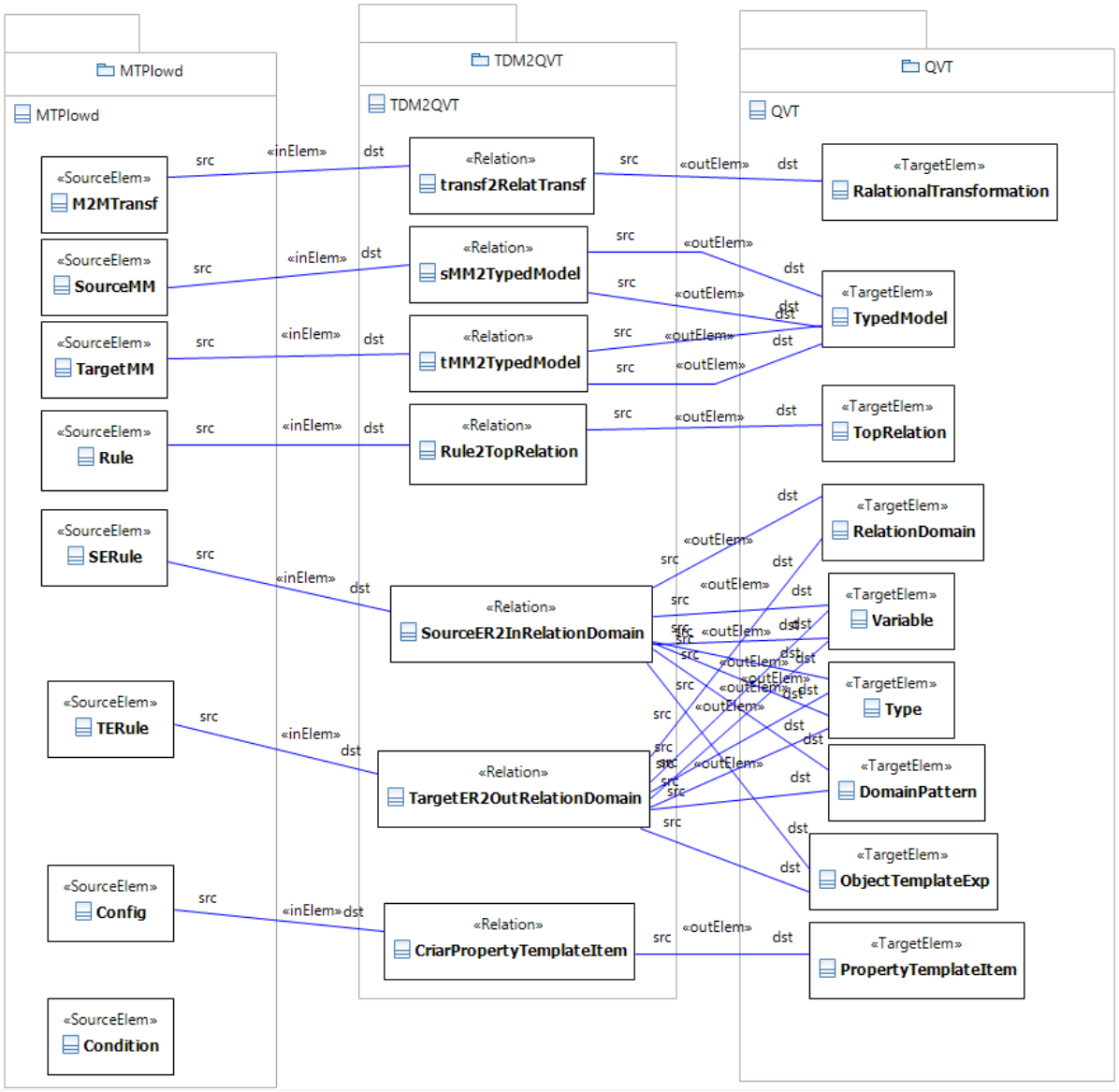


Figura 6.11 Especificação da transformação *TDM2QVT*

para o exemplo da transformação OO2RDBMS. Foi criado o módulo OO2RDBMS (linha 2) e definidas as entradas e saídas da transformação (linha 4). Em seguida, a regra *MapearClasse* é descrita a partir da linha 7. Cada elemento de saída é inicializado de acordo com a configuração definida no projeto de baixo nível. Por exemplo, as linhas 12 a 15 apresentam a configuração do elemento *RdbmsColunm*, gerado pela regra.

6.2.7 Transformação CodeQVT.mtl

A transformação *CodeQVT.mtl* tem o objetivo de gerar o código da transformação na linguagem QVT a partir de um modelo de entrada representado por um diagrama de classes em conformidade com o perfil *PQVT* (descrito no apêndice H). Analogamente à

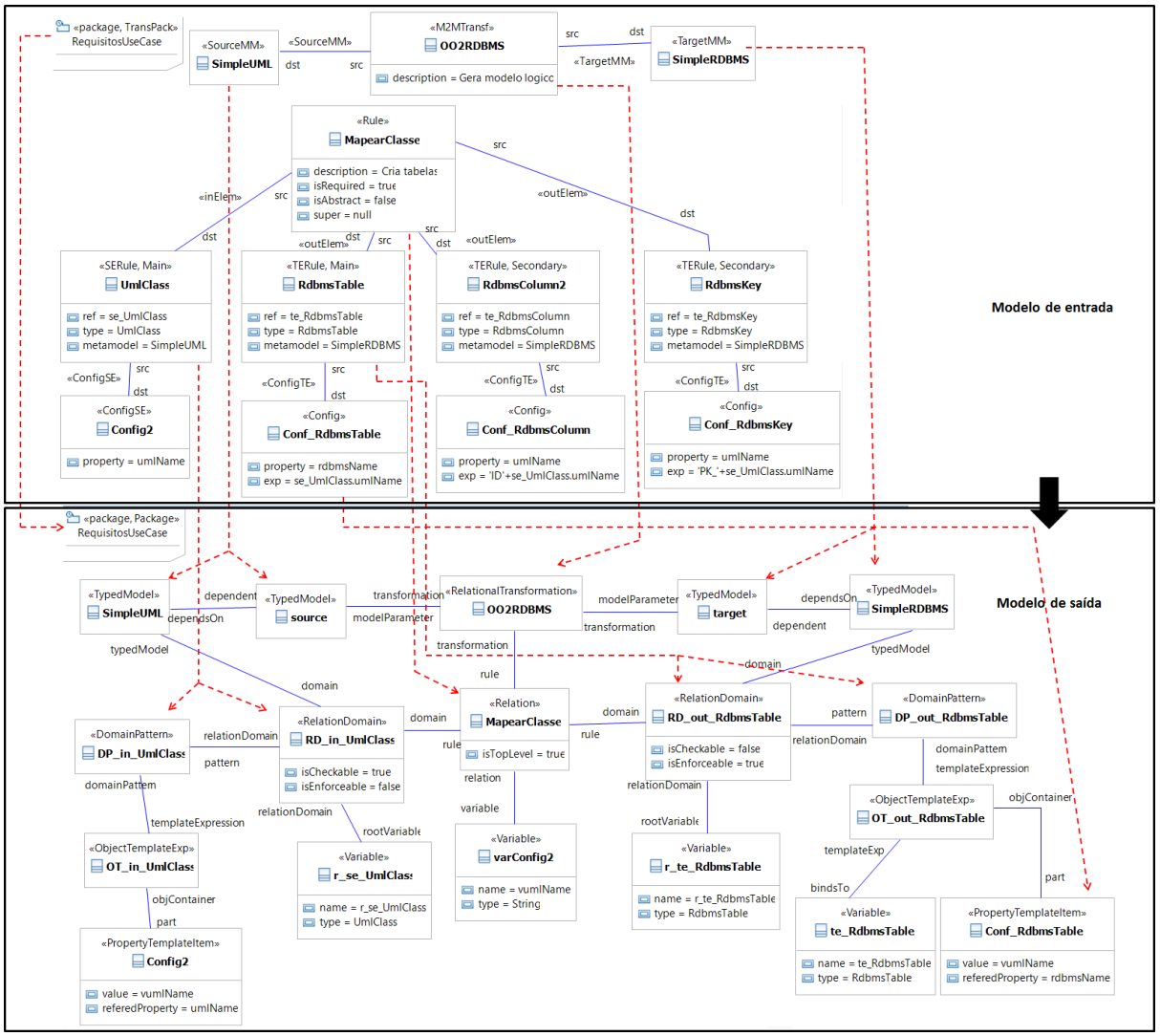


Figura 6.12 Exemplo do modelo de entrada e saída da transformação *TDM2TQVT*

transformação descrita na seção anterior, essa transformação foi construída na linguagem MTL.

A Figura 6.14 apresenta parte do código gerado pela transformação *CodeQVT.mtl* para o exemplo da transformação OO2RDBMS. A transformação OO2RDBMS recebe como parâmetros para execução o modelo *source* do tipo *SimpleUML* e o modelo *target* do tipo *SimpleRDBMS* (linha 2). Em seguida são descritas as relações *top relation MapearModelo* e *top relation MapearClasse*.

6.3 CONSIDERAÇÕES SOBRE O CAPÍTULO

Neste capítulo foram definidos o ambiente de desenvolvimento e a cadeia de transformações necessários para viabilizar o uso da abordagem proposta. O ambiente de desenvolvimento, diferentemente das soluções apresentadas na literatura, integra ferra-

```
1.  -- *** Gera modelo logico ***
2.  module OO2RDBMS;
3.  --*** modelos de saida e de entrada ***
4.    create OUT_SimpleRDBMS:SimpleRDBMS   from IN_SimpleUML:SimpleUML;
5.  -- *** regras da transformação ***
6.    -- *** Cria tabelas ***
7.   rule MapearClasse{
8.   from
9.        se_UmlClass:SimpleUML!UmlClass
10.  to
11.     te_RdbmsColumn:SimpleRDBMS!RdbmsColumn(
12.     rdbmsKey<-te_RdbmsKey,
13.     rdbmsOwner<-te_RdbmsTable,
14.     rdbmsName<-'ID_'+se_UmlClass.umlName,
15.     rdbmsType<-'INTEGER'
16.     ),
17.     te_RdbmsTable:SimpleRDBMS!RdbmsTable(
18.     rdbmsName<-se_UmlClass.umlName,
19.     rdbmsSchema<-thisModule.resolveTemp(se_UmlClass.umlNamespace,'te_RdbmsSchema')
20.     ),
21.     te_RdbmsKey:SimpleRDBMS!RdbmsKey(
22.     rdbmsOwner<-te_RdbmsTable,
23.     rdbmsName<-'PK_'+se_UmlClass.umlName
24.     )
25. } ....
```

Figura 6.13 Código da transformação OO2RDBMS em ATL gerada pelo framework

mentas *open source* já disponíveis no mercado, dissociando a abordagem da utilização de ferramentas específicas. A cadeia de transformações implementadas para (semi) automatizar o processo de desenvolvimento contempla não só o projeto e a geração do código, mas também os estágios iniciais de identificação de requisitos, característica que não foi observada em nenhuma proposta analisada.

```
1.  --*** transformation code ***
2.  transformation OO2RDBMS(  source:SimpleUML,  target:SimpleRDBMS)
3.  {
4.     top relation MapearModelo{
5.      vumlName:String;
6.      checkonly domain source r_se_UmlPackage:SimpleUML::UmlPackage{
7.       umlName=vumlName
8.      };
9.      enforce domain target
    r_te_RdbmsSchema:SimpleRDBMS::RdbmsSchema{
10.      rdbmsName=vumlName
11.     };
12.   }
13.   top relation MapearClasse{
14.    vumlName:String;
15.    checkonly domain source r_se_UmlClass:SimpleUML::UmlClass{
16.     umlName=vumlName
17.    };
18.    enforce domain target r_te_RdbmsTable:SimpleRDBMS::RdbmsTable{
19.     rdbmsKey=te_RdbmsKey:SimpleRDBMS::RdbmsKey{
20.        umlName='PK_'+vumlName
21.       },
22.     rdbmsColumn=te_RdbmsColumn:SimpleRDBMS::RdbmsColumn{
23.        umlName='ID'+vumlName
24.       },
25.     rdbmsName=vumlName
26.    };
27.  }...
```

Figura 6.14 Código da transformação OO2RDBMS em QVT gerada pelo framework

Capítulo 7

AVALIAÇÃO DO FRAMEWORK MDTD

O elemento central do framework MDTD é o processo de desenvolvimento MDTDproc que, apoiado em perfis, cadeia de transformação e ambiente, conduz o desenvolvimento de transformações de modelos. Desta forma, a avaliação do framework está centrada na avaliação do processo MDTDproc.

Processos de desenvolvimento de software são inerentemente complexos, pois envolvem um elevado número de atividades executadas por pessoas com diferentes *expertises*. Como consequência, avaliações de processos de softwares são geralmente realizadas em etapas que se complementam (SOMMERVILLE, 2010). Desta forma, a avaliação do MDTDproc foi dividida em duas etapas: um estudo de caso, para avaliar a eficácia do processo na condução do desenvolvimeto de transformações, e um experimento controlado, para avaliar a qualidade das transformações desenvolvidas com o processo MDTDproc em relação às transformações desenvolvidas de maneira *ad-hoc*. Uma comparação detalhada do processo MDTDproc com trabalhos relacionados (descritos no Capítulo 8) não foi realizada, pois não dispomos de acesso à especificação detalhada destes trabalhos. O perfil MTP foi avaliado através de um estudo exploratório. As Seções 7.1 e 7.2 apresentam respectivamente o estudo de caso e o experimento controlado e a Seção 7.3 discorre sobre a avaliação do perfil MTP.

7.1 AVALIAÇÃO DA EFICÁCIA DO PROCESSO MDTD$_{PROC}$

Para avaliar a eficácia do processo MDTDproc utilizamos a norma ISO/IEC-15504 para avaliação de processos de software (ISO, 2005). A descrição geral da norma pode ser encontrada no apêndice A. Definimos o estudo de caso com base nas diretrizes para experimentação de software de (WOHLIN et al., 2012) e o projeto do estudo de acordo com o *template Goal-Question-Metric (GQM)* (CALDIERA; ROMBACH, 1994).

Processos são avaliados segundo a norma ISO/IEC-15504 em duas dimensões: capacidade e processos. Neste trabalho, avaliamos o MDTDproc para capacidade de nível 1 em relação aos seguintes processos de referência: ENG.1 - Elicitação de requisitos de software; ENG.4 - Análise de requisitos de software; ENG.5: Projeto de software; ENG.6:

Construção de software, adaptados para o domínio de transformações de modelo. Esta adaptação está descrita no apêndice A.

De uma meneira geral, a norma define para cada processo de referência um conjunto de indicadores que devem ser avaliados como amplamente ou complemetamente alcançados (de acordo com uma escala também definida pela norma) para se atingir o nível de capacidade desejado. Os indicadores avaliam práticas básicas de engenharia de software, artefatos produzidos pelo desenvolvedor e recursos utilizados ao longo do desenvolvimento. A avaliação deve coletar evidências de que os indicadores foram satisfeitos. Maiores detalhes sobre como funciona a norma podem ser encontrados no apêndice A.

O estudo de caso está estruturado em quatro etapas: projeto do estudo; preparação para coleta de dados; coleta de dados; e análise dos dados coletados, detalhadas nos subtópicos a seguir.

7.1.1 Projeto do Estudo

O projeto do estudo consiste na definição dos seguintes itens: objetivos do estudo, questões de pesquisa, métricas, definição do cenário e definição do contexto.

7.1.1.1 Objetivos do Estudo O objetivo do estudo está definido na Figura 7.1 de acordo com o template GQM.

Analisar o processo de desenvolvimento de transformações *MDTDproc*
Com o propósito de avaliar a eficácia no desenvolvimento de transformações de modelos
Com respeito aos indicadores definidos para o nível de capacidade 1 da norma ISO/IEC -15504 de avaliação de processos de software nos processos relacionados a requisitos, análise, projeto e construção de software
Do ponto de vista do pesquisador como gerente de projeto
No contexto de estudantes de graduação e pós-graduação e especialistas DDM desenvolvendo transformações de modelos

Figura 7.1 Objetivo do estudo de caso de avaliação do processo MDTDproc

7.1.1.2 Questões de Pesquisa As seguintes quetões de pesquisa (QP) nortearam o estudo:

- **QP1: O processo conduz o desenvolvedor no desenvolvimento da transformação?** Nesta questão busca-se avaliar se as tarefas especificadas para o processo resultam em uma transformação de modelos.

- **QP2: O processo ajuda na construção dos modelos (artefatos) que deverão gerar a transformação?** Nesta questão busca-se avaliar se o processo conduz a construção dos artefatos necessários para alcançar o produto final que é a transformação de modelos, ou seja, se os passos definidos para a tarefa guiam o desenvolvedor na construção dos artefatos necessários à construção da transformação.

- **QP3: O processo pode ser executado com as ferramentas existentes atualmente?** Esta questão busca identificar se recursos tais como o uso de ferramentas de modelagem e engenhos de transformação disponíveis são suficientes para apoiar o processo.

7.1.1.3 Métrica Conforme especificado na norma ISO/IEC-15504 a métrica utilizada para avaliar o processo no nível de capacidade 1 é a *performance do processo*. A *performance do processo* é medida como:

$$PerformanceDoProcesso = GP + GR \tag{7.1}$$

onde, GP e GR representam respectivamente as práticas genéricas e os recursos genéricos definidos nos processos utilizados como referência para a avaliação. Adicionalmente, as práticas genéricas (GP) compreendem as práticas básicas (BP) e os *workproducts* (WP) também definidos pela norma para cada um dosprocessos de referência.

As práticas básicas (BP) consideradas relevantes para o contexto de transformações de modelos estão identificadas na tabela 7.1. Cada prática básica (BP) listada na tabela está relacionada também a uma questão de pesquisa definida para o estudo de caso (coluna 3 da tabela 7.1).

Os *workproducts* (WP) considerados relevantes para o domínio de transformações estão listados na tabela 7.2 a seguir. Cada *workproduct* está associado a uma ou mais questões de pesquisa (ver coluna 3 da tabela).

Os recursos genéricos (GR) considerados relevantes ao desenvolvimento na abordagem DDM estão listados na tabela 7.3.

7.1.1.4 Definição do Cenário O cenário do estudo consistiu em desenvolver uma transformação de modelo que representa um caso clássico no desenvolvimento de software na indústria atual. A transformação recebe como entrada um modelo conceitual de classes de um sistema e o transforma em um modelo de projeto na arquitetura Model View Controller (MVC)(DORAY, 2006). Por não ser um caso real, o cenário é classificado como um *toy problem*. A descrição detalhada do cenário utilizado no estudo pode ser encontrada no Apêndice D.

7.1.1.5 Definição do Contexto O estudo foi conduzido em ambiente acadêmico com estudantes de graduação e pós-graduação e por alguns especialistas em MDD selecionados de acordo com um perfil pré-definido. Para realizar essa seleção foi aplicado um questionário com o objetivo de avaliar o nível de conhecimento do aluno em DDM, em UML, em metamodelagem e em desenvolvimento de transformações. Somente foram selecionadas pessoas que tinham conhecimento em DDM, em UML e em metamodelagem. Conhecimento no domínio de transformações de modelo não foi obrigatório, embora fosse desejável ter participantes selecionados com e sem conhecimento em linguagens de transformação. Ao todo 34 pessoas responderam ao questionário que avaliava o perfil do candidato e dessas, 30 pessoas foram selecionadas, as demais não tinham o conhecimento mínimo requerido para realizar o estudo.

Tabela 7.1 Práticas básicas dos processos ENG1, ENG4, ENG5 e ENG6 da ISO/IEC-15504 adaptados para o domínio de transformações de modelos

Id	BP - Práticas Básicas	QP
1	ENG1.BP1: Obter requisitos do processo DDM a ser automatizado	1,2
2	ENG4.BP1: Identificar os requisitos da transformação	1,2
3	ENG4.BP2: Analisar os requisitos da transformação em termos de risco e capacidade de teste	1,2
4	ENG4.BP4: Priorizar e categorizar os requisitos	1,2
5	ENG4.BP6: Garantir a consistência e rastreabilidade bilateral entre os requisitos da transformação e os requisitos do processo DDM	1
6	ENG5.BP1: Desenvolver a arquitetura da transformação	1,2
7	ENG5.BP2: Alocar os requisitos da transformação aos componentes da arquitetura	1,2
8	ENG5.BP3: Definir interfaces	1,2
9	ENG5.PB4: Definir o comportamento dinâmico	1,2
10	ENG5.BP6: Desenvolver o projeto detalhado	1,2
11	ENG5.BP7: Desenvolver o critério de verificação	1,2
12	ENG5.BP8: Verificar o projeto da transformação	1,2
13	ENG5.BP9: Garantir a consistência e rastreabilidade bilateral entre o projeto da arquitetura da transformação e os requisitos da transformação	1
14	ENG6.BP1: Definir uma estratégia de verificação da unidade	1
15	ENG6.BP2: Desenvolver um critério de verificação da unidade	1
16	ENG6.BP3: Gerar o código da transformação	1,3
17	ENG6.BP4: Verificar unidade de software	1,2
18	ENG6.BP5: Gravar resultados de verificação de unidade	1
19	ENG6.BP6: Garantir consistência e rastreabilidade bilateral do projeto detalhado e unidades	1
20	ENG6.BP8: Garantir consistência e rastreabilidade bilateral dos casos de teste e unidades	1

Dois estudos foram planejados, o estudo piloto, para validar o projeto do estudo, e o estudo principal, que efetivamente foi utilizado para avaliar o processo MDTDproc.

7.1.2 Preparação para Coleta dos Dados

Dois métodos de coleta de dados foram utilizados no estudo: a coleta de dados indireta e a coleta de dados independente.

A coleta dos dados *indireta* foi realizada através da aplicação de questionários (disponíveis no apêndice B) que os participantes responderam ao final da execução de cada fase do processo.

Tabela 7.2 Artefatos requeridos pelos processos de referência ENG1,ENG4,ENG5 e ENG6 da ISO/IEC-15504 relacionados a transformação de modelos

Id	WP - Workproduct	QP
1	ENG01 - Requisitos do processo DDM	1,2
2	ENG04 - Plano de versão	1,2
3	ENG04 - Registro de rastreabilidade	1,2
4	ENG04 - Relatório de análise	1,2
5	ENG04 - Especificação dos requisitos de software	1,2
6	ENG04 - Critério de verificação	1,2
7	ENG05 - Projeto de arquitetura do software	1,2
8	ENG06 - Projeto detalhado do software	1,2
9	ENG05 - Registro de rastreabilidade	1,2
10	ENG06 - Plano de teste	1,2
11	ENG06 - Especificação de teste	1,2
12	ENG06 - Resultado do teste	1,2
13	ENG06 - Unidade de software	1,3
14	ENG06 - Registro de rastreabilidade	1

Tabela 7.3 Recursos que serão avaliados para o processo MDTDproc

Id	GR - Recursos genéricos	QP
1	Ferramentas de modelagem	3
2	Engenhos de transformação	3

A coleta de dados *independente* foi realizada através da análise dos artefatos produzidos pelos participantes. Os artefatos produzidos pelos participantes ao longo do estudo foram comparados com os documentos criados pelo próprio pesquisador. A partir desta análise o pesquisador preencheu um barema de avaliação dos artefatos. (disponível no apêndice C).

7.1.3 Execução do Estudo

Nesta etapa o estudo foi executado com base no projeto definido anteriormente e os dados foram coletados para serem posteriormente analisados.

Os participantes selecionados para o estudo foram arranjados, de acordo com suas respectivas disponibilidades de horário, em diversas replicações do estudo. Um total de seis replicações foram realizadas.

O estudo foi iniciado com um treinamento sobre o framework MDTD que compreendeu os seguintes itens: uma visão geral do processo MDTDproc; apresentação da linguagem de modelagem (perfil MTP); e uma introdução ao ambiente Eclipse configurado para executar o processo (ambiente de modelagem e execução de transformações). Foi utilizado como base para o treinamento um exemplo clássico de transformação, a transformação do modelo de classes para o modelo lógico de banco de dados. Os seguintes artefatos foram produzidos e entregues aos participantes do estudo de caso:

- apresentação do framework utilizada no treinamento dos participantes do estudo;
- documento com o cenário do estudo de caso;
- processo MDTDproc gerado pela ferramenta EPF sob a forma de um *site*;
- ambiente Eclipse com todos os *plug-ins* instalados, as transformações que automatizam o processo de desenvolvimento e o exemplo de transformação OO2DBMS desenvolvida no framework;
- questionários *on-line* para serem respondidos após cada fase do desenvolvimento da transformação;
- um exemplo de modelo de entrada para testar a transformação construída no final do estudo de caso.

O cenário desenvolvido no estudo foi o mesmo para todos os participantes e cada participante desenvolveu a transformação proposta em uma máquina independente. O desenvolvimento da transformação foi conduzido pelo processo MDTDproc, onde cada participante desempenhou todos os papeis definidos no processo. Após a execução de cada fase do processo o participante respondeu a um questionário *on-line* de coleta de dados. No final do estudo foi realizado um backup da *workspace* de cada participante para que o pesquisador pudesse fazer a análise dos artefatos produzidos definida na coleta independente dos dados (estas *workspaces* podem ser encontradas no *site* do projeto (MAGALHAES, 2016).

Os estudos foram acompanhados pelo pesquisador que teve participação como instrutor, durante o treinamento, e observador, durante a execução do estudo.

7.1.3.1 Detalhamento do Estudo de Caso Piloto O objetivo do estudo piloto foi identificar falhas no projeto do estudo (no treinamento e nos documentos entregues aos participantes). Por este motivo, foram selecionados para participar deste estudo dois alunos integrantes de um projeto de iniciação científica relacionado ao desenvolvimento do framework proposto. Estes alunos já tinham conhecimento do projeto, do processo e da linguagem proposta.

O estudo foi realizado em Julho de 2015 e teve duração de quatro dias (3 horas por dia). Na ocasião da realização deste estudo, o ambiente Eclipse ainda não estava pronto. Desta forma foi utilizado o software *Enterprise Architect (EA)* para modelagem da transformação proposta no cenário do estudo. Neste ambiente não foi possível executar de maneira automática as transformações que automatizam o processo proposto. Assim, os alunos tiveram que executá-las manualmente, ou seja, gerar manualmente o modelo inicial de cada fase do processo de desenvolvimento. Por este motivo as fases de projeto específico de plataforma e codificação não foram executadas no estudo piloto.

O estudo piloto foi importante, pois detectou necessidades de melhorias no projeto do estudo de caso, das quais detalhamos:

- modificação na forma como o treinamento deve ser aplicado aos participantes. Em vez de ser realizado de uma só vez antes de iniciar o estudo, foi dividido em fases, correspondentes às fases do processo, e realizado antes da execução de cada fase pelo participante;

- modificação do documento que descreve o cenário. Deve ser incluído neste documento um exemplo de modelo para cada metamodelo utilizado, pois os participantes estavam com dificuldade para entender estes metamodelos;

- revisão de algumas perguntas contidas no questionário de avaliação do processo que foram reescritas para terem um melhor entendimento;

- criação de documentos de apoio a algumas atividades do processo. As atividades do processo julgadas pelos participantes como de maior complexidade foram enriquecidas com documentos de apoio, anexados ao processo sob a forma de guias. Por exemplo, foram criados guias para execução de transformação e guias para aplicação de estereótipos nas diversas fases do processo.

7.1.3.2 Detalhamento do Estudo de Caso Principal O estudo de caso principal foi realizado com 30 pessoas divididas em seis replicações de acordo com a disponibilidade dos participantes. Na ocasião do estudo principal (em todas as replicações) o ambiente Eclipse já estava pronto e foi utilizado pelos participantes.

A primeira replicação do estudo foi realizada com nove participantes, seis que não tinham conhecimento em desenvolvimento de transformação e três que tinham experiência com desenvolvimento de transformações em projetos de pesquisa. Dentre os participantes que não tinham conhecimento em transformação, é relevante destacar que um deles não era estudante e sim doutor especialista em DDM. Os demais eram estudantes de graduação e pós-graduação. O estudo foi realizado em Setembro de 2015 e teve duração de três dias (3 horas cada dia).

A segunda replicação do estudo foi realizada com cinco alunos de graduação[1] sem experiência em desenvolvimento de transformações. O estudo foi realizado em outubro de 2015 e teve duração de cinco dias (1h50min cada dia).

A terceira replicação foi realizada em novembro de 2015 não com um estudante, mas com um doutor especialista em DDM e em metamodelagem, sem experiência em desenvolvimento de transformações.

A quarta replicação foi realizada também em novembro com um aluno do Mestrado em computação da Ufba que, por motivos particulares, não pode participar da primeira replicação.

A quinta replicação foi realizada no início de dezembro de 2015 com alunos de uma turma de pós-graduação em Engenharia de Software[2] que cursavam a disciplina de Desenvolvimento Dirigido a Modelos. Estes alunos tiveram contato com a abordagem DDM

[1]Disciplina de Tópicos em Engenharia de Software do curso de Sistemas de Informação da Universidade do Estado da Bahia, cujo tema abordado é o Desenvolvimento Dirigido a Modelos

[2]Faculdade Ruy Barbosa

durante os 15 dias da disciplina e então participaram do estudo. Não tinham, portanto, experiência em desenvolvimento de transformações.

A sexta replicação foi realizada no final de dezembro de 2015 com um aluno do Mestrado em Computação da Ufba que também, por motivos particulares, não pode participar das replicações anteriores.

7.1.4 Validação e Análise dos Dados

O estudo de caso piloto foi fundamental na preparação do estudo principal. Porem, considerando as condições de realização deste estudo (ferramenta diferente, falta de automação, dentre outros), os dados obtidos não foram utilizados na avaliação do processo. Portanto, a avaliação do processo foi realizada somente com os dados coletados no estudo principal.

As seções a seguir mostram os resultados destas análises. Inicialmente é apresentado o resultado do questionário aplicado antes do estudo de caso, destacando o perfil dos participantes selecionados. Em seguida, são apresentados os resultados da avaliação do processo propriamente dita. Os dados foram analisados utilizando a ferramenta de estatística R[3] cujos *scripts* e bases de dados estão disponíveis no site do projeto (MAGALHAES, 2016).

7.1.4.1 Perfil dos Participantes do Estudo de Caso Conforme mencionado anteriormente somente participaram do estudo pessoas com conhecimento em UML e em DDM. Para os selecionados, buscou-se então identificar qual o nível de experiência que eles tinham em DDM no que se refere à forma de aquisição do conhecimento e ao tempo de experiência. A Figura 7.2(A) mostra que a maioria dos participantes adquiriu conhecimento em DDM cursando disciplinas relacionadas à área ou através de projeto de pesquisa, não há participantes com experiência em DDM na indústria. A Figura 7.2(B) mostra o tempo de experiência dos participantes. Metade deles afirmam que apenas realizaram pequenos exercícios em disciplinas relacionadas, os demais têm experiência fora de sala de aula em projetos de pesquisa relacionados ao tema.

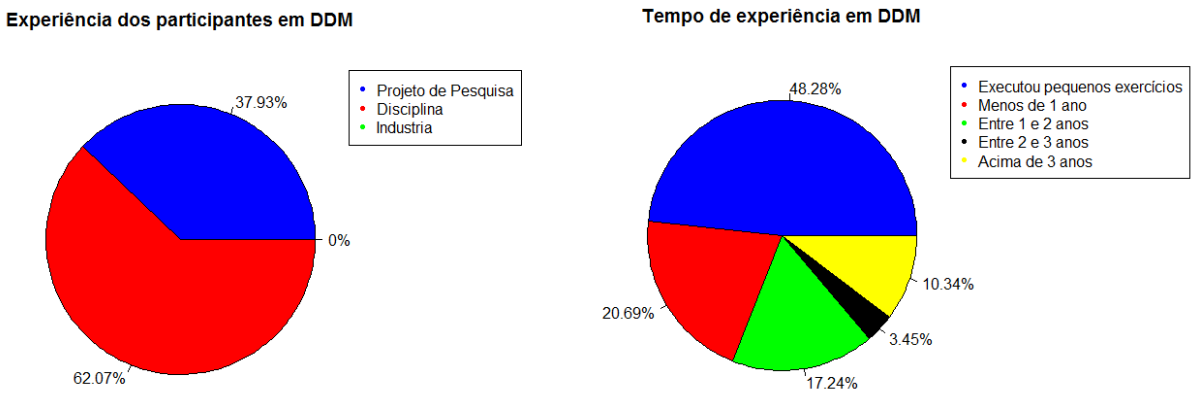


Figura 7.2 (A) Forma de aquisição do conhecimento em DDM; e (B) Tempo de experiência

[3]R Software - www.r-project.org

Buscou-se identificar também se os participantes tinham conhecimento em desenvolvimento de transformações (Figura 7.3). Das 30 pessoas que participaram do estudo 55,17% são usuários DDM, mas nunca desenvolveram uma transformação. 10,34% tem experiência com especificação de transformação, mas nunca as implementou. 27,29% já tiveram contato com alguma linguagem de transformação para desenvolver pequenas transformações e somente 6,9% se declaram experientes em desenvolvimento de transformações.

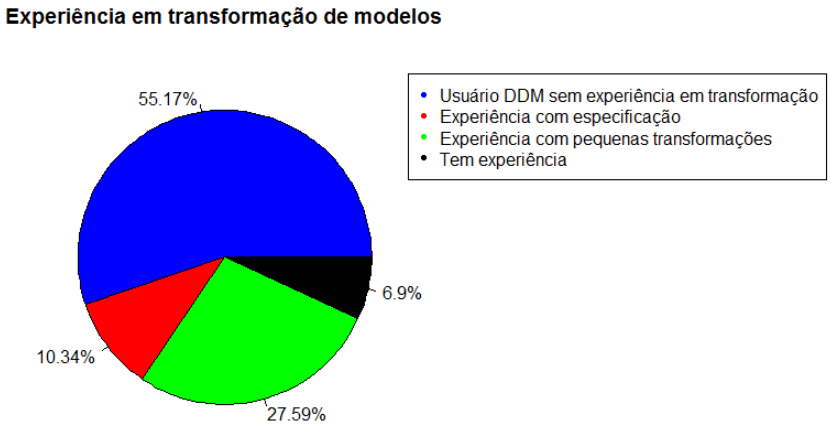


Figura 7.3 Experiência em desenvolvimento de transformações de modelos

7.1.4.2 Resultados da Avaliação do Processo O processo MDTDproc foi avaliado com base nos processos de referência definidos na norma ISO/IEC-15504, onde cada processo de referência corresponde a uma fase do MDTDproc. Desta forma, os resultados estão organizados nas subseções a seguir de acordo com cada fase do processo MDTDproc avaliada. São apresentados os indicadores que foram avaliados, práticas básicas (BP), *workproducts* (WP) e recursos genéricos (RG), e seus respectivos resultados. No final de cada subseção é apresentado também um quadro que resume a avaliação da fase.

Fase de planejamento (TCP) e especificação (TRM)

A avaliação das fases TCP e TRM foi realizada com base nos processos de referência ENG.1 (elicitação de requisitos) e ENG.4 (análise de requisitos de software).

A avaliação do indicador de *Prática Básica* (BP) foi realizada a partir dos dados coletados do questionário (apêndice B) aplicado aos participantes após a execução das fases TCP e TRM. O gráfico da Figura 7.4 mostra o resultado da avaliação de cada prática básica (BP) de acordo com a escala proposta pela norma, F (completamente alcançada), L (largamente alcançada), P (parcialmente alcançada) e N (não alcançada). O eixo x apresenta a BP avaliada e o eixo y o percentual atingido para cada nível da escala. Por exemplo, para o processo ENG1, a prática básica *ENG1.BP1: Obter requisitos do processo DDM a ser automatizado* foi considerada alcançada ou largamente alcançada por 88,45%

dos participantes do estudo (na figura representada pela soma das colunas azul e vermelha). As práticas básicas avaliadas estão definidas na tabela 7.1 do projeto do estudo de caso e resumidas na legenda da Figura 7.4 (na base ao lado esquerdo). Adicionalmente, o gráfico também apresenta a avaliação do indicador de prática genérica (*GP – entendimento das tarefas*) avaliado por 87,88% como completamente e largamente alcançado. Em suma, todas as práticas básicas foram avaliadas pela maioria dos participantes como *completamente alcançado* e *largamente alcançado*.

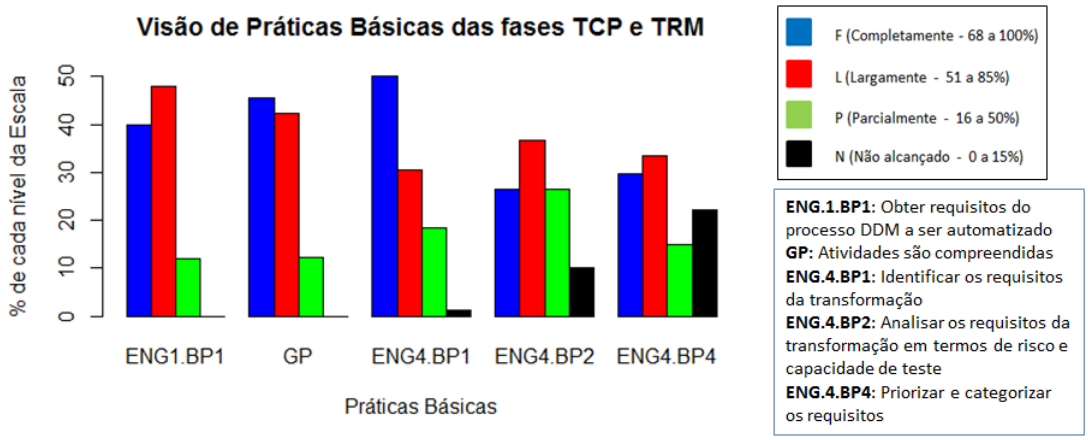


Figura 7.4 Nível de alcance da prática básica avaliada no processo ENG1 e ENG4

O indicador *Workproduct* (WP) foi avaliado com base na análise dos artefatos produzidos pelos participantes. No projeto do estudo de caso cinco artefatos foram definidos como relevantes para o contexto de desenvolvimento de transformações (ver tabela 7.2): *relatório de análise*, *plano de versão*, *registro de rastreabilidade*, *critério de verificação* e *especificação dos requisitos*. Foram avaliados neste estudo o *relatório de análise*, o *plano de versão* e a *especificação dos requisitos*. O *registro de rastreabilidade* não foi avaliado porque o ambiente proposto ainda não contempla recursos de rastreabilidade. O *critério de verificação* foi predefinido para todos os participantes como sendo o *teste de software*.

