Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Coordenação de Pós-Graduação em Ciência da Computação

Uma Abordagem para Análise de Impacto de Mudanças em Transformações de Modelos

Andreza de Sousa Vieira

Tese submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Campina Grande - Campus I como parte dos requisitos necessários para obtenção do grau de Doutor em Ciência da Computação.

Área de Concentração: Ciência da Computação
Linha de Pesquisa: Engenharia de Software

Franklin de Souza Ramalho
(Orientador)

Campina Grande, Paraíba, Brasil

Resumo

MDD (*Model-Driven Development*) tem como principal objetivo deslocar o foco da implementação do código fonte para o desenvolvimento de modelos dentro do processo de desenvolvimento de software. Elementos que desempenham um papel muito importante dentro de MDD são as transformações de modelos, isto é, regras que descrevem como modelos de origem devem ser usados para gerar automaticamente um ou mais modelos de destino. Como qualquer projeto de software, os projetos baseados em MDD evoluem ao longo do ciclo de vida, tendo em vista que mudanças em suas transformações são frequentes. Antes de aplicar qualquer mudança em uma transformação é importante analisar como ela pode afetar o projeto como um todo. No entanto, atualmente não há nenhuma técnica para auxiliar os gerentes de projeto ou desenvolvedores nesse sentido. O objetivo deste trabalho é propor uma abordagem para análise do impacto de mudanças em transformações de modelos. Baseada na análise estática, a abordagem proposta visa: (i) identificar o conjunto de elementos impactados por uma mudança na transformação; e (ii) mensurar o impacto da mudança através de um conjunto de métricas. Dentre outros benefícios, a abordagem proposta pode auxiliar: (i) os gerentes de projeto a melhor escalonar e priorizar mudanças, uma vez que eles podem estimar os custos da aplicação de cada uma delas; e (ii) os desenvolvedores a identificar, automaticamente, todos os elementos impactados com uma mudança, assim, economizando esforço e tempo de desenvolvimento. Um estudo de caso foi conduzido para identificar novas métricas e para avaliar a abordagem proposta em relação à análise manual do impacto de mudanças em diversas transformações reais. Com o estudo, foi possível perceber que os resultados obtidos pela abordagem proposta e pela análise manual foram bastante próximos. Porém, considerando o esforço e o tempo de análise, os resultados da abordagem proposta foram melhores.

Abstract

The main goal of the MDD (Model-Driven Development) approach is to change the focus from source code to models within the software development process. Elements that play important role in MDD are model transformations, *i.e.* rules that describe how source models should be used to automatically generate one or more target models. As any software project, MDD-based projects evolve along their lifecycle in such way that changes in their transformations are frequent. Before applying any change to a transformation, it is important to analyze how it would impact the whole project. However, there is currently no technique to help project managers or developers in this direction. The objective of this work is to propose an approach to analyze the impact of changes resulting from model transformations. Based on static analysis, the proposed approach aims at: (i) identifying the set of elements impacted with a change applied to a transformation; and (ii) measuring the impact of a change through a set of metrics. Among other benefits, the proposed approach can help: (i) project managers to better prioritize and schedule changes, since they can estimate the costs to apply each of them; and (ii) developers to automatically identify all elements impacted with a change, thus saving effort and development time. A case study was conducted to identify new metrics and to evaluate the proposed approach regarding to the manual change impact analysis considering real transformations. This study revealed that the results obtained by using the proposed aproach and using the manual analysis were very similar. However, regarding effort and time, we observed that the results obtained from the proposed approach were superior than the manual analysis.

Agradecimentos

Agradeço, primeiramente, à Deus pelo dom da vida e por me permitir realizar sempre novas conquistas.

Aos meus pais, ao meu esposo, ao meu irmão e aos meus amigos, que sempre me ajudaram com conselhos, incentivos e levantando a minha cabeça nos momentos difíceis. Obrigada pela paciência e compreensão dos momentos em que estive ausente em suas vidas.

Ao meu orientador, o professor Franklin Ramalho. Sou muito grata pela paciência, por todo o auxílio, pela dedicação empenhada, pelas motivações e pelos rápidos e construtivos *feedbacks*.

Aos professores e demais funcionários que fazem o Curso de Pós-Graduação em Ciência da Computação da UFCG, que prestaram auxílio direta ou indiretamente à minha pesquisa.

Aos professores, colegas e amigos do SPLab que, direta ou indiretamente, me ajudaram sempre que precisei. Agradeço também à CAPES, que colaborou financeiramente para a realização deste trabalho.

Conteúdo

Lista de Símbolos

ATL - *ATLAS Transformation Language*

GPL - *General Public License*

HOT - *Higher-Order Transformations*

ICC - *Intraclass Correlation*

KM3 - *Kernel MetaMetaModel*

MDD - *Model-Driven Development*

MOPS - *MOdelchecking Programs for Security properties*

OMG - *Object Management Group*

OWL - *Web Ontology Language*

QVT - *Query/View/Transformation*

RTEC - *Relevance of the Impacted Module Elements*

RTERT - *Relevance of the Transformation Element Relation Types*

RITE - *Relevance of the Impacted Transformation Elements*

RCTAE - *Relevance of the Change Type Applied to an Element*

RCITOM - *Relevance of the Change Impact in the Transformation Output Model*

ROMTC - *Relevance of the Output Model in Transformation Chains*

RTEATM - *Relevance of the Transformation Element Across Transformation Modules*

UML - *Unified Modeling Language*

VTCL - *Viatra Textual Command Language*

Yasca - *Yet Another Source Code Analyzer*

Lista de Figuras

Lista de Tabelas

Lista de Códigos Fonte

Lista de Equações

Capítulo 1

Introdução e Motivação

Novas abordagens e tecnologias no âmbito da Engenharia de Software estão surgindo no intuito de oferecer uma maior automação e produtividade no processo de desenvolvimento de sistemas. MDD (*Model-Driven Development*) [Selic, 2003] é um exemplo de abordagem que atua nesse sentido propondo separar a especificação de um sistema de seus detalhes de implementação.

Com a abordagem de MDD, modelos de entrada podem ser automaticamente transformados em modelos de saída através de definições de transformações, que são regras de transformação. Linguagens de transformação são utilizadas para expressar definições de transformação, um exemplo deste tipo de linguagem é a ATL (*ATLAS Transformation Language*) [Jouault e Kurtev, 2006]. Ela é usada para especificar transformações do tipo modelo-modelo, no qual modelos de entrada são transformados em modelos de saída. Por exemplo, um diagrama de classes é transformado em um modelo Java.

Um sistema geralmente passa por várias manutenções e evoluções durante o seu ciclo de vida, sendo assim, diversas partes de um sistema podem ser modificadas. Da mesma maneira, sistemas baseados na abordagem de MDD evoluem e, segundo Amstel e Brand [van Amstel e van den Brand, 2011], suas definições de transformações passam por mudanças ao passo que os requisitos mudam. Antes de aplicar qualquer mudança em uma transformação, é importante: (i) entender a transformação, ou seja, as dependências e as relações entre os seus elementos; e (ii) conhecer as consequências da mudança, ou seja, os elementos dentro da transformação afetados com a mudança. Uma abordagem que pode ser utilizada para obter estas informações é a análise de impacto de mudanças, que é o processo onde é possível

identificar as potenciais consequências de uma mudança e estimar o que precisa ser modificado para realizá-la [Bohner e Arnold, 1996]. Segundo Bohner e Arnold, analisar o impacto de mudanças antes mesmo de aplicá-las é uma importante tarefa no ciclo de desenvolvimento de um software. A análise de impacto de mudanças pode ser utilizada por meio de técnicas de: (i) análise estática, que identifica todos os possíveis impactos de uma mudança considerando todas as possíveis execuções de um programa; (ii) análise dinâmica, que identifica os impactos de uma mudança considerando os rastros de execução do programa; e (iii) análise híbrida, que combina as duas técnicas citadas anteriormente.

A análise de impacto de mudanças ainda é uma abordagem incipiente no contexto de MDD, onde ela é realizada manualmente. De acordo com Wimmer et al. [Wimmer et al., 2012], atualmente não existem técnicas disponíveis focadas na manutenção de transformações, então ele afirma que os desenvolvedores realizam esta tarefa manualmente e que um suporte para auxiliá-los nesse sentido se faz necessário. Por ser uma tarefa manual, ela é trabalhosa, demorada e está sujeita a erros dado que os desenvolvedores têm que procurar manualmente pelas dependências e relações do elemento da transformação que será modificado [Mendez et al., 2010]. Por exemplo, antes de mudar os parâmetros de uma determinada regra *R1* da transformação, é necessário saber se ela é chamada por outra regra dentro da mesma transformação e/ou em outra transformação do projeto. Se esta mudança é aplicada na transformação e os elementos afetados pela mudança não são adaptados, a integridade da transformação pode ser comprometida.

Existem vários trabalhos voltados para a análise de impacto de mudanças no campo da Engenharia de Software [Hattori et al., 2008] [Maia et al., 2010] [Ren et al., 2004] [Zhang et al., 2008]. Contudo, no contexto de MDD há uma carência muito grande de técnicas que ajudem os desenvolvedores a analisar mudanças nos artefatos de projetos. Em metamodelos, encontramos os trabalhos propostos por Garcia e Díaz [Garcia e Díaz, 2010], Mendez et al. [Mendez et al., 2010], Garcés et al. [Garcés et al., 2012] e Iovino et al. [Iovino et al., 2012], que focam na análise do impacto de mudanças em metamodelos. Eles propõem uma abordagem para ajudar os desenvolvedores a adaptar suas transformações de modelos de acordo com as mudanças realizadas no metamodelo. Porém, estes trabalhos não permitem nem a identificação dos elementos impactados nem a medição do impacto causado pela mudança de um elemento da transformação, o foco deles é apenas nos metamodelos. Em transfor-

mações de modelos, encontramos apenas dois trabalhos: (i) o proposto por Marzullo et al. [Marzullo et al., 2010], que sugere o uso de técnicas de MDD juntamente com anotações no modelo dos projetos para automatizar a análise de impacto de mudanças. Porém, o seu foco é mais voltado para o rastreamento dos requisitos de um software para seu código fonte e vice-versa. Marzullo et al. não propõem técnicas para análise de impacto de mudanças no contexto de MDD; e (ii) o proposto por Wimmer et al. [Wimmer et al., 2012], que define um catálogo de refatorações para melhorar os atributos de qualidade relacionados à manutenção de transformações de modelo. Embora este seja o trabalho que mais se aproximou do trabalho proposto nesta tese, as mudanças usadas nele são apenas de refatoração. Além disso, ele não identifica os elementos impactados nem mensura o impacto da mudança.

A necessidade de técnicas que auxiliem os desenvolvedores na aplicação de mudanças (seja devido à manutenção ou evolução) em transformações de modelos é expressa por diversos autores da área, tais como Mendez et al., Garcia e Díaz, e Iovino et al. Segundo Mendez et al., a detecção manual de inconsistências (impacto de mudanças) em transformações de modelos pode ser difícil sem o conhecimento prévio da transformação e da mudança. De acordo com Garcia e Díaz, mudanças em transformações tende a ser uma tarefa complexa, que exige a assistência de uma abordagem automática para guiar o desenvolvedor. Iovino et al. dizem que os desenvolvedores podem adaptar suas transformações de modelos depois de aplicar uma mudança através da inspeção manual do código, porém, eles dizem que realizar esta tarefa sem uma técnica especializada pode ser difícil. Como se pode notar, este é um tema bastante relevante e ainda há muito o que explorar no que tange manutenção e evolução de projetos baseados em MDD no escopo de transformações de modelos.

Nesta tese, é proposta uma abordagem para medir o impacto de mudanças em transformações de modelos antes que a mudança seja aplicada. A abordagem proposta permite aos usuários: (i) identificar os elementos impactados por uma mudança a ser aplicada em uma transformação; e (ii) calcular o valor de impacto para a mudança através de um conjunto de métricas. Com esta abordagem, a análise de impacto de mudanças no contexto de transformações de modelos se torna uma tarefa mais simples e eficaz, pois permite ao gerente de projetos melhor escalonar e priorizar as mudanças a serem realizadas com base em informações concretas, isto é, ele poderá mensurar o impacto de cada mudança e priorizá-las de acordo com o impacto que será causado. Além disso, a abordagem irá auxiliar os desenvol-

vedores durante a manutenção das transformações depois que uma mudança é aplicada, pois a abordagem identifica todas as partes da transformação que precisarão ser adaptadas depois que a mudança é aplicada.

Um estudo de caso foi conduzido neste trabalho com dois objetivos: servir como alicerce e embasamento para a *identificação de métricas* para medir o impacto de mudanças em transformações de modelos; e *avaliar a abordagem proposta*. Para que o primeiro objetivo do estudo fosse alcançado, foi elaborada a seguinte questão de pesquisa: *Quais critérios os usuários adotam para identificar os elementos impactados e medir o impacto de mudanças em transformações de modelos?* Como resultado do estudo, foram identificadas e formalizadas cinco métricas. As duas outras métricas propostas neste trabalho foram definidas com base em estudos sobre análise de impacto de mudanças e baseado também no desenvolvimento com linguagens de transformações de modelos. Por outro lado, para que o segundo objetivo deste estudo fosse alcançado, foi elaborada a seguinte questão de pesquisa: *Quão divergentes são os resultados da análise de impacto de mudanças realizada por nossa abordagem e pela análise manual?* Com o estudo foi possível concluir que os resultados da abordagem proposta foram melhores que os da abordagem manual em relação ao esforço e tempo de análise.

Nas seções seguintes, serão apresentados: o escopo em que este trabalho está inserido (Seção 1.1), uma breve contextualização do problema abordado (Seção 1.2), os objetivos gerais e específicos deste trabalho (Seção 1.3), a metodologia adotada para o desenvolvimento deste trabalho (Seção 1.4) e a forma como este documento está estruturado (Seção 1.5).

1.1 Escopo do Trabalho

Este trabalho de doutorado propõe uma abordagem baseada na análise estática para a análise do impacto de mudanças em transformações de modelos especificadas em qualquer linguagem de transformação, como por exemplo ATL e QVT [Object Management Group, 2011a].

Contudo, para a aplicação e avaliação da abordagem proposta foi necessário escolher uma linguagem específica. Nesse sentido, a linguagem ATL foi escolhida porque ela oferece uma documentação bastante completa e um rico repositório [Eclipse Foundation, 2012] com diversas transformações de modelos, as quais foram utilizadas na avaliação realizada neste

trabalho (apresentada no Capítulo 8).

1.2 Contextualização do Problema

O principal problema a ser tratado neste trabalho é a carência de abordagens para a análise de impacto de mudanças em transformações de modelos dentro de um projeto baseado em MDD. Atualmente, os desenvolvedores realizam mudanças em suas transformações sem saber do impacto que elas irão causar no projeto como um todo.

Dado um conjunto de mudanças que devem ser aplicadas em uma transformação, como (e baseado em que critério) o gerente do projeto deve escalonar e priorizar cada uma das mudanças? Como identificar, previamente, os elementos que serão impactados por cada mudança e o impacto que será causado? Para facilitar a realização de manutenções e evoluções com sucesso em transformações de modelos, é interessante que estas perguntas sejam corretamente respondidas. No caso de transformações muito extensas, com regras não triviais, fica inviável para o desenvolvedor analisar e mensurar manualmente o impacto que determinada mudança irá causar na transformação (e no projeto como um todo), assim como identificar os elementos impactados.

Segundo Mendez et al., sem o uso de uma abordagem para análise do impacto de mudanças qualquer modificação pode causar, mais facilmente, inconsistência no projeto como um todo, o que exigirá ainda mais esforço do desenvolvedor e tempo de desenvolvimento. Por exemplo, se uma regra é removida de uma transformação e o impacto desta remoção não é analisado, as demais regras dessa transformação que fazem referência à regra removida ficarão inconsistentes. Adicionalmente, estas tarefas realizadas de forma *ad-hoc* e manual são suscetíveis a erros.

1.3 Objetivos Gerais e Específicos

O objetivo geral deste trabalho é propor uma abordagem, amparada na análise estática, para analisar previamente o impacto de uma mudança a ser aplicada em uma transformação de modelo especificada em qualquer linguagem de transformação. A seguir, são enumerados os objetivos específicos a serem alcançados com este trabalho:

- Proposta de uma técnica para análise estática de transformações através da inspeção do código fonte da transformação, que servirá de auxílio para a abordagem proposta neste trabalho;

- Proposta de um metamodelo para análise de impacto de mudanças que defina os conceitos relacionados à aplicação de uma mudança a um elemento da transformação, de forma que seja genérico para qualquer tipo de mudança a ser aplicada em transformações especificadas em qualquer linguagem;

- Proposta de um conjunto de métricas para calcular o impacto de aplicar uma mudança em uma transformação, que serão utilizadas pela abordagem proposta neste trabalho;

- Proposta de uma abordagem amparada na análise estática que permita analisar o impacto de mudanças em transformações de modelos antes que elas sejam aplicadas, utilizando o analisador estático e o conjunto de métricas previamente definidas;

- Realização de um estudo de caso no intuito de aplicar a abordagem proposta e avaliar a sua eficácia, bem como para a identificação de novas métricas.

1.4 Metodologia

Após a realização de estudos dirigidos sobre análise de impacto de mudanças e técnicas existentes para análise estática de programas, neste trabalho foi desenvolvido um analisador estático para o contexto de transformações de modelos. Em seguida, com o auxílio desse analisador estático foi desenvolvida uma abordagem amparada na análise estática para a análise de impacto de mudanças em transformações de modelos. Essa abordagem foi generalizada com o uso de um metamodelo para análise de impacto de mudanças a fim de permitir sua aplicação em qualquer linguagem de transformação. Por fim, foi elaborado um estudo de caso tanto para a definição de novas métricas a fim de serem usadas na abordagem quanto para avaliar a abordagem proposta.

O desenvolvimento deste trabalho foi planejado e dividido em seis macro atividades, as quais são apresentadas a seguir.

- **A1** - Realização de estudos dirigidos sobre análise de impacto de mudanças em geral e revisão bibliográfica contínua sobre técnicas e ferramentas para análise de impacto de mudanças, amparadas na análise estática, no escopo de MDD;

- **A2** - Investigação e levantamento das principais técnicas já existentes em outras áreas para a análise estática de programas. Para isto, essas técnicas foram analisadas e relacionadas com possíveis aplicações em transformações de modelos;

- **A3** - Desenvolvimento de um analisador estático no intuito de inspecionar o código fonte de transformações ATL e obter informações sobre cada um dos elementos especificados de forma a auxiliar a abordagem de análise de impacto de mudanças proposta;

- **A4** - Proposta de uma abordagem para análise de impacto de mudanças em transformações ATL amparada na análise estática, com o uso do analisador estático desenvolvido em *A3*. Para isto, as seguintes atividades foram desenvolvidas:

  - **A4.1** - Proposta dos principais tipos de mudanças que podem ocorrer em transformações ATL;
  - **A4.2** - Proposta de um conjunto de métricas a fim de calcular o valor de impacto para a aplicação de uma mudança em um elemento da transformação;
  - **A4.3** - Definição de um mecanismo para identificar, dentro da transformação, o conjunto de elementos impactados com a aplicação de uma mudança;

- **A5** - Proposta de uma abordagem genérica para análise de impacto de mudanças em transformações de modelos amparada na análise estática. Para isto, as seguintes atividades foram desenvolvidas:

  - **A5.1** - Redefinição dos principais tipos de mudanças previamente definidos em *A4.1*, de forma que sejam genéricos para qualquer linguagem de transformação de modelos;
  - **A5.2** - Redefinição das métricas previamente definidas em *A4.2*, de forma que sejam genéricas para qualquer linguagem de transformação de modelos. Para isto, elas foram definidas de acordo com um metamodelo de análise de impacto de mudanças construído em *A5.3*;

  - **A5.3** - Proposta de um metamodelo genérico para análise de impacto de mudanças usado na abordagem para formalizar a definição das métricas para calcular o impacto de mudanças;

- **A6** - Elaboração e execução de um estudo de caso onde os participantes tinham que analisar um conjunto de mudanças a serem aplicadas em transformações de modelos e informar o impacto de cada uma delas, justificando os critérios adotados para isto. O estudo teve dois objetivos principais: (i) identificar novas métricas para calcular o impacto de mudanças em transformações de modelos; e (ii) avaliar a abordagem proposta neste trabalho. Foi escolhido o estudo de caso para a execução neste trabalho tendo em vista que o objetivo era conduzir um estudo exploratório, no sentido de compreender como os desenvolvedores atualmente analisam o impacto de mudanças em transformações de modelos e, a partir de dados coletados do estudo, tentar aprimorar a abordagem proposta. Nesse sentido, com os dados coletados a partir do estudo de caso exploratório foi possível a identificação de novas métricas e a avaliação da abordagem proposta comparando-a com a análise manual do impacto de mudanças. Durante a definição do estudo de caso, duas questões de pesquisa foram elaboradas. Vale ressaltar que para estas questões de pesquisa não foram formuladas hipóteses nem testes de hipótese[1], pois trata-se de questões de pesquisa qualitativa que, segundo [Sarantakos, 1998], tendem a ser abertas e descritivas, não exigindo obrigatoriamente a formulação de hipóteses. Ao contrário, em questões de pesquisa quantitativa são formuladas hipóteses a serem comprovadas através de testes estatísticos. As questões de pesquisa definidas nesta tese são apresentadas a seguir.

  - **A6.1** - *Quais critérios os usuários adotam para identificar os elementos impactados e medir o impacto de mudanças em transformações de modelos?* Esta questão foi respondida a fim de atingir o primeiro objetivo do estudo. Para isto, foram analisados os critérios adotados pelos participantes para analisar o impacto de mudanças. As novas métricas definidas foram formalizadas de acordo com o metamodelo de mudanças definido em *A5.3*;

[1] Um teste de hipótese é um método de inferência estatística que usa dados de um estudo científico. Trata-se de um procedimento estatístico baseado na análise de uma amostra, através da teoria de probabilidades, usado para avaliar determinados parâmetros que são desconhecidos numa população.

  - **A6.2** - *Quão divergentes são os resultados da análise de impacto de mudanças realizada pela abordagem proposta e pela análise manual?* Esta questão foi respondida a fim de atingir o segundo objetivo do estudo. Para isto, foi feita uma comparação da análise do impacto das mudanças feita manualmente pelos participantes e feita automaticamente pela abordagem proposta.

1.5 Estrutura do Documento

Esta tese de doutorado encontra-se organizada em onze capítulos, incluindo esta introdução. A seguir, é apresentada uma breve descrição de cada um dos capítulos restantes:

- Capítulo 2 (Fundamentação Teórica): Apresenta os conceitos básicos para o melhor entendimento deste trabalho. Serão abordados fundamentos de: (i) MDD e transformações de modelos; (ii) ATL; (iii) análise de impacto de mudanças; e (v) métricas;

- Capítulo 3 (Exemplo Motivacional): Apresenta um breve exemplo ilustrando a motivação da abordagem proposta neste trabalho;

- Capítulo 4 (Metamodelo para Análise de Impacto de Mudanças): Apresenta um metamodelo construído para definir os conceitos relativos à *mudanças em transformações* que foi adotado ao longo do trabalho;

- Capítulo 5 (Abordagem para Análise de Impacto de Mudanças): Apresenta a abordagem, amparada na análise estática, que foi desenvolvida neste trabalho para analisar o impacto de mudanças em transformações de modelos;

- Capítulo 6 (Analisador Estático para ATL): Apresenta o analisador estático que foi desenvolvido neste trabalho para a linguagem ATL;

- Capítulo 7 (Exemplo de Aplicação): Apresenta um exemplo de aplicação da abordagem proposta neste trabalho;

- Capítulo 8 (Estudo de Caso): Apresenta um estudo de caso conduzido para a identificação de métricas e para a avaliação da abordagem de análise de impacto de mudanças proposta;

- Capítulo 9 (Trabalhos Relacionados): Apresenta os trabalhos considerados mais relevantes e relacionados ao trabalho proposto;

- Capítulo 10 (Considerações Finais): Apresenta uma compilação dos resultados e contribuições obtidas com este trabalho e algumas sugestões de trabalhos futuros que podem ser desenvolvidos.