Para avaliar o *WP Especificação de Requisitos*, foi feita uma comparação entre os requisitos definidos por cada participante e o documento de requisitos definido como barema pelo pesquisador (apêndice C). No documento de requisitos definido pelo pesquisador foram identificados três requisitos (RF01, RF02 e RF03) considerados essenciais para o desenvolvimento da transformação e outros três requisitos (RF04, RF05, RF06) que representam o detalhamento dos requisitos essenciais e podem ser opcionalmente especificados. A comparação entre os *workproducts* evidenciou que os participantes identificaram de forma satisfatória os requisitos considerados essenciais para o desenvolvimento da transformação (Figura 7.5) e que alguns participantes foram capazes de enriquecer a especificação detalhando esses requisitos. Esse detalhamento está relacionado, por exemplo, à utilização de especializações e inclusões no diagrama de casos de uso de requisitos. Esses detalhes podem ajudar, por exemplo, na construção de regras mais modulares, mas a não especificação destes não implica em erro de concepção da transformação.

A Figura 7.6 mostra o resultado da avaliação dos *WP relatório de análise* e do *plano*

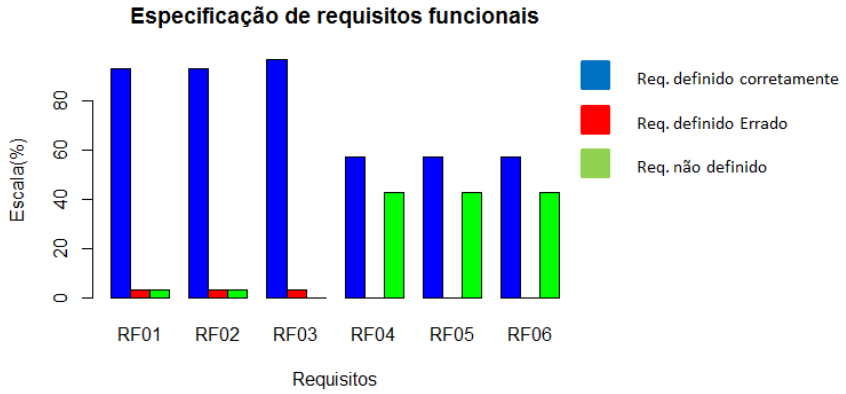


Figura 7.5 Análise dos requisitos funcionais definidos pelos participantes do estudo

de versão. O relatório de análise deve conter as informações definidas na análise dos requisitos da transformação, são elas: objetivo da transformação e objetivo dos requisitos (no gráfico representado pela coluna *Descrições*) e os metamodelos fonte e alvo (no gráfico representado pela coluna *Metamodelos*). O plano de versão é definido de acordo com a *priorização dos requisitos* (no gráfico representado pela coluna *Priorização*). Observou-se que os participantes definiram todos esses itens de forma satisfatória, (80%).

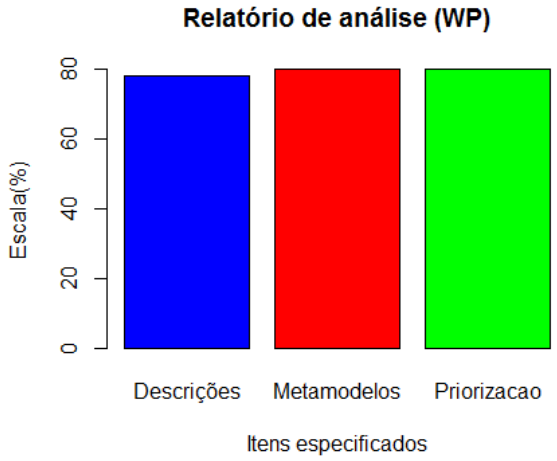


Figura 7.6 Análise dos artefatos produzidos pelos participantes

Para avaliar o indicador recursos genéricos (GR) foram utilizadas as respostas dos participantes no questionário de requisitos (apêndice B) aplicado após a execução das fases TCP e TRM. De acordo com o projeto do estudo de caso dois recursos são relevantes no contexto de desenvolvimento de transformação, as *ferramentas de modelagem* e os *engenhos de transformação*. Em relação a ferramentas de modelagem (Figura 7.7), a maioria dos participantes avaliou satisfatoriamente a ferramenta utilizada, contudo observou-se certa insatisfação de alguns pelo fato de ter sido usada uma ferramenta de modelagem UML que não estava totalmente customizada para o perfil MTP adotado. Em relação

ao engenho de transformação, os participantes em sua maioria ficaram satisfeitos com a ferramenta utilizada (Figura 7.7).

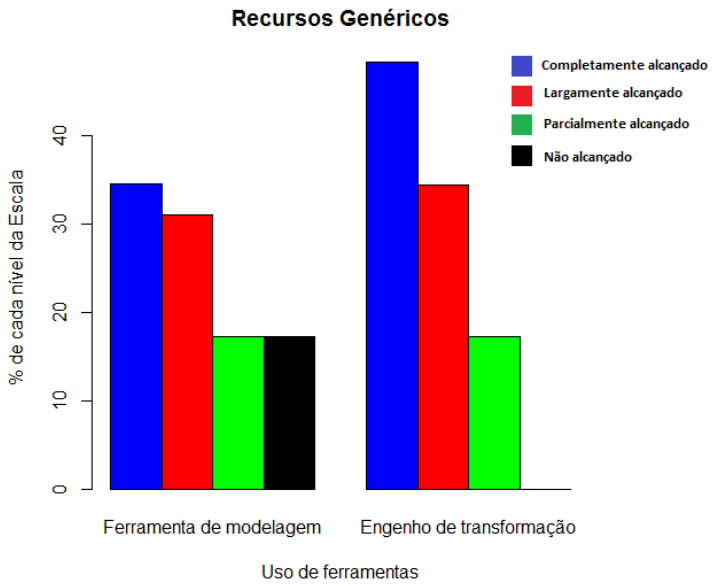


Figura 7.7 Nível de alcance dos recursos genéricos (GR) de ENG4

A tabela 7.4 apresenta um resumo dos resultados obtidos na análise do processo MDTDproc nas fases TCP e TRM. Todos os indicadores de práticas básicas (BP), *work-product* (WP) e recursos genéricos (GR) foram avaliados como completamente alcançados (F) e largamente alcançados (L) pelos participantes. Portanto, consideramos que para esta fase o processo alcaça seus objetivos.

Fase de projeto independente de plataforma (TDM)

A avaliação da fase TDM utilizou como referência o processo ENG.5 relacionado ao projeto de software.

A avaliação do indicador de prática básica (BP) foi realizada com base nos dados coletados do questionário aplicado aos participantes após a conclusão da fase TDM. Oito práticas básicas foram consideradas relevantes para o desenvolvimento de transformações (ver tabela 7.1 no projeto do estudo): *BP1 - desenvolver a arquitetura da transformação*; *BP2 – alocar os requisitos da transformação aos componentes da arquitetura*; *BP3 – definir interfaces*; *BP4 – descrever comportamento dinâmico*; *BP6 – desenvolver o projeto detalhado*; *BP7 – desenvolver critério de verificação*; *BP8 – verificar o projeto*; e *BP9 – garantir a consistência e rastreabilidade*; além da prática genérica (GP) *Entendimento das tarefas*.

Como pode ser observado na Figura 7.8, em relação ao entendimento geral das tarefas (GP), mais de 80% dos participantes declararam ter entendido o que devia ser realizado em cada tarefa. Das oito práticas básicas consideradas relevantes para o desenvolvimento

Tabela 7.4 Resumo do resultado da avaliação do processo MDTDproc nas fases TCP e TRM

Id	Práticas Básicas (BP) e Práticas Genéricas (GP)	RQ	Resultado
1	Práticas genéricas (o processo pode ser entendido)	1	87,78%
2	ENG1BP1 - Requisitos do domínio	1,2	88,00%
3	ENG4BP1 - Identificar os requisitos da transformação	1,2	80,49%
4	ENG4BP2 - Analisar riscos e capacidade de teste	1,2	63,27%
5	ENG4BP4 - Priorizar e categorizar os requisitos	1,2	62,96%
	Recursos Genéricos (GR)		
1	Ferramentas de modelagem	3	67,84%
2	Engenhos de transformação	3	82,14%
	Workproduct (WP)		
1	Requisitos do processo MDD	1,2	86,67%
2	Plano de versão	1,2	86,67%
3	Relatório de análise	1,2	79,44%
4	Especificação dos requisitos de software	1,2	77,78%

de transformações, cinco foram executadas, *BP1, BP2, BP3, BP4* e *BP6* e avaliadas como completamente alcançadas ou largamente alcançadas pela maioria dos participantes. As práticas *BP7* e *BP8*, relacionadas à verificação da transformação, não foram, executadas, pois o processo ainda não fornece apoio completo às atividades relacionadas a essa prática. Além disso, demandam conhecimento em métodos formais que os participantes não tem. A *BP9*, relacionada à rastreabilidade, também não foi executada, pois o ambiente ainda não provê este recurso.

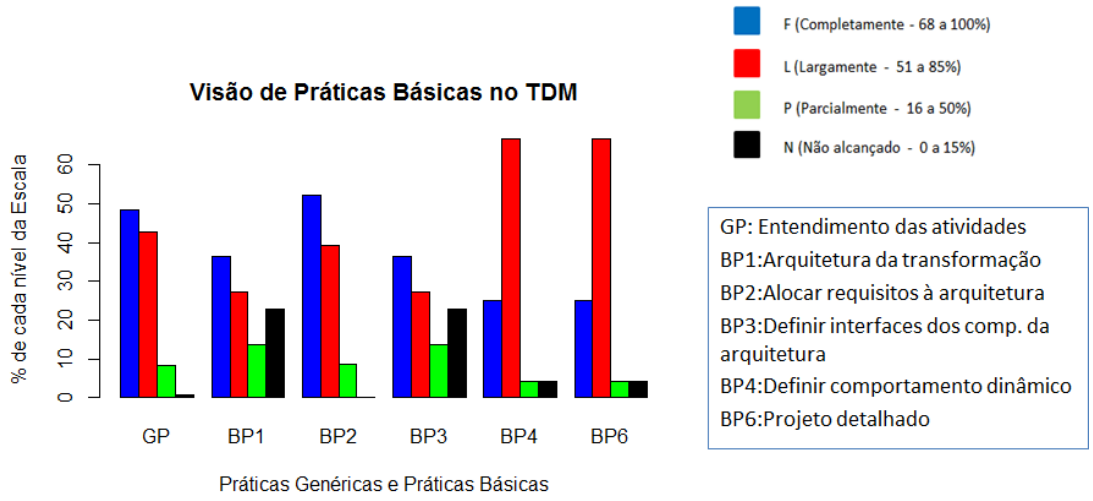


Figura 7.8 Nível de alcance da prática básica avaliadas para a fase TDM

O indicador *workproducts* (WP) foi avaliado com base nos dados coletados após análise dos artefatos produzidos pelos participantes. Conforme descrito na tabela 7.2 do projeto do estudo, os seguintes artefatos foram considerados relevantes para o contexto de trans-

formação de modelos: *projeto de arquitetura do software*; *projeto detalhado do software*; e *registro de rastreabilidade*. A arquitetura da transformação desenvolvida continha um único componente. Para esse componente cada participante deveria então realizar o projeto de alto nível e o projeto de baixo nível.

No projeto de alto nível foi avaliado se o participante conseguiu definir as relações entre os elementos dos metamodelos fonte e alvo. Três relacionamentos entre metamodelos deveriam ser definidos segundo o barema do pesquisador (apêndice C). A Figura 7.9 mostra que a maioria dos participantes conseguiu definir corretamente esses três relacionamentos. Alguns também definiram o relacionamento de maneira parcial, por exemplo, definiram relações do tipo 1 para N como sendo apenas 1 para 1.

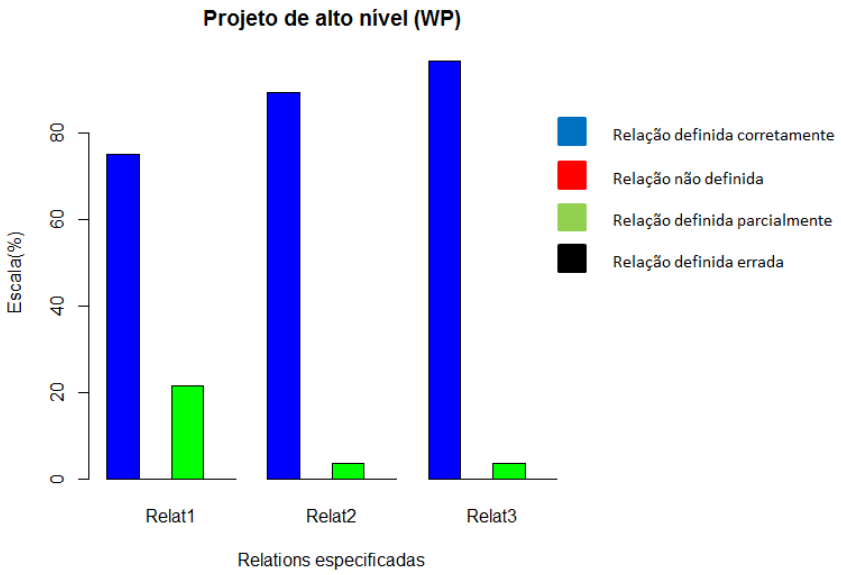


Figura 7.9 Análise das relações especificadas no projeto de alto nível

No projeto de baixo nível avaliou-se a definição das regras e a configuração dos elementos de saída de cada regra. A Figura 7.10 mostra o resultado da análise. Em 7.10(A), observa-se que a definição das regras foi feita corretamente por quase 100% dos participantes, pois ela é gerada automaticamente pelo processo a partir do modelo de alto nível. Somente nos casos em que o participante fez alterações após a geração, algum erro foi inserido.

Em relação à configuração dos elementos de saída da transformação, a Figura 7.10(B) mostra duas visões para cada regra, a visão de *configuração simples* e a visão de *configuração complexa*. Foi considerada uma *configuração simples* aquela que consiste de uma operação de atribuição simples e apenas atribui um valor a um elemento do modelo gerado. Foi considerada uma *configuração complexa* aquela que inicializa objetos usando expressões detalhadas da linguagem textual definida no MTP (ex. chamada a outras regras). Como pode ser observado houve uma incidência maior de erros nesta etapa do processo do que nas fases anteriores, pois esse é o mais baixo nível da modelagem e requer maior conhecimento dos metamodelos envolvidos. Particularmente, esses erros

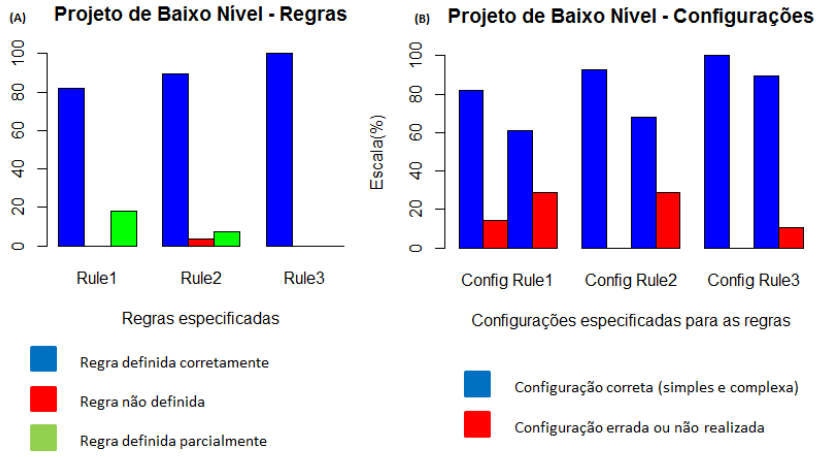


Figura 7.10 Nível de alcance do *workproduct* do projeto de baixo nível

foram mais expressivos em *configurações complexas* que requerem maior conhecimento da linguagem textual proposta.

Para avaliar o indicador de recursos genéricos (GR) foram utilizados os dados coletados no questionário respondido pelos participantes após a fase de projeto (apêndice B). Para ambas os tipos de ferramenta avaliados, ferramentas de modelagem e engenho de transformação, a avaliação foi satisfatória com a maioria dos participantes concordando que as ferramentas atenderam completamente e largamente às necessidade do processo (Figura 7.11).

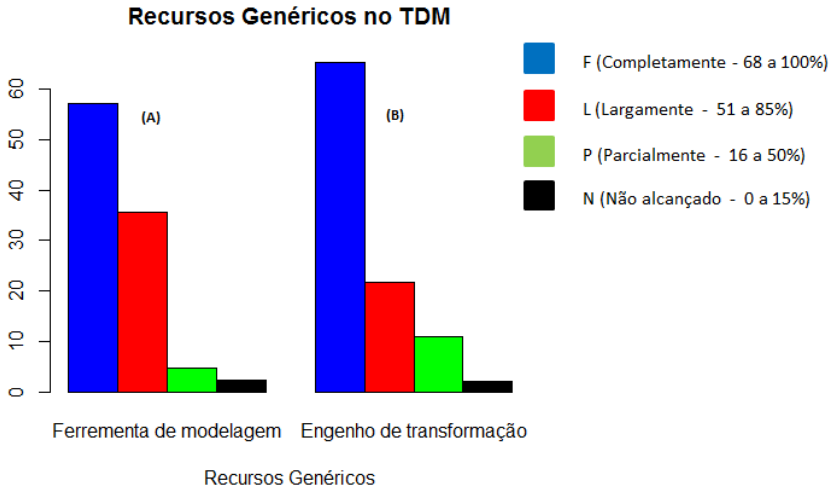


Figura 7.11 Nível de alcance do dos recursos genéricos na fase TDM

A tabela 7.5 apresenta um resumo dos resultados obtidos na análise do processo MDTDproc na fase TDM. As práticas básicas relacionadas à verificação da transformação e ao registro de rastreabilidade não foram avaliadas neste estudo, pois os participantes

não tinham conhecimento para tal ou a funcionalidade ainda não é contemplada pelo framework. Todas as demais práticas básicas (BP) e a prática genérica (GP) foram avaliadas pelos participantes como largamente alcançada ou completamente alcançada. Os *workproducts* (WP) e recursos genéricos (GR) também foram avaliados satisfatoriamente. Os indicadores relacionados à definição da arquitetura da transformação (id 2 e 4) tiveram índice inferior aos demais indicadores, pois como a transformação desenvolvida era formada por um único componente alguns participantes não definiram a arquitetura. Desta forma, será importante reavaliar esses indicadores em novos estudos de caso. Todos os indicadores analisados, contudo, foram avaliados como completamente e largamente alcançados. Portanto, considerameos que processo alcança seus objetivos para a fase TDM.

Tabela 7.5 Resumo do resultado da avaliação do processo MDTDproc na fase TDM

Id	**Práticas Básicas (BP) e Práticas Genéricas (GP)**	QP	Resultado
1	Práticas genéricas	1	90,96%
2	ENG5BP1 - Desenvolver a arquitetura da transformação	1,2	63,67%
3	ENG5BP2 - Alocar os requisitos da transformação aos componentes da arquitetura	1,2	91,33%
4	ENG5BP3 - Definir interfaces	1,2	63,67%
5	ENG5BP4 - Descrever o comportameto dinâmico	1,2	91,67%
6	ENG6BP6 - Desenvolver o projeto detalhado	1,2	91,67%
	Recursos Genéricos (GR)		
1	Ferramentas de modelagem	3	92,81%
2	Engenho de transformação	3	86,94%
	Workproduct (WP)		
1	Projeto da arquitetura	1,2	100%
2	Projeto detalhado	1,2	79,26%

Fase de projeto específico de plataforma (TSM) e codificação (Code)

A fase TSM corresponde ao projeto da transformação em uma plataforma específica. Neste estudo de caso os participantes transformaram o projeto independente de plataforma construído no final da fase TRM em um projeto específico na linguagem ATL. O modelo ATL foi gerado, conforme a especificação realizada, por 100% dos participantes que chegaram a esta fase do processo. Os participantes visualizaram o modelo gerado (em um diagrama de classes), mas nenhuma modificação foi feita neste modelo. Ele foi utilizado para gerar o código fonte da transformação em ATL.

A avaliação da fase *Condificação* utilizou como referência o processo ENG.6 relacionado à construção de software. Dentre as práticas básicas definidas na ISO/IEC-15504 para este processo, sete foram consideradas importantes para o domínio de transformação (ver na tabela 7.1): *BP1 - definir uma estratégia de verificação da unidade; BP2 - desenvolver um critério de verificação da unidade; BP3 - gerar o código da transformação; BP4 - verificar unidade de software; BP5 - gravar resultado da verificação da unidade;*

BP6 - garantir consistência e rastreabilidade entre projeto e unidades; e BP8 - garantir consistência e rastreabilidade entre casos de teste e unidade. Neste estudo de caso foram avaliadas as práticas básicas *BP3 - gerar código da transformação, BP4 - verificar unidade de software e BP5, gravar resultados.* As *BP1* e *BP2* se referem ao critério de verificação, que neste estudo foi previamente definido como sendo o teste de software. Analogamente às fases anteriores, as *BP6* e *BP8* não foram avaliadas, pois se referem à rastreabilidade, ainda não contemplada no trabalho.

Referente à *BP3*, dos 30 participantes 28 tiveram o código da transformação gerado conforme sua especificação, pois 2 participantes não chegaram a concluir o estudo. Para os códigos gerados, 19 transformações não apresentaram erro de sintaxe (67,86% na Figura 7.12A) e 9 transformações apresentaram algum tipo de erro (32,14% na Figura 7.12A) decorrente de definições incompletas ou incorretas de configuração. A especificação da configuração dos elementos de saída é realizada utilizando a linguagem OCL e algumas instruções especificadas para o metamodelo MMTlowdesign. Para estas instruções não foi implementado um compilador que avalie as configurações definidas pelos participantes do estudo. Como consequencia, erros de conformidade da configuração especificada pelo participante em relação à instrução definida no metamodelo levaram a erros de sintaxe no código ATL gerado. Das 28 transformações geradas 10 foram modificadas manualmente, pela inclusão e ou alteração de algumas linhas. Observa-se que a quantidade de linhas de código (LOC) incluídas (1,55% na Figura 7.12B) e ou alteradas (1,61% na Figura 7.12B) foi bastante reduzida em relação ao LOC gerado automaticamente pelo framework (96,84% na Figura 7.12B).

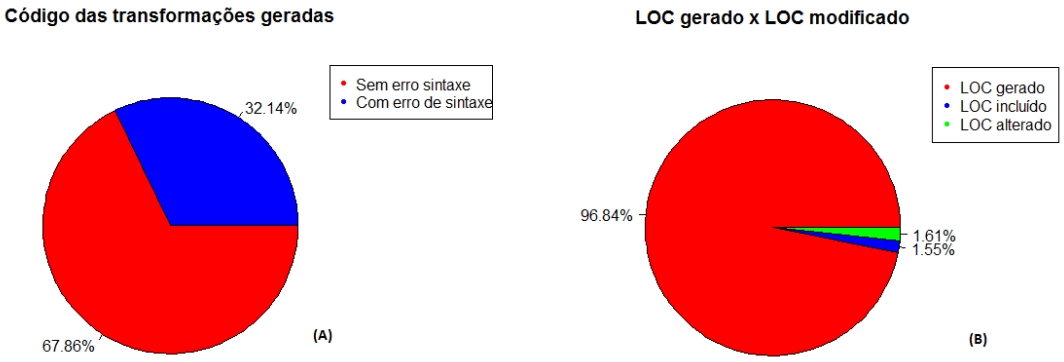


Figura 7.12 Análise dos códigos produzidos pelos participantes

Referente à *BP4*, o critério de verificação utilizado para validar a transformação construída foi o teste. Na fase de projeto, foram definidos pelos participantes os casos de teste da transformação. 24 transformações foram executadas (85,71%). Destas, 19 produziram o modelo de saída esperado (79,17% na Figura 7.13) em relação ao barema utilizado pelo pesquisador, 4 transformações geraram parcialmente o modelo de saída (16,7% na Figura 7.13), pois a transformação não foi completamente especificada, e somente uma transformação (4,17% na Figura 7.13) não conseguiu gerar um modelo de saída. É importante colocar que das 19 pessoas que produziram modelo de saída esperado 7 tinham

se declarado conhecedoras de linguagem de transformação, as demais conhecem DDM, mas nunca haviam desenvolvido transformações.

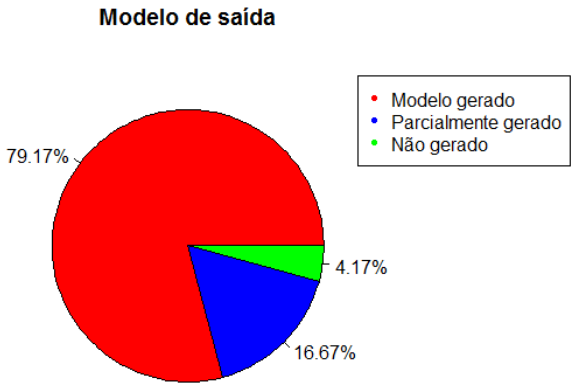


Figura 7.13 Análise do modelo de saída gerado pelas transformações construídas

A tabela 7.6 apresenta um resumo dos resultados obtidos na análise do processo MDTDproc na fase de codificação. Embora o MDTDproc possua diversas tarefas relacionadas à verificação de transformação, essas tarefas não foram avaliadas. Assumiu-se, portanto, que a validação da transformação seria feita a partir de casos de teste. Os participantes especificaram, em linguagem natural, diversos casos de teste que foram utilizados no final do processo para testar se a transformação gerava o modelo desejado como saída. Quanto a *BP3*, 100% dos participantes que alcançaram essa fase do processo conseguiram gerar o código da transformação conforme sua especificação. Em relação a *BP4*, dentre as transformações geradas, 85,71 foram testadas e todos os resultados foram registrados (*BP5*).

Nesta fase do processo foi avaliado o recurso genérico (GR), engenho de transformação. Em todos os casos o engenho foi executado satisfatoriamente e o código foi gerado. Como reflexo da avaliação das práticas básicas, todos os *workproducts* (WP) foram satisfatoriamente avaliados.

Em suma, todos os indicadores foram avaliados como completamente e largamente alcançados pela maioria dos participantes do estudo. Portanto, consideramos que o processo alcança seus resultados para a fase de TSM e codificação.

7.1.4.3 Validade da Análise Esta seção descreve as ameaças à validade interna, validade externa, validade de contrução e confiança da análise.

Validade interna. Ameaças à validade interna estão relacionadas à possibilidade de fatores não controlados influenciarem nos resultados obtidos.

Este estudo de caso está baseado no uso do processo MDTDproc pelos participantes do estudo e sua observação quanto às evidências de que os atributos medidos foram alcançados. Portanto, a experiência do participante no desenvolvimento de transformações pode ser determinante na sua avaliação. Desta forma algumas medidas foram tomadas para diminuir as ameaças relacionadas ao nível de experiência dos participantes: a seleção

Tabela 7.6 Resumo do resultado da avaliação do processo MDTDproc na fase Codificação

Id	**Práticas Básicas (BP)**	QP	**Resultado**
1	ENG6BP3 - Gerar o código da transformação	1,3	100% gerado
2	ENG6BP4 - Verificar unidade de software	1,2	85,71% executadas e testadas
3	ENG6BP5 - Gravar o resultado da verificação da unidade	1	Todos gravaram
	Recursos Genéricos (GR)		
1	Engenhos de transformação	3	100% gerado
	Workproducts (WP)		
1	Plano de teste	1,2	100%
2	Especificação de teste	1,2	100%
3	Resultado de teste	1,2	100%
4	Unidade de software	1,3	Código gerado

dos participantes considerou dentre o universo de alunos de cursos relacionados à ciência da computação, apenas aqueles que tinham conhecimento em DDM e em UML. Também foi identificado o nível de conhecimento e experiência desses participantes em desenvolvimento de transformações de modelos. Com base nesses dados os participantes foram então selecionados; garantiu-se com isso que todos os alunos têm conhecimento em DDM e que existem alunos com e sem experiência de desenvolvimento de transformações participando do estudo. Adicionalmente, a interpretação dos participantes no que se refere ao alcance ou não dos atributos medidos pode ser subjetiva e influenciar no resultado final do estudo. Desta forma, foi utilizada como escala para medir o alcance dos atributos a escala utilizada pela norma ISSO/IEC-15504 de avaliação de processo de software.

Outro ponto considerado está relacionado à experiência do participante no processo de desenvolvimento MDTDproc. O estudo envolve a avaliação de um processo novo que está sendo utilizado pelo participante pela primeira vez. Sendo assim, para diminuir o risco de mau uso do processo, foi disponibilizado para cada participante um site com todas as informações contidas no processo, para serem acessadas pelos participantes durante a execução do estudo. Além disso, foi realizada uma apresentação prévia do processo proposto e da linguagem de modelagem (MTP) utilizado no processo. O desenvolvimento de software na abordagem DDM envolve também o uso de ferramentas de modelagem e engenho de transformação. Para evitar problemas com o uso do ambiente de desenvolvimento, os participantes também foram treinados no uso do ambiente.

Validade externa. Ameaças à validade externa estão relacionadas à possibilidade de generalização dos resultados para outros cenários.

O desenvolvimento dirigido a modelos é uma abordagem recente, mas com significativo potencial em algumas áreas de desenvolvimento na indústria (HUTCHINSON; WHITTLE, 2011). Portanto o uso de um processo de desenvolvimento de transformações é relevante não só na academia, mas também na indústria. Contudo, por ser uma aborda-

gem recente, não foi possível utilizar neste estudo um caso real da indústria para realizar a avaliação do processo. Para diminuir esse risco escolheu-se construir uma transformação comumente utilizada no desenvolvimento de software convencional, o uso da arquitetura MVC. A generalização do resultado deste estudo para outros ambientes requer, contudo, a realização de novos estudos de caso em ambiente reais de desenvolvimento usando a abordagem de DDM. O estudo foi replicado seis vezes. Novas replicações podem ser planejadas com casos reais.

Outro ponto importante a ser analisado é o tamanho da amostra utilizada no estudo. O conhecimento necessário para realização do estudo (DDM e UML) foi determinante no tamanho da amostra (30 participantes), pois se trata de um perfil especializado de participante, difícil de ser encontrado. Desta forma, realizar o estudo com um número maior de participantes também é importante para validar e generalizar os resultados obtidos.

Adicionalmente, a complexidade da transformação desenvolvida também pode impactar nos resultados obtidos. Este estudo de caso construiu uma transformação comumente utilizada no desenvolvimento de software, mas que não tem um grau de complexidade muito elevado. Portanto, a realização de estudos que envolvam transformações mais complexas também se faz necessário para confirmar os resultados obtidos.

Validade de construção. Ameaças à validade de contrução estão relacionadas às medidas utilizadas pelo pesquisador, se estas representam realmente o que se pretende avaliar.

Medir a eficácia de um processo é uma tarefa difícil, uma vez que envolve pessoas executando tarefas. Como consequência, a avaliação dependente da opinião dessas pessoas. Para diminuir o risco desta avaliação foi utilizado como referência o modelo de avaliação do processo de software sugerido pela ISO/IEC-15504. Os atributos medidos referem-se aos processos de referência para requisitos, análise, projeto e construção de software adaptados ao domínio de transformações. É importante colocar que essa adaptação foi realizada pelo pesquisador com base nas necessidades de desenvolvimento de transformações, mas também pode representar uma ameaça ao estudo. A medição foi realizada através de questionários que buscaram coletar as evidências de alcance de cada atributo medido. Para diminuir o risco de entendimento das perguntas que compõe cada questionário, as perguntas possuem uma explicação indicando o que deve ser observado no processo para responder a pergunta.

Confiança. Ameaças à confiança do estudo estão relacionadas ao nível de dependência, dos dados e do estudo, ao pesquisador.

O estudo foi conduzido pelo próprio pesquisador em todas as replicações. Para diminuir as ameaças em replicação do estudo por outros pesquisadores, todo o material utilizado no estudo foi disponibilizado eletronicamente (MAGALHAES, 2016): ambiente de desenvolvimento; apresentações do treinamento realizado previamente com os participantes; e os questionários aplicados. Todos os questionários foram automatizados para serem enviados eletronicamente aos participantes. Adicionalmente, os sripts de análise dos dados na ferramenta R estão disponibilizados para serem executados pelos pesquisa-

dores em estudos posteriores.

7.1.5 Dificuldades Encontradas

Durante o estudo o pesquisador, no papel de observador, registrou algumas dificuldades encontradas pelos participantes.

A primeira dificuldade observada está relacionada ao uso de processos de desenvolvimento de software pelos participantes. Dada à importância dos processos no desenvolvimento de software atual, esperava-se que os participantes do estudo de caso já tivessem utilizado algum processo para desenvolvimento de software. Contudo o primeiro contato dos participantes com o MDTDproc evidenciou um desconhecimento dos elementos que compõe um processo de software o que consequentemente levou a uma dificuldade inicial em executar o estudo. Outro ponto importante que também está relacionado ao uso de processo foi a observação de certa resistência de alguns participantes em seguir os passos indicados no processo. Como consequência esses participantes tiveram dúvidas que, em sua maioria, estavam detalhadas no processo, mas que não tinham sido adequadamente executadas.

A dificuldade de compreensão de alguns conceitos utilizados pelo MDTD também foi observada. Por exemplo, distinguir os conceitos de requisitos do domínio, requisitos funcionais e requisitos não funcionais. Essas dúvidas foram reduzidas à medida que os participantes exploravam os recursos oferecidos pelo processo, tais como guias e exemplo, que detalhavam esses conceitos.

Dificuldades relacionadas ao entendimento das abstrações entre modelos, metamodelos e metametamodelos também foram observadas. Alguns participantes demoraram algum tempo para perceber que transformações processam modelos, mas são escritas em relação aos metamodelos. Assim fizeram a especificação inicial usando os elementos dos modelos fornecidos como exemplo. Em seguida tiveram que refazer a especificação usando os elementos do metamodelo.

Conhecimento em metamodelagem é essencial para desenvolver transformações, pois é fundamental entender os metamodelos fonte e alvo utilizados. A dificuldade de alguns participantes em entender os metamodelos utilizados na transformação desenvolvida dificultou também a especificação da inicialização dos atributos dos elementos que seriam gerados pela transformação. Alguns não conseguiram compreender o metamodelo e deixaram a especificação incompleta.

Além das dificuldades observadas pelo pesquisador, após o estudo os participantes também reportaram algumas dificuldades encontradas ao longo do desenvolvimento. As principais estão listadas abaixo:

- dificuldade para aplicar o perfil aos modelos gerados, lembrar quais os elementos que deveria ser estereotipados;
- dificuldade em conceitos básicos relacionados à transformação de modelos, por exemplo, o que significa relacionar elementos dos metamodelos;
- dificuldade para identificar quais os elementos que participam de uma relação;

- dificuldade em identificar qual o tipo da relação a ser criada, por exemplo, duas relações de 1-1 ou uma relação de 1-N;
- dificuldade em definir a inicialização dos elementos no projeto de baixo nível, principalmente quando envolve dependência entre regras;
- dificuldade em definir casos de testes para a transformação.

7.1.6 Considerações Sobre a Avaliação do Processo

Este estudo de caso buscou avaliar a eficácia do processo MDTDproc utilizando como base os indicadores propostos pela ISO/IEC-15504. Para isso, a norma foi analisada e adaptada para o domínio de transformações selecionando-se os itens considerados relevantes para um processo DDM. Embora alguns itens da norma tenham sido selecionados como relevantes, eles não foram avaliados neste estudo de caso, a exemplo da rastreabilidade dos modelos ao longo do processo e da verificação formal da transformação, pois ainda não são contemplados na versão atual do processo ou exigem uma *expertise* que os participantes do estudo não tinham. Alem disso, alguns indicadores, embora tenham alcançado o valor aceitável pela norma, evidenciaram que algumas tarefas do processo ainda precisam ser melhor detalhadas para facilitar a sua execução, como, por exemplo, a definição da arquitetura da transformação e a especificação do projeto de baixo nível. Em relação à arquitetura, novos estudos que envolvam transformações com n componentes são necessários para melhor avaliar essas tarefas. Em relação ao projeto de baixo nível, identificou-se uma dificuldade relacionada ao uso da liguagem textual definida, que deve ser melhor detalhada e exemplificada.

Três indicadores foram avaliados no processo, de acordo com a norma ISO/IEC-15504: as práticas básicas e genéricas (BP e GP), os *workproducts* (WP) e os recursos genéricos (GR). Em todos eles, para os itens avaliados, a meta estabelecida pela norma foi alcançada, ou seja, o indicador foi largamente ou completamente atingido. Portanto o processo MDTDproc atinge o seu objetivo, para o caso estudado. Evidenciamos, contudo, a necessidade de novos estudos para que esse resultado possa ser generalizado.

Com base nos resultados obtidos pelos indicadores avaliados, pode-se dizer também que às três questões de pesquisa definidas na Seção 7.1.1.2 foram alcançadas. As questões Q1 e Q2, que avaliam se o processo conduz o desenvolvedor na construção de transformações e na construçaõ dos artefatos intermediários necessários para chegar ao código da transformação, foram avaliadas com base nos indicadores de práticas genéricas, práticas básicas e artefatos construídos (ver mapeameneto de indicadores com as questões de pesquisa, por exemplo, na tabela 7.1). Independente destes indicadores, o estudo de caso mostrou que desenvolvedores com pouca ou nenhuma experiência em transformação de modelos conseguiram desenvolver a transformação proposta seguindo o processo MDTDproc e geraram um código executável e testado. Analogamente, a questão de pesquisa Q3, que avalia a eficácia das ferramentas utilizadas no estudo, pode ser respondida tanto pelos indicadores de recursos genéricos, que foram amplamente atendidos, quanto pelos resultados obtidos pelos participantes com a geração automática dos artefatos e do código da transformação no final do processo.

Consideramos os resultados deste estudo um indicativo de que a abstração proposta no desenvolvimento de transformações através do uso do DDM e de um perfil independente de linguagem apoiados por um processo de desenvolvimento reduz a complexidade inerente ao desenvolvimento de transformações.

7.2 EXPERIMENTO CONTROLADO PARA AVALIAÇÃO DA QUALIDADE DA TRANSFORMAÇÃO

Esta seção apresenta o experimento controlado realizado para avaliar a qualidade das transformações construídas com o framework MDTD em relação às transformações construídas diretamente em código.