Capítulo 2

Fundamentação Teórica

Este capítulo tem a finalidade de apresentar os conceitos necessários para o melhor entendimento do presente trabalho. Serão abordadas definições sobre MDD e Transformações de Modelos (Seção 2.1), a linguagem de transformação ATL (Seção 2.2), Análise de Impacto de Mudanças (Seção 2.3) e Métricas (Seção 2.4).

2.1 MDD e Transformações de Modelos

O principal objetivo de MDD (*Model-Driven Development*) é deslocar o esforço e tempo durante o ciclo de vida de um software das tarefas de implementação e testes para tarefas de modelagem, metamodelagem e transformações de modelos.

Os modelos, metamodelos e transformações são artefatos fundamentais dentro da abordagem MDD. Os modelos são representações de um sistema em diferentes níveis de abstração. Já os metamodelos, são modelos que descrevem outros modelos, isto é, descrevem a sintaxe abstrata de linguagens e domínios. Um exemplo de metamodelo é o metamodelo da UML introduzido na especificação de UML [Object Management Group, 2011b]. Portanto, em UML podemos usar classes, atributos, associações e outros tipos de elementos porque no seu metamodelo esses elementos são definidos. As transformações, por sua vez, consistem na geração automática de modelos de saída a partir de modelos de entrada, de acordo com uma definição de transformação. Esta definição de transformação é um conjunto de regras que, juntas, definem como modelos de entrada devem ser usados para gerar automaticamente modelos de saída.

Existem dois tipos de transformações: modelo-modelo e modelo-texto. Em transformações modelo-modelo (também chamadas transformações de modelos), modelos de entrada são transformados em um ou mais modelos de saída. Por exemplo, um diagrama de classes é transformado em um modelo Java. Neste caso, o modelo de destino não é um artefato executável nem compilável, mas sim um modelo Java no formato XMI (*XML Metadata Interchange*) [Object Management Group, 2007] contendo todas as informações necessárias para uma futura geração do código fonte Java. As transformações modelo-modelo geralmente são passos intermediários para se chegar à geração de artefatos textuais, tais como código Java, páginas HTML, documentações, código Javascript, etc [Czarnecki e Helsen, 2003]. Por outro lado, em transformações modelo-texto (também chamadas transformações textuais), os modelos de entrada são transformados em sintaxe concreta. Seguindo o exemplo anterior, um diagrama de classes pode ser transformado diretamente em código Java ou um modelo Java (saída da transformação modelo-modelo anterior) pode ser transformado em código Java.

As regras de transformações são expressas através de uma linguagem de transformação que deve ser entendida por ferramentas denominadas engenhos de transformação, capazes de total e automaticamente executá-las. Como exemplos de linguagens de transformação modelo-modelo podemos citar QVT e ATL. Exemplos de linguagens de transformação modelo-texto são MOF2Text (*MOF Model to Text Transformation*) [Object Management Group, 2008], MOFScript [Oldevik, 2006], Xpand [Eclipse Foundation, 2008] e Acceleo [Eclipse Foundation, 2009].

2.2 ATL (*ATLAS Transformation Language*)

ATL é uma linguagem de transformação popular empregada para especificar transformações de modelos. Ela permite especificar três tipos de unidades, que são definidas em arquivos ATL próprios e distintos: (i) *ATL Library* - especifica funções e constantes que podem ser importadas a partir de diferentes tipos de unidades de ATL; (ii) *ATL Query* - especifica uma operação para calcular um valor primitivo a partir de um conjunto de modelos e/ou elementos de modelos de entrada; e (iii) *ATL Module* - especifica a forma que os modelos de saída serão automaticamente gerados a partir de modelos de entrada. Dentro das unidades *Li-*

brary e *Query*, os desenvolvedores podem especificar apenas *helpers*, enquanto que dentro da unidade *Module* eles podem especificar *helpers* e regras.

ATL oferece suporte a dois tipos de *helpers*: *functional* e *attribute*, que podem ser vistos como métodos e constantes, respectivamente. Ambos podem ser referenciados a partir de diferentes pontos de uma transformação ATL. Por outro lado, ATL oferece suporte a três tipos de regras: *matched rule*, *lazy matched rule* e *called rule*. Uma *matched rule* é uma regra declarativa que especifica para quais tipos de elementos de origem os elementos de destino devem ser gerados. Esta regra não recebe parâmetros e é automaticamente executada quando a transformação ATL é executada. Uma *lazy matched rule* é um tipo de *matched rule*, porém, ela deve ser explicitamente invocada por outra regra para que seja executada. Uma *called rule* é uma regra imperativa que pode receber parâmetros e deve ser invocada por outra regra para que seja executada e ela pode, opcionalmente, gerar elementos do modelo de destino. De acordo com o metamodelo de ATL [Jouault, 2007], todos os elementos previamente discutidos suportados por ATL são também chamados de *module element* (elemento do módulo), dado que eles estendem a metaclasse abstrata `ModuleElement` deste metamodelo.

Para ilustrar um exemplo de transformação ATL, o Código Fonte 2.1 apresenta um trecho da transformação UML2Java [Eclipse Foundation, 2005b], qual gera um modelo Java a partir de um modelo UML. O modelo Java a ser gerado possuirá informações para a criação de classes Java, principalmente, sobre a sua estrutura, atributos e métodos. De acordo com o Código Fonte 2.1, na linha 1 foi especificado o nome do módulo (`UML2JAVA`). Na linha 2 foi especificado tanto o nome do modelo de saída que será gerado (`OUT`), seguido do nome do seu metamodelo (`JAVA`), quanto o nome do modelo de entrada (`IN`), seguido do nome do seu metamodelo (`UML`). Os caminhos dos arquivos correspondentes ao modelo de entrada, modelo de saída e seus respectivos metamodelos são informados pelo usuário durante a configuração da transformação no plug-in do Eclipse [Eclipse Foundation, 2014c]. Nas linhas 3-4 foi definido um *helper* (`isPublic()`) que verifica se um elemento do modelo UML tem visibilidade pública ou não. Este *helper* foi definido no contexto de um `ModelElement` do metamodelo de UML. A regra `AttributeToField` foi especificada nas linhas 5-12. Ela gera um campo no modelo Java a partir de um atributo do modelo UML. Para isto, a regra associa cada propriedade do modelo Java com cada propriedade do modelo UML.

Código Fonte 2.1: Trecho da transformação UML2Java.atl

```
1  module UML2JAVA;
2  create OUT : JAVA from IN : UML;
3  helper context UML!ModelElement def: isPublic() :
4           Boolean = self.visibility = #vk_public;
5  rule AttributeToField {
6     from e : UML!Attribute
7     to out : JAVA!Field (
8             name <- e.name,
9             isPublic <- e.isPublic(),
10            type <- e.type
11            )
12 }
```

2.3 Análise de Impacto de Mudanças

Softwares evoluem ao longo do seu ciclo de vida ao passo que novos requisitos são implementados e os existentes são modificados para satisfazer, por exemplo, as expectativas dos usuários, as novas necessidades do ambiente operacional e as melhorias do *design* do software. As mudanças de um software dentro do processo de desenvolvimento devem ser cuidadosamente gerenciadas, tendo em vista que mudanças nos requisitos podem resultar em grandes mudanças no software e em consequências imprevisíveis que muitas vezes atrasam a sua implementação. É importante compreender o software que será modificado e conhecer as consequências da mudança. A análise de impacto de mudanças é uma maneira de conseguir tais informações.

Embora a análise de impacto de mudanças tenha sido praticada há muitos anos, ainda não há um consenso para sua definição. De acordo com Pfleeger e Bohner [Pfleeger e Bohner, 1990], a análise de impacto de mudanças é a avaliação dos diversos riscos associados à mudança, incluindo a estimativa dos efeitos sobre os recursos, esforço e escalonamento. Por outro lado, Turver e Munro [Turver e Munro, 1994] definem a análise de impacto de mudanças como a avaliação, nos outros módulos do sistema, das consequências de uma mudança feita em um módulo. Neste trabalho, será usada a definição apresentada por Bohner e Arnold [Bohner e Arnold, 1996]: a análise de impacto de mudanças é o processo no qual é possível identificar as potenciais consequências de uma mudança e estimar o que precisa ser modificado para realizá-la.

Existem três maneiras nas quais a análise de impacto de mudanças pode ser utilizada: (i)

por meio da análise estática, que é realizada na estrutura do programa através da inspeção do seu código fonte ou bytecode para detectar os impactos de uma mudança antes dela ser aplicada; (ii) por meio da análise dinâmica, que é realizada nos rastros de execução do programa para detectar os impactos de uma mudança depois que ela já foi aplicada; e (iii) por meio de uma abordagem híbrida, que combina a análise estática e a análise dinâmica.

Ferramentas de análise estática bem conhecidas são o FindBugs [FindBugs, 2013] e o Checkstyle [Checkstyle, 2014], ambas para código Java. Também podemos citar o MOPS [Chen e Wagner, 2002] e o UNO [Holzmann, 2002], ambas para código C. Ferramentas para análise estática são amplamente utilizadas para ajudar os desenvolvedores a depurar o código fonte de programas e identificar possíveis erros ou vulnerabilidades de codificação. No entanto, ferramentas de análise estática são suscetíveis a detectar falsos-positivos, pois elas consideram todas as possíveis entradas e todos os possíveis comportamentos do programa, inclusive aqueles impossíveis de ocorrer. Nesse contexto, falso-positivos são os possíveis resultados (sejam erros, vulnerabilidades, etc.) identificados pela técnica que não ocorrem de fato.

As informações obtidas a partir de ferramentas de análise estática também podem ser aplicadas para outros fins, tais como: (i) para detectar pontos fracos no código fonte em um local exato; (ii) para verificar se o código fonte está em conformidade com diretrizes de codificação; (iii) para provar propriedades sobre um determinado programa; e (iv) para analisar o impacto da mudança de um software.

2.4 Métricas

De acordo com Sommerville [Sommerville, 2003], uma métrica de software é qualquer tipo de medição que se refira a um sistema de software, processo ou documentação relacionada. As métricas de software assumem um importante papel dentro da engenharia de software, especialmente na gerência de projetos, e fornecem uma base quantitativa para a validação. Pressman [Pressman, 1995] classifica as métricas de software em duas categorias: medidas diretas e medidas indiretas.

- *Medidas Diretas:* são realizadas em termos de atributos observáveis. Pode-se considerar o custo e o esforço aplicados ao desenvolvimento e manutenção de um software, a

quantidade de linhas de código produzidas, o total de defeitos registrados durante um determinado período de tempo e a velocidade de execução;

- *Medidas Indiretas:* são métricas que podem ser obtidas através de outras métricas. Pode-se considerar a qualidade, confiabilidade, manutenibilidade, funcionalidade, complexidade e capacidade de manutenção do software.

Para medir software, são utilizadas diversas métricas que são como tipos de medições aplicadas a um sistema de software, documentação ou processo relacionado. Através dessas métricas é possível determinar o esforço ou tempo para realização de uma tarefa, por exemplo. Além disso, as métricas de software são facilmente calculadas, entendidas e testadas e independem do observador que as aplica, sendo também uma boa fonte para estudos estatísticos acerca do ciclo de vida de um software.