O experimento utilizou como referência o trabalho de (AMSTEL; MARCEL; BRAND, 2010) sobre avaliação da qualidade de transformações de modelos. Foi avaliada a *qualidade interna da transformação*[4], que segundo (AMSTEL; MARCEL; BRAND, 2010) é avaliada de acordo com os seguites atributos, utilizados para nortear esse experimento:

- *Understandability*, relacionado à quantidade de esforço requerido para entender o propósito de uma transformação;
- *Modifiability*, relacionado à quantidade de esforço requerido para adaptar a transformação com novos requisitos;
- *Reusability*, relacionado às partes de uma transformação que podem ser reusadas em outras transformações;
- *Modularity*, relacionada à capacidade das transformações de modelo serem sistematicamente separadas e estruturadas;
- *Completeness*, indica que qualquer modelo instância do metamodelo pode ser processado pela transformação;
- *Consistency*, relacionado à uniformidade de implementação da transformação, ou seja, se é utilizado um estilo de programação uniforme em toda a transformação; e
- *Conciseness*, relacionado à existência de elementos supérfluos na transformação (ex. variáveis nunca utilizadas), uma transformação concisa é livre de elementos supérfulos.

O experimento está estruturado em quatro etapas, *escopo*, *planejamento*, *operação*, e *análise e interpretaçao*, detalhadas nos subtópicos a seguir.

7.2.1 Escopo do Experimento

Na etapa de escopo é definido o objetivo do experimento e suas respectivas questões de pesquisa. Para apoiar essa definição utilizamos a técnica de *Goal-Question-Metric (GQM)* proposta por (CALDIERA; ROMBACH, 1994).

[4]Avaliação do código do software que representa a transformação.

7.2.1.1 Objetivo do Experimento Neste experimento buscamos avaliar a qualidade do código das transformações de modelo desenvolvidas com o framework MDTD em relação às transformações desenvolvidas em código. Esse objetivo está sumarizado na Figura 7.14 de acordo com o *template GQM* (CALDIERA; ROMBACH, 1994).

Analisar o código de transformações de modelo
Com o propósito de avaliar a qualidade da transformação construída
Com respeito à *Understandability, Modifiability, Reusability, Modularity Completeness, Consistency, Conciseness*
Do ponto de vista do método de desenvolvimento utilizado
No contexto de estudantes de graduação e pós-graduação e especialistas DDM desenvolvendo uma transformação

Figura 7.14 Objetivo do experimento de avaliação da qualidade da transformação

7.2.1.2 Questão de Pesquisa Com base no objetivo definido anteriormente, a seguinte questão de pesquisa foi estabelecida: **O uso do framework MDTD influencia na qualidade do código da transformação em relação às transformações desenvolvidas diretamente em código?** Considerando que geradores de código tendem a gerar códigos mais difíceis de serem manipulados diretamente pelo desenvolvedor, queremos avaliar se há diferença na qualidade do código que está sendo gerado pelo framework em relação à qualidade do código produzido diretamente pelos programadores.

7.2.1.3 Métrica Os atributos de qualidade avaliados são os propostos por (AMSTEL; MARCEL; BRAND, 2010), *Understandability*, *Modifiability*, *Reusability*, *Modularity*, *Completeness*, *Consistency*, *Conciseness*. Assim como no trabalho do autor, os atributos foram avaliados com base em um questionário de avaliação, o mesmo utilizado no trabalho de Amstel, composto por 21 questões. A avaliação é realizada atribuindo-se uma nota de 1 a 5, onde 1 representa "muito baixo" e 5 "muito alto" a cada questão. Cada atributo é medido por pelo menos três questões. Assim, a avaliação de cada atributo consiste na media das questões associadas a ele. O questionátio está disponível no apêndice I. A seguir é apresentado o cálculo realizado para encontrar a média de cada atributo.

$$Media_Understandability = (Q1 + Q7 + Q21)/3 \tag{7.2}$$

$$Media_Modifiability = (Q6 + Q11 + Q14 + Q18)/4 \tag{7.3}$$

$$Media_Reusability = (Q5 + Q10 + Q19)/3 \tag{7.4}$$

$$Media_Modularity = (Q5 + Q6 + Q13)/3 \tag{7.5}$$

$$edia_Completeness = (Q4 + Q12 + Q15 + Q16)/4 \tag{7.6}$$

$$Media_Consistency = (Q2 + Q20 + Q9)/3 \tag{7.7}$$

$$Media_Conciseness = (Q3 + Q8 + Q17)/3 \tag{7.8}$$

Onde *Qn*, corresponde à questão n do questionário.

Embora o objetivo do estudo não seja a avaliação da *performance* de desenvolvimento, o tempo gasto para executar o experimento foi registrado para todos os participantes.

7.2.2 Planejamento do Experimento

Nesta etapa o experimento é planejado em termos de seleção do contexto, formulação das hipóteses, definição das variáveis dependentes e independentes, tipo do experimento e instrumentação.

7.2.2.1 Definição do Contexto O experimento foi conduzido, assim como no estudo de caso, em ambiente acadêmico formado em sua maioria por alunos de graduação e pós-graduação em cursos relacionados à ciência da computação além de alguns especialistas em DDM. Diferentemente do estudo de caso, neste experimento além de conhecimento em UML e DDM também foi exigido dos participantes conhecimento em transformação de modelos. Para garantir que os alunos tinham este conhecimento, foram selecionados somente alunos que participam e/ou já participaram de projeto de iniciação científica na área de DDM ou alunos que fizeram seu trabalho de conclusão de curso (TCC) na área de DDM. Uma vez feita uma pré-seleção dos participantes foi aplicado um questionário (apêndice B) para atestar esse conhecimento. Neste questionário constam inclusive perguntas que avaliam o conhecimento do participante em codificar transformações.

7.2.2.2 Tipo do Experimento O tipo de *design* utilizado neste experimento compreendeu um fator, a *abordagem de desenvolvimento*, e dois tratamentos, a *abordagem MDTD* e a *abordagem ad-hoc*. Desta forma o experimento foi realizado em dois grupos organizados da seguinte maneira: *Grupo MDTD*, para desenvolvimento da transformação usando a abordagem MDTD; e *Grupo Controlado*, para desenvolvimento da transformação de maneira *ad-hoc*, direto em código.

7.2.2.3 Formulação das Hipóteses Para conduzir a avaliação foram formuladas hipóteses correspondentes a cada atributo avaliado. Desta forma sete grupos de hipóteses nulas e alternativas foram definidos. A hipótese nula indica que a média do atributo, para ambos os grupos analisados é a mesma. A hipótese alternativa indica que as médias alcançadas em cada grupo são diferentes. Adicionalmente, a hipótese alternativa foi dividida em duas outras sub hipóteses, onde a primeira indica que a média do *Grupo MDTD* é superior que a média do *Grupo Controlado* e a segunda indica que a média do *Grupo MDTD* é inferior a média do *Grupo Controlado*.

- Atributo *Understandability*
  - $H1_0$: O uso do framework *não influencia* no esforço requerido para entender o propósito da transformação.
    - $Media\_Understandability_{MDTD} = Media\_Undertandability_{Controlado}$
  - $H1_{a1}$: O uso do framework *influencia* no esforço requerido para entender o propósito da transformação.
    - $Media\_Understandability_{MDTD} <> Media\_Undertandability_{Controlado}$
    - $H1_{a1}$ $Media\_Understandability_{MDTD} > Media\_Undertandability_{Controlado}$
    - $H1_{a2}$ $Media\_Understandability_{MDTD} < Media\_Undertandability_{Controlado}$
- Atributo *Consistency*
  - $H2_0$: O uso do framework *não influencia* a padronização do código da transformação.
    - $Media\_Consistency_{MDTD} = Media\_Consistency_{Controlado}$
  - $H2_a$: O uso do framework *influencia* a padronização do código da transformação.
    - $Media\_Consistency_{MDTD} <> Media\_Consistency_{Controlado}$
    - $H2_{a1}$: $Media\_Consistency_{MDTD} > Media\_Consistency_{Controlado}$
    - $H2_{a2}$: $Media\_Consistency_{MDTD} < Media\_Consistency_{Controlado}$
- Atributo *Conciseness*
  - $H3_0$: O uso do framework *não influencia* no nível de concisão do código da transformação.
    - $Media\_Conciseness_{MDTD} = Media\_Conciseness_{Controlado}$
  - $H3_a$: O uso do framework *influencia* no nível de concisão do código da transformação.
    - $Media\_Conciseness_{MDTD} <> Media\_Conciseness_{Controlado}$
    - $H3_{a1}$: $Media\_Conciseness_{MDTD} > Media\_Conciseness_{Controlado}$
    - $H3_{a2}$: $Media\_Conciseness_{MDTD} < Media\_Conciseness_{Controlado}$
- Atributo *Completeness*
  - $H4_0$: O uso do framework *não influencia* na construção de transformações completas.
    - $Media\_Completeness_{MDTD} = Media\_Completeness_{Controlado}$
  - $H4_a$: O uso do framework *influencia* na construção de transformações completas.
    - $Media\_Completeness_{MDTD} <> Media\_Completeness_{Controlado}$
    - $H4_{a1}$: $Media\_Completeness_{MDTD} > Media\_Completeness_{Controlado}$
    - $H4_{a2}$: $Media\_Completeness_{MDTD} < Media\_Completeness_{Controlado}$
- Atributo *Reusability*
  - $H5_0$: O uso do framework *não influencia* no reuso de transformações já existentes na construção de novas transformações.
    - $Media\_Reusability_{MDTD} = Media\_Reusability_{Controlado}$
  - $H5_a$: O uso do framework *influencia* no reuso de transformações já existentes na construção de novas transformações.
    - $Media\_Reusability_{MDTD} <> Media\_Reusability_{Controlado}$
    - $H5_{a1}$: $Media\_Reusability_{MDTD} > Media\_Reusability_{Controlado}$
    - $H5_{a2}$: $Media\_Reusability_{MDTD} < Media\_Reusability_{Controlado}$
- Atributo *Modifiability*
  - $H6_0$: O uso do framework *não influencia* no esforço requerido para modificar a transformação.
    - $Media\_Modifiability_{MDTD} = Media\_Modifiability_{Controlado}$
  - $H6_a$: O uso do framework *influencia* no esforço requerido para modificar a transformação.
    - $Media\_Modifiability_{MDTD} <> Media\_Modifiability_{Controlado}$
    - $H6_{a1}$: $Media\_Modifiability_{MDTD} > Media\_Modifiability_{Controlado}$
    - $H6_{a2}$: $Media\_Modifiability_{MDTD} < Media\_Modifiability_{Controlado}$
- Atributo *Modularity*
  - $H7_0$: O uso do framework *não influencia* no nível de modularidade da transformação.
    - $Media\_Modularity_{MDTD} = Media\_Modularity_{Controlado}$
  - $H7_a$: O uso do framework *influencia* no nível de modularidade da transformação.
    - $Media\_Modularity_{MDTD} <> Media\_Modularity_{Controlado}$
    - $H7_{a1}$: $Media\_Modularity_{MDTD} > Media\_Modularity_{Controlado}$
    - $H7_{a2}$: $Media\_Modularity_{MDTD} < Media\_Modularity_{Controlado}$

Figura 7.15 Hipóteses formuladas para o experimento

7.2.2.4 Definição das Variáveis Variáveis *independentes* são aquelas que podem ser controladas durante o experimento e variáveis *dependentes* são as variáveis observadas para que se possa medir o efeito do tratamento (WOHLIN et al., 2012). Neste experimento, a variável independente do experimento consistiu na abordagem de desenvolvimento que assumiu dois níveis, o desenvolvimento *ad-hoc* e desenvolvimento usando o framework. As variáveis dependentes foram os atributos de qualidade da transformação, *understandability, modifiability ,reusability, modularity, completeness, consistency e conciseness*, diretamente relacioanadas às medidas para teste das hipóteses.

7.2.2.5 Instrumentação O cenário utilizado como objeto de estudo foi o mesmo para ambos os grupos e consistiu no desenvolvimento de uma transformação de modelo, definida pelo pesquisador, cujo objetivo foi transformar um modelo conceitual de classes de um sistema em um modelo de projeto na arquitetura MVC (detalhada no apêndice D). Para a execução do experimento foram disponibilizados os documentos necessários (ex. apresentações, questionário, exemplos), para o *Grupo MDTD*, e a implementação dos metamodelos envolvidos na transformação para ambos os grupos.

O experimento foi conduzido em três tarefas: a *tarefa 1*, treinamento dos participantes; a *tarefa 2*, desenvolvimento da transformação; e a *tarefa 3*, avaliação da qualidade. O treinamento (*tarefa 1*) foi realizado apenas com os participantes do *Grupo MDTD* e consistiu em uma palestra para apresentação do framework e do ambiente Eclipse customizado com os *plug-ins* de modelagem e engenhos de transformação utilizados no experimento. O desenvolvimento da transformação (*tarefa 2*) foi realizado por ambos os grupos. Os participantes do *Grupo MDTD* desenvolveram a transformação usando o framework.

Os participantes do *Grupo Controlado* desenvolveram a transformação na linguagem ATL usando seus conhecimentos em desenvolvimento de sistemas e em desenvolvimento de transformações. A avaliação da qualidade (*tarefa 3*) foi realizada mediante questionário respondido por todos os participantes. Assim, após o experimento, o código das transformações construídas por ambos os grupos foi enviado aleatoriamente para ser analisado pelos participantes, não sendo permitido que um participante avaliasse o seu próprio código. Adicionalmente, para aumentar o número de avaliações, os códigos foram enviados também para alguns participantes do estudo de caso para que eles também fizessem a avaliação da qualidade.

7.2.3 Operação do Experimento

A etapa de operação compreende basicamente em três passos, preparação, execução e coleta dos dados.

7.2.3.1 Preparação A preparação envolveu selecionar os participantes de acordo com o perfil definido no contexto e notificá-los, além de preparar o material necessário para o experimento. A seleção dos participantes para o experimento foi uma tarefa árdua devido à dificuldade em encontrar pessoas com conhecimento em linguagem de transformação. Finalmente 10 pessoas foram selecionadas com o perfil desejado. A distribuição dos participantes entre os grupos foi inicialmente feita de maneira aleatória. Contudo, três

participantes não concordaram em desenvolver a transformação diretamente no código. Para garantir uma divisão balanceada dos grupos, foi necessário fazer uma troca de participantes entre os grupos.

O material preparado para o experimento envolveu documentos, artefatos de software e ferramentas. Os documentos produzidos foram os seguintes:

- Apresentação do framework para ser ministrada para o *Grupo MDTD*;
- Exemplo de transformação desenvolvida com o framework proposto, para ser entregue ao *Grupo MDTD* (*exemploClasse2Banco.doc*);
- Ambiente de modelagem customizado (Eclipse com plug-ins);
- Cenário do estudo;
- Questionário de avaliação da qualidade a ser respondido pelos participantes;
- Documentos para possibilitar o desenvolvimento da transformação definida como cenário (para ambos os gupos): O Metamodelo fonte (chamado *MMConceitual*) da transformação implementado em formato Ecore; O Metamodelo alvo (chamado *MMProjeto*) da transformação implementado em formato Ecore. Um modelo instância de *MMConceitual* e um model instância de *MMProjeto* para um sistema de vendas de uma loja. Esses dois modelos serão utilizados pelos participantes para identificar os requisitos da transformação.

7.2.3.2 Execução A execução do experimento foi realizada pelos dois grupos em momentos distintos. O *Grupo MDTD* realizou o experimento no laboratório da Universidade Federal da Bahia (UFBA) sob a supervisão do pesquisador. O *Grupo Controlado* realizou o experimento de maneira virtual, sem a supervisão do pesquisador. Neste grupo o material necessário (cenário e metamodelos) foi enviado por email e foi solicitado que a transformação fosse desenvolvida e o tempo gasto registrado. Após o desenvolvimento os participantes enviaram o código da transformação para o pesquisador, que a executou e testou.

Somente após o desenvolvimento de todas as transformações foi iniciada a avaliação da qualidade das transformações, pois foi necessário aguardar que os códigos estivessem prontos para serem avaliados. Assim, os códigos produzidos foram enviados para os avaliadores que analisaram o código e responderam ao questionário. O questionário foi respondido *on-line* por todos os avaliadores. Cada código foi avaliado por pelo menos três participantes.

7.2.3.3 Coleta dos Dados A Figura 7.16 mostra os dados coletados ao final do experimento.

	Grupo	Avaliações individuais																
		AV1	AV2	AV3	AV4	AV5	AV6	AV7	AV8	AV9	AV10	AV11	AV12	AV13	AV14	AV15	AV16	AV17	AV18
Understandability	Grupo Controlado	3,33	3,67	3,33	2,67	2,67	2,67	2,67	3,0	3,33	2,67	2,33	3,33	2,67	2,33	3,00	3,00	4,00	3,33
	Grupo MDTD	2,00	3,00	2,67	3,33	2,67	2,33	1,67	2,67	3,33	4,00	3,67	3,00	3,00	2,33	3,00	3,00	2,67	2,67
Consistency	Grupo Controlado	3,67	3,33	3,00	2,00	2,67	3,00	3,67	4,33	3,33	3,33	3,67	3,00	3,00	3,67	3,00	3,67	2,67	3.00
	Grupo MDTD	3,33	3,00	3,67	3,33	3,00	3,67	3,67	3,33	4,00	4,00	3,67	3,00	3,67	3,33	3,00	4,00	3,00	3,67
Conciseness	Grupo Controlado	4,00	3,00	3,67	2,00	2,67	3,00	2,33	3,00	1,67	3,00	3,67	3,33	2,33	2,00	3,67	2,67	3,33	2,33
	Grupo MDTD	3,33	4,00	2,67	3,33	2,33	4,33	2,33	2,67	4,00	3,33	3,00	3,67	3,33	2,33	2,67	2,67	3,33	2,33
Completeness	Grupo Controlado	4,75	2,75	3,50	2,50	4,25	4,00	4,50	4,50	3,50	4,00	5,00	3,00	4,50	4,25	4,00	3,50	3,25	3,75
	Grupo MDTD	3,50	2,00	4,25	4,50	3,25	4,50	5,00	4,00	3,25	4,50	4,25	4,00	3,25	4,50	3,50	4,00	3,25	1,15
Reusability	Grupo Controlado	4,33	3,00	3,33	2,00	1,67	1,67	3,67	4,00	3,00	3,33	3,33	4,00	2,67	2,67	3,33	2,33	1,33	2,67
	Grupo MDTD	3,33	3,67	2,67	3,67	4,00	4,33	2,33	3,33	3,00	3,33	3,00	1,67	4,00	2,67	3,00	4,00	3,67	4,67
Modifiability	Grupo Controlado	2,50	2,75	3,75	3,25	3,50	3,75	2,50	3,25	3,25	3,25	3,00	3,50	2,75	2,75	3,25	3,00	3,50	3,75
	Grupo MDTD	2,50	3,00	3,00	2,25	2,25	2,00	3,25	2,50	3,25	2,50	3,50	4,00	2,75	2,75	3,50	2,50	3,00	2,75
Modularity	Grupo Controlado	2,33	2,67	3,33	3,67	1,67	2,00	1,33	3,33	3,00	1,67	1,67	3,00	1,67	2,00	3,00	2,33	3,00	3,33
	Grupo MDTD	1,33	2,67	2,33	2,00	3,33	1,67	2,00	2,33	3,00	2,67	2,67	2,67	2,67	2,67	3,00	1,67	1,67	2,67

Figura 7.16 Dados coletados na avaliação dos atributos de qualidade

A primeira coluna indica o atributo avaliado. Para cada atributo dois conjuntos de dados foram coletados, correspondentes aos dois grupos participantes do estudo, o *Grupo Controlado* e o *Grupo MDTD*. As demais colunas representam as médias das questões respondidas por cada avaliador em cada transformação por ele avaliada para cada atributo. Por exemplo, para o atributo *Understandability* do *Grupo Controlado*, a coluna AV1 indica qual a nota atribuída pelo avaliador para aquele atributo, no exemplo a nota foi *3,33*. Note que de acordo com a métrica estabelecida esta nota é um média de três questões do questionário que avaliam o mesmo atributo. É considerada uma avaliação (ex. AV1) um questionário respondido por um avaliador avaliando uma transformação específica. Foi coletado um total de 36 avaliações, 18 de cada grupo.

7.2.3.4 Validação dos Dados A validação é realizada após a execução do experimento para avaliar se existe alguma inconsistência nos dados coletados. Para o atributo *Understandability* foi identificado no *Grupo MDTD* quatro valores discrepantes. Esses valores foram descartados, pois se constatou que os avaliadores não tinham entendimento correto do atributo. Nenhuma outra inconsistência foi encontrada, portanto, todas as

demais respostas foram utilizadas na análise dos dados.

7.2.4 Análise e Interpretação dos Dados

Esta etapa é responsável pela análise dos dados coletados para que se possa tirar conclusões sobre o estudo. A Figura 7.17 apresenta as medidas descritivas calculadas com base nos dados coletados. Para cada atributo e grupo avaliado é apresentado o *mínimo valor observado* (*Min*), o *primeiro quartil* (*Q1*), a *mediana*, a *média*, o *terceiro quartil* (*Q3*), o *máximo valor observado* (*Max*), o *desvio padrão* (*S*) e *coeficiente de variação* (*CV*).

Atributo	Grupo	Min	Q1	Mediana	Média	Q3	Max	S	CV(%)
Understandability	MDTD	2,33	2,67	2,83	2,83	3,00	3,33	0,31	11,04
	Controlado	2.33	2,67	3,00	3,00	3,33	4,00	0,46	15,22
Consistency	MDTD	3,00	3,08	3,50	3,46	3,67	4,00	0,36	10,54
	Controlado	2,00	3,00	3,16	3,22	3,67	4,33	0,52	16,25
Conciseness	MDTD	2,33	2,67	3,16	3,09	3,33	4,33	0,63	20,52
	Controlado	1,67	2,33	3,00	2,87	3,33	4,00	0,67	23,28
Completeness	MDTD	2,00	3,31	4,00	3,86	4,44	5,00	0,71	18,34
	Controlado	2,50	3,50	4,00	3,86	4,44	5,00	0,70	18,07
Reusability	MDTD	1,67	3,00	3,33	3,35	3,92	4,67	0,75	22,22
	Controlado	1,33	2,41	3,00	2,91	3,33	4,33	0,86	29,61
Modifiability	MDTD	2,00	2,50	2,75	2,85	3,19	4,00	0,51	18,10
	Controlado	2,50	2,81	3,25	3,18	3,50	3,75	0,41	12,87
Modularity	MDTD	1,33	2.00	2,67	2,39	2,67	3,33	0,55	23,07
	Controlado	1,33	1,75	2,50	2,50	3,00	3,67	0,73	29,34

Figura 7.17 Dados descritivos da avaliação dos atributos

Para o atributo *Understandability*, a média das avaliações de cada grupo indica que o código produzido pelo *Grupo MDTD* (média = 2,83) é mais difícil de ser entendido que o código produzido pelo *Grupo Controlado* (média = 3.0), correspondente à hipótese $H1_{a2}$ *(Grupo MDTD <Grupo Controlado)*. Contudo se analisarmos o coeficiente devariaçã(CV), uma medida de dispersão que considera tanto a média quanto o desvio padrão, observamos que as avaliações do *Grupo MDTD* (11,04%) tiveram uma variação menor do que as do *Grupo Controlado* (15,22%). Essa variação pode ser vista também no gráfico *boxplot* da Figura 7.18(A), pois tanto a amplitude do *box* quanto os valores mínimo e máximo do *Grupo Controlado* são maiores que os do *Grupo MDTD*. Essa discrepância pode ter influenciado para aumentar o valor da média do *Grupo Cotrolado*.

Para o atributo *Consistency*, analisando os dados da Figura 7.17 observamos que

na média, o código produzido pelo *Grupo MDTD* se mostrou mais consistente que o código produzido pelo *Grupo Controlado*, correspondente à hipótese $H2_{a1}$.*(Grupo MDTD >Grupo Controlado)* Essa observação se confirma pelo coeficiente de variação inferior do *Grupo MDTD* ilustrada também no gráfico da Figura 7.18(B). Além disso, observa-se no gráfico a partir da mediana (linha que divide o *box*) que em geral os valores das avaliações do *Grupo MDTD* foram superiores aos valores do *Grupo Controlado*. A uniformidade observada no *Grupo MDTD* pode ser o reflexo do código padronizado gerado pelo framework. Em contra partida, a disparidade dos valores observados no *Grupo Controlado* também pode ser reflexo da estratégia de programação utilizada por cada desenvolvedor.

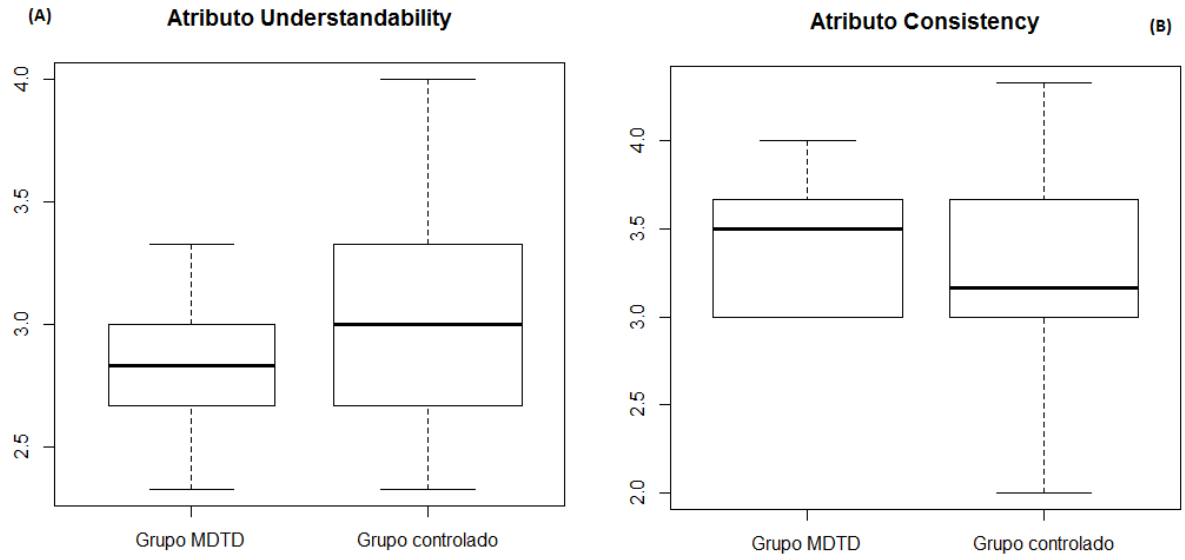


Figura 7.18 Comparação entre *Grupo MDTD* e o *Grupo Controlado* para os atributos *understandability*(A) e *Concistency*(B)

No atributo *Conciseness* observa-se que pela média o código gerado pelo *Grupo MDTD* (média = 3,09) foi avaliado como mais conciso que o código gerado pelo *Grupo Controlado* (média = 2,87), correspondente à hipótese $H3_{a1}$ *(Grupo MDTD >Grupo Controlado)*. Essa avaliação é reforçada pelo o coeficiente de variação e pelo gráfico boxplot da Figura 7.19(A) que mostram que as avaliações do *Grupo MDTD* também foram mais uniformes (observe que o *box* do *Grupo MDTD* é menor que o do *Grupo Controlado*) do que as avaliações do *Grupo Controlado*. Além disso, se compararmos a mediana dos dois grupos, observarmos que as avaliações individuais do *Grupo MDTD* também foram superiores que as do *Grupo Controlado* confirmando uma melhor concisão do código gerado pelo *Grupo MDTD*. Vale ressaltar que embora exista um valor discrepante na avaliação do *Grupo MDTD* (representado pelo ponto no gráfico), não foi observado nenhum motivo relevante para retirnarmos esse valor da amostra.

A análise dos dados coletados para o atributo *Completeness* evidenciou que não houve diferença entre os dois grupos (Figura 7.19B), correspondente à hipótese $H4_0$ *(Grupo MDTD = Grupo Controlado)*. Média, mediana e outros valores calculados para ambos

os grupos foram praticamente iguais indicando que eles produziram código com o mesmo nível de completude.

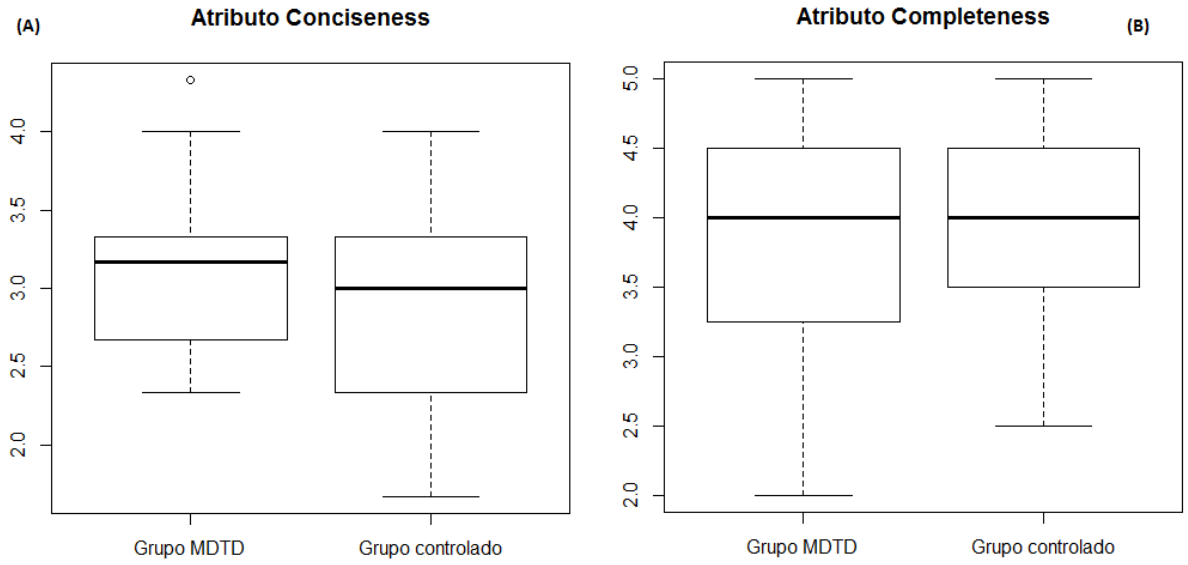


Figura 7.19 Comparação entre *Grupo MDTD* e o *Grupo Controlado* para os atributos *Conciseness*(A) e *Completeness*(B)

Para o atributo *Reusability*, conforme observado na Figura 7.17, o código construído pelo *Grupo MDTD* obteve maior capacidade de reuso do que o código construído pelo *Grupo Controlado*, correspondente à hipótese $H5_{a1}$ *(Grupo MDTD >Grupo Controlado)*. Essa informação se confirma pela maior uniformidade das avaliações para este grupo, observada tanto no coeficiente de variação quanto no gráfico apresentado na Figura 7.20(B). Outro dado interessante está na mediana (linha que divide o box da Figura 7.20(A)) que indica que 50% das avaliações realizadas no código do *Grupo MDTD* receberam nota acima de 3,3. Já no *Grupo Controlado* a maior parte das avaliações se encontra abaixo deste valor. Essa observação é considerada bastante pertinente no contexto de geradores de código.

Diferentemente de *Reusability*, no atributo *Modifiability* observa-se que em média o código produzido pelo *Grupo MDTD* (media = 2,85) é mais difícil de ser modificado do que o código produzido pelo *Grupo Controlado* (média = 3,18) conforme dados da Figura 7.17, correspondente à hipótese $H6_{a2}$ *(Grupo MDTD <Grupo Controlado)*. Essa diferença fica bem evidente no gráfico *boxplot* da Figura 7.20(B), pois além das avaliações do *Grupo Controlado* serem mais uniformes (ver variação do gráfico), em geral elas foram superiores às avaliações do *Grupo MDTD* (ver mediana no gráfico).

Finalmente, para o atributo *Modularity* na média as avaliações do *Grupo MDTD* se mostraram menos modularizadas que as do *Grupo Controlado* correspondente à hipótese $H7_{a2}$ *(Grupo MDTD <Grupo Controlado)*. Contudo há uma grande variação nas avaliações do *Grupo Controlado* (CV = 29,34) em relação ao *Grupo MDTD* (CV = 23,07) cujas avaliações são bem mais uniformes (ver Figura 7.21), o que levanta a possibilidade

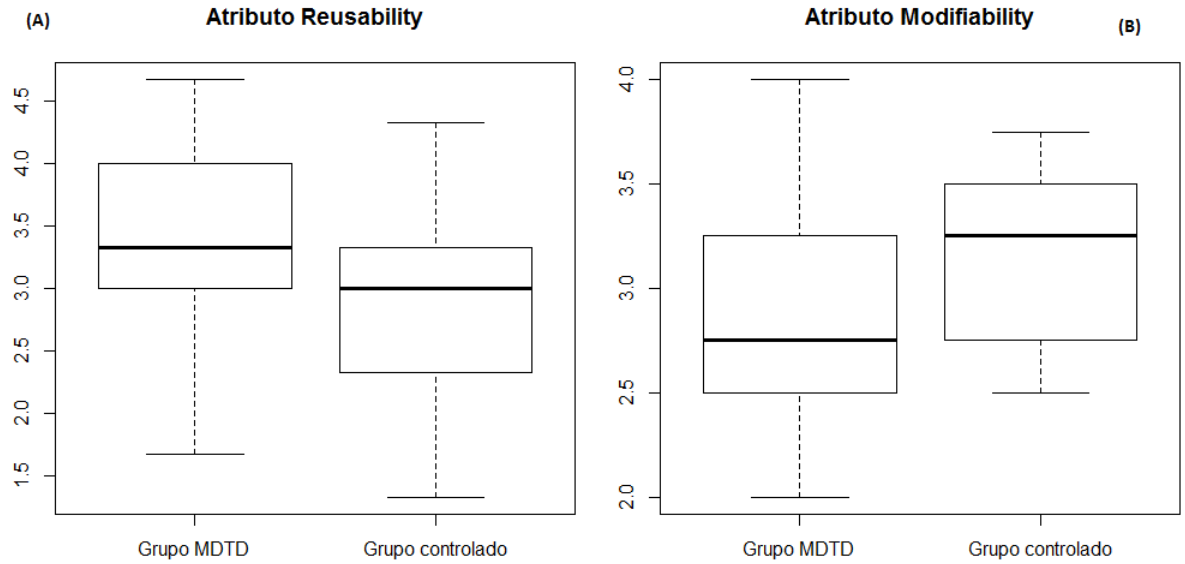


Figura 7.20 Comparação entre *Grupo MDTD* e o *Grupo Controlado* para os atributos *Reusability*(A) e *Modifiability(B)*

da média do *Grupo Controlado* ter sido superior devido à grande variação dos dados coletados.

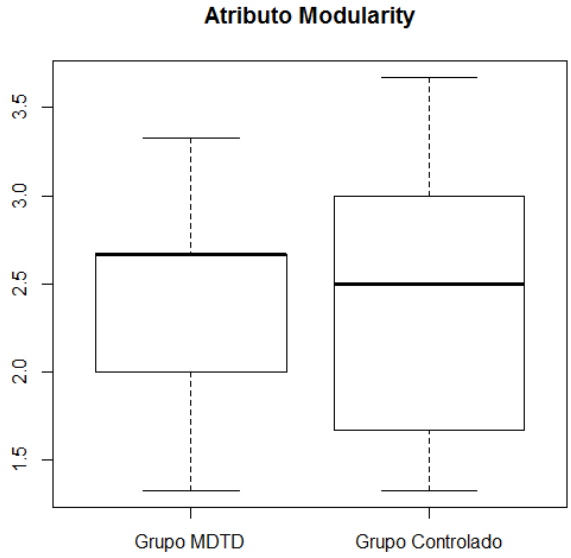


Figura 7.21 Comparação entre *Grupo MDTD* e o *Grupo Controlado* para o atributo *Modularity*

Em suma, pode-se observar que não houve grandes variações na qualidade do código produzido por ambos os grupos e que os resultados refletiram o que se espera para uma abordagem de geração de código. Por exemplo, percebe-se uma melhor avaliação do *Grupo MDTD* para atributos como *Reusability* e *Modularity* que realmente são diferenciais de

abordagens de geração de código. Em contrapartida percebe-se um menor aproveitamento do *Grupo MDTD* em atributos como *Understandability* e *Modifiability*, uma vez que geradores de código produzem códigos mais genéricos e complexos.

7.2.4.1 Avaliação das Hipóteses Para verificar se existe diferença significante na qualidade do código produzido pelos dois grupos relacionado aos atributos considerados, executamos o teste de hipóteses (WOHLIN et al., 2012) a partir do teste *Student's t-test*. A Figura 7.22 apresenta para cada atributo avaliado qual a probabilidade sobre a hipótese nula (*p-value*).

Para aceitar ou rejeitar as hipóteses foi utilizado o *p-value* calculado para cada atributo considerando o nível de significância *p=0,05*. Portanto, se *p-value <0.05* o resultado é significativo e a hipótese nula pode ser rejeitada.

O resultado do teste mostrou que, com exceção do atributo *Modifiability* cuja hipótese nula foi rejeitada de acordo com o *p-value*, todos os outros atributos avaliados tiveram *p-value >0.05*. Isso mostra que a diferença entre os grupos não é significativa para os atributos de qualidade medidos. Mesmo que os dados descritivos apresentados aterior-mente (na Figura 7.17 e sob a forma de gráficos *boxplot*) tenham evidenciado algum tipo de melhoria ou perda de qualidade para os atributo, não existem evidências suficientes para afirmar que o framework influencia na qualidade do código das transformações construídas. Contudo, é importante colocar que um aumento da amostra ou do nível de significância pode levar a outras conclusões sobre as hipóteses formuladas.

	p-value
Understandability	0,3379
Consistency	0,1203
Conciseness	0,3159
Completeness	1
Reusability	0,1066
Modifiability	0,03929
Modularity	0,6146

Figura 7.22 Dados coletados na avaliação dos atributos de qualidade

É evidente que a qualidade do produto é uma meta para qualquer processo de desenvolvimento. Portanto, a avaliação da qualidade foi importante para complementar o estudo de caso que já tinha sido realizado, pois assim foi possível mostrar que o framework conduz o desenvolvimento de transformações de modelos sem influenciar significativamente

na qualidade do código gerado. Desta forma, mesmo pessoas sem conhecimento em linguagem de transformação, perfil dominante no estudo de caso mostrado anteriormente, puderam construir transformações e com qualidade semelhante à de programadores experientes. Em relação ao atributo *Modifiability* cuja hipótese nula foi refutada indicando que o código produzido manualmente é mais facilmente alterado do que o código gerado pela nossa abordagem é importante considerar que nesta tese propomos construir transformações usando DDM e que, portanto, modificações não seriam realizadas diretamente no código, mas nos modelos da transformação.

Embora o objetivo deste experimento não tenha sido avaliar a *performance* do desenvolvimento, o tempo gasto na execução do experimento (em minutos) foi também coletado para ambos os grupos.

A Figura 7.23 mostra uma comparação entre o tempo médio de desenvolvimento da transformação pelo *Grupo MDTD* e pelo *Grupo controlado*.

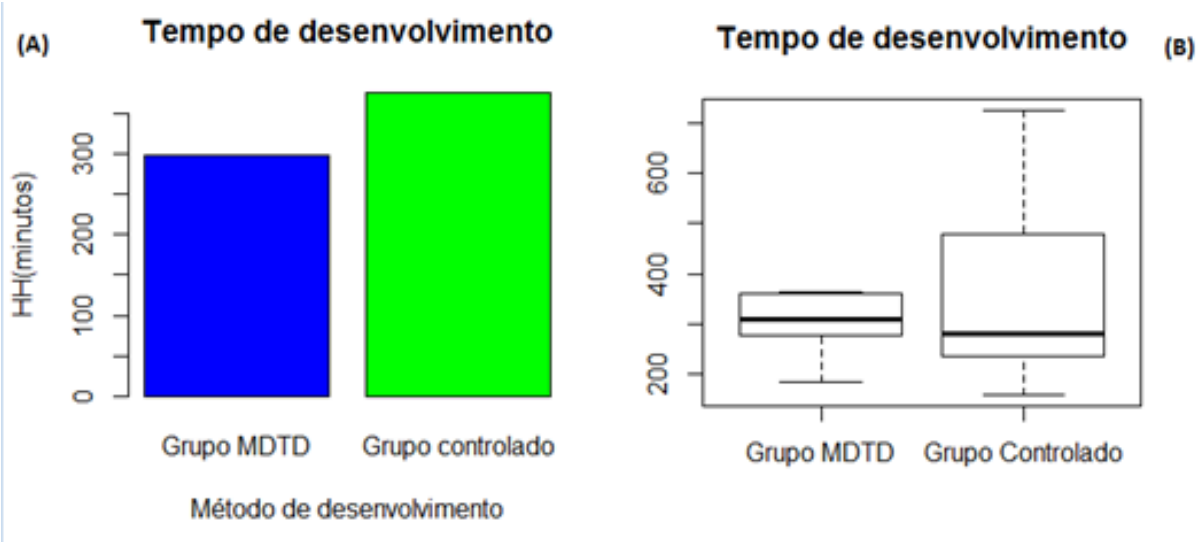


Figura 7.23 Comparação entre o tempo gasto pelo *Grupo MDTD* e pelo *Grupo Controlado*

Pode-se observar que o *Grupo MDTD* teve uma performance ligeiramente melhor (Figura 7.23A) e mais uniforme (Figura 7.23B) do que o Grupo controlado. Esse é um dado importante, pois o processo compreende um conjunto de atividades que, em geral, não é realizado quando a transformação é construída diretamente no código, como, por exemplo, a definição de requisitos, casos de teste e identificação de riscos, mas que consomem tempo do desenvolvimento.

7.2.4.2 Validade da Análise Esta seção apresenta as ameaças à validação interna, externa, de construção e de conclusão do experimento.

Validade interna. Ameaças à validação interna estão relacionadas a fatores não controlados que podem influenciar os efeitos dos tratamentos nas variáveis.

Para diminuir as ameaças quanto ao nível de *expertise* dos participantes de cada grupo, foi requerido que todos os participantes tivessem conhecimento em alguma linguagem específica para transformação de modelos. Para garantir esse conhecimento um questionário prévio foi aplicado com questões que incluíam trechos de código em linguagens específicas. Além disso, a distribuição dos participantes entre os grupos foi realizada, na medida do possível, de maneira randômica.

Para evitar desvios no desenvolvimento, somente após o término da construção da transformação os participantes foram informados que seria avaliada a qualidade da transformação e quais os atributos que seriam avaliados. Neste momento então as transformações construídas por cada participante foram distribuídas de forma aleatória e anônima entre os avaliadores, ou seja, o avaliador recebeu um código para avaliar, mas não teve conhecimento de quem foi o autor do código. Cada transformação foi avaliada por pelo menos três avaliadores. Além disso, nenhum participante avaliou o seu próprio código.

Os participantes do *Grupo MDTD* foram acompanhados pelo pesquisador e a interação entre eles não foi permitida. Os participantes do *Grupo Controlado* não foram acompanhados pelo pesquisador. Para prevenir a comunicação entre eles, não foi divulgado o nome de quem estava participando do experimento. Além disso, os participates não se conheciam previamente.

Validade externa. Ameaças à validade externa estão relacionadas à possibilidade de generalização dos resultados para outros ambientes.

Para minimizar o risco de representatividade da amostra o código das transformações produzidas por ambos os grupos foi enviado para ser analisado também por pessoas que não participaram do experimento. Assim, em vez de termos 10 avaliações tivemos 36 avaliações. Cada avaliador externo avaliou duas transformações, uma de cada grupo, distribuídas de forma aleatória.

A representatividade do domínio escolhido bem como o tamanho e a complexidade da transformação desenvolvida são outras ameaças ao experimento. Como não havia possibilidade de usar um caso de transformação real para ser desenvolvido, foi escolhida uma transformação bastante utilizada no desenvolvimento de software atual que é a transformação de um modelo conceitual de classes para a arquitetura MVC. Contudo, experimentos com casos reais são necessários para validar os resultados do estudo.

O conhecimento dos avaliadores no domínio da transformação também pode influenciar a validade do experimento. Garantimos então que os avaliadores foram também participantes do estudo de caso ou do experimento, portanto tinham conhecimento sobre o domínio da transformação. Adicionalmente, o conhecimento dos avaliadores em relação aos atributos de qualidade avaliados também foi considerado importante, pois interpretações erradas sobre esses atributos poderiam levar a distorções na avaliação. Assim, incluímos no questionário de avaliação uma explicação sobre cada um dos atributos que seriam avaliados.

Validade de construção. Ameaças à validade de construção estão relacionadas ao

projeto do experimento.

A transformação desenvolvida neste experimento foi a mesma utilizada no estudo de caso. Essa estratégia foi utilizada para diminuir as ameaças em relação à construção de todo o material utilizado, pois o estudo de caso anterior foi conduzido em diversas replicações e, portanto, o material utilizado (descrição do cenário, ambiente, metamodelos) foi considerado suficientemente validado.

Em relação à medição da qualidade, sabe-se que é possível medir a qualidade sob siversos aspectos e em geral essa é uma característica difícil de medida. Para diminuir os riscos desta medição foi utilizado o mesmo questionário já aplicado por (AMSTEL; MARCEL; BRAND, 2010) na coleta de dados sobre qualidade de transformação. Este questionário busca diminuir a subjetividade das perguntas utilizando estratégias diferentes para colher o mesmo dado, ou seja, para cada dado colhido foi realizado pelo menos três perguntas diferentes distribuídas de forma aleatória ao longo do questionário. Com isso, buscou-se também minimizar a influência do cansaço do participante durante a avaliação da transformação. Como consequência, o resultado da avaliação de cada participante consistiu na média das questões relacionadas a cada atributo. Adicionalmente, antes do experimento não foi informado aos participantes que seria avaliada a qualidade das transformações construídas. Distorções no entendimento dos atributos de qualidade avaliados foram também minimizadas através da inserção no questionário de uma explicação sobre o propósito de cada atributo.

Validade de conclusão. A validade de conclusão está realacionada as questões que influenciam nas conclusões obtidas com o experimento.

Uma questão que pode influenciar na conclusão é o método estatístico utilizado na análise dos dados. Utilizamos um método considerado robusto para avaliação de hipóteses quando temos um fator e dois tratamentos. Além disso, a análise foi discutida com um especialista que ajudou a escolher o teste mais apropriado para o experimento.

Outro ponto que pode influenciar o resultado é julgamento dos atributos através a avaliação de humanos. Para minimizar o risco de respostas inconsistentes, utilizando um questionário que contém pelo menos três perguntas para avaliar cada atributo de qualidade e utiliza como resposta a média dessas perguntas. As avalições discrepantes foram também desconsiderados, conforme indicado na Seção 7.2.3.4. Adicionalmente, foi utilizada como referência uma medida de qualidade já testada em outros trabalhos acadêmicos.

7.3 AVALIAÇÃO DA LINGUAGEM DE MODELAGEM

A especificação da linguagem de modelagem proposta neste trabalho utilizou como base os metamodelos das linguagens de transformação ATL e QVT. Esses metamodelos foram analisados e seus conceitos foram abstraídos em construçoes independentes de plataforma, seguindo o objetivo de definir uma linguagem de modelagem, declarativa, para especificação de transformações de modelos unidirecionais. A linguagem ATL foi se-

lecionada por ser a linguagem mais utilizada pela comunidade de desenvolvimento de transformações e a linguagem QVT por ser o padrão OMG para desenvolvimento de transformações. Partimos da premissa de que ao utilizar os metamodelos dessas linguagens como base para definir os construtores da nossa linguagem de modelagem já alcançamos um nível de cobertura satisfatório dos conceitos necessários para especificar transformações de modelos. Ainda assim, achamos pertinente realizar uma avaliação tanto do metamodelo MMT quanto do perfil MTP proposto nesta tese.

Nesta direção, inicialmente avaliamos o nível de cobertura dos construtores do metamodelo MMT em relação à teoria de transformação de modelos. Para isso, comparamos os construtores definidos para o MMT com a taxonomia de transformações definida em (MENS; CZARNECKI; GORP, 2006), ilustrada na Figura 7.24. A coluna *Taxonomia* apresenta os elementos utilizados para a comparação, classificados em conceitos básicos e tipos de transformação. A taxonomia contém outros tipos de classificações que não foram utilizados nesta comparação, pois não foram considerados relevantes para a validação do metamodelo e do perfil, a exemplo de características do ambiente de desenvolvimento. A coluna *Metamodelo MMT* apresenta os construtores do MMT relacionados a cada um dos elementos citados na taxonomia. Observa-se que dentre os conceitos propostos na taxonomia, o MMT não contempla apenas a especificação de transformações de programas, referente à construção de transformações modelo-a-texto.

	Taxonomia	**Metamodelo *MMT***
Conceitos básicos	Definição de transformação	TransformationSpecification (*MMTspec*) Transformation (*MMThighdesign*)
	Regras de transformação	Relation (*MMThighdesign*) Rule (*MMTlowdesign*)
	Modelo fonte	InModel
	Modelo alvo	OutModel
	Linguagem fonte	SourceMM
	Linguagem alvo	TargetMM
	Construtores das linguagens fonte e alvo	ModelElement
Tipos de transformação	Transformações de modelos	M2MTransformation
	Transformações de programa	Não contempla
	Transformações endógenas	SourceMM igual ao TargetMM
	Transformações exógenas	SourceMM diferente do TargetMM
	Transformações horizontais	Contempla
	Transformações verticais	Contempla
	Mesmo espaço tecnológico	Contempla
	Espaço tecnológico diferente	Contempla

Figura 7.24 Comparação MMT versus Taxonomia de (MENS; CZARNECKI; GORP, 2006)

A segunda etapa da avaliação consistiu em um estudo exploratório realizado através da engenharia reversa de transformações já existentes.

O objetivo geral deste estudo foi avaliar a expressividade do metamodelo MMT na especificação de transformações de modelos visando detectar a falta ou necessidade de melhoria de construtores e avaliar a expressividade dos diagramas UML selecionados para o perfil MTP na especificação de transformações. A expressividade dos construtores do

metamodelo MMT foi avaliada em relação aos construtores das linguagens ATL e QVT, utilizadas como base para definir o MMT.

Para conduzir este estudo as seguintes questões de pesquisa (QP) foram definidas:

- **QP1: Os construtores do MMT são suficientes para especificar transformações de modelos escritas em ATL e QVT?**
- **QP2: É necessário adicionar novos construtores no MMT para possibilitar a especificação de transformações escritas em ATL/QVT?**
- **QP3: Os diagramas UML selecionados são suficientes para as necessidades de especificação de transformações?**

A avaliação foi realizada através da engenharia reversa de transformações já existentes escritas nas linguagens ATL e QVT. Para cada transformação selecionada foram construídos os modelos correspondentes no perfil MTP e em seguida preenchido um formulário de coleta de dados, conforme modelo ilustrado na Figura 7.25. O formulário está dividido em duas partes: *Identificação*, para registrar qual a transformação modelada, seu objetivo, em que linguagem ela está desenvolvida, quantas linhas de código possui e quais os artefatos disponíveis para a especificação; e *Resultados*, preenchido após a especificação da transformação identificando quais os construtores do metamodelo MMT que foram usados e quais os construtores que precisariam ser inseridos e ou alterados (necessidades de novos construtores).

Parte1: Identificação			
Nome da transformação: Objetivo:			
Artefatos encontrados:			
Linguagem:		LOC:	
Parte2: Resultado			
Pergunta			**Resposta**
1) Existe necessidade de incluir novos construtores? (sim/não)			
2) Existe necessidade de incluir novas associações? (sim/não) 3) Existe necessidade de incluir novos atributos? (sim/não)			
4) Existe necessidade de alterar os construtores existentes? (sim/não)			
5) Existe necessidade de alterar a multiplicidade das associações? (sim/não)			
Parte3: Detalhamento das respostas			
Descrição da necessidade		Inclusão	Alteração

Figura 7.25 Questionário de avaliação do MMT

O estudo foi realizado no período de maio a setembro de 2014. No primeiro momento foram modeladas as transformaçóes do processo MDA para serviços específicos de middleware proposto em (MACIEL; SILVA; MASCARENHAS, 2006). Esta cadeia de transformações havia sido desenvolvida diretamente em código ATL por integrantes do

grupo de pesquisa ligado a esta tese. Ela foi selecionada para ser usada neste estudo por já ter sido utilizada em vários projetos e por ser uma transformação conhecida dos pesquisadores, diminuindo o risco de conhecimento do domínio da transformação. Em seguida o estudo foi realizado com transformações selecionadas ao caso em repositórios públicos de transformações. Um total de sete transformações diferentes foram especificadas neste segundo momento.

Ajustes no metamodelo e no perfil foram feitos de maneira incremental ao longo do estudo após a especificação de cada transformação de acordo com as necessidades identificadas. Quando foi observado que não havia mais necessidade de incluir e ou adicionar nenhum construtor ao metamodelo e que os diagramas selecionados da UML eram suficientes para a modelagem, o processo de validação foi finalizado.

Esta avaliação embora não esteja apoiada em técnicas de experimentação foi representativa para considerarmos o perfil apto para ser utilizado nas demais avaliações apresentadas neste capítulo.

7.4 CONSIDERAÇÕES SOBRE O CAPÍTULO

Neste capítulo apresentamos a validação da abordagem proposta, realizada a partir de estudo de caso e experimento controlado.

No estudo de caso, participantes com diferentes níveis de conhecimento em DDM, inclusive sem conhecimento em transformações, foram capazes de desenvolver código executável testado. Evidenciamos com isso a eficácia do processo na condução do desenvolvimento. No experimento, buscamos comparar a qualidade do código gerado pelo framework à qualidade do código construído diretamente pelo programador. Utilizamos participantes com conhecimento em desenvolvimento de transformação, mas sem experiência na indústria. Mostramos que os códigos produzidos não apresentaram diferença significativa de qualidade.

O framework, portanto, conduziu o desenvolvimento da transformação por pessoas de diferentes perfis, que conseguiram gerar código similar ao produzido manualmente. Com esses resultados acreditamos que a abordagem tem potencial para apoiar o desenvolvimento de transformações de modelos reduzindo a sua complexidade.

Capítulo

8

TRABALHOS RELACIONADOS AO DESENVOLVIMENTO DE TRANSFORMAÇÕES DE MODELOS

Este capítulo apresenta a pesquisa bibliográfica realizada para identificar o estado da arte em desenvolvimento de transformações de modelos. Inicialmente são apresentados os trabalhos que propõem estratégias para sistematização do desenvolvimento de transformações, relacionados ao processo proposto nesta tese. Em seguida são apresentadas as propostas de linguagens para especificação de modelos de transformação de modelo.

8.1 PROPOSTAS PARA SISTEMATIZAÇÃO DO DESENVOLVIMENTO

Para mapear o estado da arte em direção à sistematização do desenvolvimento de transformações foi realizada uma pesquisa bibliográfica, com base nas orientações de (KITCHENHAM, 2004) para revisão sistemática, organizada em três etapas: *Planejamento*, onde foi definida a questão de pesquisa e o protocolo a ser utilizado para conduzir a pesquisa; *Condução da pesquisa*, onde foi executada a pesquisa de acordo com o protocolo definido; e *Sumarização*, onde são apresentados os resultados da pesquisa.

A questão de pesquisa (QP) que norteia este estudo busca identificar as estratégias propostas na literatura para conduzir o desenvolvimento de transformações de modelos e está formulada da seguinte maneira:

QP: Quais as estratégias utilizadas para conduzir o desenvolvimento de transformações de modelos?

Com base na questão de pesquisa foi definido o protocolo do estudo que compreendeu: a definição das bases de dados utilizadas na pesquisa; a definição da *String* de busca para filtrar os trabalhos; os critérios de inclusão e exclusão dos artigos encontrados; e o procedimento de extração de artigos relevantes.

Os trabalhos foram analisados de acordo com características consideradas relevantes ao contexto do ciclo de vida de desenvolvimento de transformações. A Figura 8.1 apresenta as características que foram analisadas.

Nr	Característica analisada	Como a característica será avaliada
C1	Estratégia utilizada para conduzir o desenvolvimento	Indicar qual foi à estratégia usada
C2	Contempla a especificação de requisitos?	1 – não contempla 2 – contempla informalmente 3 – contempla e detalha o que deve ser especificado 4 – Não informa
C3	Contempla o projeto independente de plataforma	1 - Não contempla 2 – Contempla 3 – Não informa
C4	Define como mapear requisitos em projeto	1 – Não define 2 – Define, mas não automatiza 3 – Define e fornece suporte automatizado
C5	Contempla o projeto específico de plataforma	1 – Não contempla 2 – Contempla 3 – Não informa
C6	Define como mapear o projeto independente de plataforma em projeto específico de plataforma	1 – Não define 2 – Define, mas não automatiza 3 – Define e fornece suporte automatizado
C7	Contempla a fase de codificação	1 – Não contempla 2 – Contempla 3 – Não informa
C8	Contempla a geração de código em linguagem específica	1 – Não contempla 2 – Contempla 3 – Não informa
C9	Contempla atividades de validação e verificação	1 – Não contempla 2 – Contempla, mas não automatiza 3 – Contempla e automatiza 4 – Não informa
C10	É uma abordagem DDM	1 – Não 2 – Sim
C11	Adota / define uma linguagem de modelagem	1 – Não adota nenhuma linguagem 2 – Utiliza puramente a UML ou extensão leve 3 – Utiliza uma extensão pesada da UML 4 – Define uma notação própria 5 – Utiliza uma linguagem formal
C12	Depende de ferramenta proprietária (fornecida pela abordadagem) para ser executada	1 – Não 2 – Sim 3 – Não informa
C13	Validação da proposta	1 - Não 2 – Sim, exemplo 3 – Sim, estudo de caso/experimento

Figura 8.1 Características analisadas nos trabalhos relacionados à sistematização do desenvolvimento de transformações

A primeira e segunda coluna da figura apresentam respectivametne o número e a descrição da característica avaliada. A terceira coluna indica como a característica foi

avaliada em cada trabalho. Estas características foram definidas com base na revisão sistemática realizada no trabalho de (BOLLATI et al., 2013).

As bases de dados utilizadas para a realização da pesquisa foram *ACM Digital Library, IEEE Digital Library, Science Direct* e *Springer*, as quais foram submetidas à seguinte *String* de busca:

"(MDE or MDD or MDA or Model Driven Development) and (model transformation) and (specification or development or approach or strategy or framework or systematic or process or methodology or method or life cycle)"

Foram *incluídos* na pesquisa os trabalhos publicados em inglês, com propostas da indústria ou da academia e que datavam de 2003 a 2015. Foram *excluídos* da pesquisa: os trabalhos que não correspondiam a estratégias relacionadas ao desenvolvimento de transformações; os trabalhos que tratavam de transformação modelo-a-texto; os trabalhos sobre transformações não relacionais (ex. transformações de grafos); e propostas voltadas a domínios específicos (ex. propostas de transformações para aplicações web, software embarcado, etc.).

Os critérios de extração utilizados para selecionar os trabalhos foram título, resumo e introdução, nesta ordem. Após esta triagem os artigos selecionados foram lidos na íntegra. A Figura 8.2 apresenta um resumo da quantidade de trabalhos de acordo com os critérios de extração por base de dados: (i) trabalhos encontrados por base de dados após a aplicação da *String* de busca; (ii) selecionados após a triagem por título; (iii) selecionados após a triagem por resumo; (iv) selecionados após a triagem pela introdução; e (v) relevantes, que foram analisados. Dois trabalhos foram adicionados com o título de "outros", encontrados nas referências dos trabalhos pesquisados. Conforme observado na tabela, dos 21 trabalhos lidos na íntegra, 10 trabalhos foram considerados relevantes no contexto de estratégias para conduzir o desenvolvimento de transformações e estão detalhados a seguir. Quatro trabalhos não foram considerados, pois se tratavam de outras publicações relacionadas aos trabalhos já selecionados, e dois trabalhos tratavam puramente de linguagens de modelagem e estão descritos na Seção 8.2.

Base de pesquisa	Critério (quantidade de trabalhos)				
	Encontrados	Triagem por título	Triagem por resumo	Triagem por introdução	Trabalhos relevantes, que representam o estado da arte
ACM Digital Library	156	34	10	2	1
IEEE Digital Library	694	89	26	9	3
Science Direct	152	28	9	4	2
Springer	395	9	6	4	2
Outros	---	---	---	2	2
Total	1397	160	51	21	10

Figura 8.2 Resumo dos trabalhos encontrados

Os itens a seguir apresentam um resumo dos trabalhos considerados relevantes (coluna 6 da Figura 8.2) organizados em ordem cronológica. Na Seção 8.1.1 é realizada uma

análise dos trabalhos apresentados sob o ponto de vista das características definidas anteriormente na Figura 8.1. Existem trabalhos e ferramentas mais genéricas relacionadas à DDM, a exemplo de (ANDROMDA, 2016) e (SILVESTR; BASTARRICA; OCHOA, 2014), que não foram abordados nesta revisão porque não estão diretamente relacionados ao desenvolvimento de transformações de modelos.

- **A Systematic Approach to Design Model Transformation (KUSTER; RYNDUNA; HAUSER, 2005)**

  As primeiras ideias encontradas que tratam da sistematização das atividades de desenvolvimento de transformações foram propostas por (KUSTER; RYNDUNA; HAUSER, 2005) em uma abordagem sistemática que abrange as fases de análise de requisitos, projeto, implementação e teste de transformações. Nesta abordagem, o desenvolvimento se inicia com a identificação dos requisitos e metamodelos envolvidos na transformação. Em seguida o projeto da transformação é realizado em dois níveis de abstração, o projeto de alto nível, onde são documentadas as regras que deverão compor a transformação, e o projeto de baixo nível, onde essas regras são detalhadas em termos de mapeamento entre metamodelos. A fase de projeto se encerra com a verificação sintática da transformação, onde é recomentado o uso da técnica de inspeção de modelos (BRAUN; MARCHALL, 2003), e a verificação semântica da transformação, realizada através da definição de casos de testes que posteriormente avaliam os modelos de entrada e saída processados pela transformação. Finalmente, na fase de implementação o código é manualmente construído.

  O trabalho de (KUSTER; RYNDUNA; HAUSER, 2005) embora organize em fases as atividades que devem ser realizadas ao longo de desenvolvimento da transformação, está especificado em um nível de granularidade que não contempla a definição de elementos, tais como artefatos, papeis e guias, essenciais para facilitar o entendimento e a execução da abordagem. Desta forma, se concentra em definir *o que* fazer, mas não define *quem* nem *como* cada atividade deve ser executada. Em especial, na fase de análise de requisitos nenhuma estratégia é utilizada para apoiar o desenvolvedor na identificação dos requisitos e metamodelos, assume-se que essas atividades serão *de alguma forma* executadas pelo desenvolvedor antes de iniciar o projeto. A proposta também não contempla automação entre as fases nem geração de código.

- **Methodological Approach to Developing Model Transformations (VIGNAGA, 2007)**

  Seguindo a mesma linha de pesquisa desta tese, Vignaga apresenta uma proposta de doutorado para definir uma metodologia de desenvolvimento de transformação de modelos. A proposta visa definir uma metodologia iterativa e incremental, expressa na linguagem SPEM, para o desenvolvimento de transformações de modelos estruturada nas fases especificação de requisitos, análise, projeto, implementação, teste e gerenciamento. A especificação da metodologia deverá contemplar a definição de

papéis, artefatos, disciplinas, passos e guias de forma detalhada para guiar o desenvolvedor. Contudo não foi encontrado nenhuma publicação de trabalhos com os resultados desta proposta.

- **Transformation have to be Developed ReST Assured (SIIKARLA et al., 2008)**

  No trabalho apresentado em (SIIKARLA et al., 2008) os autores propõem um processo iterativo e incremental para desenvolvimento de transformações com foco em reduzir a complexidade e o custo de desenvolvimeto. O processo compreende as fases de especificação, projeto e implementação da transformação e para cada fase são identificados os artefatos que devem ser produzidos e quem é o responsável pela construção do artefato. O desenvolvimento se inicia com a identificação de correspondências entre modelos de entrada e saída da transformação. Essas correspondências são utilizadas para selecionar *padrões transformacionais*, soluções prontas para problemas recorrentes relacionados ao contexto de transformações, que poderão ser aplicados no projeto da transformação. Em seguida são definidos a estrutura e o comportamento da transformação para finalmente o código ser implementado em uma linguagem específica.

  Assim como os trabalhos apresentados anteriormente, este processo não está especificado em nenhuma linguagem de modelagem de processo (PML) o que pode dificultar seu entendimento e execução por parte dos desenvolvedores. Esse cenário se agrava pela inexistência de uma notação para documentar a especificação dos artefatos. Adicionalmente, o processo não oferece qualquer tipo de automação entre as fases definidas nem tampouco gera o código da transformação.

- **Towards the Efficient Development of Model Transformations Using Model Weaving and Matching Transformations (FABRO; VALDURIEZ, 2009)**

  (FABRO; VALDURIEZ, 2009) propõe a semi-automatização do desenvolvimento de transformações através da técnica de *weaving model*. Nesta proposta, o desevolvimento se inicia informando quais os metamodelos fonte e alvo que participam da transformação. Esses metamodelos são automaticamente analisados por uma sequencia de *matching transformations*, transformações que identificam mapeamentos entre metamodelos, e gera como resultado um *weaving model*, um modelo contendo os *links* entre os elementos dos metamodelos fonte e alvo da transformação. O desenvolvedor então verifica se há inconsistências nos *links*, identificados automaticamente pela ferramenta, e em seguida gera o modelo da transformação. A abordagem contém transformações que recebem o *weaving model* como entrada e produzem um modelo da transformação como saída. Estes modelos podem ser transformados em código ATL ou XSLT para serem executados.

  Diferentemente dos trabalhos encontrados anteriormente, esse trabalho introduz o uso de ferramentas automatizadas para conduzir o desenvolvimento da transformação e encapsula o processo de desenvolvimento na ferramenta. Desta forma, a

utilização da abordagem só é possível através da ferramenta proposta. A ferramenta é aplicada especificamente para as fases de projeto independente de plataforma e codificação da transformação em uma linguagem específica, não tratando das demais fases do ciclo de vida de desenvolvimento de software.

- **Method of Constructing Model Transformation Rule Based on Reusable Pattern (LI; YIN, 2010)**

  O trabalho (LI; YIN, 2010) propõe um framework para transformação de modelos baseado em padrões e utiliza um algoritmo para conduzir o desenvolvimento da transformação. São definidos sete diferentes padrões de transformação e nove estratégias para extração de regras com base nestes padrões. A construção das regras que compõem a transformação está baseada em um algoritmo que incorpora as estratégias propostas e guia o desenvolvedor em como utilizar os padrões.

  Este trabalho introduz o uso de algoritmos para conduzir o desenvolvimento de transformações. Trata-se de um trabalho com foco específico em reuso de padrões e que contempla somente transformações entre modelos independentes de plataforma e modelos específicos de plataforma. O trabalho menciona a utilização de uma linguagem de modelagem independente de plataforma e fácil de ser utilizada, mas não apresenta nenhum detalhe nem exemplo da linguagem citada. O algoritmo definido não está automatizado.

- **Model Transformation Specification for Automated Formal Verification (SANI; POLACK; PAIGE, 2011)**

  Este trabalho propõe o framework *Trans-DV* (*Transformation Specification Development and Verification framework*), uma abordagem sistemática para especificação de transformações com foco no uso de métodos formais. *Trans-DV* associa a especificação da transformação a um conjunto de *templates* que podem ser instanciados para automaticamente produzir a especificação formal da transformação e gerar assertivas para verificação de propriedades.

  O trabalho compreende as etapas de elicitação de requisitos, projeto e verificação formal de transformações de modelos. A especificação se inicia com a identificação dos requisitos e condições associadas à transformação, documentados no *Model Transformation Requirement Model (MTRM)*. Em seguida, na fase de projeto, estes requisitos são refinados para definir as características estruturais e comportamentais da transformação, representadas no *Model Transformation Specification Model (MTSM)*, expresso na linguagem *Trans-ML*(GUERRA et al., 2010a) e em máquinas de estados. O framework contém um catálogo de *templates* que possibilita instanciar os modelos construídos e gerar o *Transformation Formal Specification Model (MTFM)*, um modelo formal da transformação em notação Alloy onde propriedades podem ser verificadas.

  Em suma, o *Trans-DV* sistematiza as atividades de especificação e projeto da transformação em direção a vericação formal de propriedades. As fases de projeto específico de plataforma e implementação da transformação não são abordados no

framework. A automação proposta pelo framework contempla a geração da especificação em notação Alloy, contudo os refinamentos realizados entre o modelo de requisitos e o modelo de projeto são feitos de forma manual. Ademais, assim como os trabalhos apresentados anteriormente, linguagens para especificação de processos de software também não são utilizadas na definição do processo proposto.

- **A Model Based Development Approach for Model Transformation (KOLAHDOUZ-RAHIMI; LANO, 2012b)**

  (KOLAHDOUZ-RAHIMI; LANO, 2012b) propõe um processo de desenvolvimento e uma técnica para especificação de transformações baseados nas abordagens DDM e no *Constraint Driven Development (CDD)* com foco em especificação formal de transformações.

  O processo compreende as etapas de *Requisitos, Especificação abstrata, Especificação explícita e Projeto*, e *Implementação*. Desta forma, na etapa de requisitos são definidos os requisitos da transformação, os metamodelos fonte e alvo, as restrições que devem ser estabelecidas ou preservadas pela transformação e, se necessário, os requisitos não funcionais. Em seguida, na *Especificação abstrata* são definidas as relações entre os metamodelos fonte e alvo, através de restrições, e as pré e pós-condições, sob a forma de predicados em máquinas de estados abstratas. A *Especificação explicita e Projeto*, define as regras que compõem a transformação sob a forma de operações expressas como pré e pós-condições e a ordem de execução dessas regras, em máquinas de estados. Finalmente, na *Implementação* o código executável pode ser gerado em diferentes linguagens de programação, como, por exemplo, Java, ATL ou Kermeta.

  Este trabalho, assim como os demais apresentados neste capítulo, propõe um processo para desenvolver transformações, mas não o formaliza em nenhuma notação específica. Desta forma, define *quais* as tarefas que devem ser executadas, mas não especifica, por exemplo, *como* nem *quem* deve executar essas tarefas. Adicionalmente, embora utilize a linguagem UML em algumas atividades, tanto a geração de código quanto a verificação de propriedades se baseia na especificação da transformação sob a forma de *constraints*. Como consequência requer conhecimento específico do usuário na definição de constraints. Ademais, o processo fornece automação somente para a fase de geração do código da transformação.

- **Applying MDE to the (Semi-)Automatic Development of Model Transformation (BOLLATI et al., 2013)**

  O trabalho de (BOLLATI et al., 2013) é o que mais se assemelha ao framework definido nesta tese, pois propõe uma abordagem metodológica e técnica para o uso de DDM no desenvolvimento de transformações de modelos, chamada de MeTA-GeM. Nesta, modelos de transformação são especificados textualmente em diferentes níveis de abstração similares aos níveis PIM (modelo independente de plataforma), PSM (modelo específico de plataforma) e código, propostos pela arquitetura MDA. Adicionalmente a abordagem insere o nível PDM (Modelo dependente de plata-

forma) que é uma especialização do PSM para abordagens específicas de transformação (ex. declarativa e imperativa).

O processo de desenvolvimento no MeTAGeM se inicia com a construção do *Modelo de transformação independente de plataforma (PIT)*, onde são definidos os relacionamentos entre elementos dos metamodelos fonte e alvo. Este modelo é convertido no *Modelo de transformação específico de plataforma (PST)* que representa uma abordagem particular de desenvolvimento de transformação. Atualmente o framework contempla somente a abordagem híbrida. O modelo *PST* é transformado em um *Modelo dependente de plataforma (PDM)*, que representa a transformação em uma linguagem específica (atualmente ATL ou RubTL), para então ser gerado o código fonte da transformação.

O framework MeTAGeM difere da proposta apresentada nesta tese inicialmente por encapsular o processo de desenvolvimento em uma ferramenta própria. Desta forma, o uso da abordagem está vinculado à ferramenta que de fato é a responsável por conduzir o desenvolvedor na construção dos modelos necessários e na conversão destes modelos entre os diversos níveis de abstração. Adicionalmente, o MeTAGeM não contempla a fase de especificação de requisitos no ciclo de vida, o desenvolvimento se inicia na fase de projeto da transformação. Em relação à linguagem de modelagem, diferentemente das propostas atuais de desenvolvimento de transformações, o MeTAGeM utiliza linguagens de especificação de modelos textuais em vez de linguagens visuais.

- **The Genenal Algorithm for the MDA Transformation Models (TAVAC; TAVAC, 2013)**

  Este trabalho propõe um algoritmo genérico aplicado ao projeto de transformações de modelos MDA que organiza os passos necessários à especificação de regras de mapeamentos entre os diversos metamodelos envolvidos na transformação.

  A proposta define um conjunto de tarefas que devem ser executadas para realizar o projeto da transformação, como, por exemplo, a identificação de mapeamento entre elementos dos metamodelos e a definição das regras de transformação. O algoritmo proposto consiste em um conjunto de passos necessários para executar as tarefas definidas, organizados sequencialmente em uma abordagem estruturada, com condições e laços, que levam ao projeto da transformação. No total o algoritmo contém oito passos e produz como resultado duas saídas, um metamodelo, com os estereótipos e valores rotulados necessários para a transformação, e o modelo da transformação contendo as regras definidas com base nos mapeamentos entre os construtores dos metamodelos fonte e alvo.

  Em suma, este trabalho utiliza um algoritmo para conduzir o desenvolvimento de transformações de modelos e contempla o projeto de transformações no nível PIM-PSM da MDA. O algoritmo ainda não possui implementação, deve ser executado manualemente pelo usuário, e não recomenda nenhuma linguagem de modelagem para realizar a especificação da transformação.

- **Specifying Model Transforamation by Direct Manipulation using Concrete Visual Notation and Interactive Recommendations (AVAZPOUR; GRUNDY; GRUNSKE, 2015)**

  O trabalho (AVAZPOUR; GRUNDY; GRUNSKE, 2015) propõe uma ferramenta, chamada CONVErT, que desenvolve transformações com base em exemplos concretos de modelos construídos nas linguagens fonte e alvo da transformação.

  O desenvolvimento na CONVErT se inicia com o especialista do domínio, selecionando exemplos de modelos de entrada e saída da transformação. Em seguida, esses exemplos são especificados em uma notação concreta provida pela ferramenta para que mapeamentos visuais sejam definidos com base em correspondências entre os modelos através de operações de *Drag* e *Drop*. Correspondências são sugestões feitas pelo sistema que indicam possíveis mapeamentos entre os modelos e que podem ser utilizadas ou não pelo usuário. Essas sugestões são baseadas em heurísticas tais como similaridades entre nomes, valores e estruturas dos elementos. Finalmente, o sistema gera um código de *script* da transformação em linguagem XSLT.

  Este trabalho encapsula na ferramenta CONVErT a estratégia de condução do desenvolvimento da transformação. O processo de fato consiste em um conjunto de passos que indicam como a ferramenta deve ser utilizada. Desta forma, o uso do processo requer o uso da ferramenta. Um ponto importante observado na proposta é a não manipulação de metamodelos pelo desenvolvedor, uma vez que os mapeamentos são definidos com base em exemplos. Essa característica contribui para diminuir a complexidade do desenvolvimento, pois não demanda conhecimento em técnicas de metamodelagem. Contudo, pode levar a construção de transformações incompletas, uma vez que a completude da transformação estará diretamente relacionada à expressividade do exemplo utilizado na definição das correspondências.

8.1.1 Análise dos Trabalhos

A Figura 8.3 mostra uma tabela que sumariza o resultado da pesquisa com base nos critérios estabelecidos no protocolo definido no início da Seção 8.1. A primeira coluna da tabela lista os critérios avaliados e as demais colunas a avaliação feita para cada trabalho. Na última coluna é apresentada a avaliação do framework MDTD proposto nesta tese.

Como pode ser observado na tabela, as estratégias utilizadas pelos trabalhos apresentados para conduzir o desenvolvimento de transformações em geral se concentram na definição de um conjunto de tarefas que devem ser seguidas pelo desenvolvedor ao longo do ciclo de vida do desenvolvimento. Alguns trabalhos, chamados de abordagem sistemática, organizam essas tarefas em fases indicando quais os artefatos que devem ser construídos (KITCHENHAM, 2004), (SIIKARLA et al., 2008), (KOLAHDOUZ-RAHIMI; LANO, 2012a). Outros trabalhos detalham essas tarefas em passos, representados sob a forma de algoritmo (LI; YIN, 2010), (TAVAC; TAVAC, 2013) ou as encapsulam em uma ferramenta (FABRO; VALDURIEZ, 2009), (BOLLATI et al., 2013), (AVAZPOUR; GRUNDY; GRUNSKE, 2015). Contudo, o uso de processos formalizados através de linguagens de modelagem de processos (PML), embora seja uma estratégia amplamente utilizada para

definir processos de desenvolvimento de software (ex. os processos RUP e XP), só foi proposto no trabalho de Vignaga (2007), que foi descontinuado.