Com a utilização de métricas de software é possível conseguir melhorias e resultados mais satisfatórios do software, mais segurança para os gerentes de projeto. É uma maneira de eliminar os obstáculos, corrigir erros e falhas. Seguem algumas razões para a utilização de métricas: (i) indicar a qualidade do software; (ii) indicar o grau de interdependências entre componentes de um software e prover um *feedback* para uma melhor reusabilidade; (iii) avaliar a produtividade dos que desenvolvem o software; (iv) entender e aperfeiçoar o processo de desenvolvimento; (v) determinar os benefícios derivados de novos métodos e ferramentas de engenharia de software; e (vi) calcular o impacto de mudanças, como é o caso da utilização de métricas nesta tese.

Capítulo 3

Exemplo Motivacional

Neste capítulo, será demonstrada por meio de um exemplo a importância e necessidade de se analisar o impacto de uma mudança em uma transformação antes de aplicá-la.

Como exemplo, será ilustrada a transformação KM32OWL [Eclipse Foundation, 2014b], que possui 307 linhas de código contendo 20 elementos no módulo: 1 *attribute helper*, 3 *functional helpers*, 10 *matched rules*, 3 *lazy matched rules* e 3 *called rules*. Esta transformação ATL é responsável por transformar um modelo KM3 (*Kernel MetaMetaModel*) [ATLAS Group e LINA & INRIA e Nantes, 2005] em um modelo OWL (*Web Ontology Language*) [World Wide Web Consortium, 2004]. Neste exemplo, a mudança a ser aplicada é a alteração do nome da *lazy matched rule* `FunctionalProperty` apresentada no Código Fonte 3.1 para `AddFunctionalProperty`. Esta regra é responsável por criar um elemento `OWL!FunctionalProperty` (linha 5-7) a partir de um elemento `KM3!Reference` (linha 3).

Código Fonte 3.1: Regra FunctionalProperty especificada na transformação KM32OWL.atl

```
1 lazy rule FunctionalProperty {
2     from
3       r : KM3!Reference
4     to
5       o : OWL!FunctionalProperty (
6         isDefinedBy <- r
7         )
8 }
```

A *lazy matched rule* `FunctionalProperty` é invocada por três regras dentro da transformação KM32OWL: `EnumerationProperty2ObjectProperty`, `Attribute2DataTypeProperty` e `Reference2ObjectProperty`. Por exemplo, o Código Fonte 3.2 mostra quando esta *lazy matched rule* é invocada pela *matched rule* `Attribute2DataTypeProperty` (linha 10). Se o nome da *lazy matched rule* `FunctionalProperty` for modificado, todas estas três regras que a invocam serão impactadas com a mudança. Portanto, depois de aplicar a mudança, cada uma destas regras deve ser adaptada para invocar a regra modificada pelo seu novo nome: `AddFunctionalProperty`.

Código Fonte 3.2: Regra Attribute2DataTypeProperty da transformação KM32OWL.atl

```
1  rule Attribute2DataTypeProperty extends StructuralFeature2Property {
2      from
3          f : KM3!Attribute (
4              f.type.oclIsTypeOf( KM3!DataType )
5          )
6      to
7          p : OWL!OWLDatatypeProperty ()
8      do {
9          if ( ( f.upper = 1) and ( f.lower = 1) )
10             thisModule.FunctionalProperty( f );
11         ...
12     }
13 }
```

A identificação dos elementos do módulo impactados é crucial para se conhecer os lugares exatos que devem ser adaptados devido à mudança. Além disso, também é importante saber o *valor de impacto* causado pela mudança na transformação. O valor de impacto é definido neste trabalho como um valor calculado para uma mudança no intuito de medir o seu impacto dentro da transformação através de um número.

Segundo Wimmer et al. [Wimmer et al., 2012], atualmente não existem técnicas que auxiliem os desenvolvedores na manutenção de transformações de modelos, eles realizam esta tarefa manualmente. Então, quando os desenvolvedores precisam aplicar mudanças a um elemento do módulo de uma transformação, eles identificam manualmente todos os elementos

impactados: analisando toda a transformação e procurando os elementos do módulo afetados com a mudança. Segundo Mendez et al. [Mendez et al., 2010], a detecção manual de inconsistências (impacto de mudanças) em transformações de modelos pode ser difícil sem o conhecimento prévio da transformação e da mudança. Esta tarefa manual se torna inviável para transformações muito grandes. De acordo com Garcia e Díaz [Garcia e Díaz, 2010], mudanças em transformações tende a ser uma tarefa complexa, que exige a assistência de uma abordagem automática para guiar o desenvolvedor. Além disso, a identificação manual dos elementos impactados está suscetível a erros, colocando a integridade da transformação em risco. Portanto, neste exemplo motivacional foi identificado, manualmente, os elementos impactados dado que não existe nenhum mecanismo que auxilie na identificação automática destes elementos dentro da transformação. Além disso, não existe nenhuma técnica para medir o valor de impacto causado pela mudança.

Neste cenário, é possível perceber a necessidade de uma abordagem que auxilie os desenvolvedores na tarefa de manutenção em suas transformações, previamente identificando o impacto causado por cada mudança a ser aplicada. Quando o impacto de uma mudança é previamente analisado, os desenvolvedores economizam esforço e tempo de desenvolvimento, uma vez que eles rapidamente detectam as partes afetadas da transformação. Além disso, os gerentes se sentem mais confiantes para tomar decisões de projeto. Por exemplo, um gerente de projeto poderá mais facilmente escalonar e priorizar um conjunto de mudanças de acordo com: (i) o valor de impacto causado por cada uma delas em toda a transformação; e (ii) o número de elementos impactados.

Capítulo 4

Metamodelo para Análise de Impacto de Mudanças

Para que a abordagem proposta neste trabalho seja genérica para a aplicação em qualquer transformação de modelos, se faz necessária a construção de um metamodelo para análise de impacto de mudanças. Com um metamodelo genérico, é possível abstrair conceitos e características específicas de linguagens de transformação, focando nas definições comuns entre elas. O metamodelo proposto tem o papel fundamental de formalizar os conceitos necessários à definição das métricas de análise de impacto (apresentadas na seção 5.3), tais como: os elementos impactados com a mudança, o índice de relevância do elemento a ser modificado, de suas características e relações, etc. Vale ressaltar que neste metamodelo não são definidos todos os conceitos de uma transformação, mas apenas as informações necessárias que foram utilizadas para a definição das métricas. O metamodelo foi construído de forma que seja genérico para qualquer mudança aplicada a transformações de modelos especificadas em qualquer linguagem de transformação.

4.1 Definição do Metamodelo

Como ponto de partida, para a construção deste metamodelo foi tomado como base o metamodelo das linguagens ATL e QVT. O metamodelo para análise de impacto de mudanças é composto por quatro pacotes, como ilustrado na Fig. 4.1: `ChangeImpact`, `Basic`, `SpecificLanguageDefinitions` e `TransformationElements`. Cada um deles

será explicado com detalhes a seguir.

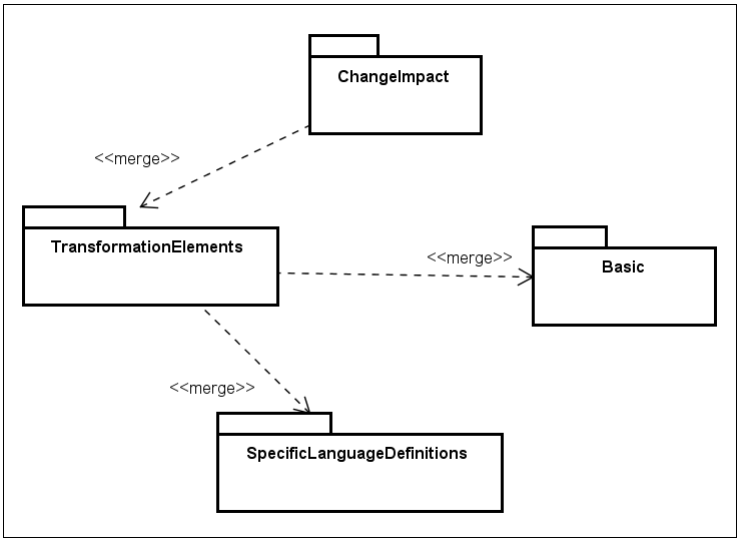

Figura 4.1: Diagrama de Pacotes.

Pacote `ChangeImpact`. Ele abrange os conceitos de uma mudança a ser aplicada a um elemento de uma transformação. Conforme apresentado na Fig. 4.2, ele compreende conceitos sobre o impacto da mudança, incluindo os tipos de mudança considerados neste trabalho. A metaclasse `Change` representa a mudança a ser aplicada a um elemento da transformação (*changedElement*) e define dois atributos: (i) o tipo de mudança (*kind*); e (ii) o conjunto de ações necessárias para aplicar a mudança (*actions*). Os tipos de mudança são representados pelo *enumeration* `ChangeKind`. O conjunto de elementos impactados com a mudança é representado pelo relacionamento *impactedElements* e o valor de impacto é representado pelo atributo *score* da metaclasse `ChangeImpact`. De acordo com a restrição OCL especificada para esta metaclasse, o atributo *score* deve assumir valores entre 0 e 1.

Foram identificados os principais tipos de mudança que podem ser aplicadas a elementos dentro de transformações de modelos. Para cada tipo de mudança identificado foi atribuído um *nome*. Por exemplo, se a mudança foi "modificar o nome de uma constante" da transformação, então o tipo de mudança foi chamado de *changeConstanteName*. Eles são apresentados na Tabela 4.1, onde a coluna 1 mostra o elemento da transformação para o qual a mudança será aplicada e a coluna 2 mostra os tipos de mudança. O *enumaration* `ChangeKind` (Fig. 4.2) reflete estes tipos de mudança.

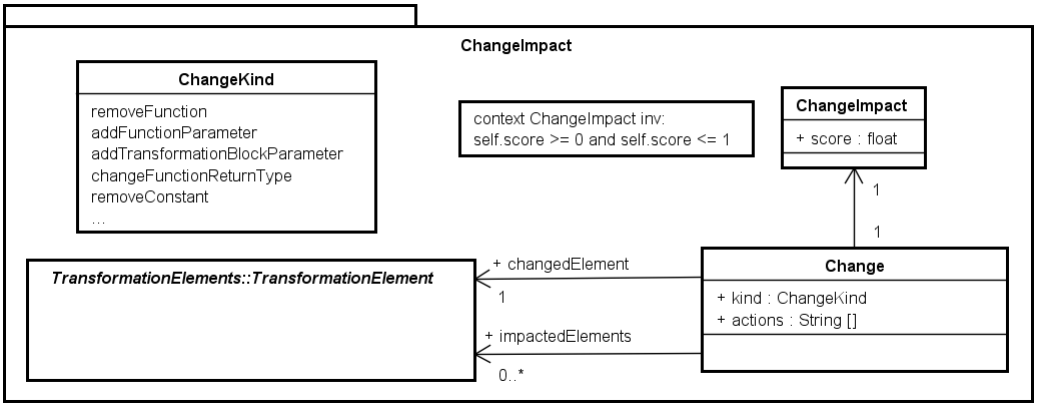

Figura 4.2: Pacote ChangeImpact.

Tabela 4.1: Classificação dos tipos de mudança.

Elemento da Transformação	Tipo de Mudança
Function	changeFunctionBody changeFunctionReturnType changeFunctionParameterType changeFunctionContextType
Constant	removeConstant changeConstantType changeConstantName changeConstantContextType
TransformationBlock	addTransformationBlock removeTransformationBlock addTransformationBlockParameter removeTransformationBlockInheritance changeTransformationBlockName changeTransformationBlockParameterType removeTransformationBlockFilter removeOutputModelElement changeOutputModelElementType removeOutputModelElementBinding

Pacote `Basic`. Ele abrange os tipos genéricos aplicáveis a qualquer linguagem de transformação, como ilustrado na Fig. 4.3. Este pacote foi estendido do metamodelo de UML [Object Management Group, 2011b]. Algumas linguagens de transformação utilizam tipos específicos. Nestes casos, o pacote `Basic` deve ser estendido com os novos tipos.