Linguagens para modelagem de processos (*Process Modeling Language – PML*) são utilizadas para definir de maneira precisa e padronizada os elementos que compõem um processo de software e possibilitam um melhor entendimento desses elementos e suas relações. Além disso, permitem que os processos sejam avaliados e melhorados, e que sejam executados em ambientes computacionais, facilitando com isso seu uso (ANDERL; RASSLER, 2008). Consideramos que a ausência de processos formalizados de desenvolvimento de transformação é ainda mais crítica quando se está inserido em um contexto de DDM, onde se busca exatamente a automação do processo de desenvolvimento de software.

A carência de trabalhos que abordem a fase de especificação da transformação também é um ponto importante observado na revisão bibliográfica. Transformações, assim como outros tipos de software, precisam ser analisadas antes de serem projetadas. Nesta tese consideramos que a identificação dos requisitos é um ponto crítico do desenvolvimento, pois em geral não existe um usuário para especificar o que será construído. Ademais, a fase de especificação dos requisitos envolve também a identificação e, em alguns casos, a definição de metamodelos, tarefa de grande complexidade e que, se realizada de forma errada, pode comprometer todo o desenvolvimento, pois a transformação é construída com base nos metamodelos envolvidos. Os trabalhos encontrados que tratam da fase de especificação de requisitos dizem em linhas gerais quais os artefatos que devem ser construídos, mas não oferecem orientações de como identificar esses requisitos. O trabalho (SANI; POLACK; PAIGE, 2011) embora aborde a especificação formal de requisitos, tem como foco a verificação de propriedades, não contemplando a fase de implementação da transformação.

Muitos trabalhos também não oferecem automação entre as fases de desenvolvimento o que torna o processo de construção da transformação mais custoso e complexo, pois o mapeamento entre os modelos até o código precisa ser feito manualmente pelo desenvolvedor.

Adicionalmente, aspectos ortogonais ao desenvolvimento são tratados somente por algumas abordagens. A definição dos requisitos não funcionais, por exemplo, é abordada em (KUSTER; RYNDUNA; HAUSER, 2005), mas orientações sobre como identificar esses requisitos não são fornecidas. Requisitos não funcionais são complexos de serem identificados e documentados mesmo em sistemas de software convencionais, pois se não forem precisamente especificados não podem ser avaliados quando o software estiver pronto. Guias que possam ajudar na identificação desses requisitos são importantes para facilitar a execução desta tarefa. Atividades de gerenciamento também não foram observadas nas abordagens propostas. Embora algumas trabalhos abordem o desenvolvimento incremental, eles não especificam tarefas para conduzir o desenvolvedor quanto à definção do escopo desses incrementos.

Observou-se também que diversas linguagens de modelagem são utilizadas pelas abordagens. Em sua maioria são definidas novas notações ou extensões pesadas[1] da UML o

[1]Extensões que incluem novos símbolos à UML.

que pode dificultar a compreensão da mesma e demandar a utilização de ambientes específicos de desenvolvimento.

Algumas abordagens oferecem automação do desenvolvimento com o objetivo de realizar verificação formal (SANI; POLACK; PAIGE, 2011) e/ou automatizar a conversão dos modelos entre as fases do desenvolviemento (FABRO; VALDURIEZ, 2009) e (BOLLATI et al., 2013) e / ou para geração de código (KOLAHDOUZ-RAHIMI; LANO, 2012a) e (AVAZPOUR; GRUNDY; GRUNSKE, 2015). Contudo, em todos os casos de automação o desenvolvimento é dependente de uma ferramenta construída especificamente para a abordagem. Nenhuma proposta de automação pode ser executada em um ambiente independente de plataforma.

Em geral as propostas analisadas foram submetidas à avaliações realizadas sob a forma de estudos de caso ou experimentos controlados com grupos de usuários, conforme ilustrado na última linha da tabela. Nenhuma das propostas utilizou a ISO/IEC-15504 como frame de referência para avaliação.

A abordagem proposta nesta tese avança no estado da arte em relação aos trabalhos aqui apresentados porque busca sistematizar o desenvolvimento de transformações de modelos através da especificação de um processo DDM, em uma PML, que contempla as diversas fases do ciclo de vida da transformação. Todo o processo é (semi) automatizado por uma cadeia de transformações que cobre desde os requisitos até a geração do código, atualmente em linguagem ATL e QVT. Ao longo dessas fases o processo cobre atividades relacionadas a aspectos não apenas técnicos, como a definições de regras de transformações, mas também a aspectos gerenciais, de planejamento e monitoramento do desenvolvimento, a questões relacionadas à validação e verificação e também à automação. O processo adota uma linguagem de modelagem que é um perfil UML definido através de uma extensão leve[2] da UML, e que, portanto, permite o uso das ferramentas de modelagem para UML já existentes no mercado. Particularmente, adotamos a independencia de plataforma como estratégia no nosso processo, tanto em relação à criação quanto em relação à conversão dos modelos.

8.2 PROPOSTAS DE LINGUAGENS PARA ESPECIFICAR TRANSFORMAÇÕES

Esta seção apresenta as propostas de linguagens de modelagem e notações visuais para especificação de transformações de modelos. Linguagens para especificação formal de transformações e linguagens baseadas em transformações de grafos não foram incluídas na pesquisa.

- **Proposed Design Notation for Model Transformation (RAHIM; MANSOOR, 2008)**

  O trabalho de (RAHIM; MANSOOR, 2008) é um dos primeiros trabalhos encontrados na literatura que propõe uma notação gráfica para projetar transformações. A proposta compreende uma sintaxe abstrata, uma sintaxe concreta e a semântica da notação. A sintaxe abstrata contém os elementos necessários para definir *metamodelos fonte*, *metamodelos resultantes* e seus respectivos elementos. A semântica e as

[2]Extensão que não acrescenta novos símbolos aos já existentes na UML

Nr	Critério	Kitchenham (2005)	Vignaga (2007)	Siikarla (2008)	Del Fabro (2009)	Jin (2010)	Sani (2011)	Rahimi (2012)	Bolatti (2013)	Tavac (2013)	Avazpour (2015)	MDTD
C1	Estratégia utilizada	Abordagem sistematica	Processo	Abordagem sistemática	Ferramenta	Algoritmo	Abordagem sistemática	Abordagem sistemática	Ferramenta	Algoritmo	Ferramenta	Processo
C2	Fase de requisitos	Sim, informal	--	Sim, informal	Não	Não	Sim, detalhada	Sim, detalhada	Não	Não	Não	Sim, detalhada
C3	Projeto independente de plataforma	Sim	--	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim
C4	Transforma requisito em projeto	Sim, manual	--	Não	Sim (com automação)	Não	Não	Não	Não	Não	Não	Sim (com automação)
C5	Projeto especifico	Não	--	Não	Não	Sim	Não	Não	Sim	Sim	Não	Sim
C6	Transforma projeto independente em projeto especifico	Não	--	Não	Não	Não	Não	Não	Sim	Não	Não	Sim
C7	Fase de implementação	Sim	--	Sim	Sim	Não informado	Não	Sim	Sim	Não	Sim	Sim
C8	Geração de código	Não	--	Não	Sim	Não informado	Não	Sim	Sim	Não	Sim	Sim
C9	Validação e verificação	Sim, manual	--	Não informado	Não informado	Não informado	Sim, automático	Sim, automático	Não informado	Não informado	Não informado	Sim manual
C10	E uma abordagem DDM	Não	--	Não	Sim	Não	Não	Sim	Sim	Não	Não	Sim
C11	DSL	Extensão pesada UML	--	Não informa	Linguagem propria	Linguagem própria	Extensão pesada UML / Formal	Extensão leveUML / Formal	Linguagem própria	Não informa	Linguagem própria	Entensão leve UML
C12	Depende de ferramenta	Não informa	--	Não informa	Sim	Não informa	Não informa	Não informa	Sim	Não	Sim	Não
C13	Validação	Não informa	Não	Pequeno estudo de caso na indústria	Sim Experimento	Sim Exemplo BPMN para JPDL	Exemplo Classe para tabela	Sim Estudos de caso	Sim Esperimento	Exemplo Banco para Classe	Sim Exemplos/ experimento	Sim ISO/IEC-15504 e Experimento

Figura 8.3 Comparação entre os trabalhos pesquisados de acordo com os critérios estabelecidos na Seção 8.1

regras de boa formação da linguagem estão especificadas em lógica de predicados. A sintaxe concreta da linguagem consiste basicamente de um diagrama de objetos da UML, que representam alguns elementos da sintaxe abstrada, e de algumas sentenças em OCL.

Esta notação representa um dos primeiros esforços na utilização de linguagens visuais para desenvolver transformações e, portanto, tem algumas limitações limitações. Ela é voltada somente à especificação do projeto da transformação, não se aplicando às demais fases do ciclo de vida do desenvolvimento da transformação. Também não propõe mapeamentos para linguagens específicas de transformação. Desta forma, se propõe a documentar o projeto da transformação, não gerando código a partir desta documentação.

- **Engineering Model Transformation with TransML (GUERRA et al., 2010b)**

  TransML é uma família de linguagens de modelagem para apoiar a especificação de transformações nas fases de requisitos, análise, projeto, implementação e teste. A especificação da *TransML* compreende metamodelos e diagramas apropriados a cada fase do desenvolvimento da transformação. Diversos diagramas são propostos a exemplo do *diagrama de requisitos* que possibilita modelar os requisitos funcionais e não funcionais da transformação, do *mapping diagram*, que modela os mapeamentos entre metamodelos e do *rule diagram*, para modelar a estrutura de uma regra.

  TranfML não foi selecionada como linguagem de modelagem do framework MDTD proposto nesta tese por ser uma extensão pesada da UML, isto é, adiciona novos símbolos definidos pela própria linguagem. Acreditamos que essa característica pode gerar impacto na curva de aprendizagem da linguagem como também pode limitar o uso do suporte ferramental existente atualmente para a UML, dificultando consequentemente a própria adoção da abordagem tanto na indústria quanto na academia.

- **Query/View/Transformation Specification (QVT)**[3]

  Query View Transformation specification (QVT) é a proposta da OMG para especificação e construção de transformações de modelos. Mais especificamente a QVT-Relation, parte declarativa da linguagem, possibilita especificação textual de transformações, usando diretamente a sintaxe da linguagem, e especificação visual de transformações através do *diagrama de classe* da UML ou do *diagrama de transformação*.

  A especificação de transformações através do *diagrama de classes* UML consiste em anotar os elementos do diagrama com os construtores da linguagem QVT. Já a especificação com o *diagrama de transformação* utiliza uma notação visual própria para representar os construtores da linguagem. Nesta tese adotamos o diagrama de classes para criar o *modelo específico de plataforma* da nossa transformação, pois

[3]QVT - http://www.omg.org/spec/QVT/1.0/PDF/

consideramos que o uso do *diagrama de transformação* vincularia a abordagem a uma ferramenta específica para construir este diagrama.

8.2.1 Considerações sobre as Linguagens de Modelagem de Transformações

O uso de linguagens para criar modelos de transformação de modelos independentes de plataforma é uma preocupação da maioria dos trabalhos analisados. Contudo, observa-se que os trabalhos têm criado notações próprias o que gera uma demanda por ferramentas de modelagem específicas ao uso da linguagem proposta além de requerer um esforço a mais relativo ao aprendizado da linguagem. Em outra direção, esta tese optou por definir um perfil UML como linguagem de modelagem e assim aproveitar o suporte ferramental e valorizar o conhecimento da UML, já difundido tanto na indústria como na academia.

8.3 CONSIDERAÇÕES SOBRE O CAPÍTULO

Este capítulo apresentou um levantamento sobre propostas utilizadas para conduzir o desenvolvimento de transformação de modelos. A partir de uma revisão sistemática foi possível observar que até o presente momento não existem processos de software especificados em linguagens para processos, voltados ao desenvolvimento de transformações de modelos. Destacou-se também a carência de abordagens que contemplem um ciclo de vida completo de desevolvimento, desde a fase de especificação de requisitos até a implementação e a validação da transformação, de maneira (semi-) automatizada. As propostas que oferecem algum recurso de automação estão vinculadas a ferramentas próprias. Adicionalmente, observamos que as linguagens para modelagem de transformação também estão vinculadas a ferramentas próprias de desenvolvimento. Esta revisão bibliográfica nos permite inserir nosso trabalho no estado da arte de transformação de modelos tanto pela definição de um processo, especificado em uma PML, que contempla diversas fases do desenvolvimento da transformação, quanto pela definição de uma linguagem de modelagem qua não demanda ferramenta específica.

Capítulo 9

CONCLUSÃO

Neste trabalho tratamos da complexidade que permeia o desenvolvimento de transformações de modelos. Identificamos que transformações de modelos são um tipo específico de software cujo desenvolvimento possui uma singularidade que o difere dos softwares convencionais, seja nas tarefas e artefatos desenvolvidos, como nas tecnologias utilizadas para apoiar este desenvolvimento. Desta forma, requerem processos, linguagens e ambientes, específicos para este domínio.

Encontramos trabalhos que auxiliam o desenvolvedor através da organização das tarefas necessárias para conduzir o desenvolvimento de transformação bem como de linguagens para especificação de transformações e ferramentas automatizadas. Contudo, observamos que existe uma carência de abordagens que tratem de todo o ciclo de vida de desenvolvimento deste tipo de software. Principalmente, no que se refere à especificação, devidamente formalizada em uma PML, de um processo que conduza a execução das atividades necessárias ao longo de todo o ciclo de vida e a integração do processo às tecnologias apropriadas.

Para contribuir com este cenário propomos sistematizar o desenvolvimento de transformações de modelos na abordagem DDM através de um framework que compreende um processo de desenvolvimento, uma linguagem de modelagem e uma cadeia de transformação, e que possibilita desenvolver transformações abstraindo detalhes específicos de plataforma.

A linguagem de modelagem proposta é um perfil UML baseado em metamodelos que tratam da especificação de modelos de transformações de modelos em diversos níveis de abstração, desde a fase de requisitos, utilizando diagramas UML. Este perfil representa uma customização da UML para o domínio de transformações. Portanto, devido a UML ser uma linguagem independente de plataforma e considerada padrão para o desenvolvimento de software, o desenvolvimento de transformações se beneficia do conhecimento dos desenvolvedores nesta linguagem e do amplo suporte ferramental existente para ela.

O processo de desenvolvimento apoia a construção de transformações de modelos de forma iterativa e incremental possibilitando a adoção da DDM pelas organizações de

maneira planejada e gradual. Por estar especificado em uma PML, o processo assiste o desenvolvedor passo a passo nas tarefas que envolvem a construção de transformações desde a especificação de requisitos até a geração do código, orientando-o também quanto a utilização dos diversos recursos que compreendem o framework, linguagens, automação, e ferramentas.

A cadeia de transformações provida pelo framework (semi) automatiza o processo de desenvolvimento e possibilita que os modelos construídos em cada fase do processo sejam transformados em modelos menos abstratos para serem complementados na fase seguinte. Assim, beneficia o desenvolvimento de transformações com características inerentes à DDM, tais como produtividade e portabilidade. Ademais, a geração semi-automática do código em linguagens específicas de transformação, particularmente ATL e QVT, também reduz a necessidade de conhecimento do desenvolvedor nestas linguagens.

O framework definido, portanto, integra processo de desenvolvimento, linguagem padrão de modelagem e recursos para a (semi) automação no âmbito de todo o ciclo de vida de desenvolvimento de transformações, na abordagem DDM. Desta forma, contribui para diminuir o nível de complexidade do desenvolvimento de transformações, pois possibilita que transformações sejam especificadas em uma linguagem e ambiente conhecido da comunidade de desenvolvimento de software, apoiadas por um processo que sistematiza todas as atividades envolvidas. Essa contribuição é evidenciada nos resultados observados no estudo de caso, onde pessoas, com diferentes níveis de conhecimento em DDM e pouco ou nenhum conhecimento em linguagem de transformação, conseguiram desenvolver código executável validado. Além disso, conforme mostrado no experimento controlado, a qualidade do código gerado pelo framework foi semelhante à qualidade do código construído diretamente por pessoas que tinham conhecimento em linguagens de transformação.

O framework MDTD foi avaliado com base na norma ISO/IEC-15504 atendendo satisfatoriamente às exigências da norma para o nível de capacidade 1. Essa avaliação, além de ofecer maior confiança aos resultados obtidos, também é um diferencial que o torna apto a ser utilizado por empresas que demandam essa maturidade no processo de desenvolvimento.

Adicionalmente às contribuições obtidas com o trabalho, a partir dos estudos de caso e experimentos realizados observamos que o framework tem potencial para ser utilizado também no ensino tanto de transformações quanto da própria abordagem DDM. A integração dos diversos elementos, linguagens, processo, automação e ambiente, sintetiza os conceitos relacioanados à DDM auxiliando no entendimento da mesma.

Por fim, acreditamos que esse trabalho pode auxiliar em uma possível expansão do uso da DDM tanto na indústria quanto na academia, uma vez que o desenvolvimento de transformações de modelos, artefato fundamental nesta abordagem, pode ser conduzido por um framework que integra os elementos essenciais a este desenvolvimento e possiblita a abstração de aspectos específicos de tecnologia.

9.1 PROPOSTA DE TRABALHOS FUTUROS

O framework proposto neste trabalho contempla vários aspectos do desenvolvimento de transformações, mas não cobre todas as especificidades deste domínio. Desta forma, sugerimos as seguintes direções de pesquisa para exploração futura:

- *Definição de uma semântica comportamental para o metamodelo proposto.* Na DDM modelos são especificados em linguagens de modelagem com sintaxe e semântica bem definida. Esses modelos podem ser utilizados para realizar simulações que auxiliem na validação da transformação ainda em estágio de especificação. Para viabilizar a simulação de modelos é necessário especificar a semântica comportamental da linguagem de modelagem. A especificação da semântica comportamental de metamodelos requer o uso de métodos formais, para prover o rigor e precisão necessária a esta especificação (GARGANTINI; RICCOBENE; SCANDURRA, 2009). Pretendemos, portanto, investigar trabalhos que propõem técnicas para especificação de semântica comportamental de metamodelos para então definir formalmente a semântica do MMT.

- *Melhoria do apoio à verificação formal de transformações.* O processo MDTDproc contém tarefas relacionadas à definição de propriedades para verificação semântica da transformação. Contudo, não adota nenhuma linguagem formal nem técnica de verificação específica. Pretendemos, portanto, identificar uma abordagem para especificação formal de propriedades e integrá-la ao nosso processo. Nesta direção, já existem propostas para especificação formal de transformações visando a geração automática de propriedades (SANI; POLACK; PAIGE, 2011). Pretendemos, investigar essas propostas e analisar uma possível integração das mesmas ao nosso processo.

- *Geração automática de testes de software.* A geração automática de casos de teste é uma abordagem já utilizada no desenvolvimento de software atual. É importante considerar, então, a possibilidade de gerarmos casos de testes automáticos para os modelos de transformação de modelos construídos.

- *Execução de novos estudos de casos e experimentos.* O estudo de caso apresentado neste trabalho é considerado um ponto de partida para o aperfeiçoamento da abordagem. Por ter sido realizado em ambiente acadêmico e com um cenário fictício, consideramos pertinente a realização de novos estudos para avaliar e melhorar a abordagem proposta. Adicionalmente pretendemos investigar para esses novos estudos o uso de métricas mais objetivas, tais como número de regras e LOC, que possam ser utilizadas para avaliar o código gerado pelo framework.

- *Definição e implementação de recursos de rastreabilidade.* A rastreabilidade entre os artefatos gerados ao longo de um ciclo de vida de desenvolvimento de software é fundamental para a evolução do sistema. Em processos DDM esta rastreabilidade é ainda mais importante para propagar mudanças nos artefatos gerados, bem como

para prover recursos de engenharia reversa. Portanto, achamos relevante implementar esse recurso na abordagem proposta.

- *Especificação de transformações bidirecionais.* Nossa abordagem trata do desenvolvimento de transformações unidirecionais. Pretendemos também analisar técnicas para especificação de transformações bidirecionais, pois este tipo de transformação favorece a sincronização e a rastreabilidade entre os artefatos produzidos ao longo do desenvolvimento, características fundamentais no contexto da DDM, conforme mencionado no item anterior.

- *Melhoria do framework MDTD.* Adicionalmente às linhas de pesquisa indicadas nos itens anteriores, pretendemos também implementar melhorias pontuais no framework proposto, tais como: aumentar a expressividade da geração de código nas linguagens ATL e QVT, pois não contemplamos todos os comandos dessas linguagens; implementar novas transformações para a cadeia que automatiza o processo MDTDproc, como, por exemplo, geração do modelo inicial de arquitetura na fase TDM, que atualmente é uma atividade manual; prover geração de código para outras linguagens de transformação; entre outros.

REFERÊNCIAS BIBLIOGRÁFICAS

AMSTEL, V.; MARCEL, F.; BRAND, M. Van den. Quality assessment of atl model transformations using metrics. Spain, 2010. Proceedings of the 2nd International Workshop on Model Transformation with ATL.

ANDERL, R.; RASSLER, J. Pml, an object oriented process modeling language. In: _____. *Computer-Aided Innovation (CAI): IFIP 20th World Computer Congress, Proceedings of the Second Topical Session on Computer-Aided Innovation, WG 5.4/TC 5 Computer-Aided Innovation, September 7-10, 2008, Milano, Italy*. Boston, MA: Springer US, 2008. p. 145–156. ISBN 978-0-387-09697-1. Disponível em: ⟨http://dx.doi.org/10.1007/978-0-387-09697-1_12⟩.

ANDROMDA. *AndroMDA*. 2016. Available at http://www.andromda.org/(06/10/2016).

AVAZPOUR, I.; GRUNDY, J.; GRUNSKE, L. Specifying model transformation by direct manipulation using concrete visual notation and interactive recommendations. Science Direct, p. 195 – 211, 2015. Journal of Visualization Language and Computing.

BEZIVIN, J. et al. Model transformations? transformation models? Springer-Verlag, Berlin Heidelberg, p. 440 – 453, 2006. MoDELS Conference.

BOLLATI, V. et al. Applying mde to the (semi-)automatic development of model transformations. Elsevier, p. 699–718, 2013. Information and Software Technology.

BOOCH; RUMBAUGH; JACOBSON, I. *UML Guia do Usuário*. [S.l.]: Addison – Wesley, 2006.

BRAGA, C. et al. On the specification verification and implementation of model transformation with transformation contracts. ACM, São Paulo, p. 108 – 123, 2011. ISSN 978-3-642-25031-6. SBMF – Simposio Brasileiro de Engenharia de Software.

BRAGA, C.; SANTOS, C.; DA SILVA, V. . Consistency of model transformation contracts. Science Direct, p. 86 – 104, 2014. Science of Computer Programming (Print).

BRAMBILLA, M.; CABOT, J.; WIMMER, M. *Model-Driven Software Engineering in Practice*. 1st. ed. [S.l.]: Morgan Claypool Publishers, 2012. ISBN 9781608458820.

BRAUN, P.; MARCHALL, F. Botl – the bidirectional object oriented transformation language. 2003. Technical Report TYM010307.

CALDIERA, V.; ROMBACH, H. Goal question metric paradigm. encyclopedia of software engineering. p. 528–532, 1994. Encyclopedia of Software Engineering.

DORAY, A. Beginning apache struts: From novice to professional. In: _____. Berkeley, CA: Apress, 2006. cap. The MVC Design Pattern, p. 37–51. ISBN 978-1-4302-0129-8. Disponível em: ⟨http://dx.doi.org/10.1007/978-1-4302-0129-8_5⟩.

FABRO, M. D.; VALDURIEZ, P. Towards the efficient development of model transformations using model weaving and matching transformations. 2009. Software System Model.

GARGANTINI, A.; RICCOBENE, E.; SCANDURRA, P. A semantic framework for metamodel-based languages. n. 3, p. 415 – 454, 2009. Automated Software Engineerin.

GUERRA, E. et al. Transml: A family of languages to model model transformations. Springer Verlag, 2010.

GUERRA, E. et al. A visual specification language for model-to-model transformations. IEEE, 2010. IEEE Symposium on Visual Languages and Human-Centric Computing.

HUTCHINSON, j.; WHITTLE, J. Empirical assessment of mde in industry. ACM, 2011. ICSE.

ISO. *ISO/IEC15504 Information technology-Process Assessment*. 2005. Available at http://www.iso.org/iso/iso$_c$*atalogue*/*catalogue*$_t$*c*/*catalogue*$_d$*etail.htm*?*csnumber* = 37458(2005).

KITCHENHAM, B. Dprocedures for performing systematic reviews. 2004. ISSN 1353-7776. TR/SE-0401.

KLEPPE, A.; WARMER, J.; BAST, W. *MDA Explained. The Model Driven Architecture: Practice and Promise.* [S.l.]: Addison – Wesley, 2003.

KOLAHDOUZ-RAHIMI, S.; LANO, K. Fundamentals of software engineering: 4th ipm international conference, fsen 2011, tehran, iran, april 20-22, 2011, revised selected papers. Springer Berlin Heidelberg, Berlin, Heidelberg, p. 48–63, 2012. Disponível em: ⟨http://dx.doi.org/10.1007/978-3-642-29320-7_4⟩.

KOLAHDOUZ-RAHIMI, S.; LANO, K. A model-based development approach for model transformations. 2012. 4th IPM International Conference on Fundamentals of Software Engineering.

KUSTER, J.; RYNDUNA, K.; HAUSER, R. A systematic approach to designing model transformations. Computer Science, 2005. Research Report. Computer Science.

LANO, K.; CLARK, D. Model transformation specification and verification. IEEE, 2008. The Eighth International Conference on Quality Software.

LI, J.; YIN, G. Method of constructing model transformation rule based on reusable pattern. IEEE, Berlin, Heidelberg, p. V8–519 – v8–524, 2010. International Conference on Computer Application and System Modeling.

MACIEL, R.; SILVA; MASCARENHAS, L. A. B. C. An edoc-based approach for specific middleware services development. IEEE, Potsdan, p. 143 – 151, 2006. Fourth Workshop on Model-Based Development of Computer-Based Systems and Third International Workshop on Model-Based Methodologies for Pervasive and Embedded Software (MBD-MOMPES'06).

MAGALHAES, A. *MDTD - Model Driven Transformation Development*. 2016. Available at homes.dcc.ufba.br/ anapfmm/mdtd/(01/05/2016).

MAGALHãES, A. P.; MACIEL, R.; ANDRADE, A. Towards a metamodel design methodology: Experiences from a model transformation metamodel design. Pitsburgh, USA, p. 625–630, 2015. The 27th International Conference on Software Engineering and Knowledge Engineering.

MELLOR, S. *MDA Distilled*. [S.l.]: Addisson-Wesley, 2004.

MENS, T.; CZARNECKI, K.; GORP, P. A taxonomy of model transformation. p. 125 – 142, 2006. Proceedings of the International Workshop on Graph and Model Transformation (GraMoT 2005)Graph and Model Transformation.

OMG. *Unified Modeling Language*. 2005. Available at http://www.omg.org/spec/UML/2.5/PDF/(09/03/2015).

OMG. Software process engineering metamodel specification. 2008. Formal/08-04-01.

OMG. *Model Driven Architecture*. 2014. Available at http://www.omg.org/mda/specs.htm(23/08/2016).

RAHIM, L.; MANSOOR, S. Proposed design notation for model transformation. IEEE, p. 589 – 598, 2008. 19th Australian Conference on Software Engineering.

SANI, A.; POLACK, F.; PAIGE, R. Model transformation specification for automated formal verification. IEEE, p. 76 – 81, 2011. 5th Malaysian Conference in Software Engineering.

SIIKARLA, M. et al. Theory and practice of model transformations: First international conference, icmt 2008, zürich, switzerland, july 1-2, 2008 proceedings. Springer Berlin Heidelberg, Berlin, Heidelberg, p. 1–15, 2008. Disponível em: ⟨http://dx.doi.org/10.1007/978-3-540-69927-9_1⟩.

SILVESTR, L.; BASTARRICA, M. C.; OCHOA, S. F. A model-based tool for generating software process model tailoring transformations. SCITEPRESS, 2014. Model-Driven Engineering and Software Development (MODELSWARD).

SOMMERVILLE, I. *Software Enginnering*. [S.l.]: Pearson, 2010.

STAHL, T.; VOLTER, M. *Model-Driven Software Development. Technology, Engineering, management*. [S.l.]: Wiley, 2010.

TAVAC, M.; TAVAC, V. The general algorithm for the design of the mda transformation models. 2013. Fifth International Conference on Computational Intelligence, Communication Systems and Networks.

VARRO, D.; PATARICZA, A. Automated formal verification of model transformation. 2003. CSDUML 2003: Critical Systems Development in UML.

VIGNAGA, A. A methodological approach to developing model transformations. 2007. Model Driven Engineering Languages and Systems.

WAGELSAR, D. Composition techniques for rule-based model transformation languages. Lecture Notes in Computer Science, p. 152 – 167, 2008. Theory and Practice of Model Transformations.

WOHLIN, C. et al. *Experimentation in Software Engineering*. [S.l.]: Springer, 2012. ISBN 978-3-642-29043-5.

Apêndice

A

ADAPTAÇÃO DA NORMA ISO/IEC-15504 PARA O DOMÍNIO DE TRANSFORMAÇÕES DE MODELOS

A *ISO/ICE-15504 – Information Technology – Software Process Assessment* (ISO, 2005) provê um framework para avaliação de processos de software que possibilita avaliar quando os processos efetivamente alcançam seus objetivos e identificar as causas da pouca qualidade que influenciam no prazo e custo do desenvolvimento de software. O framework pode ser utilizado para vários propósitos, dentre eles, para possibilitar que uma organização entenda o estado do seu processo e possa então melhorá-lo.

A avaliação de um processo é realizada com base em um modelo de avaliação utilizado como referência. Este modelo está organizado em duas dimensões: processos; e capacidades.

A *dimensão de processo* identifica qual o processo que será avaliado. Nesta dimensão os processos são classificados por categoria. Existem três categorias onde os processos são agrupados, são elas: processos de ciclo de vida primário, que compreendem os processos de aquisição, fornecedores e engenharia; processos de ciclo de vida organizacional; e processos de ciclo de vida de suporte. Cada processo tem um conjunto de propósitos associados a resultados que devem ser alcançados. Neste trabalho foram utilizados processos do ciclo de vida primário relacionados à *engenharia*.

O propósito dos processos de engenharia é transformar um conjunto de requisitos em um produto de software funcional que atende às necessidades do cliente. Dentre os processos especificado para engenharia este trabalho utiliza o *ENG.1 Requirements elicitation*, *ENG.4 Software requirement analysis*, *ENG.5 Software design* e *ENG.6 Software construction*.

A dimensão de *capacidade* envolve a capacidade do processo. A capacidade é expressa em termos de atributos do processo. Atributos são características que podem ser avaliadas em uma escala a ser alcançada. Esses atributos são agrupados nos seguintes níveis de capacidade:

- Nivel 0: *Incomplete*, neste nível há falhas para atender ao propósito do processo. Existem poucos artefatos produzidos como saída ou os artefatos são difíceis de serem identificados;

- Nível 1: *Performed*, neste nível o propósito do processo é alcançado, mas não se pode rastrear como eles foram alcançados. Há um consenso em quais e como as tarefas devem ser realizadas. Os artefatos de saída são identificados e atestam que o propósito do processo foi alcançado;
- Nivel 2: *Managed*, neste nível o processo resulta em artefatos de acordo com os procedimentos especificados e planejados e pode ser rastreado. Os artefatos de saída expressam requisitos de qualidade;
- Nível 3: *Established*, neste nível o processo é executado e gerenciado usando um processo definido baseado em bons princípios da engenharia de software;
- Nível 4: *Predictable*, neste nível o processo é executado de forma consistente com limites de controle para alcançar os seus objetivos. Medidas detalhadas são coletadas e analisadas levando a um entendimento quantitativo da capacidade do processo. A qualidade dos artefatos são quantitativamente conhecidas;
- Nível 5: *Optimizing*, neste nível a performance do processo é otimizada considerando necessidades atuais e futuras do negócio. O processo alcança repetidamente os objetivos do negócio. Objetivos de eficiência e performance são estabelecidos, baseado nos objetivos da organização.

Indicadores de avaliação

O modelo de avaliação de processos se baseia no princípio de que a capacidade do processo é definida demonstrando que os atributos do processo foram alcançados com base em evidências relatadas pelos indicadores de avaliação. Existem dois tipos de indicadores (ISO, 2005): *Indicadores de Performance*; e *Indicadores de Capacidade*.

Os *Indicadores de Performance*, aplicado ao nível 1, indicam que o propósito do processo foi alcançado. Esses indicadores são classificados em dois tipos: *Base Practices* (BP); e *Workproducts* (WP). As *Base Practices* (BP) indicam que as atividades desempenhadas atendem ao propósito do processo, ou seja, identificam o que precisa ser feito; Os *WorkProducts* (WP), representam os artefatos produzidos na execução do processo para alcançar o objetivo. Adicionalmente, os indicadores de performance estão relacionados a processos individuais da dimensão de processos (ex. processos de Engenharia). Desta forma, há um conjunto de BP pré-definido e associado a cada processo.

Os *Indicadores de Capacidade* do processo, aplicados aos níveis 1 a 5, estão relacionados a atividades significativas, recursos e resultados associados com o alcance dos atributos do processo. São classificados em dois tipos: *Generic Practice* (GP); e *Generic Resources* (GR). As *Generic Practices* (GP) são atividades genéricas e guias para implementação dos atributos. São projetadas de acordo com o que se pretende alcançar e muitas delas estão relacionadas a práticas de gerenciamento e práticas estabelecidas para apoiar a performance do processo no nível 1. As *Generic Resources* (GR) estão associadas a recursos que podem ser usados quando o processo é executado para alcançar os objetivos. Ex.: recursos humanos, ferramentas, métodos e infraestrutura.

A existência de *Base Practices* (BP), *Workproducts* (WP), *Generic Practices* (GP) e *Generic Resouces* (GR) provêm evidências de que o propósito do processo foi alcançado.

Estas evidências são coletas durante a avaliação do processo e mapeadas em um conjunto de indicadores possibilitando uma relação entre o processo avaliado e o modelo de processo usado na avaliação. Assim, o avaliador utiliza estes indicadores para definir a capacidade do processo. As evidências são coletadas a partir da análise dos artefatos produzidos ou a partir de sentenças definidas pelos executores do processo ou gerentes do processo.

Como avaliar os atributos

Um atributo representa uma característica mensurável de um processo. A escala para medir atributos consiste de uma percentagem de 0 a 100% que representa o quanto o atributo foi alcançado. A calibração da escala é realizada da seguinte maneira:

- (N) Não alcançado (0 a 15%) indica que tem pouca ou nenhuma evidência de que o atributo foi atingido;
- (P) Parcialmente alcançado (16% a 50%) indica que tem uma evidência de abordagem aparentemente sistemática de que alcançou o atributo;
- (L) Largamente alcançado (51% a 85%) indica que existe evidência de abordagem aparentemente sistemática que alcançou significativamente o atributo;
- (F) Completamente alcançado (86% a 100%) indica que existe evidência de abordagem completa e sistemática para alcançar completamente o atributo.

Para obter o nível de capacidade 1, utilizado neste trabalho, é necessário que os atributos de *performance* sejam considerados *largamente* ou *completamente alcançados*. Os demais níveis não serão detalhados, pois não serão abordados neste trabalho.

Os indicadores de *Práticas Genéricas* (GP), *Práticas Básicas* (BP), *Workproducts* (WP) e *Recursos Genéricos* (GR) que devem ser medidos estão associados a cada processo de referência utilizado. Nesta tese utilizamos os processos de engenharia, sub processos *ENG.1 Software requeriment elicitation, ENG.4 Software requeriment analysis; ENG.5 Software design;* e *ENG.6 Software construction*.

Adaptação da norma para o domínio de transformações de modelos

Os processos de referência descritos na norma são genéricos para qualquer domínio de software. Para melhor avaliar um sistema de transformação de modelos foi definido um mapeamento destes processos de referência para o domínio de transformações de modelos (apresentados nas tabelas A.1 a A.4 a seguir). Neste mapeamento foram selecionados os itens de avaliação pertinentes ao domínio de transformação e excluídos os itens que não se aplicam a este domínio (assinalados nas tabelas como N/A).

Cada uma das tabelas que define os processos utilizados como referência para avaliação contêm duas colunas principais. A primeira coluna descreve o processo original definido pela ISO/IEC-15504. Esta coluna está subdividida em duas outras colunas que detalham para cada item do processo a sua descrição na norma. A segunda coluna apresenta a adaptação definida para o domínio de transformações de modelos.

A coleta de evidências para cada uma das práticas básicas (BP) definidas nas tabelas a seguir será feita mediante a aplicação de questionários (apêndice B) respondidos pelos participantes do estudo de caso. A coleta de evidências dos *workproducts* (WP) será

realizada com base na análise dos artefatos produzidos pelos participantes do estudo de caso.

Modelo de referencia do processo ENG.1 da ISO/IEC-15504		Adaptação do modelo para o domínio de transformações
ID do processo	ENG.1	ENG.1
Nome do processo	Elicitação de requisitos de Software	Elicitação de requisitos do domínio
Propósito do processo	Estabelecer o conjunto de necessidades do cliente que serão utilizadas como base para definir os requisitos do produto.	Estabelecer o conjunto de necessidades das aplicações DDM que serão utilizadas como base para definir os requisitos da transformação. Transformações são definidas para automatizar o processo de desenvolvimento na abordagem DDM e estão relacionadas a um conjunto de aplicações de um domínio. Os requisitos de uma transformação não são definidos por necessidades de um cliente e sim pelas necessidades desse domínio de aplicações.
Resultados esperados	1) É estabelecida uma comunicação continuada com o cliente.	N/A
	2) São definidos os requisitos do cliente.	2) São definidos os requisitos do processo DDM que será automatizado (e das aplicações que fazem parte de um domínio).
	3) É estabelecido um mecanismo de avaliação e incorporação de mudanças para os requisitos do cliente baseado nas mudanças das necessidades dos clientes.	N/A
	4) É estabelecido um mecanismo de monitoramento contínuo das necessidades do cliente.	N/A
	5) É estabelecido um mecanismo para garantir que os clientes possam facilmente determinar o status e a disposição de suas requisições.	N/A
	6) Mudanças provenientes de mudança de tecnologia e necessidades do cliente são identificadas, riscos são associados e seus impactos gerenciados.	N/A
Práticas Básicas (BP)	**ENG.1.BP1: Obter requisitos e requisições do cliente.** Obter os requisitos e requisições através de solicitação direta do cliente e através de revisões do propósito do negócio, ambientes de hardware, dentre outros. (resultado 1,4)	**ENG.1.BP1: Obter requisitos do processo DDM a ser automatizado. (resultado 2)**
	ENG.1.BP2: Entender as expectativas do cliente. Garantir que tanto o fornecedor quanto o cliente entendem cada requisito da mesma forma. (resultado 2)	N/A
	ENG.1.BP3: concordar com os requisitos. Obter uma concordância explicita de todas as partes relevantes envolvidas com os requisitos (resultado 2)	N/A
	ENG.1.BP4: Estabelecer baseline para os requisitos do cliente. Formalizar os requisitos do cliente e estabelecer um baseline para monitorar o projeto em relação às necessidades do cliente (resultado 2,3).	**ENG.1.BP4: Estabelecer baseline para os requisitos do processo DDM.** Formalizar os requisitos do processo DDM e estabelecer um baseline para monitorar o projeto em relação às necessidades do domínio (resultado 2).
	ENG.1.BP5: Gerenciar as mudanças nos requisitos do cliente. Gerenciar todas as mudanças feitas nos requisitos do cliente em relação ao baseline (resultado 3,6)	N/A
	ENG.1.BP6: Estabelecer o mecanismo de comunicação entre fornecedor e cliente (resultado 5).	N/A
Work Product (WP)	**a)** Registro (resultado 4,5)	N/A
	b) Registro de comunicação (resultado 1,4)	N/A
	c) Registro de controle de mudança (resultado 3,4)	N/A
	d) Relatório de análise (resultado 2, 3,6)	N/A
	e) Plano de gerenciamento de riscos (resultado 6)	N/A
	f) Plano de mitigação de riscos (resultado 6)	N/A
	g) Requisitos do cliente (resultado 1,2)	Requisitos do processo DDM (resultado 2)

Figura A.1 ENG1 – Elicitação de requisitos de software

Modelo de referencia do processo ENG.4 da ISO/IEC-15504		Adaptação do modelo para o domínio de transformações
ID do processo	ENG.4	ENG.4
Nome do processo	Analise de requisitos de software	Análise de requisitos de transformação
Propósito do processo	Estabelecer os requisitos do software.	Estabelecer os requisitos da transformação
Resultados esperados	1) Os requisitos do software são definidos.	1) Os requisitos da transformação são definidos.
	2) Os requisitos do software são categorizados e analisados para correção e teste. - A categorização é feita em termos de viabilidade e risco. - A análise dos requisitos para teste inclui desenvolver um critério de verificação. - O critério de verificação define quantitativa e qualitativamente critérios para verificação do requisito. - Os aspectos de correção envolvem verificar características tais como "completeness, understandability, testability, consistency, etc.".	2) Os requisitos da transformação são categorizados e analisados para correção e teste. - A categorização é feita em termos de viabilidade e risco. - A análise dos requisitos para teste inclui desenvolver um critério de verificação dos modelos gerados pela transformação. - Os aspectos de correção envolvem verificar os atributos de qualidade da transformação tais como "Understandability, Modifiability, Reusability, Modularity, Completeness, Consistency e Conciseness"
	3) O impacto dos requisitos do software no ambiente operacional é avaliado.	N/A
	4) A priorização para implementação dos requisitos do software é definida. 5) Os requisitos do software são aprovados e atualizados se necessário.	4) A priorização para implementação dos requisitos da transformação é definida. N/A
	6) A consistência e rastreabilidade bilateral é estabelecida entre os requisitos do software e as necessidades do cliente. 7) Mudanças nos requisitos são avaliadas em termos de tempo, custo e impacto técnico.	6) A consistência e rastreabilidade bilateral é estabelecida entre os requisitos da transformação e as necessidades do processo DDM a ser automatizado. N/A
	8) Os requisitos são divulgados às partes afetadas **ENG.4.BP1: Identificar os requisitos do software** (resultado 1) Identificar os requisitos funcionais e não funcionas do software e documentar em um documento de requisitos de software.	N/A **ENG.4.BP1: Identificar os requisitos da transformação** (resultado 1)
	ENG.4.BP2: analisar requisitos de software em termos de viabilidade técnica (N/A), riscos e capacidade de teste. - Deve ser definido o critério de verificação para os requisitos que depois serão utilizados para definir os casos de teste (resultado 2)	**ENG.4.BP2: Analisar os requisitos da transformação em termos de risco e capacidade de teste.** - deve ser definido o critério de verificação para os requisitos que depois serão utilizados para definir os casos de teste (resultado 2)
Práticas Básicas	**ENG.4.BP3: determinar o impacto do ambiente operacional** (resultado 3) **ENG.4.BP4: Priorizar e categorizar os requisitos** (resultado 2,4) - Priorizar e categorizar os requisitos analisados para mapear as futuras versões.	N/A **ENG.4.BP4: Priorizar e categorizar os requisitos** (resultado 2,4) - Priorizar e categorizar os requisitos analisados para mapear as futuras versões.
	ENG.4.BP5: Avaliar e atualizar os requisitos (resultado 5,7)	N/A
	ENG.4.BP6: Garantir a consistência e rastreabilidade bilateral entre requisitos do sistema e requisitos do usuário(resultado 6,1)	**ENG.4.BP6: Garantir a consistência e rastreabilidade bilateral entre requisitos da transformação e requisitos do processo DDM**(resultado 6,1) - Deve ser possível rastrear quais os requisitos da transformação relacionados aos requisitos do processo DDM
Workproduct (WP)	a) Plano de versão (resultado 4,5)	a) Plano de versão (resultado 4,5)
	b) Registro de mudanças (resultado 7)	N/A
	c) Registro de rastreabilidade (resultado 1,6)	c) Registro de rastreabilidade (resultado 1,6)
	d) Relatório de análise (resultado 2, 7)	d) Relatório de análise (resultado 2, 7)
	e) Especificação das interfaces dos requisitos (resultado 1) relacionado as interfaces entre a arquitetura do Sistema e os requisitos de software	N/A
	f) Especificação dos requisitos de software (resultados 1,2,4,5,6)	f) Especificação dos requisitos de software (resultados 1,2,4,5,6)
	g) Critério de verificação (resultado 2)	g) Critério de verificação (resultado 2)

Figura A.2 ENG4 – Análise de requisitos de software

Modelo de referencia do processo ENG.5 da ISO/IEC-15504		Adaptação do modelo para o domínio de transformações
ID do processo Nome do processo	ENG.5 Projeto do software	ENG.5 Projeto da transformação
Propósito do processo	Provê um projeto para que o software possa ser implementado e testado de acordo com os requisitos.	Provê um projeto para que a transformação possa ser implementada e testada de acordo com os requisitos
Resultados esperados	1) Uma arquitetura é definida com os componentes do software de acordo com os requisitos.	1) Uma arquitetura é definida com os componentes da transformação de acordo com os requisitos.
	2) Os requisitos do software são alocados aos elementos do software.	2) Os requisitos são alocados aos elementos da transformação.
	3) Interfaces internas e externas são definidas para cada componente.	3) Interfaces internas e externas são definidas para cada componente.
	4) O comportamento dinâmico e os objetivos de consumo de recursos dos componentes do software são definidos.	4) O comportamento dinâmico dos componentes da transformação são definidos.
	5) Um projeto detalhado é desenvolvido descrevendo as unidades do software que devem ser implementadas e testadas.	5) Um projeto detalhado é desenvolvido descrevendo as unidades da transformação que devem ser implementadas e testadas.
	6) Consistência e rastreabilidade bilateral são estabelecidas entre os requisitos do software e o projeto da arquitetura	6) Consistência e rastreabilidade bilateral são estabelecidas entre os requisitos da transformação e o projeto da arquitetura
	7) Consistência e rastreabilidade bilateral são estabelecidas entre o projeto da arquitetura do software e o projeto detalhado.	7) Consistência e rastreabilidade bilateral são estabelecidas entre o projeto da arquitetura do software e o projeto detalhado.
Práticas Básicas (BP)	ENG.5.BP1: Desenvolver a arquitetura do software (resultado 1)	**ENG.5.BP1: Desenvolver a arquitetura da transformação** (resultado 1)
	ENG.5.BP2: Alocar os requisitos do software aos componentes da arquitetura (resultado 2)	**ENG.5.BP2: Alocar os requisitos da transformação aos componentes da arquitetura** (resultado 2)
	ENG.5.BP3: Definir interfaces (resultado 3)	**ENG.5.BP3: Definir interfaces** (resultado 3)
	ENG.5.BP4: Descrever o comportamento dinâmico (resultado 4)	**ENG.5.BP4: Descrever o comportamento dinâmico** (resultado 4)
	ENG.5.BP5: Definir os objetivos de consumo de recursos (memória, processador, etc.) (resultado 4)	N/A
	ENG.5.BP6: Desenvolver o projeto detalhado (resultado 5)	**ENG.5.BP6: Desenvolver o projeto detalhado** (resultado 5)
	ENG.5.BP7: Desenvolver critério de verificação. Desenvolver o critério de verificação para cada componente de acordo com o comportamento dinâmico, interface e consumo de recurso. (resultado 5)	**ENG.5.BP7: Desenvolver critério de verificação**.(resultado 5) - Definir critério de verificação considerando os atributos de qualidade relacionados a transformação de modelos (Understandability, Modifiability, Reusability, Modularity, Completeness, Consistency e Conciseness)
	ENG.5.BP8: Verificar o projeto do software. Garantir que o projeto está de acordo com os requisitos (resultado 4,5)	**ENG.5.BP8: Verificar o projeto do software**. Garantir que o projeto está de acordo com os requisitos (resultado 4,5)
	ENG.5.BP9: garantir a consistência e rastreabilidade bilateral entre a projeto de arquitetura do software e os requisitos do software (resultado 6)	**ENG.5.BP9: garantir a consistência e rastreabilidade bilateral entre a projeto de arquitetura do software e os requisitos do software** (resultado 6) - Garantir que os requisitos especificados no modelo independente de computação podem ser rastreados em relação ao modelo independente de plataforma
Workproduct (WP)	a) Projeto de arquitetura do software (resultado 1,2,3,4,6)	a) Projeto de arquitetura do software (resultado 1,2,3,4,6)
	b) Projeto detalhado do software (resultado 2,3,4,5,6)	b) Projeto detalhado do software (resultado 2,3,4,5,6)
	c) Registro de rastreabilidade (resultado 2,6,7)	c) Registro de rastreabilidade (resultado 2,6,7)

Figura A.3 ENG5 – Projeto do software

Modelo de referencia do processo: ENG.6 da ISO/IEC-15504		Adaptação do modelo para o domínio de transformações
ID do processo	ENG.6	ENG.6
Nome do processo	Construção do software	Construção da transformação
Propósito do processo	Produzir unidades de código verificáveis que reflitam o projeto do software.	Produzir unidades de código verificáveis que reflitam o projeto da transformação
Resultados esperados	1) É definida uma estratégia de verificação de unidade.	1) É definida uma estratégia de verificação de unidade.
	2) Unidades de software definidas no projeto do software são produzidas.	2) Unidades de software definidas no projeto do software são produzidas.
	3) Consistência e rastreabilidade bilateral são estabelecidas entre o projeto detalhado e as unidades de software. 4) Unidades de software são verificadas de acordo com a estratégia de verificação de unidade.	3) Consistência e rastreabilidade bilateral são estabelecidas entre o projeto detalhado e as unidades de software. 4) Unidades de software são verificadas de acordo com a estratégia de verificação de unidade.
	5) Os resultados da verificação são registrados.	5) Os resultados da verificação são registrados.
Práticas Básicas (BP)	ENG.6.BP1: Definir uma estratégia de verificação de unidade. A estratégia deve definir como a qualidade desejada será alcançada com as técnicas disponíveis (resultado 1)	**ENG.6.BP1: Definir uma estratégia de verificação de unidade**. A estratégia deve definir como a qualidade desejada será alcançada com as técnicas disponíveis (resultado 1) - Definir um plano de teste para a transformação + Considerar aspectos de qualidade externa na transformação, analisar os modelos fonte e alvo envolvidos na transformação (ex. correção sintática e semântica) + Analisar os critérios de qualidade interna de transformação (Understandability, Modifiability, Reusability, Modularity, Completeness, Consistency e Conciseness)
	ENG.6.BP2: Desenvolver um critério de verificação de unidade. Critério para verificar se cada unidade satisfaz o projeto e os requisitos na estratégia de verificação (resultado 2)	**ENG.6.BP2: Desenvolver um critério de verificação de unidade**. Definir os critérios que serão utilizados para verificar se cada unidade satisfaz o projeto e os requisitos na estratégia de verificação (resultado 2) - Para a qualidade externa da transformação é importante a definição de propriedades que devem ser checadas nos modelos fonte e alvo - Para qualidade interna da transformação é importante o uso de métricas de avaliação da qualidade do código da transformação.
	ENG.6.BP3: Desenvolver unidades de software. Desenvolver e documentar a representação executável de cada unidade. (resultado 3)	**ENG.6.BP3: Gerar o código da transformação (complementar se necessário)**. (resultado 3)
	ENG.6.BP4: Verificar unidade de software. Verificar em relação ao projeto, com base na estratégia de verificação (resultado 4)	**ENG.6.BP4: Verificar unidade de software**. Verificar em relação ao projeto, com base na estratégia de verificação (resultado 4)
	ENG.6.BP5: Gravar resultados da verificação de unidade. Documentar os resultados (resultado 5)	**ENG.6.BP5: Gravar resultados da verificação de unidade**. Documentar os resultados (resultado 5)
	ENG.6.BP6: Garantir consistência e rastreabilidade bilateral do projeto detalhado e unidades (resultado 3)	**ENG.6.BP6: Garantir consistência e rastreabilidade bilateral do projeto detalhado e unidades** (resultado 3) - Garantir que o modelo específico de plataforma possa ser rastreado em relação ao modelo independente de plataforma - Garantir que o o código possa ser rastreado em relação ao modelo específico de plataforma.
	ENG.6.BP7: Garantir consistência e rastreabilidade bilateral dos requisitos e unidades (somente se não for possível garantir com o BP6) (resultado 3) (Não precisa)	NA
	ENG.6.BP8: Garantir consistência e rastreabilidade bilateral dos casos de teste e unidades (resultado 3)	**ENG.6.BP8: Garantir consistência e rastreabilidade bilateral dos casos de teste e unidades** (resultado 3)
Workproduct (WP)	a) Plano de teste (resultado 1)	a) Plano de teste (resultado 1)
	b) Especificação de teste (resultado 1,4)	b) Especificação de teste (resultado 1,4)
	c) Resultado do teste (resultado 4,5)	c) Resultado do teste (resultado 4,5)
	d) Unidade de software (resultado 2)	d) Unidade de software (resultado 2)
	e) Registro de rastreabilidade (resultado 3)	e) Registro de rastreabilidade (resultado 3)

Figura A.4 ENG6 – Construção do software

Apêndice

B

QUESTIONÁRIOS DE COLETA DE DADOS PARA A AVALIAÇÃO DO FRAMEWORK

Este apêndice apresenta os questionários respondidos pelos participantes durante a avaliação do framework.

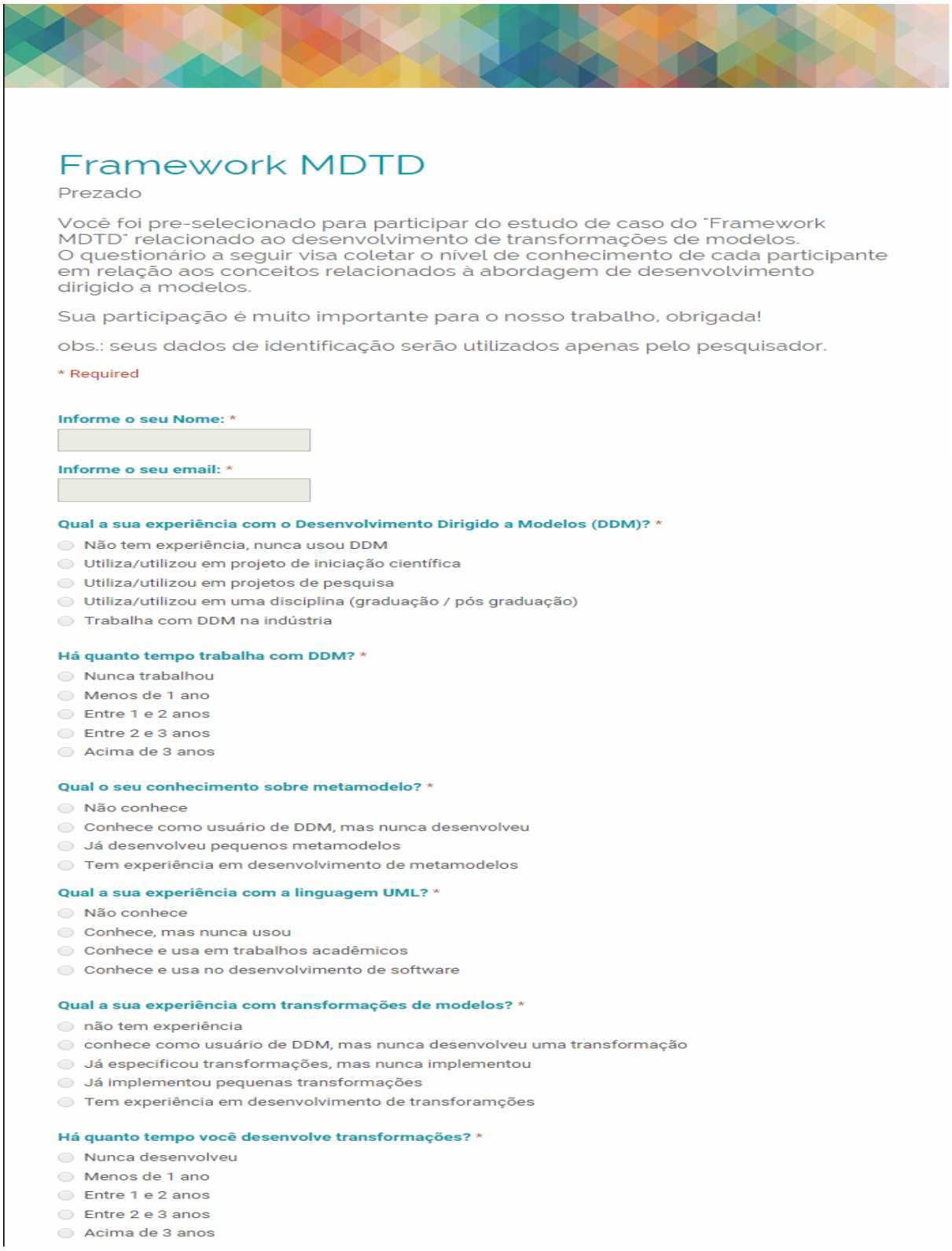
Questionário para avaliação do conhecimento do participante

O questionário apresentado nas Figuras B.1 e B.2 foi aplicado aos participantes do estudo de caso e do experimento controlado com o objetivo de identificar o conhecimento destes sobre assuntos tais como DDM, UML, metamodelagem e transformação de modelos.

Questionários para avaliação do processo MDTDproc

Os questionários apresentados objetivam coletar evidências para avaliação dos indicadores de práticas básicas, práticas genéricas e recursos genéricos. Cada um dos questionários contém uma seção inicial de identificação do participante e em seguida uma lista de perguntas que devem ser respondidas de acodo com a seguinte escala proposta pela norma ISO/IEC-15504: N, não alcançado; P, parcialmente alcançado; L, largamente alcançado; e F, completamente alcançada. Adicionalmente a última coluna do questionário registra qual o indicador que está sendo avaliado naquela pergunta específica. Esta coluna é de uso exclusivo do pesquisador, não foi apresentada para os participantes do estudo. Estes questionários foram disponibilizados para os participantes em formato eletrônico. Os seguintes questionários foram elaborados:

- Questionário aplicado após a fase TCP e TRM do processo MDTDproc (Figura B.3)
- Questionário aplicado após a fase TDM do processo MDTDproc (Figura B.4)
- Questionário aplicado após a fase TSM e códico do processo MDTDproc (Figura B.5)



Framework MDTD

Prezado

Você foi pre-selecionado para participar do estudo de caso do "Framework MDTD" relacionado ao desenvolvimento de transformações de modelos.
O questionário a seguir visa coletar o nível de conhecimento de cada participante em relação aos conceitos relacionados à abordagem de desenvolvimento dirigido a modelos.

Sua participação é muito importante para o nosso trabalho, obrigada!

obs.: seus dados de identificação serão utilizados apenas pelo pesquisador.

* Required

Informe o seu Nome: *

Informe o seu email: *

Qual a sua experiência com o Desenvolvimento Dirigido a Modelos (DDM)? *

- Não tem experiência, nunca usou DDM
- Utiliza/utilizou em projeto de iniciação científica
- Utiliza/utilizou em projetos de pesquisa
- Utiliza/utilizou em uma disciplina (graduação / pós graduação)
- Trabalha com DDM na indústria

Há quanto tempo trabalha com DDM? *

- Nunca trabalhou
- Menos de 1 ano
- Entre 1 e 2 anos
- Entre 2 e 3 anos
- Acima de 3 anos

Qual o seu conhecimento sobre metamodelo? *

- Não conhece
- Conhece como usuário de DDM, mas nunca desenvolveu
- Já desenvolveu pequenos metamodelos
- Tem experiência em desenvolvimento de metamodelos

Qual a sua experiência com a linguagem UML? *

- Não conhece
- Conhece, mas nunca usou
- Conhece e usa em trabalhos acadêmicos
- Conhece e usa no desenvolvimento de software

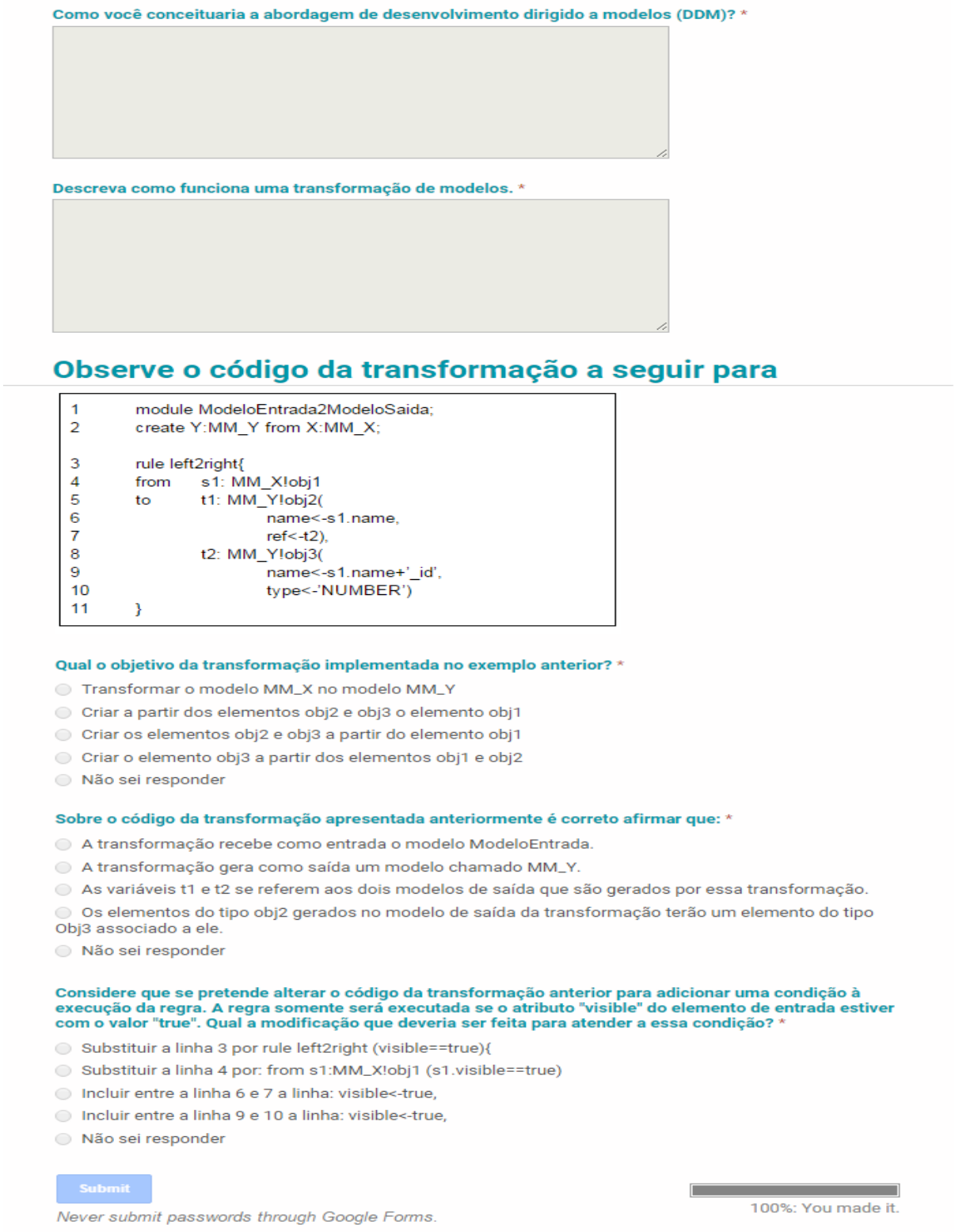
Qual a sua experiência com transformações de modelos? *

- não tem experiência
- conhece como usuário de DDM, mas nunca desenvolveu uma transformação
- Já especificou transformações, mas nunca implementou
- Já implementou pequenas transformações
- Tem experiência em desenvolvimento de transforamções

Há quanto tempo você desenvolve transformações? *

- Nunca desenvolveu
- Menos de 1 ano
- Entre 1 e 2 anos
- Entre 2 e 3 anos
- Acima de 3 anos

Figura B.1 Questionário aplicado antes do estudo e experimento - parte 1



Como você conceituaria a abordagem de desenvolvimento dirigido a modelos (DDM)? *

Descreva como funciona uma transformação de modelos. *

Observe o código da transformação a seguir para

```
1       module ModeloEntrada2ModeloSaida;
2       create Y:MM_Y from X:MM_X;

3       rule left2right{
4       from    s1: MM_X!obj1
5       to      t1: MM_Y!obj2(
6                       name<-s1.name,
7                       ref<-t2),
8               t2: MM_Y!obj3(
9                       name<-s1.name+'_id',
10                      type<-'NUMBER')
11      }
```

Qual o objetivo da transformação implementada no exemplo anterior? *

- Transformar o modelo MM_X no modelo MM_Y
- Criar a partir dos elementos obj2 e obj3 o elemento obj1
- Criar os elementos obj2 e obj3 a partir do elemento obj1
- Criar o elemento obj3 a partir dos elementos obj1 e obj2
- Não sei responder

Sobre o código da transformação apresentada anteriormente é correto afirmar que: *

- A transformação recebe como entrada o modelo ModeloEntrada.
- A transformação gera como saída um modelo chamado MM_Y.
- As variáveis t1 e t2 se referem aos dois modelos de saída que são gerados por essa transformação.
- Os elementos do tipo obj2 gerados no modelo de saída da transformação terão um elemento do tipo Obj3 associado a ele.
- Não sei responder

Considere que se pretende alterar o código da transformação anterior para adicionar uma condição à execução da regra. A regra somente será executada se o atributo "visible" do elemento de entrada estiver com o valor "true". Qual a modificação que deveria ser feita para atender a essa condição? *

- Substituir a linha 3 por rule left2right (visible==true){
- Substituir a linha 4 por: from s1:MM_X!obj1 (s1.visible==true)
- Incluir entre a linha 6 e 7 a linha: visible<-true,
- Incluir entre a linha 9 e 10 a linha: visible<-true,
- Não sei responder

Submit

100%: You made it.

Never submit passwords through Google Forms.

Figura B.2 Questionário aplicado antes do estudo e experimento - parte 2

Questionário para coleta de dados do processo de especificação e análise dos requisitos da transformação					
Estudo:					
Participante:					
Transformação:					
Registre o tempo de início e término da sessão Inicio: ___:____ Termino ____:____					
Pergunta	N	P	L	F	Indicador
Q1-O conjunto de atividades da primeira fase do processo é suficiente para identificar os requisitos da transformação?					ENG2.BP1
Q2-A definição dos objetivos de cada atividade está clara, ou seja, possibilita que você entenda qual o propósito da atividade?					GP
Q3-Os passos que definem como a atividade deve ser executada foram suficientes para te guiar na construção do artefato de saída da atividade?					GP
Q4-Você teve dificuldade para executar os passos definidos nas atividades, ou seja, a explicação de como executar cada passo está satisfatória?					GP
Q5-Os documentos de apoio (ex. exemplo de transformação) ajuda na execução das atividades?					GP
Q6-Os artefatos de entrada e saída de cada atividade estão definidos? Para cada atividade são identificados os artefatos que devem ser utilizados como entrada e quais os artefatos que devem ser produzidos?					GP
Q7-Você conseguiu identificar os requisitos do domínio?					ENG1.BP1
Q8-Você conseguiu identificar os requisitos da transformação?					ENG2.BP1
Q9-Você conseguiu identificar quais os metamodelos de entrada e saída da transformação?					ENG2.BP1
Q10-Foram identificados os riscos para o desenvolvimento da transformação / o processo conduz a identificação dos riscos no desenvolvimento da transformação?					ENG2.BP2
Q11-Foram definidas propriedades para verificação da transformação / existe atividade para especificação de propriedades para verificação da correção da transformação?					ENG2.BP2
Q12-Os requisitos da transformação foram categorizados, priorizados e distribuídos em versões?					ENG2.BP4
Q13-Você identifica atividades de controle de mudanças e versões? Quando uma nova versão da transformação é desenvolvida, existe um controle do que foi adicionado / modificado na nova versão?					ENG2.BP5
Q14-Existe rastreabilidade entre os requisitos do cliente e os requisitos da transformação?					ENG2.BP6
Q15-Os papeis de cada participante do processo estavam definidos? Foi possível identificar quais as tarefas do processo eram de sua responsabilidade?					GR
Q16-Foram utilizadas ferramentas de modelagem customizadas para o perfil *MTPspec*?					GR
Q17-Foram utilizados engenhos de transformação para automatizar a geração de artefatos de saída?					GR
Q18-O modelo de requisitos gerado ao final da fase apresenta claramente os requisitos e metamodelos envolvidos na transformação? O modelo pode ser compreendido?					GP

Figura B.3 Questionário aplicado após as fases TCP e TRM

Questionário para coleta de dados do processo de projeto da transformação					
Replicação:					
Participante: Transformação:					
Registre o tempo de início e término da sessão Inicio: ___:____ Termino ____:____					
Pergunta	N	P	L	F	Indicador
Q1 - O modelo inicial de projeto foi gerado satisfatoriamente pelo processo a partir do modelo de requisitos, ou seja, os dados gerados são suficientes para você iniciar o projeto?					GR
Q2 - A definição dos objetivos de cada atividade está clara, ou seja, possibilita que você entenda qual o propósito da atividade? Q3 - As atividades definidas para o projeto da transformação foram suficientes para te conduzir na definição da arquitetura da transformação? As atividades definidas guiam você na definição dos componentes que serão desenvolvidos?					GP GP
Q4 - As atividades especificadas foram suficientes para conduzir a identificação dos relacionamentos entre os elementos dos metamodelos fonte e alvo?					GP
Q5 - Os artefatos de entrada e saída de cada atividade estão definidos? Para cada atividade são identificados os artefatos que devem ser utilizados como base e quais os artefatos que devem ser produzidos?					GP
Q6 - Os passos (de uma atividade) ajudam na construção dos artefatos de saída? Apresentam o passo a passo de como construir o artefato de saída de cada atividades? Ex. como construir o diagrama de classes com o mapeamento dos relacionamentos entre metamodelos fonte e alvo.					GP
Q7 - A arquitetura da transformação foi definida? Foi construido um diagrama de componentes ilustrando a transformação como um componente (ou composta por vários componentes) e suas interfaces (metamodelos de entrada e saida)?					BP1/BP3
Q8 - Os relacionamentos entre os elementos dos metamodelos fonte e alvo foram identificados?					BP6
Q9 - Os relacionamentos identificados contemplam todos os requisitos?					BP2
Q10 - O comportamento de cada relacionamento foi detalhado ilustrando a inicialização dos atributos do elemento gerado no modelo alvo?					BP4, BP6
Q11 - Propriedades para verificação da transformação foram formalizadas em uma linguagem apropriada/ existe atividade para definição de propriedades para verificação da correção da transformação?					BP7
Q12 - Foram definidos os casos de teste da transformação / existe atividade relacionada a definição de casos de teste da transformação?					BP7
Q13 - Existe rastreabilidaentre os requisitos às Relações?					BP8/BP9
Q14 - Os papeis de cada participante do processo estão definidos? Foi possivel identificar quais as tarefas do processo eram de sua responsabilidade? Q15 - Foram utilizadas ferramentas de modelagem customizadas para o perfil *MTPdesign*?					GR GR
Q16 - Foram utilizados engenhos de transformação para geração de artefatos?					GR
Q17 - O modelo de projeto gerado ao final da fase apresenta claramente as regras que compõe a transformação especificada? O modelo pode ser compreendido?					GP

Figura B.4 Questionário aplicado após a fase TDM

Questionário para coleta de dados do processo de construção da transformação					
Replicação:					
Participante: Transformação:					
Registre o tempo de início e término da sessão Inicio: ___:____ Termino ____:____					
Pergunta	N	P	L	F	Indicador
Q1 - As atividades da fase de construção estão claramente definidas? Ex. os passos para avaliação do código gerado e realização dos testes estão definidos?					GP
Q2 - O plano de teste foi montado com a especificação dos testes que devem ser executados?					BP1/BP2
Q3 - Os casos de teste foram gerados?					BP1
Q4 - O código fonte da transformação foi gerado? Foi criado o cabeçalho da transformação, com os modelos de entrada /saída e o cabeçalho das regras que compõe a transformação?					BP3
Q5 - O código fonte de cada regra foi gerado? Foi gerado o código referente a identificação dos elementos de entrada, identificação dos elementos de saída e a inicialização dos atributos dos elementos de saída?					BP3
Q6 - As propriedades foram verificadas em um verificador de modelos Existe atividade referente a verificação de propriedades?					BP1,P4
Q7 - Os testes foram executados?					BP5
Q8 - Foi gerada uma matriz de rastreabilidade entre o código e as regras especificadas?					BP6
Q9 - Foi gerada uma matriz de rastreabilidade entre o código e os casos de teste?					BP7

Figura B.5 Questionário aplicado após a fases de TSM e Códificação

Apêndice

C

BAREMA DO PESQUISADOR

Este apêndice contém os baremas utilizados pelo pesquisador para analisar os artefatos produzidos pelos participantes do estudo de caso. Os seguintes baremas foram definidos: lista de requisitos do domínio, lista de requisitos funcionais, detalhamento do diagrama de requisitos (diagrama de classes), projeto de alto nível, projeto de baixo nível, projeto específico de plataforma e código.

A Figura C.1 mostra o barema utilizado para analisar a lista de requisitos de domínio produzida pelos participantes. É analisado se os participantes identificaram os requisitos necessários. Para isso utilizamos a seguinte escala: (C) definiram corretamente o requisito, (N) não definiram o requisito ou (R) definiram o requisito de forma errado. A primeira coluna da tabela indica qual foi o item avaliado e as demais colunas indicam quem foi o participante avaliado.

A Figura C.2 mostra o barema utilizado para registrar dados coletados dos diagramas de casos de uso com os requisitos funcionais da transformação, definidos por cada praticante do estudo de caso. É analisado se o participante identificou o requisitos necessários e se o diagrama foi corretamente definido (ex. se os estereótipos foram corretamente aplicados). A avaliação utiliza a seguinte escala: (S) Sim, (N) Não, (P) Parcialmente. A primeira coluna da tabela indica qual foi o item avaliado e as demais colunas indicam quem foi o participante avaliado. Todas os baremas, a partir deste, possuem uma numeração (ex. Q1, Q2, etc.) que é utilizada nos scripts do software R para identificar cada item utilizado nas análises.

A Figura C.3 apresenta o barema utilizado para analisar o diagrama de classes com os requisitos construídos na primeira fase do processo de desenvolvimento da transformação. A primeira coluna indica qual foi o item avaliado e as demais colunas indicam quem foi o participante avaliado.

A Figura C.4 apresenta o barema utilizado para analisar o diagrama de classes com o projeto de alto nível da transformação. É analisado se os participantes definiram todos os elementos do diagrama de acordo com a escala: (S) Sim, (N) Não, (P) Parcialmente. A primeira coluna indica qual foi o item avaliado e as demais colunas indicam quem foi o participante avaliado.

Escala: (C) Definiu Corretamente, (N) Não Definiu (E) Definiu Errado o requisito

Pesquisador Requisitos de domínio identificados	Parcipantes P1	P2	...
A transformação deve gerar um sistema a partir do negócio modelado A transformação deve gerar três camadas (model, view e controller) a partir do negócio modelado			
A transformação deve gerar elemento de interface, elemento de controle, elemento de regra de negocio e elemento de persistência a partir de entidade de negócio			
Elemento de interface deve pertencer a camada de interface			
Elemento de controle deve pertencer a camada de controle			
Elemento de regra de negócio e persistência deve pertencer a camada model Mapear os dados de cada entidade em tributos.			
Os atributos deverão pertencer ao elemento regra de negócio.			

Figura C.1 Barema para avaliar a lista de requisitos de domínio

A Figura C.5 mostra o barema utilizado para analisar o diagrama de classes com o projeto de alto baixo da transformação. É analisado se os participantes definiram todos os elementos do diagrama de acordo com a escala: (S) Sim, (N) Não, (P) Parcialmente. A primeira coluna indica qual foi o item avaliado e as demais colunas indicam quem foi o participante avaliado.

A Figura C.6 apresenta o barema utilizado para analisar o diagrama de classes com o projeto específico de plataforma. É analisado se os participantes definiram todos os elementos do diagrama de acordo com a escala: (S) Sim, (N) Não, (P) Parcialmente. A primeira coluna da tabela indica qual foi o item avaliado e as demais colunas indicam quem foi o participante avaliado.

A Figura C.7 mostra o barema utilizado para analisar o código gerado para a transformação. É analisado se os participantes definiram todos os elementos do diagrama de acordo com a escala: (S) Sim, (N) Não, (P) Parcialmente. A primeira coluna da tabela indica qual foi o item avaliado e as demais colunas indicam quem foi o participante avaliado.