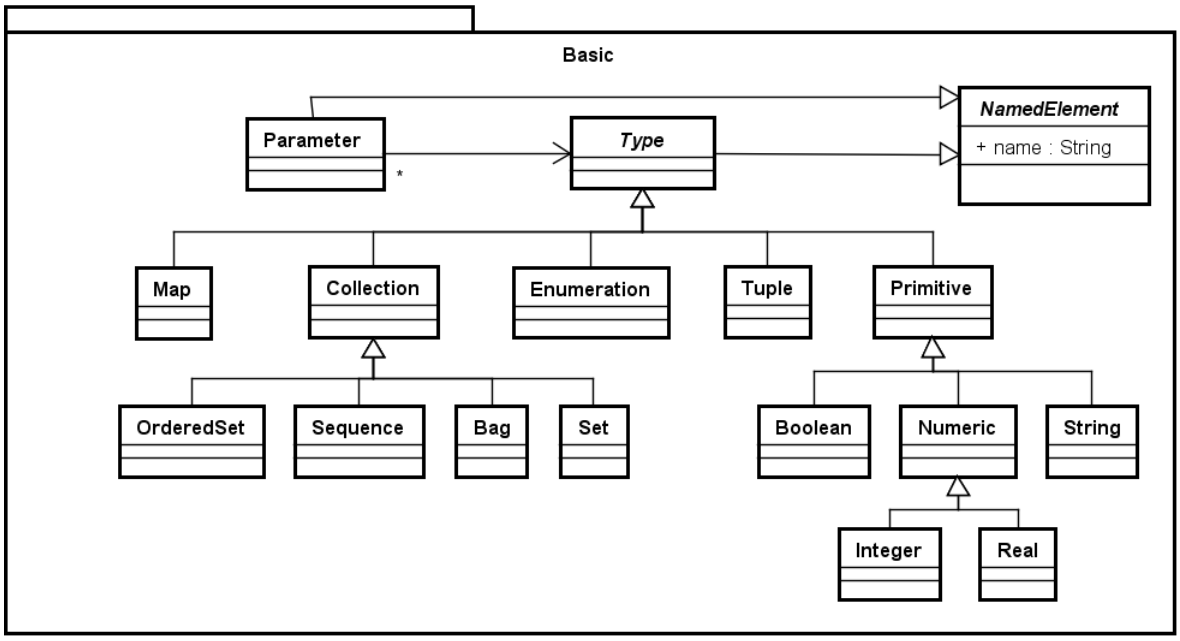

Figura 4.3: Pacote Basic.

Pacote `SpecificLanguageDefinitions`. Ele abrange as duas únicas metaclasses do metamodelo que precisam ser definidas quando a abordagem proposta nesta tese for utilizada em uma determinada linguagem de transformação de modelos, como ilustrado na Fig. 4.4. `CharacteristicKind` é um *enumeration* que define as características que uma transformação especificada em determinada linguagem deve assumir. Para este *enumeration*, as características a serem definidas são literais, como por exemplo, *isAbstract*, *hasIfExpression* e *hasParameter*. Por outro lado, *`RelevanceIndexDefinition`* especifica um índice de relevância (*relevanceIndex*), que significa um grau de influência. Como as metaclasses `Relation`, `Characteristic` e *`TransformationElement`* (posteriormente apresentadas na Fig. 4.5) estendem *`RelevanceIndexDefinition`*, isto significa que um índice de relevância deve ser definido, respectivamente, para: cada relação de uma transformação, cada característica de uma transformação e cada tipo de elemento de uma transformação. O índice de relevância a ser definido deve ser um valor dentro do intervalo entre 0 e 1.

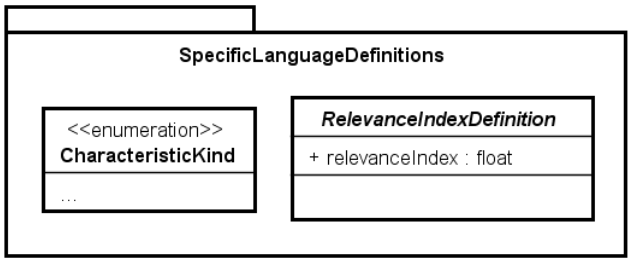

Figura 4.4: Pacote SpecificLanguageDefinitions.

Pacote `TransformationElements`. Ele abrange as relações e características dos elementos que podem ser especificados em uma transformação (*Transformation Elements*).

Como ilustrado na Fig. 4.5, a metaclasse `Relation` representa a dependência entre dois elementos de uma transformação e define dois atributos: o tipo (*kind*) e o índice de relevância (*relevanceIndex*) da relação, este último é estendido da metaclasse `RelevanceIndexDefinition` (Fig. 4.4). O *enumeration* `RelationKind` ilustra os tipos de relação definidos para contemplar as dependências entre os elementos, que são usados pelas métricas para calcular o impacto. Por exemplo, quando uma *function A* invoca uma *constant B* há uma relação do tipo *invokes* entre *A* e *B*, assim como uma relação do tipo *isInvoked* entre *B* e *A*. As relações serão discutidas em detalhes na Subseção 5.3.2.

A metaclasse `Characteristic` representa uma característica que um elemento assume[1] na transformação e define dois atributos: o tipo (*kind*) e o índice de relevância (*relevanceIndex*) da característica, este último é estendido da metaclasse `RelevanceIndexDefinition` (Fig. 4.4). O *enumeration* `CharacteristicKind` (Fig. 4.4) ilustra apenas algumas das características definidas, mais detalhes serão discutidos na Subseção 5.3.1.

O índice de relevância de cada tipo de relação e de cada característica é definido neste trabalho como um valor em um intervalo entre 0 e 1, que significa o grau de influência que cada um deles pode ter dentro da transformação. Esse índice será utilizado pelas métricas definidas para calcular o impacto de mudanças.

Como ilustrado na Fig. 4.5, um `TransformationElement` pertence a um mó-

[1]Quando um elemento assume uma característica significa que ele foi especificado de forma a apresentar esta característica na sua estrutura. Por exemplo, uma regra assume a característica *hasUsingBlock* se ela tem um bloco *using* em sua definição para declaração e inicialização de variáveis.

dulo de transformação[2] (metaclasse `Module`) que, por sua vez, gera um modelo de saída (metaclasse `OutputModel`) com vários elementos do modelo de saída. Um *`TransformationElement`* representa os seguintes elementos que podem ser especificados em uma transformação de modelos:

- `TransformationBlock` - é uma metaclasse que generaliza as regras de transformação, as quais podem receber parâmetros, opcionalmente. Um bloco de transformação pode, opcionalmente, gerar um conjunto de elementos do modelo de saída (`OutputModelElement`) do módulo de transformação;

- `Function` - é uma metaclasse que representa uma função que pode receber parâmetros e ser invocada em qualquer parte da transformação. Ela possui um tipo de retorno (*returnType*), um tipo de contexto (*context*) onde ela pode ser aplicada e uma expressão (*bodyExp*) onde o corpo da função é especificado;

- `Constant` - é uma metaclasse que representa uma constante que pode ser invocada em qualquer parte da transformação. Ela possui uma expressão inicial (*initExpression*) que é opcional e um tipo (*type*).

4.2 Guia de Utilização do Metamodelo

O metamodelo deve ser usado para guiar desenvolvedores na aplicação da abordagem proposta em outras linguagens de transformação. Os pacotes `ChangeImpact` e `TransformationElements` do metamodelo são genéricos para qualquer linguagem de transformação de modelos e não precisam ser estendidos. Por outro lado, o pacote `SpecificLanguageDefinitions` é o único que contém definições específicas para determinada linguagem de transformação e, portanto, precisa ser estendido. Para aplicar o metamodelo proposto no contexto específico de uma linguagem de transformação, é necessário seguir os dois passos como descritos a seguir.

- *Definição de um enumeration de acordo com a linguagem desejada* - o *enumeration* `CharacteristicKind` (Fig. 4.4) deve ser definido, pois em cada linguagem

[2]Um *módulo de transformação* também é chamado apenas de *transformação* ao longo deste documento.

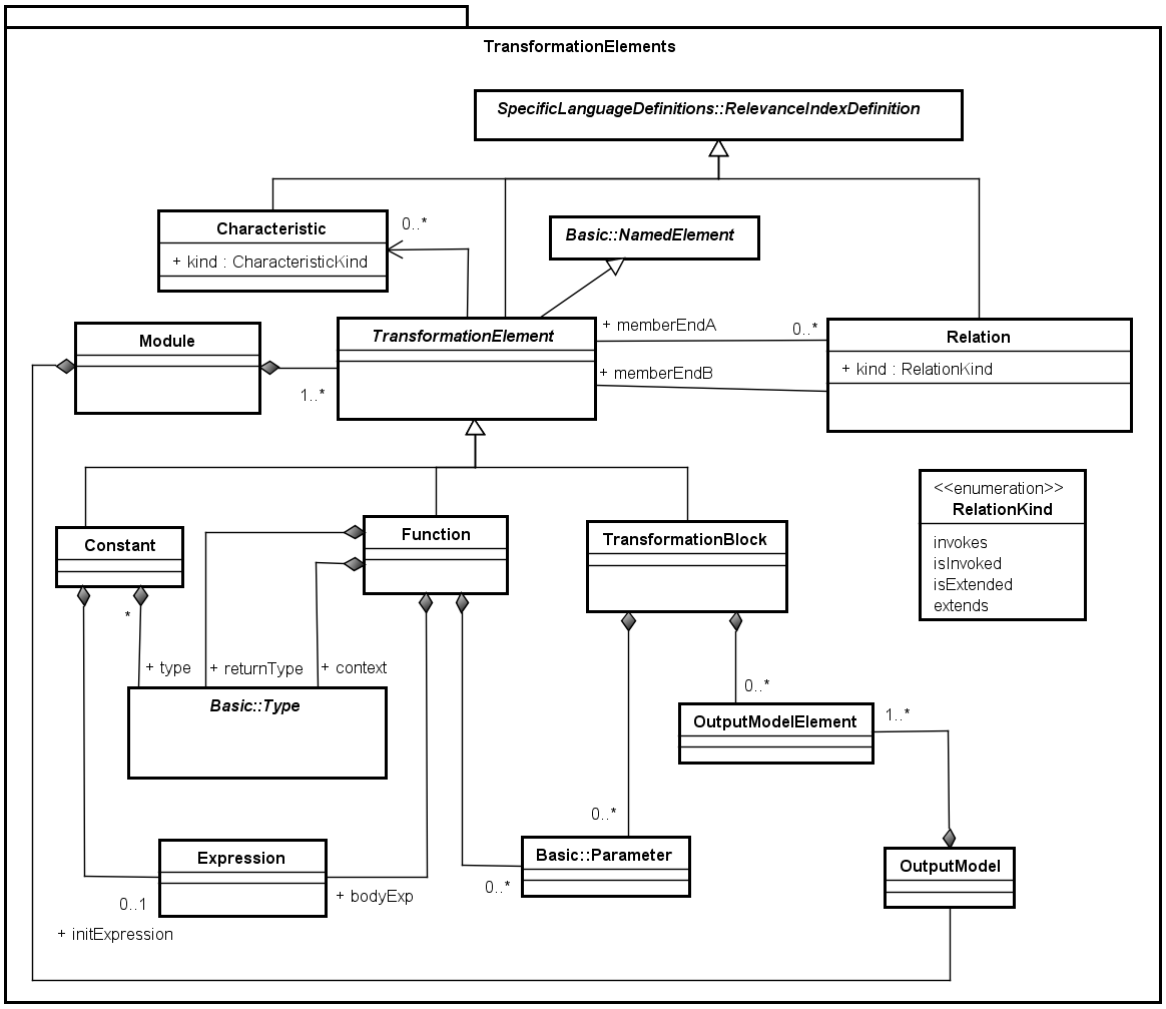

Figura 4.5: Pacote TransformationElements.

de transformação os elementos possuem características próprias, não havendo como generalizá-las para qualquer tipo de linguagem;

- *Definição dos índices de relevância* - devem ser definidos os índices de relevância para: (i) as características de um elemento (metaclasse `Characteristic`); (ii) os tipos de relação (metaclasse `Relation`); e (iii) os elementos da transformação (metaclasse `TransformationElement`). Estas três metaclasses estendem a metaclasse abstrata *`RelevanceIndexDefinition`* e, consequentemente, estendem o atributo `relevanceIndex` especificado nela. Os valores para esses índices de relevância devem ser obtidos através de estudos ou definidos por profissionais experientes na linguagem de transformação para a qual o metamodelo será aplicado. Como cada linguagem de transformação tem suas próprias construções e definições, de fato, os

índices de relevância devem variar de acordo com a linguagem.