Artefato: Requisitos Funcionais - diagrama de casos de uso
Escala: (S) Sim, (N) Não (E) Parcialmente

Pesquisador	Parcipantes		
Requisitos funcionais e outros itens analisados	P1	P2	...
Q1 - Definiu o nome da transformação (ator)			
Q2 - Mapear Sistema Q3 - Mapear Entidade Negocio			
Q4 - Mapear Dados			
Q5 - Criar View (comp)			
Q6 - Criar Controller (comp)			
Q7 - Criar Model (comp) Q8 - Mapear Interface (ref)			
Q9 - Mapear Controle (ref)			
Q10 - Mapear RN (ref)			
Q11 - Mapear Persistencia (ref)			
Q12 - Definiu restrições?			
Q13 - Definiu os estereotipos corretamente?			

Figura C.2 Barema para avaliar o diagrama de casos de uso de requisitos

Artefato: Requisitos Funcionais - digrama de classes

Pesquisador	Parcipantes		
Itens analisados	P1	P2	...
Q14 - Definiu o objetivo da transformação?			
Q15 - Definiu o objetivo de cada requisitos?			
Q16 - Definiu o metamodelo source?			
Q17 - Definiu o metamodelo target?			
Q18 - Definiu qual o arquivo Ecore que contem os metamodelos source e target?			
Q19 - Especificou a versão			
Q20 - Especificou o critério de verificação? Q21 - Especificou a prioridade?			
Q22 - Especificou a rastreabilidade com o requisito de domínio?			

Figura C.3 Barema para avaliar o diagrama de classes de requisitos

Artefato: Projeto de alto nível - diagrama de classes

Pesquisador Itens analisados	Parcipantes P1	P2	...
Q1 - Colocou os elementos do metamodelo source no pacote?			
Q2 - Associou os elementos do metamodelo source a classe que representa o metamodelo? Q3 - Estereotipou a associação entre metamodelo source e seus elementos?			
Q4 - Colocou os elementos do metamodelo target no pacote?			
Q5 - Associou os elementos do metamodelo target a classe que representa o metamodelo?			
Q6 - Estereotipou a associação entre metamodelo target e seus elementos?			
Q7 - Nomeou os elementos do metamodelo com o mesmo nome do que foi especificado no metamodelo (ecore)?			
Q8 - Definiu a Relação negocio para sistema e camadas			
Q9 - Definiu a Relação entre entidade de negocio e os diversos elementos?			
Q10 - Definiu a Relação entre dado e atributo?			
Q11 - Definiu os estereótipos de entrada da relação?			
Q12 - Definiu os estereótipos de saída da relação?			

Figura C.4 Barema para avaliar o projeto de alto nível

Artefato: Projeto de Baixo nível - diagrama de classes

Pesquisador	Parcipantes		
Itens analisados	P1	P2	...
Q13 - Criou a regra negocio para sistema e camadas			
Q14 - Criou a regra entidade de negocio para os diversos elementos?			
Q15 - Criou a regra dado para atributo?			
Q16 - Especificou a conFiguração do nome do sistema?			
Q17 - Especificou a conFiguração do nome de cada camada?			
Q18 - Especificou a conFiguração do sistema ao qual a camada está inserido?			
Q19 - Especificou a conFiguração do nome de cada elemento?			
Q20 - Especificou a camada em que cada elemento deve estar?			
Q21 - Especificou a conFiguração do nome do atributo?			
Q22 - Especificou a conFiguração do tipo do atributo?			
Q23 - Especificou a conFiguração do elemento ao qual o atributo está associado?			
24 - Definiu o estereótipo das novas conFigurações criadas?			
25 - Definiu a associação entre TERule e suas conFigurações?			
26 - As expressões foram corretamente definidas (sintaxe)?			

Figura C.5 Barema para avaliar o projeto de baixo nível

Artefato: Modelo específico de plataforma - digrama de classe

Pesquisador	Parcipantes		
Itens analisados	P1	P2	...
O modelo ATL foi gerado?			
O modelo ATL foi alterado?			

Figura C.6 Barema para avaliar o projeto específico de plataforma

Artefato: Código

Pesquisador Itens analisados	Parcipantes P1	P2	...
Q1 - O código foi gerado?			
Q2 - O cabeçalho da transformação foi criado com o nome do módulo?			
Q3 - Os modelos e metamodelos source e target foram definidos?			
Q4 - As regras foram criadas conforme especificado? Foi criada uma regra para cada <<Rule>> definida?			
Q5 - Os elementos de entrada da regra foram gerados?			
Q6 - Os elementos de saída da regra foram gerados conforme especificado?			
Q7 - A inicialização dos elementos de saída da regra foi gerada conforme especificado?			
Q8 - Foi gerado comentário da transformação de acordo com a descrição inserida na especificação?			
Q9 - Foi gerado comentário de cada regra de acordo com a descrição inserida na especificação dos requisitos que deram origem a regra?			
Q10 - Existe erro de sintaxe no código gerado?			
Q11 - Total de linhas geradas			
Q12 - Linhas incluidas no codigo			
Q13 - Linhas alteradas no código			
Q14 - A transformação foi executada?			
Q15 - arquivo de saída foi corretamente gerado? (Conforme o modelo)			

Figura C.7 Barema para avaliar o código gerado para a transformação

Apêndice

D

CENÁRIO DA AVALIAÇÃO DO FRAMEWORK

Este apêndice apresenta o cenário utilizado para realização do estudo de caso e do experimento controlado de avaliação da abordagem proposta nesta tese.

O estudo consiste na construção de uma transformação modelo-a-modelo que transforma um modelo conceitual de classes de um sistema de informação, em conformidade com o metamodelo *MMConceitual*, em um modelo de projeto na arquitetura MVC, em conformidade o metamodelo *MMProjeto*.

Os seguintes itens são especificados: (i) metamodelo conceitual, chamado de *MMConceitual* que representa o metamodelo fonte da transformação; (ii) exemplo de modelo, instância de *MMConceitual*, que especifica um sistema de vendas de uma loja; (iii) metamodelo de projeto , chamado de *MMProjeto* que representa o metamodelo alvo da transformação; (iv) exemplo de modelo, instância de *MMProjeto*, para o sistema de vendas de uma loja; (v) atividade a ser desenvolvida no estudo de caso.

A Figura D.1 ilustra o metamodelo *MMConceitual* em formato visual e em formato ECore. Neste um *Negocio* possui um *nome* e é composto de *EntidadesNegocio*. Cada uma dessas entidades também possui um *nome* e podem ser compostas de *Dado*. Um *Dado* contém um *nome* e um *tipo*.

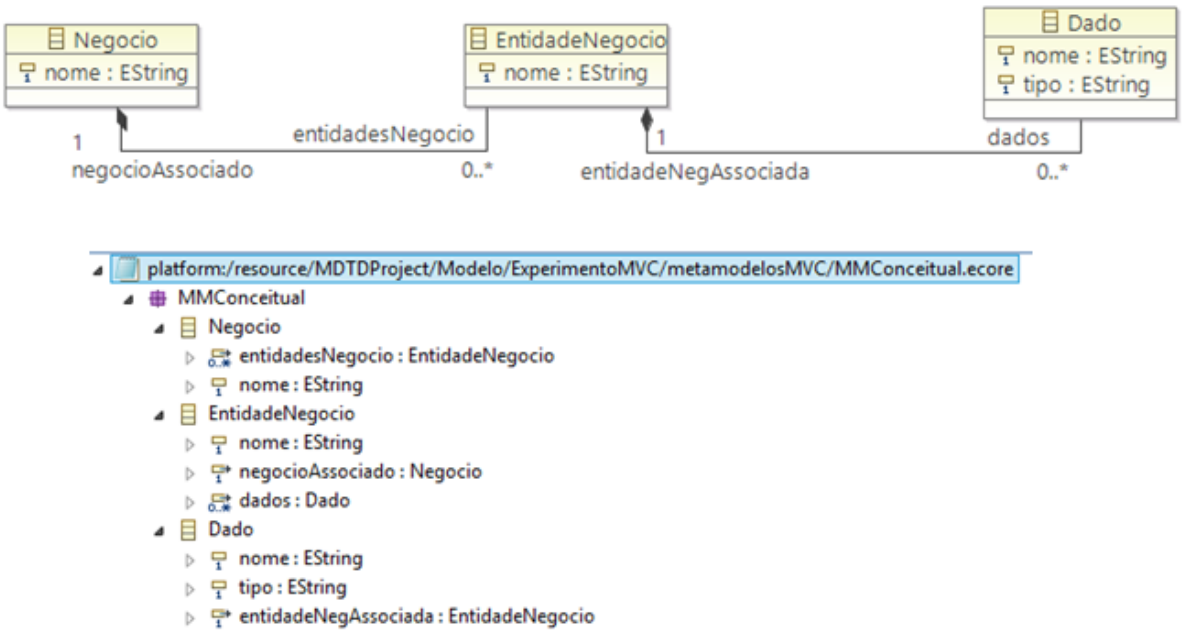


Figura D.1 Metamodelo *MMConceitual* (representação gráfica e em Ecore)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns="MMConceitual">
    <Negocio nome="Loja">
        <entidadesNegocio nome="Cliente">
          <dados nome="cpf" tipo="String"/>
          <dados nome="nome" tipo="String"/>
          <dados nome="email" tipo="String"/>
          <dados nome="telefone" tipo="String"/>
        </entidadesNegocio>
        <entidadesNegocio nome="Compra">
          <dados nome="notaFiscal" tipo="int"/>
          <dados nome="data" tipo="date"/>
          <dados nome="comprador" tipo="Cliente"/>
          <dados nome="total" tipo="float"/>
          <dados nome="produto" tipo="Produto"/>
        </entidadesNegocio>
        <entidadesNegocio nome="Produto">
          <dados nome="codico" tipo="int"/>
          <dados nome="nomeDoProduto" tipo="String"/>
          <dados nome="precoVenda" tipo="float"/>
        </entidadesNegocio>
    </Negocio>
</xmi:XMI>
```

Figura D.2 Modelo de um sistema de loja, instância de *MMConceitual*

A Figura D.2 mostra um exemplo de modelo, instância de *MMConceitual*, criado

para representar o sistema de um loja. Neste, o *Negocio Loja* possui as *EntidadeNegocio Cliente*, *Compra* e *Produto*. A entidade *Cliente* contém os *Dado cpf, nome, email* e *telefone*. Cada um destes dados possui um *nome* e um *tipo*. Por exemplo, o *cpf* é do *tipo String*.

A Figura D.3 ilustra o metamodelo *MMProjeto* no formto visual e Ecore. Neste um *Sistema* possui um *nome* e é composto de *Camada* (seguindo a arquitetura MVC) que podem ser de três tipos: *Model, View* ou *Controller*. Cada *Camada* possui um nome e um conjunto de *Elemento* que podem ser do tipo *EleInterface*, *EleControle*, *EleRegraNegocio* ou *ElePersistencia*. Um *Elemento ElePersistencia* é composto de *Atributo*. Um *Atributo* contém um *nome* e um *tipo*.

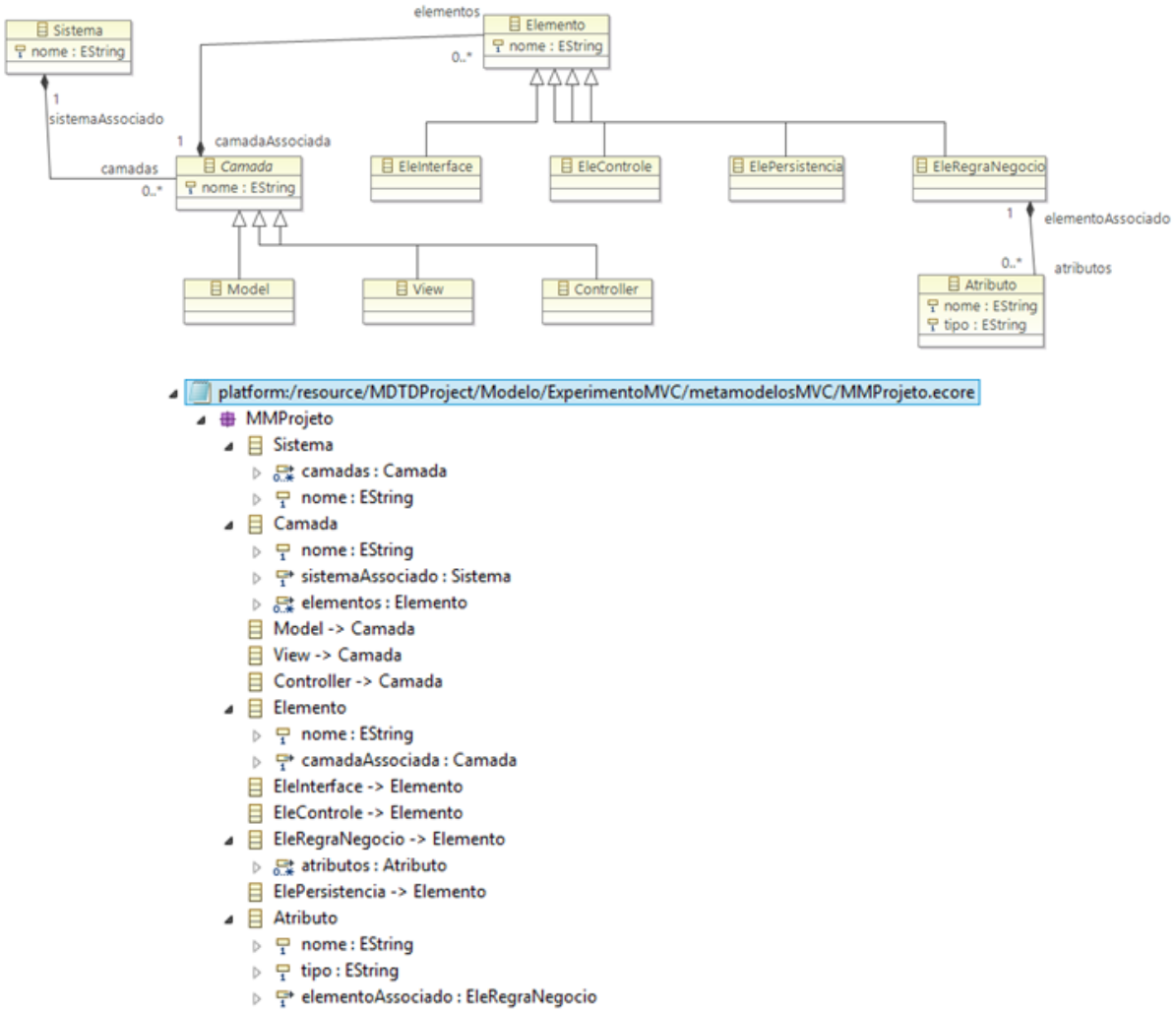


Figura D.3 Metamodelo *MMProjeto* (representação gráfica e em Ecore)

A Figura D.4 mostra um exemplo de modelo, instância do metamodelo *MMProjeto*, criado para representar o sistema de um loja.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="MMProjeto">
 <Sistema nome="Loja">
  <camadas xsi:type="View" nome="CamadaView_Loja">
   <elementos xsi:type="EleInterface" nome="VW_Cliente"/>
   <elementos xsi:type="EleInterface" nome="VW_Compra"/>
   <elementos xsi:type="EleInterface" nome="VW_Produto"/>
  </camadas>
  <camadas xsi:type="Controller" nome="CamadaController_Loja">
   <elementos xsi:type="EleControle" nome="CTR_Cliente"/>
   <elementos xsi:type="EleControle" nome="CTR_Compra"/>
   <elementos xsi:type="EleControle" nome="CTR_Produto"/>
  </camadas>
  <camadas xsi:type="Model" nome="CamadaModel_Loja">
   <elementos xsi:type="EleRegraNegocio" nome="RN_Cliente">
    <atributos nome="cpf" tipo="String"/>
    <atributos nome="nome" tipo="String"/>
    <atributos nome="email" tipo="String"/>
    <atributos nome="telefone" tipo="String"/>
   </elementos>
   <elementos xsi:type="EleRegraNegocio" nome="RN_Compra">
    <atributos nome="notaFiscal" tipo="int"/>
    <atributos nome="data" tipo="date"/>
    <atributos nome="comprador" tipo="Cliente"/>
    <atributos nome="total" tipo="float"/>
    <atributos nome="produto" tipo="Produto"/>
   </elementos>
   <elementos xsi:type="EleRegraNegocio" nome="RN_Produto">
    <atributos nome="codico" tipo="int"/>
    <atributos nome="nomeDoProduto" tipo="String"/>
    <atributos nome="precoVenda" tipo="float"/>
   </elementos>
   <elementos xsi:type="ElePersistencia" nome="ClienteDAO"/>
   <elementos xsi:type="ElePersistencia" nome="CompraDAO"/>
   <elementos xsi:type="ElePersistencia" nome="ProdutoDAO"/>
  </camadas>
 </Sistema>
```

Figura D.4 Modelo de um sistema de loja, instância de *MMProjeto*

Neste, o *Sistema Loja* possui três camadas: *CamadaView* do tipo *View*, *CamadaController* do tipo *Controller* e a *CamadaModel* do tipo *Model*. A *CamadaView* contem três elementos do tipo *Interface*: *VW_Cliente, VW_Compra* e *VW_Produto*.A *CamadaController* contém três elementos do tipo *Controle*: *ClienteCTR, CompraCTR* e *ProdutoCTR*. A *CamadaModel* contém três elementos do tipo *EleRegraNegocio* e três elementos do tipo *ElePersistencia*: *ClienteRN, CompraRN, ProdutoRN, ClienteDAO, CompraDAO* e *ProdutoDAO*. Os elementos do tipo *EleRegraNegocio* possuem atributos.

Com base nos modelos e metamodelos fornecidos construa uma transformação utilizando o framework MDTD que transforme um modelo (conforme o metamodelo descrito em i) em um modelo (conforme o metamodleo descrito em ii). Será utilizado o ambi-

ente Eclipse para o desenvolvimento da transformação. O processo MDTDproc deve ser seguido.

Apêndice

E

DOCUMENTOS PRODUZIDOS NA AVALIAÇÃO DO FRAMEWORK

Este apêndice apresenta os documentos produzidos durante o desenvolvimento da transformação proposta no cenário do estudo de caso e do experimento apresentados no capítulo 7. O cenário está descrito no apêndice D. Os artefatos aqui apresentados foram utilizados para apoiar o preenchimento dos baremas definidos (apêndice C) durante a avaliação dos artefatos produzidos pelos participantes. Os documentos estão organizados de acordo com as fases do processo MDTDproc as quais foram produzidos.

Documentos da fase TCP

De acordo com o processo MDTDproc, o primeiro artefato desenvolvido é a lista de requisitos do domínio que a transformação deverá automatizar. Essa lista é registrada em um documento textual de acordo com formato definido pelo processo MDTDproc (Figura E.1).

Nr	Descrição do requisito de domínio da transformação
RD01	Mapear um sistema a partir do modelo de negócio definido.
RD02	Criar a s camadas Model, View e Controller conforme definido no padrão MVC.
RD03	Mapear os elementos de cada camada (elementos de interface, controle, persistência e regras de negócio) para cada entidade de negócio criada.
RD04	Os elementos de negócio devem esta r localizados em suas respectivas camadas.
RD05	Mapear os dados como atributos.
RD06	Os atributos devem pertencer ao elemento regra de negócio.

Figura E.1 Requisitos do domínio da transformação

A partir da lista de requisitos do domínio é construído o diagrama de casos de uso com os requisitos da transformação (Figura E.2). Também é definida a lista de possíveis

riscos ao projeto. A Figura E.3 apresenta o *template* e alguns riscos que foram definidos para a transformação *Conceitual2Projeto* modelada para o cenário proposto no estudo de caso e no experimento controlado.

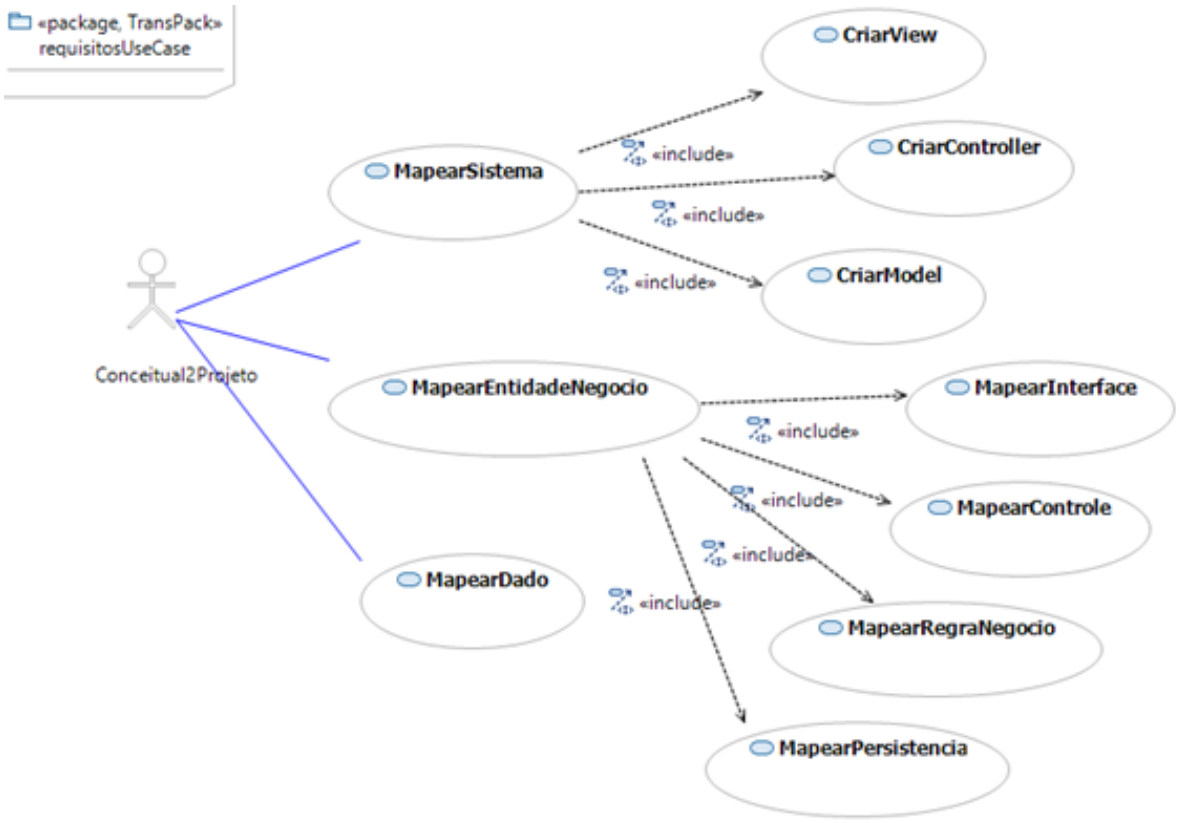


Figura E.2 Requisitos funcionais da transformação

Documentos da fase TRM

Na fase TRM os requisitos funcionais da transformação são detalhados utilizando o diagrama de classes gerado no final da fase anterior, ilustrado na Figura E.4. Adicionalmente são especificados os casos de teste para a transformação, conforme apresentado na Figura E.5.

Documentos da fase TDM

Inicialmente é modelada a arquitetura da transformação, neste caso composta por um único componente, ilustrada na Figura E.6.

Em seguida é especificado o projeto de alto nível ilustrado na Figura E.7. Para a transformação *Conceitual2Projeto* foram definidas três relações, especificadas no pacote central: *MapearSistema*, *MapearEntidadeNeg* e *MapearDado*. As duas primeiras são do tipo 1-N e a última do tipo 1-1. Por exemplo, a relação *MapearSistema* mapeia o elemento *Negocio* nos elementos *Sistema*, *Model*, *View* e *Controller*.

Finalmente, foi realizado o projeto de baixo nível da transformação ilustrado parcialmente na Figura E.8. Neste existem três regras, correspondentes às relações definidas no projeto de alto nível. Cada regra tem seus elementos de saída configurados. Por

Categoria: Habilidades Específicas			
Nr.	Descrição do Risco	Possibilidade	Forma de Mitigação
1	Falta de conhecimento do desenvolvedor na arquitetura MVC.	Media	Fornecer uma maior quantidade de modelos de exemplo desta arquitetura.
2	Conhecimento não aprofundado em metamodelagem.	Media	Explicar o metamodel o ao desenvolvedor antes de iniciar o projeto da transformação.
3	Conhecimento não aprofundado em UML	Baixa	Mostrar um exemplo de cada diagrama utilizado pelo processo *MDTDproc*.

Figura E.3 Especificação dos riscos

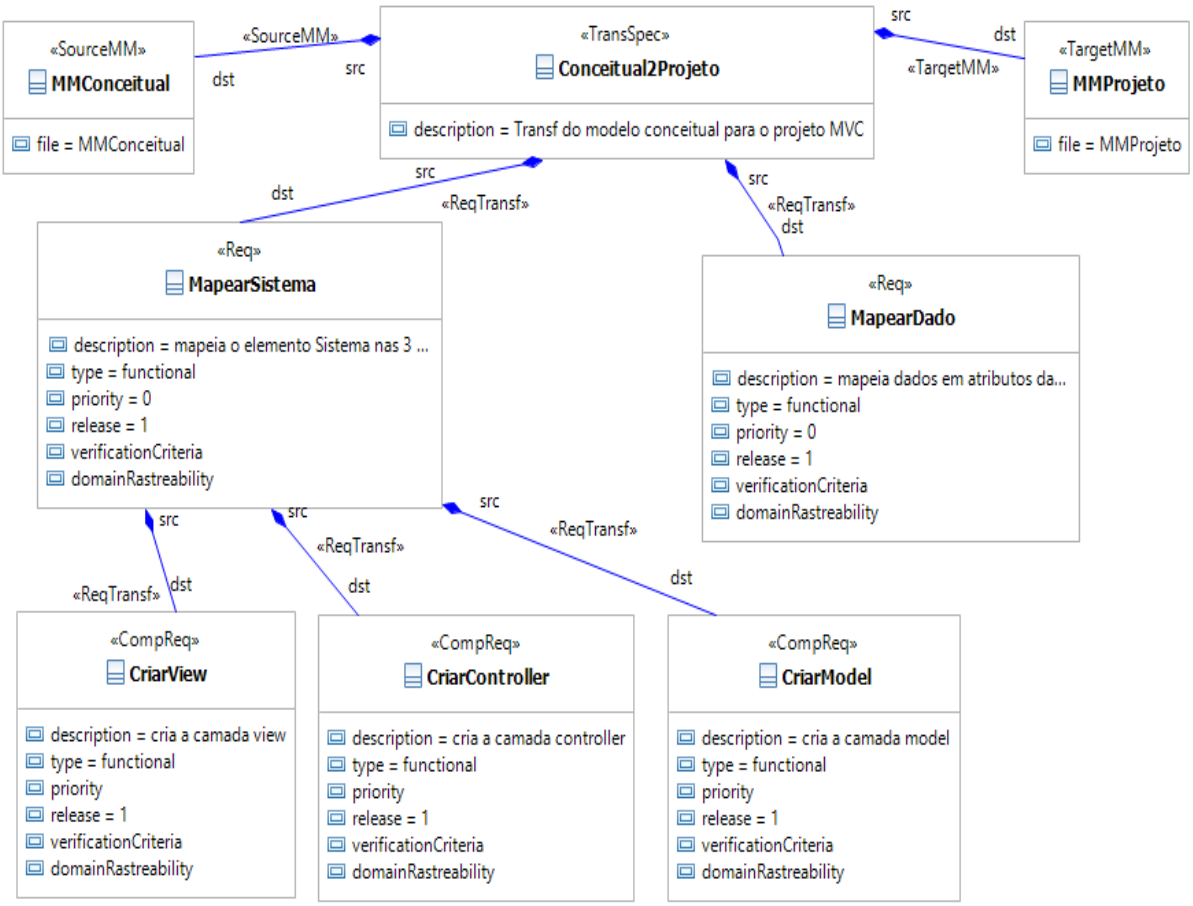


Figura E.4 Parte dos requisitos funcionais (classes)

Tipo do teste: Funcional				
Nr	O que testar	inModel	outModel	Resultado esperado
1	Modelo completo com Sistema, entidade de negócio e dado	Modelo exemplo Entrada 1	Modelo exemplo saída 1	Gerar o modelo co m as 3 camadas, cada uma com seus elementos, e os atributos associados as regras de negócio na camada model
2	Modelo contendo apenas o sistema, sem entidades de negocio	Modelo exemplo entrada 2	Modelo exemplo saída 2	Gerar o modelo com as 3 camadas vazias
3	Modelo contendo o sistema e as regras de negócio, sem dados	Modelo exemplo entrada 3	Modelo exemplo saída 3	Gerar modelo com as 3 camadas e os elementos associados a cada camadas, sem atributos

Figura E.5 Especificação de casos de teste

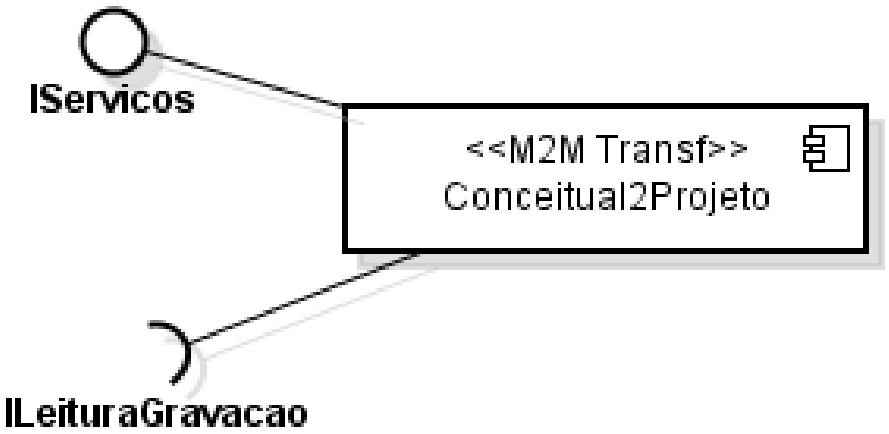


Figura E.6 Arquitetura da transformação

exemplo, para a regra *MapearDado* foram configurados duas propriedades do elemento de saída *Atributo*: a propriedade *nome*, a propriedade *tipo* e a propriedade *elementoAssociado*, esta última com a chamada a outra regra (*call MapearEntidadeNegocio*) que não está ilustrada na figura.

Documentos da fase TSM e Código

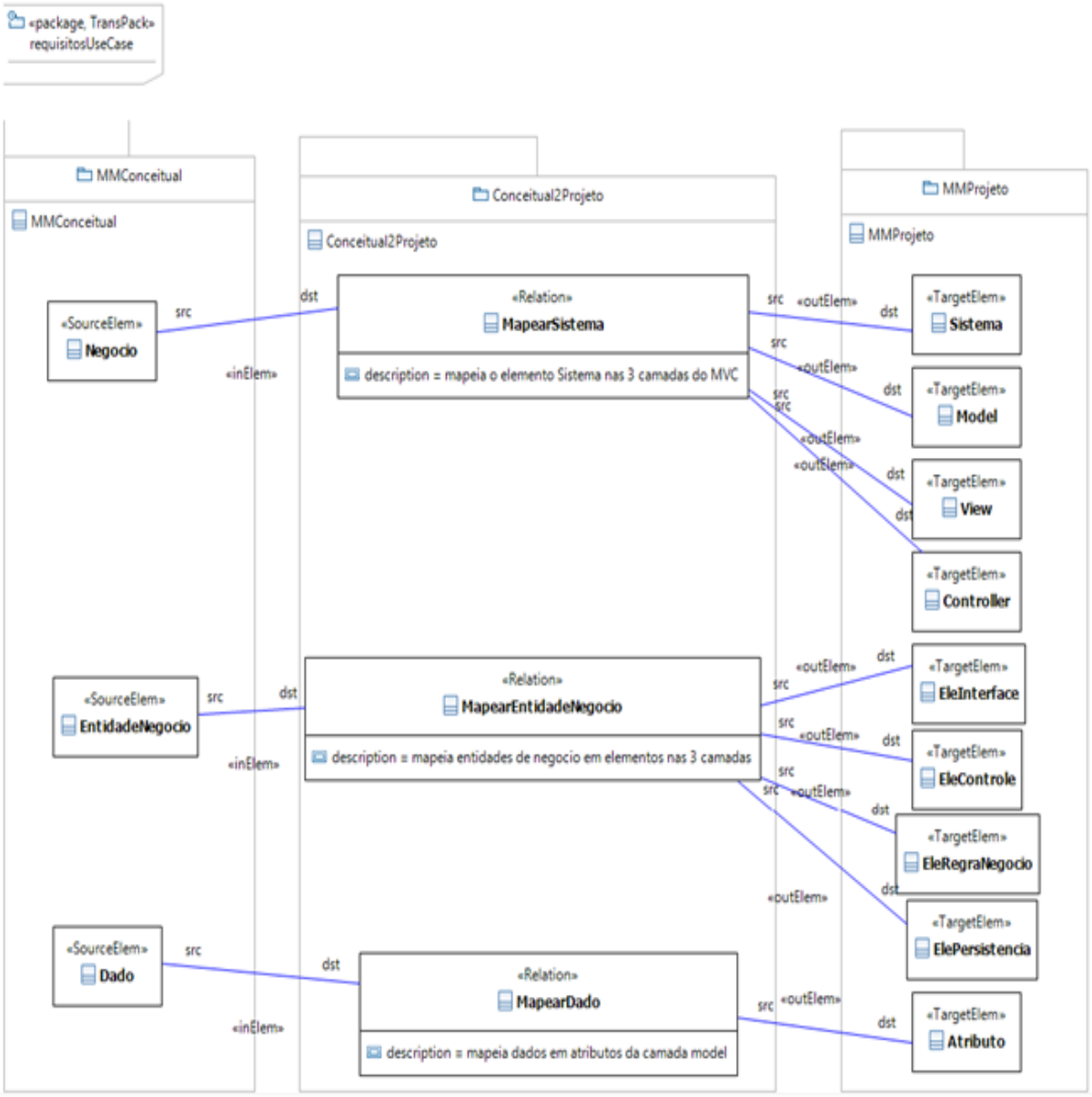


Figura E.7 Relacionamentos definidos para a transformação

A partir do projeto de baixo nível foi gerado o projeto específico (não ilustrado aqui devido ao grande tamanho da figura) e em seguida foi gerado o código da transformação, ilustrado na Figura E.9 em ATL.

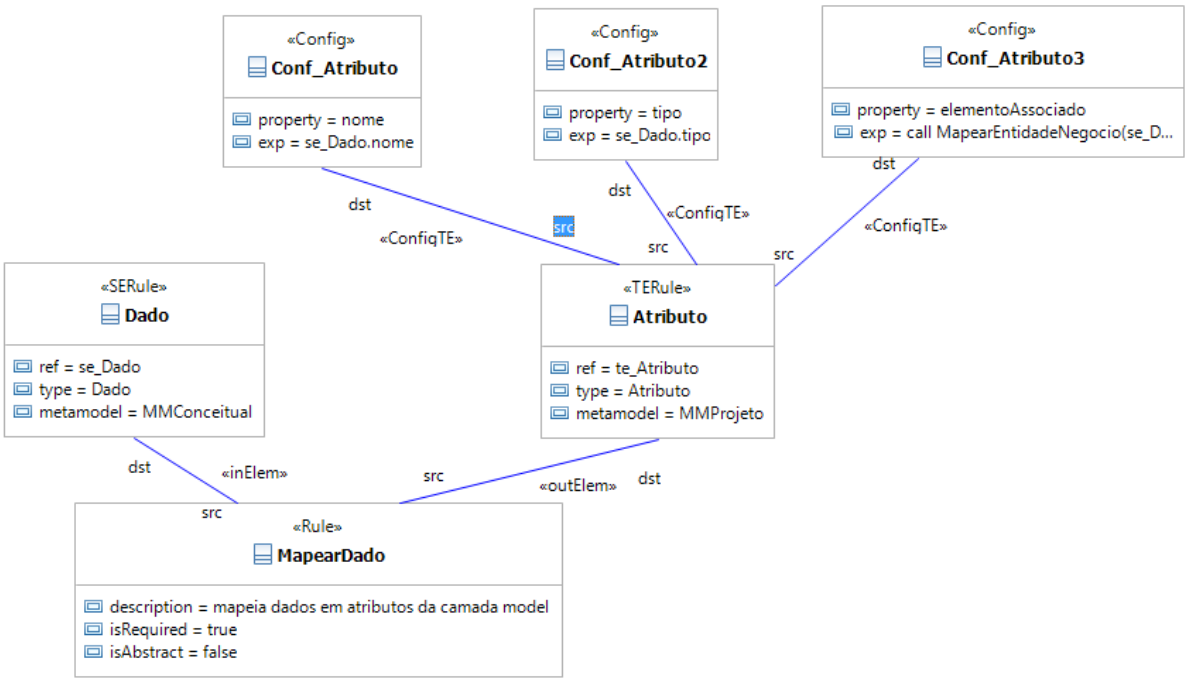


Figura E.8 Projeto de baixo nível da transformação

```
module MVC;
--*** modelos de saida e de entrada ***
   create OUT_MMProjeto:MMProjeto
   from IN_MMConceitual:MMConceitual;
uses string;
-- *** regras da transformação ***
rule MapearSistema{
 from
      se_Negocio:MMConceitual!Negocio
to
     te_View:MMProjeto!View
     (
      nome<-'CamadaView_'+se_Negocio.nome,
      elementos<-MMConceitual!EntidadeNegocio.allInstancesFrom('IN_MMConceitual')->collect(e|thisModule.resolveTemp(e,'te_EleInterface'))
),
     te_Model:MMProjeto!Model
     (
      elementos<-MMConceitual!EntidadeNegocio.allInstancesFrom('IN_MMConceitual')->collect(e|thisModule.resolveTemp(e,'te_EleRegraNegocio')),
      nome<-'CamadaModel_'+se_Negocio.nome,
      elementos<-MMConceitual!EntidadeNegocio.allInstancesFrom('IN_MMConceitual')->collect(e|thisModule.resolveTemp(e,'te_ElePersistencia'))
     ),
     te_Controller:MMProjeto!Controller
     (
      nome<-'CamadaController_'+se_Negocio.nome,
      elementos<-MMConceitual!EntidadeNegocio.allInstancesFrom('IN_MMConceitual')->collect(e|thisModule.resolveTemp(e,'te_EleControle'))
     ),
     te_Sistema:MMProjeto!Sistema
     (
      nome<-se_Negocio.nome,
      camadas<-Set{te_View,te_Model,te_Controller}
     )
}
 rule MapearEntidadeNegocio{
 from
      se_EntidadeNegocio:MMConceitual!EntidadeNegocio
  to
     te_EleInterface:MMProjeto!EleInterface
     (
      nome<-'VW_'+se_EntidadeNegocio.nome
     ),
     te_EleRegraNegocio:MMProjeto!EleRegraNegocio
     (
      atributos<-MMConceitual!Dado.allInstancesFrom('IN_MMConceitual')->select(x|x.entidadeNegAssociada.nome=se_EntidadeNegocio.nome)-
>collect(e|thisModule.resolveTemp(e,'te_Atributo')),
      nome<-'RN_'+se_EntidadeNegocio.nome
     ),
     te_ElePersistencia:MMProjeto!ElePersistencia
     (
      nome<-se_EntidadeNegocio.nome+'DAO'
     ),
     te_EleControle:MMProjeto!EleControle
     (
      nome<-'CTR_'+se_EntidadeNegocio.nome
     )
}
rule MapearDado{
 from
      se_Dado:MMConceitual!Dado
  to
     te_Atributo:MMProjeto!Atributo
     (
      tipo<-se_Dado.tipo,
      nome<-se_Dado.nome
     )
}
```

Figura E.9 Codigo ATL gerado para a transformação

Apêndice

F

METAMODELO MMT EM FORMATO ECORE

Este apêndice apresenta o metamodelo de transformação (MMT) proposto nesta tese implementado em formato Ecore.