Como a abordagem proposta neste trabalho foi aplicada no contexto de transformação de modelos ATL, o metamodelo para análise de impacto de mudanças foi estendido para esta linguagem. De acordo com o guia de utilização do metamodelo, para estendê-lo foi necessário apenas: (i) definir o *enumeration* `CharacteristicKind` para especificar as características próprias de transformações ATL; e (ii) definir os valores para o índice de relevância das metaclasses `Characteristic`, `Relation` e `TransformationElement` através de um estudo empírico, posteriormente descrito na seção 5.3.8.

Capítulo 5

Abordagem para Análise de Impacto de Mudanças

Este capítulo apresenta a abordagem proposta para a análise de impacto de mudanças em transformações de modelos amparada na análise estática, que tem o objetivo de: (i) detectar os elementos impactados por uma mudança e (ii) calcular o valor de impacto desta mudança. Vale ressaltar que a abordagem não aplica a mudança, mas permite aos seus usuários previamente mensurar o impacto de aplicá-la.

No intuito de se definir uma taxonomia para a abordagem proposta, foram seguidos os critérios de classificação apresentados por Lehnert [Lehnert, 2011] para abordagens de análise de impacto de mudanças. Portanto, a taxonomia para a abordagem proposta foi definida da seguinte forma:

- *Escopo da Análise* - a abordagem proposta inspeciona o código fonte das transformações através da análise estática;
- *Granularidade dos Artefatos, Mudanças e Elementos Afetados* - a abordagem proposta trabalha no artefato de transformações de modelos e considera os *transformation blocks*, as *constants* e as *functions* como os elementos a serem modificados em uma transformação. Os tipos de mudança suportados pela abordagem foram obtidos a partir de repositórios com transformações de modelos reais e são de granularidade baixa, isto é, eles podem ser aplicados em qualquer parte dos elementos da transformação. Como elementos afetados por uma mudança, a abordagem proposta considera qualquer um

dos elementos da transformação;

- *Estilo da Análise* - o estilo é exploratório, uma vez que a abordagem proposta fornece aos desenvolvedores métodos para obter os elementos impactados com a mudança e o valor de impacto;

- *Ferramenta de Suporte* - a abordagem proposta conta com o apoio de uma ferramenta de suporte desenvolvida em Java;

- *Escalabilidade* - a abordagem pode ser, futuramente, estendida com novas métricas para calcular o valor de impacto. Ela pode ser usada em projetos de pequeno e de grande porte;

- Linguagens Suportadas - a abordagem proposta é genérica para analisar o impacto em transformações de modelo especificadas em qualquer linguagem de transformação.

Neste capítulo, primeiramente, será apresentada uma visão geral da abordagem proposta (Seção 5.1). Depois, serão apresentados cada um dos três módulos que compõem a abordagem: *Analisador Estático* (Seção 5.2), *Métricas* (Seção 5.3) e *Ferramenta de Suporte* (Seção 5.4). Em seguida, será mostrado como a abordagem proposta identifica os elementos impactados (Seção 5.5) e como ela calcula o valor de impacto de uma mudança (Seção 5.6). Por fim, será apresentado um estudo realizado para se definir como os valores de impacto calculados pela abordagem proposta deveriam ser classificados como um impacto *baixo* ou *alto* (Seção 5.7).

5.1 Visão Geral

Uma visão geral da abordagem proposta neste trabalho é apresentada na Fig. 5.1. Como pode ser observado, o módulo principal chamado *Analisador de Impactos* recebe como entrada a *Mudança* a ser aplicada e uma *Transformação* que contém o elemento a ser modificado. Como resultado, este módulo gera os *Elementos Impactados* (obtidos a partir do analisador estático) e o *Valor de Impacto* para a mudança (calculado a partir das métricas). O *Analisador de Impactos* abrange três módulos como seguem.

- *Analisador Estático* - é uma ferramenta para analisar o código fonte das transformações e extrair informações sobre os seus elementos. Ele fornece uma API para obter essas informações;

- *Métricas* - é o conjunto de métricas definidas seguindo o metamodelo para análise de impacto de mudanças a fim de calcular o valor de impacto;

- *Ferramenta de Suporte* - é uma ferramenta para calcular automaticamente o valor de impacto de uma determinada mudança e obter o conjunto de elementos impactados com esta mudança.

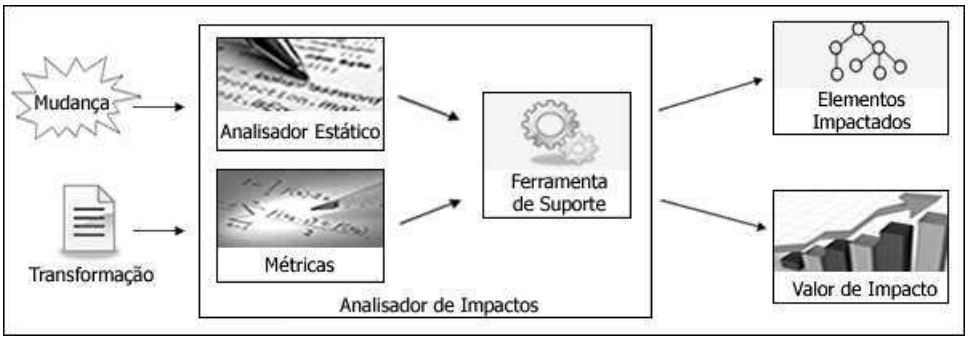

Figura 5.1: Visão geral da abordagem proposta.

A abordagem proposta é independente de qualquer linguagem de transformação de modelos. Portanto, ela pode ser utilizada para analisar o impacto de mudanças em transformações especificadas em qualquer linguagem de transformação. Os módulos *Métricas* e *Ferramenta de Suporte* são genéricos para qualquer linguagem dado que eles não lidam diretamente com os elementos da transformação. Por outro lado, para usar a abordagem proposta em uma determinada linguagem, o módulo *Analisador Estático* deve ser provido de acordo com a linguagem desejada.

5.2 Módulo: Analisador Estático

A abordagem proposta é baseada na análise estática para inspecionar o código fonte das transformações de modelos e extrair informações delas sem executá-las. Para isto, é necessário prover um analisar estático para uma determinada linguagem de transformação. Ele deve ser usado para extrair informações sobre todos os elementos de uma transformação.

Tais informações são importantes para auxiliar a abordagem proposta na identificação dos elementos impactados por uma mudança e na obtenção dos valores necessários para calcular as métricas.

Como para a aplicação da abordagem proposta foi escolhida a linguagem ATL, foi necessário desenvolver um analisador estático específico para esta linguagem. Todos os detalhes deste analisador estático são apresentados posteriormente no Capítulo 6.

No campo da engenharia de software já existem várias ferramentas para análise estática, tais como *FindBugs* para Java e *UNO* para C. No entanto, não foi possível reutilizá-los porque eles são específicos para determinadas linguagens de programação. Se fazia necessário um analisador estático para a linguagem de transformação de modelos ATL, que possui seus próprios conceitos e estrutura. Por exemplo, as transformações ATL são especificadas para transformar algum modelo de entrada em um modelo de saída modificado. No entanto, as informações sobre esses modelos de entrada/saída não são fornecidos pelas ferramentas existentes.

5.3 Módulo: Métricas

No intuito de mensurar o valor do impacto de uma mudança em uma transformação foi proposto um conjunto de métricas. Para defini-las, algumas atividades foram realizadas, como seguem.

- *Estudo de métricas já existentes* - foi feito o estudo de métricas já existentes na literatura em diferentes escopos, tais como métricas para medir a qualidade de programas Java. O intuito era encontrar métricas e reutilizá-las nesta tese. Algumas métricas foram encontradas, porém, nenhuma delas foi reutilizada porque eram muito específicas de contexto;

- *Estudo de caso* - foi elaborado um estudo de caso para o entendimento do que os usuários acreditavam ser importante considerar durante a análise do impacto de uma mudança em transformações de modelos [Vieira e Ramalho, 2014a]. Com o estudo, foram identificados vários critérios adotados pelos participantes para analisar o impacto de mudanças. Todos os detalhes do estudo de caso realizado são apresentados

no Capítulo 8;

- *Elaboração das métricas* - para cada critério identificado com o estudo, uma métrica foi definida. As demais métricas foram definidas com base em estudos sobre análise de impacto de mudanças e baseado também no desenvolvimento com linguagens de transformações de modelos.

As descobertas do estudo de caso associadas a nossa experiência e conhecimento em impacto de mudanças, métricas e transformações de modelos foram cruciais para a definição do conjunto de métricas proposto neste trabalho.

Em resumo, o estudo de caso elaborado contou com a participação de 12 usuários experientes no desenvolvimento de transformações de modelos. O papel deles era: (i) classificar o impacto de um conjunto de mudanças como *baixo* ou *alto*, justificando que critérios foram adotados para a classificação; e (ii) identificar os elementos impactados com a mudança, justificando os critérios adotados. Para a definição das métricas, todos os critérios adotados pelos participantes para analisar o impacto das mudanças foram analisados. Como resultado, segue uma compilação destes critérios.

- *Papel do elemento a ser modificado* - cada elemento desempenha um determinado papel na transformação e possui as suas próprias características. Os participantes indicaram que quanto mais relevantes forem as características de um elemento, maior será o impacto da mudança. Com base neste critério, foi definida a métrica *RTEC* (Seção 5.3.1);

- *Relações entre elementos* - dentro de uma transformação podem existir várias relações entre seus elementos. Os participantes indicaram que quanto mais relações um elemento assumir, maior será a sua relevância dentro da transformação e maior será o impacto da mudança. Com base neste critério, foi definida a métrica *RTERT* (Seção 5.3.2);

- *Número de elementos impactados* - para cada mudança a ser aplicada, os participantes identificaram o elemento a ser modificado dentro da transformação e eles procuraram as possíveis dependências deste elemento (os elementos impactados). Eles indicaram que quanto maior for o número de elementos impactados com a mudança, maior será

o impacto da mudança. Com base neste critério, foi definida a métrica *RITE* (Seção 5.3.3);

- *Tipo da mudança* - o impacto em uma transformação depende também do tipo de mudança a ser aplicada. Alguns tipos de mudança podem impactar mais do que outros. Os participantes indicaram que quanto mais complexo for o tipo de mudança, maior será o seu impacto na transformação. Com base neste critério, foi definida a métrica *RCTAE* (Seção 5.3.4);

- *Impacto no modelo de saída* - algumas mudanças podem afetar o modelo de saída gerado pelo módulo de transformação. Os participantes indicaram que quanto mais o modelo de saída for afetado, maior será o impacto da mudança. Com base neste critério, foi definida a métrica *RCITOM* (Seção 5.3.5).

Para facilitar a interpretação do valor de impacto calculado pelas métricas, foi determinado que ele deve assumir valores num intervalo entre *0* e *1*: quanto mais próximo de *0* ele for, menor será o impacto causado e, por outro lado, quanto mais próximo de *1* ele for, maior será o impacto causado.

Vale ressaltar que as métricas foram definidas seguindo o metamodelo para análise de impacto de mudanças (Capítulo 4), que define de forma genérica os conceitos de uma mudança aplicada à transformações de modelos. A seguir, são apresentadas as sete métricas propostas, das quais cinco foram definidas a partir do estudo conduzido, e suas aplicações[1] para a linguagem ATL (adotada para aplicação da abordagem proposta). Considere $E = \{e_1, ..., e_n\}$ o conjunto de elementos da transformação, onde n é o número total de elementos especificados dentro da transformação em análise.

5.3.1 RTEC

A métrica *RTEC (Relevance of the Transformation Element Characteristics)* determina a relevância do elemento da transformação a ser modificado em termos das características que ele assume. É importante considerar tais características porque elas dão significados e relevâncias diferentes ao elemento, dado que a sua presença (ou não) pode torná-lo mais

[1] Apenas as métricas *RTEC*, *RTERT* e *RCTAE* usam informações intrínsecas para uma dada linguagem de transformação e precisam de adaptação para aplicação nesta linguagem.

ou menos complexo. Quando um elemento assume uma característica, significa que ele foi especificado de forma a apresentar esta característica na sua estrutura.

A métrica *RTEC* é calculada através da função $calcRTEC(e_i)$, onde e_i é o elemento a ser modificado. Conforme apresentado na Equação 5.1, esta função soma os índices de relevância para as características que e_i assume. Esta soma nunca ultrapassa 1, dado que o índice de relevância de cada característica foi definido de forma que a soma deles para as características que um elemento pode assumir sempre é 1. Vale ressaltar que estes índices de relevância foram definidos na metaclasse `Characteristic` (Fig. 4.5) do metamodelo para análise de impacto de mudanças (Capítulo 4). Seja:

- $C = \{c_1, ..., c_s\}$ o conjunto de índices de relevância para as características $c_k (i = 1, ..., s)$ que e_i assume, onde s é o número de características que e_i assume.

$$calcRTEC(e_i) = \sum_{k=1}^{s} c_k \tag{5.1}$$

O índice de relevância representa o grau de influência que uma característica pode ter em uma transformação de forma que a sua presença (ou não) pode tornar a manutenção da transformação mais fácil (ou não). Portanto, quanto maior for o número de características que um elemento assume, mais relevante[2] considera-se que ele seja. Por outro lado, se o elemento não assumir nenhuma característica, a métrica *RTEC* deve resultar em 0 e menos relevante considera-se que ele seja.

Aplicação da métrica RTEC à linguagem ATL. De acordo com o metamodelo para análise de impacto de mudanças, para aplicar a métrica *RTEC* no contexto específico de ATL é necessário definir: (i) o *enumeration* `CharacteristicKind` (Fig. 4.4) com características próprias dos elementos de uma transformação ATL; e (ii) os índices de relevância para tais características de um elemento (metaclasse `Characteristic`, Fig. 4.5).

Portanto, foi analisado cada elemento de várias transformações ATL para que fosse possível a identificação das características que eles poderiam opcionalmente assumir. Como resultado, foram encontradas e classificadas 13 características.

[2]Relevante no contexto de análise de impacto de mudanças.

A Tabela 5.1 detalha as características, onde: (i) a coluna 1 mostra o nome da característica; (ii) a coluna 2 mostra os elementos da transformação ATL que podem assumir a característica. Por exemplo, uma *matched rule* pode, opcionalmente, assumir as características *isAbstract* e *hasUsingBlock*, isto é, ela pode ser uma regra abstrata e especificar um bloco `using` que é responsável pela declaração de variáveis; e (iii) a coluna 3 mostra o índice de relevância definido para a característica.

Tabela 5.1: Classificação das características.

Característica	**Elemento da Transf. ATL**	**Índice de Relevância**
isAbstract	Matched Rule	0.16
	Lazy Matched Rule	0.16
isEntrypoint	Called Rule	0.15
hasUsingBlock	Matched Rule	0.16
	Lazy Matched Rule	0.16
	Called Rule	0.13
hasNotPrimitiveContextType	Helper	0.17
	Attribute	0.22
hasNotPrimitiveReturnType	Helper	0.17
hasNotPrimitiveType	Attribute	0.18
hasFilter	Matched Rule	0.18
	Lazy Matched Rule	0.18
hasParameter	Called Rule	0.15
	Helper	0.19
hasImperativeBlock	Matched Rule	0.16
	Lazy Matched Rule	0.16
	Called Rule	0.13
hasToBlock	Called Rule	0.16
hasIfExpression	Helper	0.16
	Attribute	0.20
	Matched Rule	0.16
	Lazy Matched Rule	0.16
	Called Rule	0.13
hasLetExpression	Helper	0.12
	Attribute	0.16
hasCollectionExpression	Helper	0.19
	Attribute	0.24
	Matched Rule	0.18
	Lazy Matched Rule	0.18
	Called Rule	0.15

Os índices de relevância foram definidos através de um estudo onde os participantes com muita experiência em ATL tinham que atribuir um valor entre 0 e 5 (onde 0 significa *nenhuma relevância* e 5 significa *extrema relevância*) para cada característica dos elementos de uma transformação ATL, de acordo com a relevância que os participantes achavam que as características representavam na transformação. Foi calculada a média das respostas dos par-

ticipantes para cada característica. Depois, cada média calculada foi normalizada de forma que a soma dos valores normalizados (índices de relevância) para cada característica deveria ser 1. Por exemplo, a soma dos índices de relevância para as características que um *helper* pode assumir deve ser 1 (veja as linhas 7, 9, 14, 19, 24 e 26 da Tabela 5.1). A Seção 5.3.8 mostra mais detalhes deste estudo para definição dos índices de relevância.

Algumas características podem ser assumidas por apenas um elemento específico. Por exemplo, apenas *called rules* podem assumir a característica *isEntrypoint*[3], como apresentado na linha 3 da Tabela 5.1. Os elementos podem assumir nenhuma ou várias características, dependendo da sua especificação dentro da transformação.

5.3.2 RTERT

A métrica *RTERT (Relevance of the Transformation Element Relation Types)* determina a relevância do elemento da transformação a ser modificado em termos dos tipos de relação que ele assume, isto é, suas dependências. Para a identificação daqueles tipos de relações que envolvem dependência entre os elementos, foram observadas as relações entre os elementos de várias transformações. Como resultado da análise, foram identificados e classificados quatro tipos de relação: (i) *invokes*, quando um elemento da transformação invoca outro; (ii) *isInvoked*, quando um elemento da transformação é invocado por um outro; (iii) *extends*, quando um elemento da transformação estende outro; e (iv) *isExtended*, quando um elemento da transformação é estendido por outro. Um elemento da transformação pode não assumir nenhum tipo de relação ou pode, simultaneamente, assumir vários, dependendo de suas dependências dentro da transformação.

É importante considerar os tipos de relação ao mensurar o impacto de uma mudança porque eles indicam a relevância do elemento a ser modificado através de suas dependências: quanto maior for o número de relações que o elemento assume, mais elementos na transformação estão relacionados com ele e potencialmente dependem dele. Por exemplo, um elemento *A* da transformação que assume os tipos de relação *isExtended*, *invokes* e *isInvoked* possui mais dependências dentro da transformação do que um outro elemento que assume apenas a relação *isInvoked*, já que o elemento *A* assume um maior número de relações.

[3]Uma *called rule entrypoint* é implicitamente invocada no início da execução da transformação, uma vez que a fase de inicialização do módulo tenha sido concluída.

A métrica ***RTERT*** é calculada através da função $calcRTERT(e_i)$, onde e_i representa o elemento a ser modificado. Conforme apresentado na Equação 5.2, esta função soma os índices de relevância para cada relação que e_i assume, considerando todos os tipos de relação. Seja:

- $R = \{r_1, ..., r_m\}$ o conjunto de índices de relevância para a relação $r_k(i = 1, ..., m)$ que um elemento e_i assume, onde m é o número de relações que e_i assume, considerando todos os tipos de relação.

$$calcRTERT(e_i) = \sum_{k=1}^{m} r_k \tag{5.2}$$

Aplicação da métrica RTERT à linguagem ATL. De acordo com o metamodelo para análise de impacto de mudanças, para aplicar a métrica ***RTERT*** no contexto específico de ATL é necessário definir os índices de relevância para os tipos de relação (metaclasse `Relation`, Fig. 4.5) que um elemento pode assumir. Vale ressaltar que os tipos de relação (*enumeration* `RelationKind`, Fig. 4.5) definidos já são genéricos para qualquer linguagem de transformação, por isso não precisam ser redefinidos. Porém, os índices de relevância para estas relações podem ser diferentes dependendo da linguagem de transformação, por isso é necessário que sejam redefinidos para cada linguagem em que a métrica for aplicada.

Similarmente à métrica ***RTEC***, foi definido um índice de relevância para cada tipo de relação para representar o grau de influência que um tipo de relação pode ter em toda a transformação, de forma que a sua presença (ou não) pode tornar a manutenção da transformação mais fácil (ou não). Cada tipo de relação pode causar um impacto diferente no elemento, portanto, deve ter um índice de relevância diferente. A Tabela 5.2 detalha os tipos de relações, onde: (i) a coluna 1 mostra o tipo de relação; (ii) a coluna 2 mostra os elementos da transformação ATL que podem assumir o tipo de relação; e (iii) a coluna 3 mostra os índices de relevância que foram definidos para os tipos de relação. A soma dos índices de relevância para cada tipo de relação que um elemento pode assumir deve ser 1.

Os índices de relevância foram definidos através de um estudo onde os participantes tinham que atribuir um valor entre 0 e 5 (onde 0 significa *nenhuma relevância* e 5 significa

Tabela 5.2: Classificação dos tipos de relação.

Tipo de Relação	Elemento da Transf. ATL	Índice de Relevância
invokes	Helper	0.49
	Attribute	0.49
	Matched Rule	0.34
	Lazy Matched Rule	0.25
	Called Rule	0.49
isInvoked	Helper	0.51
	Attribute	0.51
	Lazy Matched Rule	0.27
	Called Rule	0.51
extends	Matched Rule	0.30
	Lazy Matched Rule	0.22
isExtended	Matched Rule	0.36
	Lazy Matched Rule	0.26

extrema relevância) para cada tipo de relação, de acordo com a relevância que eles achavam que o tipo de relação representava na transformação. Foi calculada a média das respostas dos participantes para cada tipo de relação. Depois, cada média calculada foi normalizada de forma que a soma dos valores normalizados (índices de relevância) para cada tipo de relação que um elemento pode assumir deveria ser 1. A Seção 5.3.8 mostra mais detalhes deste estudo para definição dos índices de relevância.

5.3.3 RITE

A métrica *RITE (Relevance of the Impacted Transformation Elements)* determina a relevância do elemento da transformação a ser modificado em termos dos elementos impactados quando uma mudança é aplicada. Como neste trabalho foi adotada a análise estática, a abordagem proposta considera como impactados todos os elementos que invocam ou estendem o elemento a ser modificado, mesmo se eles não o invocarem em tempo de execução.

É importante considerar os elementos impactados para mensurar o impacto de uma mudança, pois eles indicam as partes da transformação que devem ser adaptadas depois que a mudança é aplicada: quanto maior for o número de elementos impactados, maior será o esforço para adaptá-los depois da mudança. Por exemplo, é mais fácil aplicar uma mudança a uma regra (*transformation block*) que impacta dois elementos da transformação do que a uma outra regra que impacta sete elementos.