O metamodelo está dividido em três níveis de abstração: MMTspec, para especificação dos requisitos da transformação; MMThighdesign, para especificação do projeto de alto nível da transformação; e MMTlowdesign, para especificação do projeto de baixo nível da transformação.

A Figura F.1 apresenta o metamodelo MMTspec.

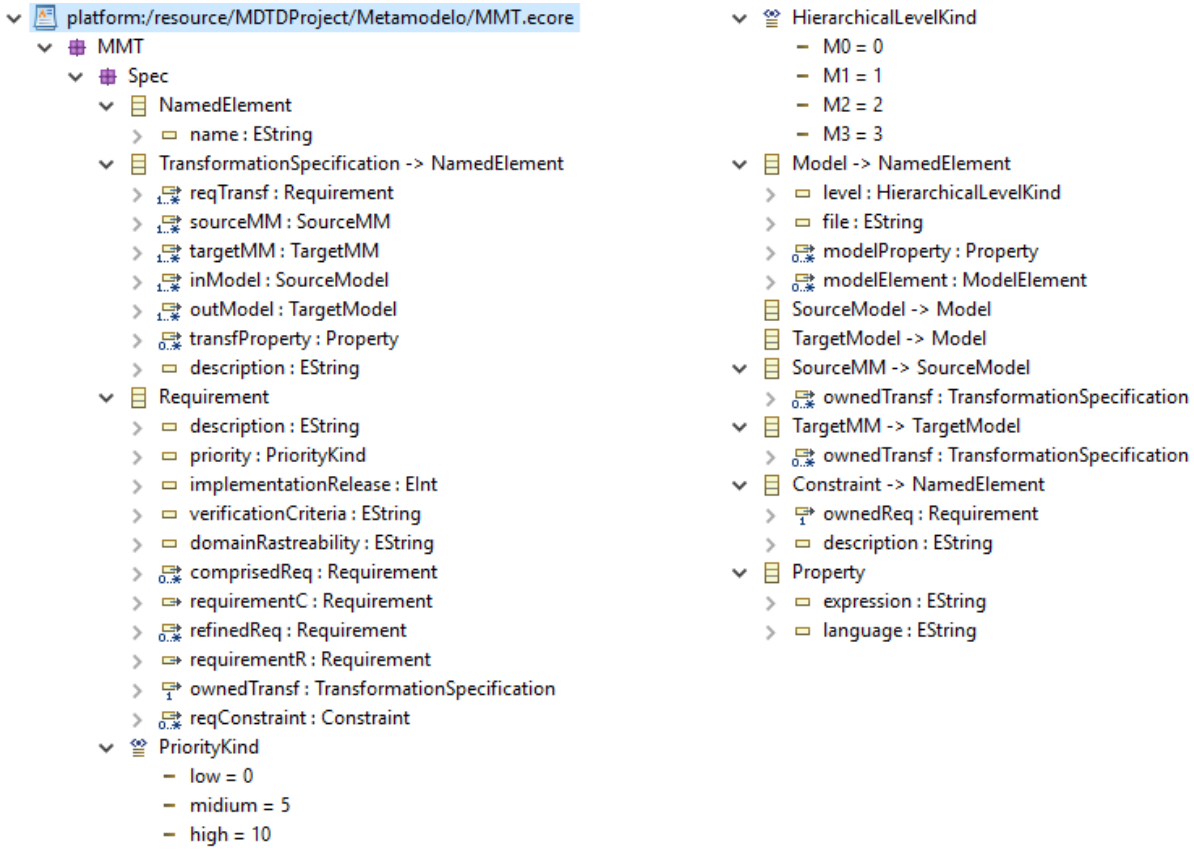


Figura F.1 Metamodelo MMTspec em formato Ecore

A Figura F.2 apresenta o metamodelo MMThighdesign.

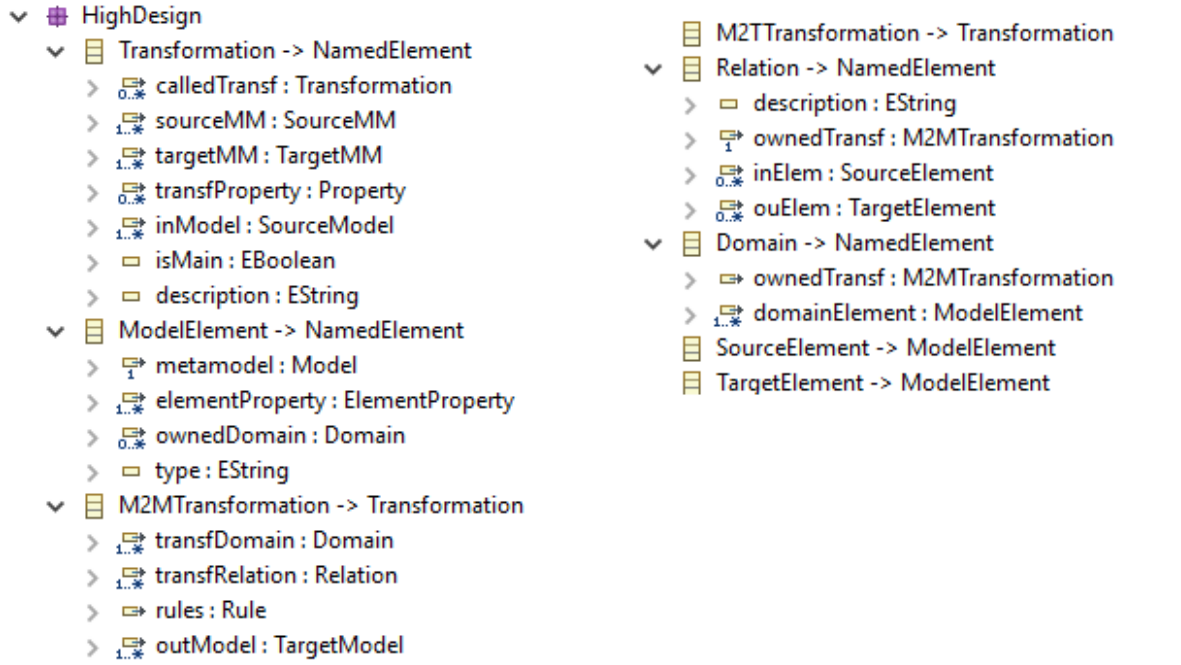


Figura F.2 Metamodelo MMThighdesign em formato Ecore

A Figura F.3 apresenta o metamodelo MMTlowdesign.

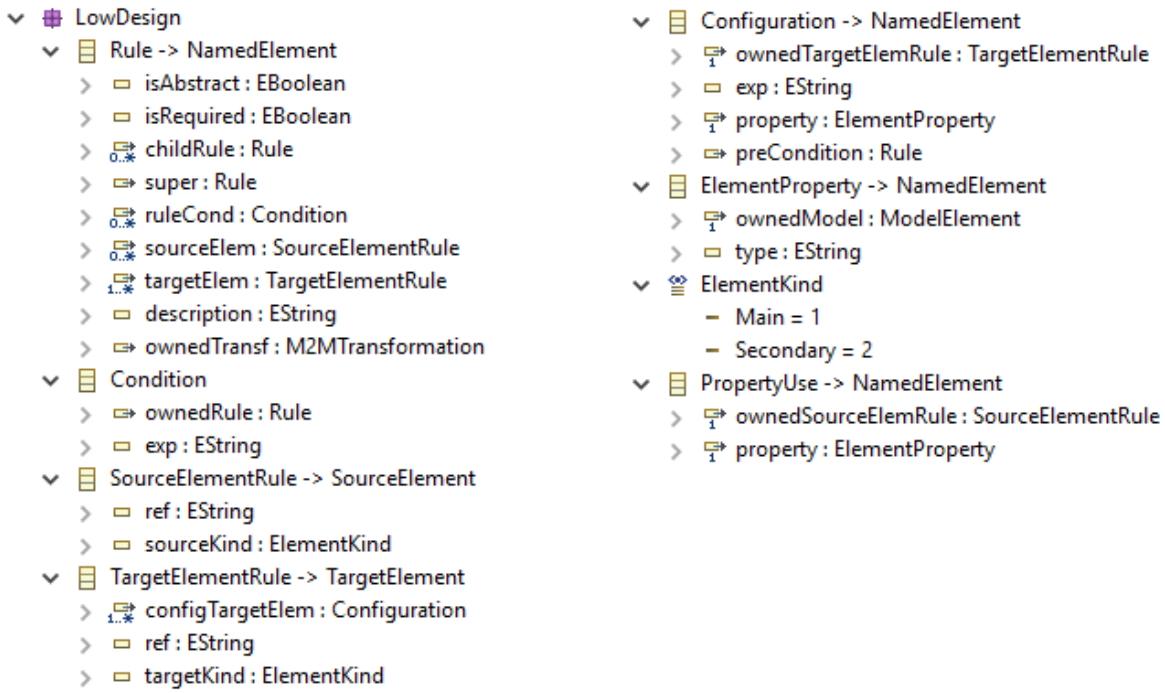


Figura F.3 Metamodelo MMTlowdesign em formato Ecore

Apêndice G

PERFIL ATL

Este apêndice apresenta o Perfil ATL (PATL) definido nesta tese para representar modelos de transformação de modelos específicos para a linguagem ATL. A seguir é apresentado o metamodelo da linguagem ATL e em seguida é o perfil definido com base neste metamodelo.

Metamodelo ATL

A Figura G.1 ilustra parte do metamodelo ATL. O metamodelo completo no formato Ecore pode ser encontrado em *www.eclipse.org/atl*.

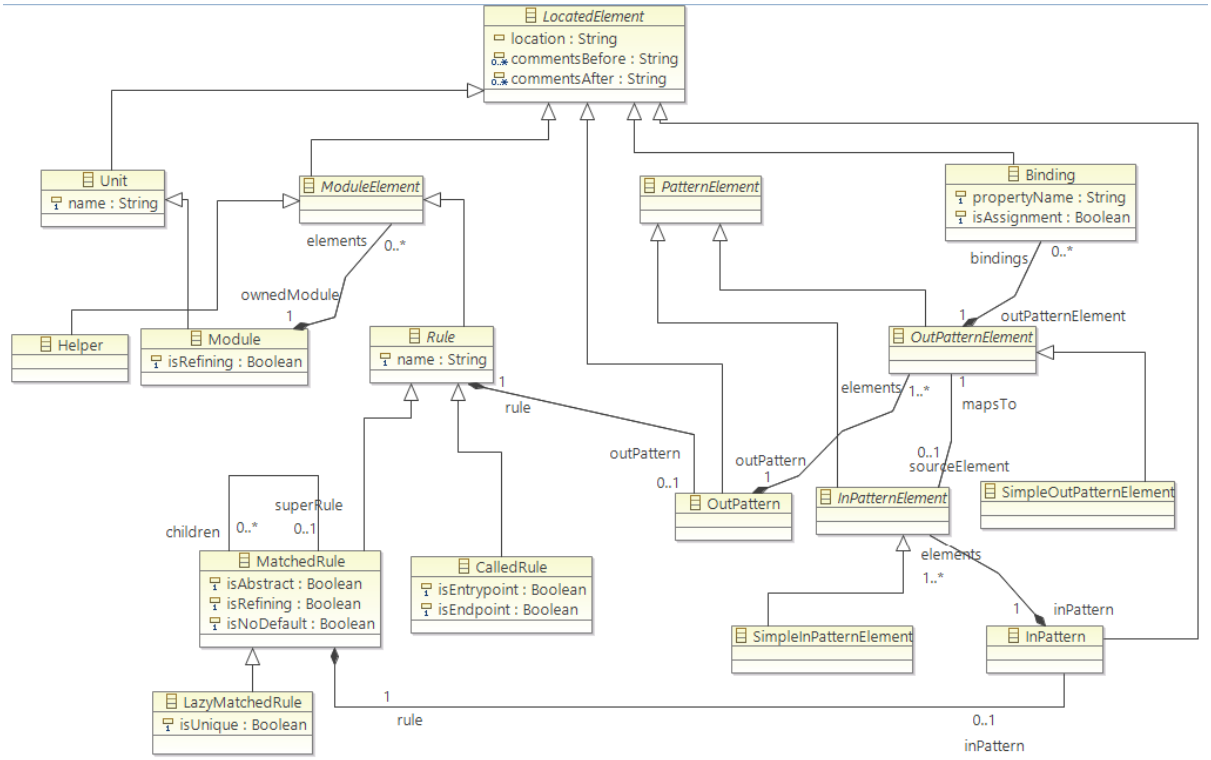


Figura G.1 Parte do metamodelo ATL

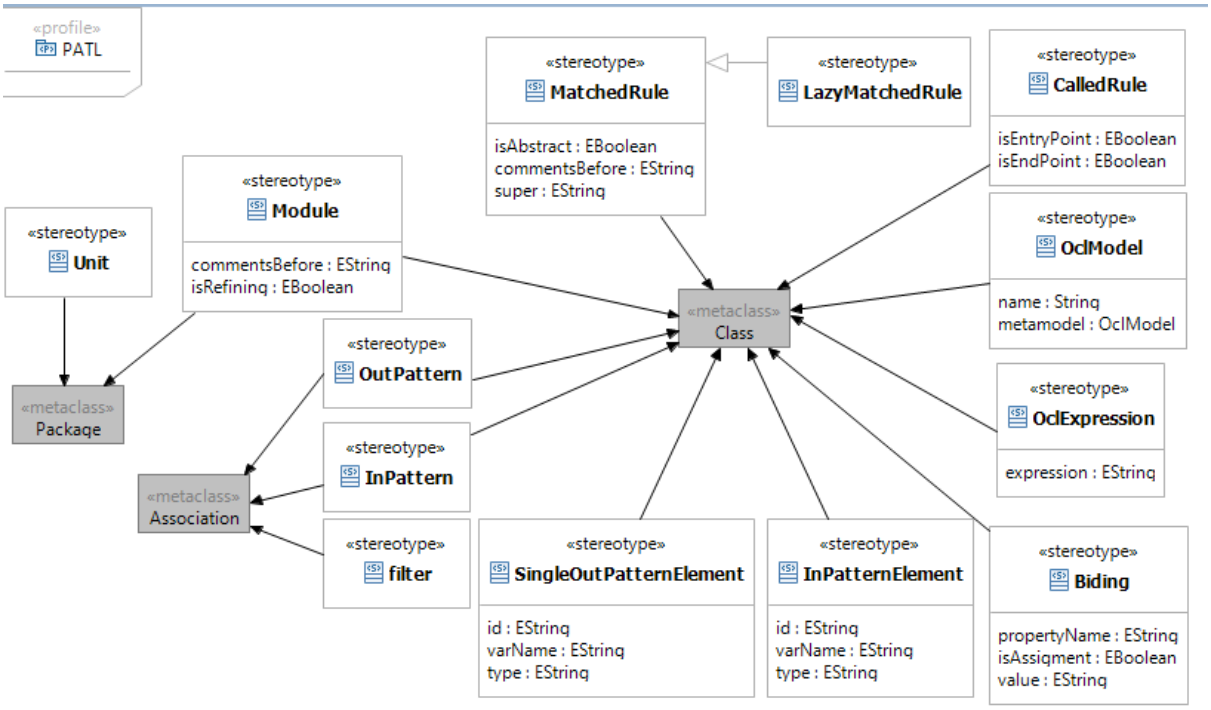


Figura G.2 perfil ATL

De uma forma geral uma transformação em ATL é um modulo (*Module*) composto de elementos (*ModuleElement*) que podem ser regras (*Rule*) ou funções (*Heper*). Um *Rule* pode ser de dois tipos, *MatchedRule* ou *CalledRule*. Uma *MatchedRule* é automticamente executada quando existe correspondência entre o elemento lido no modelo de entrada e o elemento de entrada da regra. Um tipo especial desta regra é a *LazyMatchedRule*, que precisa ser chamada para ser executada e não permite a passagem de parâmetros. A *CalledRule* é uma regra que deve ser chamada para ser executa e pode receber parâmetros. Regras são definidas usando padrões que determinam como seus elementos serão inicializados. As regras do tipo *MatchedRule* possuem padrão de entrada (*InPattern*) e de saída (*OutPattern*). As regras do tipo *CalledRule* possuem apenas padrão de saída (*OutPattern*). Esses padrões são compostos de elementos (*InPatternElement*) e (*OutPatternElement*) que guardam quais os elementos de entrada/saída da regra. Para elementos de saída são definidos *Binding* indicando como as propriedades do elemento serão inicializadas. Adicionalmente o metamodelo ATL utiliza em sua definição elementos dos metamodelo *PrimitiveType* e *OCL*.

Perfil PATL

A representação dos modelos de transformação específicos de plataforma em ATL é realizada através do perfil PATL especificado no pacote ilustrado na Figura G.2, aplicado ao diagrama de classes da UML.

Exemplo de uso do perfil PATL na especificação da transformação OO2RDBMS

A Figura G.3 ilustra um exemplo de diagrama de classes com parte da transformação OO2RDBMS, utilizada como exemplo nesta tese, estereotipado com perfil PATL.

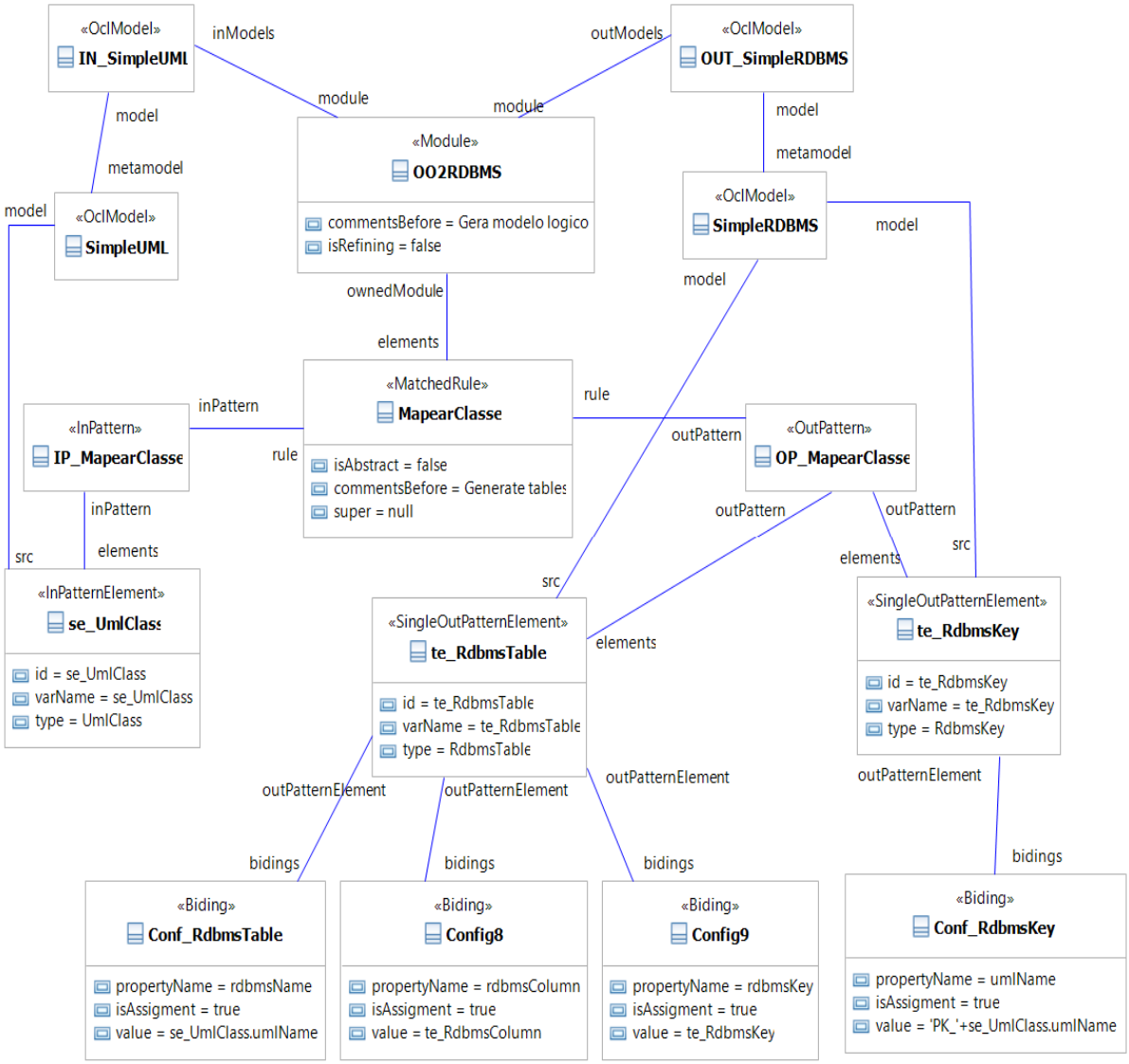


Figura G.3 Modelo da transformação OO2RDBMS em ATL

No modelo apresentado na Figura G.3, o *Module OO2RDBMS* representa a transformação. Os *OclModel* representam os metamodelos e modelos de entrada e saída da transformação. Neste exemplo apenas uma regra é apresentada a *MatchedRule Mapear-Classe* que possui um *inPattern* associada ao *InPatternElement* o elemento de entrada e um *outPattern* associado a dois *SingleOutPatternElement* com os elementos de saída da regra. Para cada elemento de saída as propriedades são inicializadas através de *Binding*.

Apêndice

H

PERFIL QVT

Este apêndice apresenta o Perfil QVT (PQVT) definido nesta tese para representar modelos de transformação de modelos específicos para a linguagem QVT. A seguir é apresentado o metamodelo da linguagem QVT e em seguida é o perfil definido com base neste metamodelo.

Metamodelo da linguagem QVT

A *Meta Object Facility (MOF) Query/View/Transformation (QVT)* é o padrão OMG para o desenvolvimento de transformações. Neste trabalho utilizamos a *QVT-Relation*, que representa a parte declarativa da QVT, para definir os modelos específicos de plataforma. As Figuras H.1 e H.2 ilustram parte do metamodelo da *QVT-Relation*, no formato visual e no formato ECore, dividido em três pacotes. O pacote *QVTBase* define de uma forma geral que uma transformação (*Transformation*) é composta de regras (*Rule*) e modelos (*TypedModel*) que compõe o domínio da transformação (*Domain*). O pacote *QVTRelation* (à esquerda) a *Transformation* é especializada em *RelationTransformation* e a *Rule* é especializada em *Relation*. Para cada *Relation* são definidos padrões (*Pattern*) especificados através de *Templates* (*TemplateExp*) de acordo com o metamodelo *QVTTemplate*.

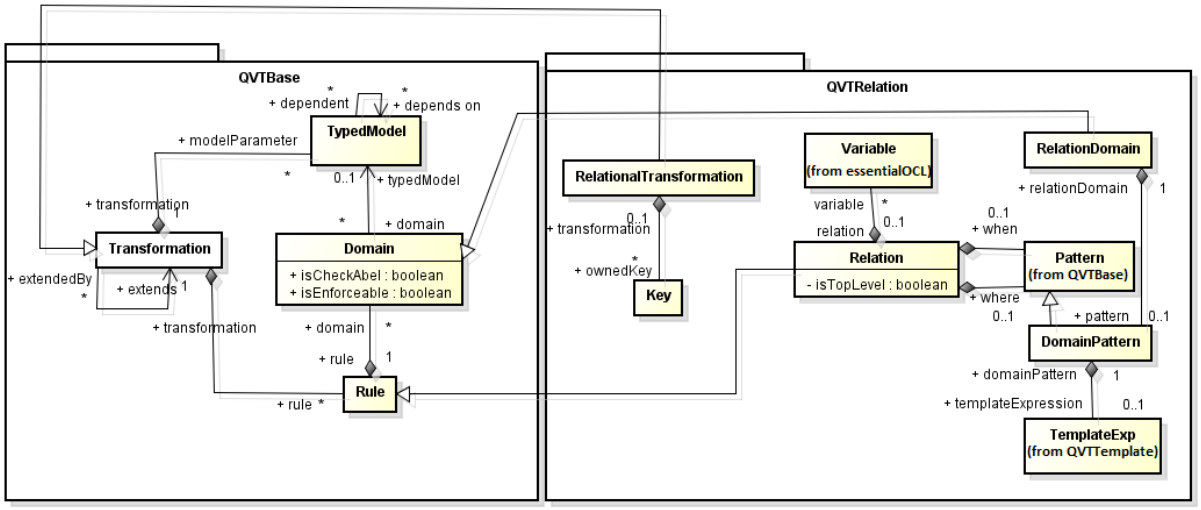


Figura H.1 Parte dos metamodelos QVTBase e QVTRelation no formato visual

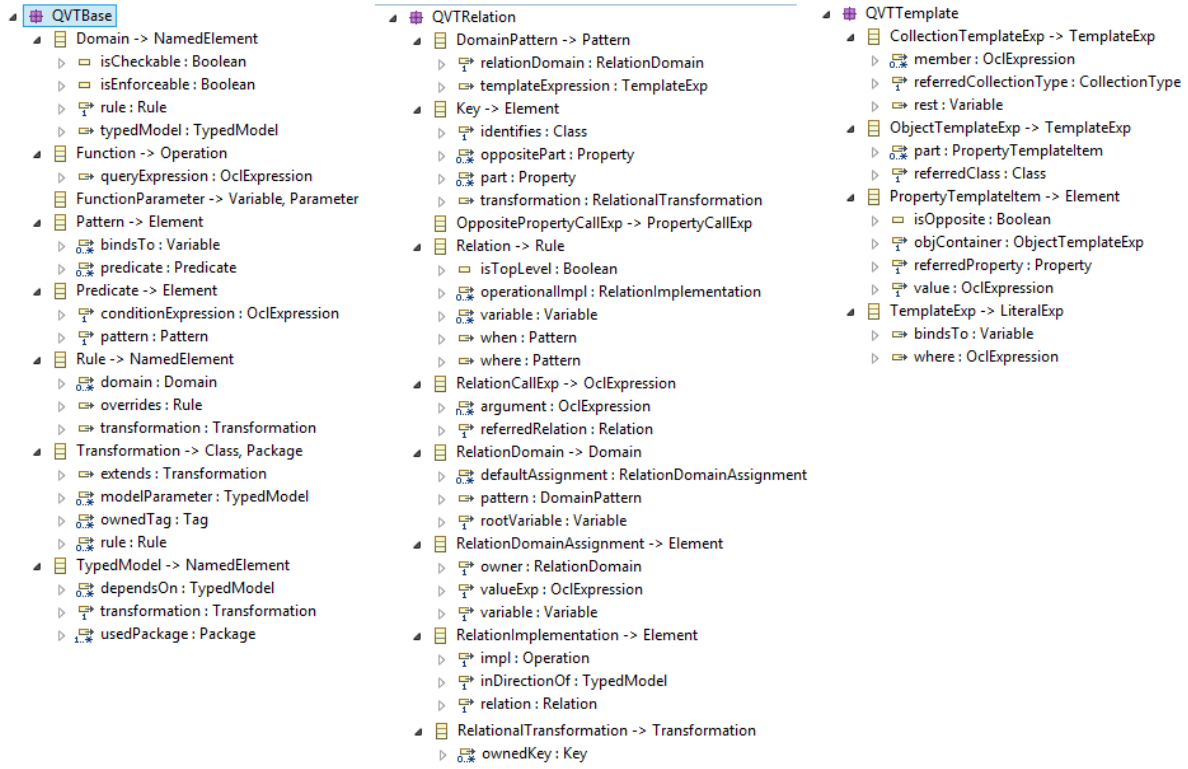


Figura H.2 Parte dos metamodelos QVTBase e QVTRelation no formato ECore

Perfil PQVT

A representação dos modelos de transformação específicos de plataforma em QVT é realizada através do perfil PQVT especificado no pacote ilustrado na Figura H.3, aplicado ao diagrama de classes da UML.

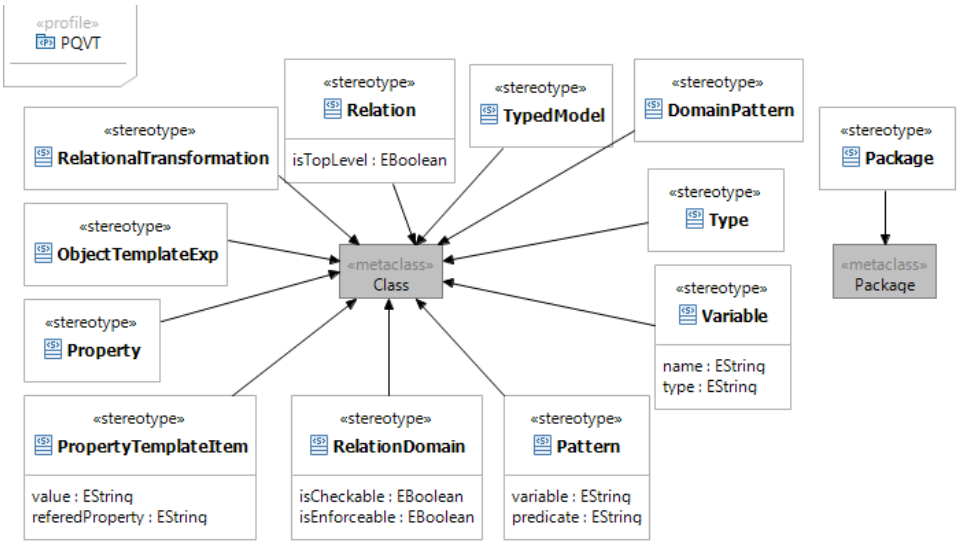


Figura H.3 Perfil QVT

O pacote profile PQVT contém estereótipos correspondentes aos conceitos do metamodelo QVT e as metaclasses estendidas (em amarelo) da UML necessárias para criar o modelo de transformação de modelo representado em um diagrama de classes.

Exemplo de uso do perfil PQVT na especificação da transformação OO2RDBMS

A Figura H.4 ilustra, em um diagrama de classes estereotipado com o perfil PQVT, uma parte da transformação OO2RDBMS utilizada como exemplo ao longo desta tese.

No modelo apresentado na Figura H.4, a *TransformationRelation OO2RDBMS* representa a transformação. Os *TypeModel* representam os metamodelos e modelos de entrada e saída da transformação. Na figura apenas parte de uma regra é apresentada a *Relation MapearClasse* com dois *RelationDomain*, um de entrada e um de saída. Cada *RelationDomain* está associado a variáveis e a um *DomainPattern* que define *Templates* (*ObjectTemplateExp*) de configuração para cada elemento gerado.

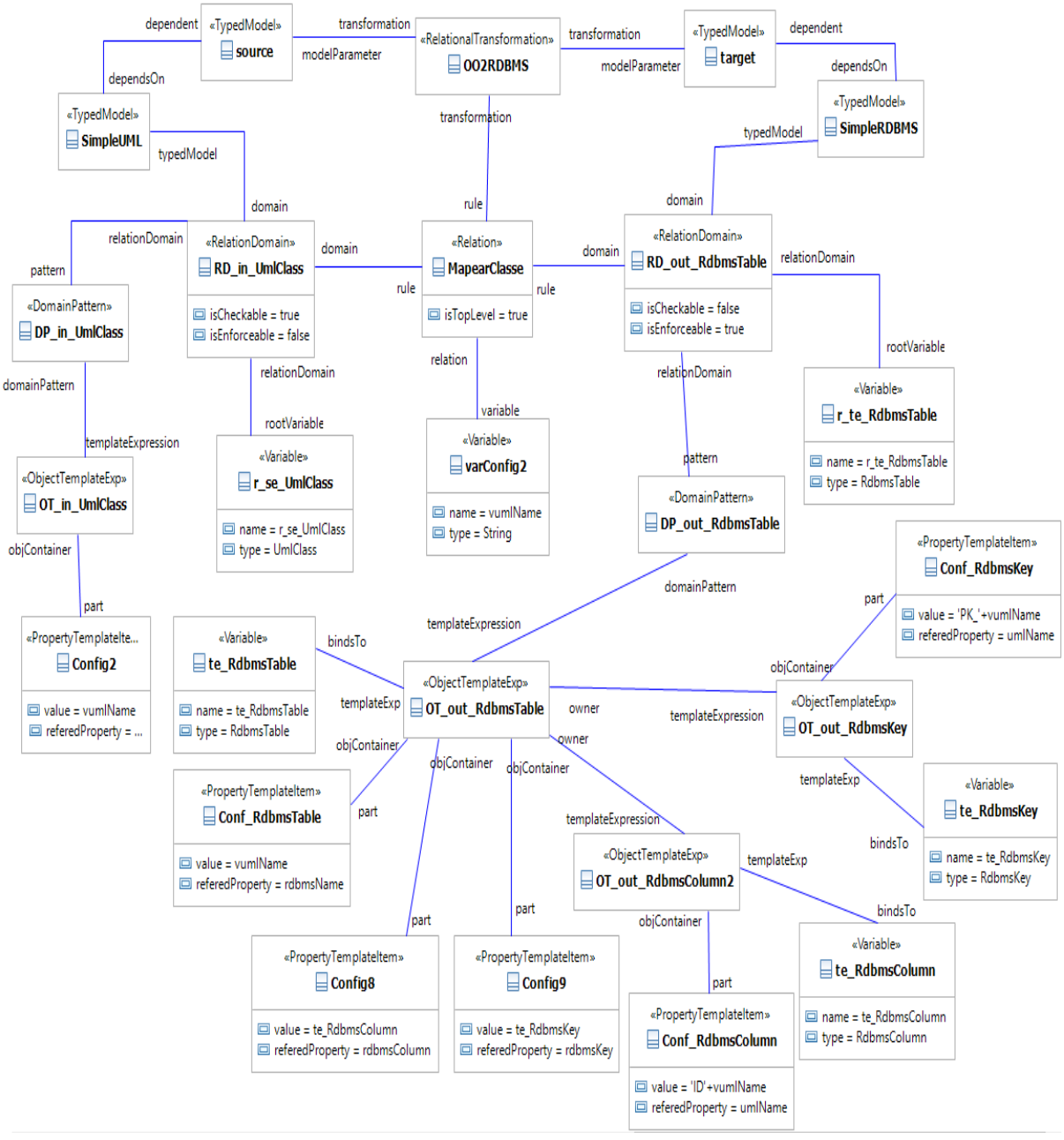


Figura H.4 Modelo da transformação OO2RDBMS em QVT

Apêndice

I

QUESTIONÁRIO PARA AVALIAÇÃO DA QUALIDADE DA TRANSFORMAÇÃO

Este apêndice apresenta o questionário para coleta de dados sobre a qualidade da transformação (Figura I.1). O questionário foi adaptado de (AMSTEL; MARCEL; BRAND, 2010), pois acrescenta uma explicação sobre cada atributo de qualidade avaliado.

Questionário para coleta de dados sobre a qualidade da transformação

Nome do avaliador:

Transformação avaliada:

A qualidade de uma transformação está relacionada a alguns atributos importantes:

- *Understandability*, relacionada à quantidade de esforço requerido para entender o propósito de uma transformação;
- *Modifiability*, relacionada à quantidade de esforço requerido para adaptar a transformação com novos requisito;
- *Reusability*, relacionada às partes de uma tran sformação podem ser reusadas em outras transformações;
- *Modularity*, relacionada à capacidade das transformações de modelo serem sistematicamente separadas e estruturadas;
- *Completeness*, indica que qualquer modelo instancia do metamodelo pode ser processado pela transformação;
- *Consistency*, relacionada a uniformidade de implementação da transformação, ou seja, se é utilizado um estilo de programação uniforme em toda a transformação;
- *Conciseness*, relacionada a existência de elementos supérfluos na transform ação (ex. variáveis nunca utilizadas).

Responda as questões utilizando a escala:

MB - muito baixo, B – Baixo, M – médio, A – alto e MA – muito alto

Pergunta	MB	B	M	A	MA
1 - Qual o esforço requerido para entender a transformação?					
2 - Você considera que a transformação tem um estilo uniforme de programação?					
3 -A transformação poderia ter sido escrita com menos elementos? (variáveis, regras, condições) atendendo a mesma especificação?					
4 - O quanto você avalia que a transformação provê as funcionalidades que deveria prover?					
5 - O quão reusável é a transformação ou partes dela na criação de outras transformações?					
6 - Qual o nível médio de dificuldade esperada para realizar modificações na transformação?					
7 – Qual a sua avaliação sobre o nível de esforço necessário para o desenvolvedor entender à transformação o suficiente para encontrar erros nela?					
8 - O quão concisa é a transformação? (não existem elementos supérfluos elementos que não precisavam ter sido criados?)					
9 - Qual o grau de elementos conflitantes existentes na transformação (ex. variáveis com mesmo nome em regras diferentes)					
10 -Quão usável é a transformação como ponto de partida para criar uma nova transformação?					
11 - Quão modificável é a transformação?					
12 - Quão aderente é a transformação aos seus requisitos?					
13 - Quanto esforço em media e necessario para entender o propósito de uma função (ex. uma regra)					
14 - Qual o esforço necessário para modificar a transformação quando houver uma mudança nos metamodelos envolvidos?					
15 - Quão completa é a transformação?					
16 - Quantos elementos do metamodelo fonte podem ser transformados pela transformação?					
17 - Ate que ponto a transformação é livre de elementos redundantes (ex. variáveis e regras)?					
18 - Qual o esforço necessário para adaptar a transformação em mudança de requisitos?					
19 - Como você avalia a propriedade de reusabilidade da transformação?					
20 - Como você avalia a propriedade de consistência da transformação?					
21 - Como você avalia a propriedade de *Understandability* da transformação?					

Figura I.1 Questionário para coleta de dados sobre a qualidade da transformação