A métrica ***RITE*** é calculada através da função $calcRITE(e_i)$, onde e_i representa o elemento a ser modificado. Conforme apresentado na Equação 5.3, esta função calcula a razão entre o número de elementos impactados com a mudança (t) e o número total de elementos da transformação (n). Seja:

- $E' = \{e'_1, ..., e'_t\}$; $t < n$; $E' \subseteq E$ o subconjunto de elementos impactados $e'_i (i = 1, ..., t)$ quando uma mudança é aplicada a e_i, onde t é o número de elementos impactados.

$$calcRITE(e_i) = \frac{t}{n} \tag{5.3}$$

5.3.4 RCTAE

A métrica ***RCTAE (Relevance of the Change Type Applied to an Element)*** determina a relevância do elemento da transformação a ser modificado em termos da dificuldade de se aplicar um determinado tipo de mudança. Essa dificuldade foi mensurada através do número de ações necessárias para aplicar a mudança, bem como da relevância do elemento a ser modificado. É importante considerar esta métrica porque cada mudança causa um impacto diferente no elemento, dependendo da sua dificuldade. Se essa métrica não fosse considerada, tipos de mudança como *removeTransformationBlock* e *changeTransformationBlockName* teriam o mesmo impacto. No entanto, o impacto de se remover uma regra é geralmente maior do que o impacto de se modificar o nome desta mesma regra. Uma tabela com todas as ações possíveis a serem executadas para aplicar cada tipo de mudança foi definida e é apresentada no Apêndice B deste documento.

É importante ressaltar que o número de ações necessárias para aplicar um dado tipo de mudança em um elemento é automaticamente obtido pela ferramenta de suporte a partir da lista de ações possíveis definidas no Apêndice B, que é analisada para a identificação das ações realmente necessárias para aplicar tal mudança (ao invés de todas as ações possíveis). Por exemplo, para aplicar o tipo de mudança *removeConstant* a uma constante *A1*, um total de quatro ações são possíveis: (i) verificar se algum elemento da transformação ou de outro módulo de transformação do projeto invoca *A1*; (ii) se *A1* for invocada, remover as invocações; (iii) se *A1* for invocada, adaptar os elementos que a invocam após a remoção,

se necessário; e (iv) remover *A1* da transformação. No entanto, apenas duas destas ações (a primeira e a última) são de fato necessárias para remover essa constante, uma vez que ela não é invocada.

Além da dificuldade de se aplicar um tipo de mudança, a métrica *RCTAE* também considera a relevância do elemento a ser modificado. Dado que cada tipo de elemento tem um papel diferente dentro da transformação, eles devem ter uma relevância diferente e devem causar um impacto diferente quando uma mudança é aplicada.

A métrica *RCTAE* é calculada através da função $calcRCTAE(e_i, ch)$, onde e_i representa o elemento a ser modificado e ch representa o tipo de mudança a ser aplicado. Conforme apresentado na Equação 5.4, esta função calcula a razão entre o número de ações necessárias para aplicar a mudança ch e o número máximo de ações possíveis para aplicar uma mudança (considerando todos os tipos de mudança) (p/d). Depois, é calculada a média entre este resultado e o índice de relevância do elemento a ser modificado e_i. Sejam:

- $A = \{a_1, ..., a_p\}$ o conjunto de ações $a_i (i = 1, ..., p)$ necessárias que devem ser executadas para aplicar um dado tipo de mudança ch a um elemento e_i da transformação, onde p é o número de ações necessárias;
- d o número máximo de ações possíveis para aplicar uma mudança, considerando todos os tipos de mudança que foram definidos. Para obter este número, as ações possíveis para cada tipo de mudança foram enumeradas. Dentre elas, o número máximo de ações possíveis é 08 (para o tipo de mudança *removeOutputModelElement*);
- b é o índice de relevância para o elemento a ser modificado.

$$calcRCTAE(e_i, ch) = \frac{p/d + b}{2} \tag{5.4}$$

Aplicação da métrica RCTAE à linguagem ATL. De acordo com o metamodelo para análise de impacto de mudanças, para aplicar a métrica *RCTAE* no contexto específico de ATL é necessário definir os índices de relevância para os elementos de uma transformação ATL (metaclasse `TransformationElement`, Fig. 4.5).

Cada elemento da transformação tem uma relevância diferente, pois cada um tem um papel diferente na transformação. Por exemplo, em ATL os *helpers* são usados como métodos

para retornar algum valor enquanto que as *matched rules* são usadas para gerar elementos de destino a partir de elementos de origem. Assim, *helpers* e *matched rules* podem causar um impacto diferente na transformação quando modificados. Portanto, para considerar esta relevância na métrica *RCTAE* foi definido um valor diferente para cada elemento da transformação, isto é, um diferente índice de relevância. A Tabela 5.3 mostra os elementos de uma transformação ATL e os índices de relevância definidos.

Tabela 5.3: Classificação dos tipos de elementos de uma transformação ATL.

Elemento da Transformação ATL	**Índice de Relevância**
Called Rule	0.67
Attribute	0.76
Lazy Matched Rule	0.78
Helper	0.82
Matched Rule	0.89

O resultado da métrica *RCTAE* nunca poderá ser 1, dado que os índices de relevância para os elementos da transformação foram definidos através de um estudo onde os participantes tinham que atribuir um valor entre 0 e 5 para cada um dos elementos de uma transformação ATL. Para cada elemento da transformação, foi calculada a média das respostas dos participantes e, em seguida, o valor resultante foi normalizado para que ficasse entre 0 e 1. Como após a normalização nenhum índice de relevância atingiu o valor 1, esta métrica nunca poderá resultar em 1. A Seção 5.3.8 mostra mais detalhes deste estudo para definição dos índices de relevância.

5.3.5 RCITOM

A métrica *RCITOM (Relevance of the Change Impact in the Transformation Output Model)* determina a relevância do elemento da transformação a ser modificado em termos do impacto causado no modelo de saída do módulo de transformação. Quando determinados tipos de mudança são aplicados, os elementos do modelo de saída dos *transformation blocks* são afetados e, consequentemente, o modelo de saída gerado pelo módulo de transformação é impactado.

A Fig. 5.2 ilustra um trecho do pacote `TransformationElements` pertencente ao

metamodelo para análise de impacto de mudanças (Capítulo 4). Como se pode observar, um *transformation block* é um elemento da transformação que representa uma regra de transformação podendo, opcionalmente, gerar vários elementos no modelo de saída do módulo de transformação. Para a métrica *RCITOM*, são considerados apenas os *transformation blocks* que de fato geram elementos no modelo de saída.

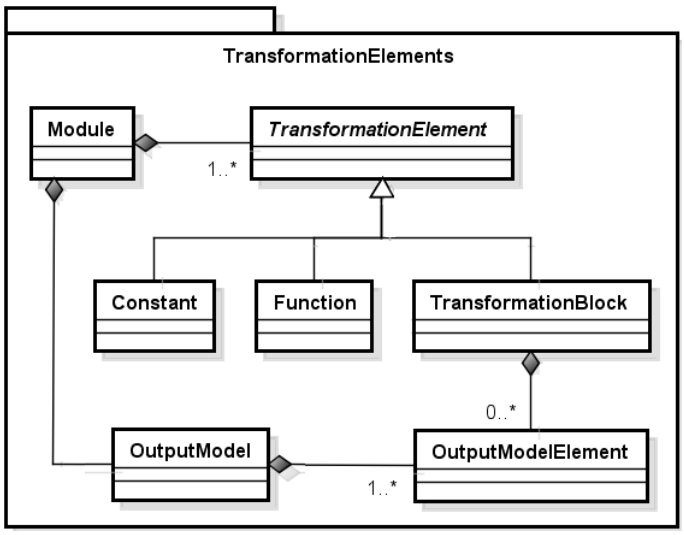

Figura 5.2: Trecho do pacote TransformationElements.

Um elemento do modelo de saída é considerado como impactado quando ele é afetado por uma mudança aplicada a um elemento do módulo de transformação. Por exemplo, suponha que uma dada constante deve ser removida da transformação. Portanto, todos os elementos do modelo de saída que fazem referência a essa constante serão afetados e, consequentemente, os *transformation blocks* que geram estes elementos do modelo de saída afetados também serão impactados. Os seguintes tipos de mudança podem impactar diretamente os elementos do modelo de saída dos *transformation blocks* e, consequentemente, o modelo de saída do módulo de transformação: *changeFunction/removeFunction* (quando a função é referenciada por qualquer elemento do modelo de saída), *changeConstant/removeConstant* (quando a constante é referenciada por qualquer elemento do modelo de saída), *removeTransformationBlockFilter / changeTransformationBlockFilter / addTransformationBlockFilter, removeTransformationBlock / addTransformationBlock, removeTransformationBlockInheritance, removeOutputModelElement / addOutputModelElement, changeOutputModelElementType* e *addOutputModelElementBinding / removeOutputModelElementBinding / changeOutputModelElementBinding*. Por outro lado, os tipos de mudança,

como *changeFunctionName* e *changeTransformationBlockName* não impactam o modelo de saída de uma transformação porque os elementos do modelo de saída de suas regras não são afetados com a mudança.

A métrica *RCITOM* é calculada através da função $calcRCITOM(e_i)$, onde e_i representa o elemento da transformação a ser modificado. Conforme apresentado na Equação 5.5, esta função primeiro calcula a razão entre o número de *transformation blocks* impactados com a mudança e o número total de *transformation blocks* (z/x) do módulo de transformação. É importante mensurar como a mudança afeta os *transformation blocks*, pois são eles que geram os elementos no modelo de saída da transformação. Depois, a função calcula a razão entre o número de elementos do modelo de saída impactados (g) e o número total de elementos do modelo de saída (f) para cada *transformation block* impactado, e em seguida, calcula a média para estes resultados $((\sum_{k=1}^{z} g/f)/z)$. É importante mensurar como a mudança afeta os elementos do modelo de saída de cada *transformation block* impactado, pois eles impactam diretamente o modelo de saída do módulo de transformação. Por fim, o resultado do numerador é dividido por 2 para assegurar que o resultado da função seja um valor entre 0 e 1. Sejam:

- $H = \{h_1, ..., h_x\}$ o conjunto de todos os *transformation blocks* $h_i(i = 1, ..., x)$ da transformação que geram elementos no modelo de saída, onde x é o número destes *transformation blocks*;

- $H' = \{h'_1, ..., h'_z\}$; $z <= x$; $H' \subseteq H$ o subconjunto de *transformation blocks* $h'_i(i = 1, ..., z)$ da transformação cujo pelo menos um de seus elementos do modelo de saída é impactado com a mudança[4], onde z é o número de *transformation blocks* impactados;

- $Q = \{q_1, ..., q_f\}$ o conjunto de todos os elementos do modelo de saída $q_i(i = 1, ..., f)$ de um *transformation block*, onde f é o número destes elementos;

- $Q' = \{q'_1, ..., q'_g\}$; $g <= f$; $Q' \subseteq Q$ o subconjunto de elementos do modelo de saída de um *transformation block* que são impactados $q'_i(i = 1, ..., g)$ com a mudança, onde g é o número destes elementos impactados.

[4]Se pelo menos um dos elementos do modelo de saída de um *transformation block* é impactado com a mudança, o próprio *transformation block* também é considerado impactado.

$$calcRCITOM(e_i) = \frac{z/x + (\sum_{k=1}^{z} g/f)/z}{2} \tag{5.5}$$

5.3.6 ROMTC

A métrica *ROMTC (Relevance of the Output Model in Transformation Chains)* determina a relevância do elemento da transformação a ser modificado e_i considerando o impacto nos modelos de saída em cadeias de transformações. Uma cadeia de transformações é um conjunto de módulos de transformações onde o modelo de saída gerado por uma transformação é usado como modelo de entrada em outra transformação. Em uma cadeia de transformações, quando alguma mudança impacta o modelo de saída de uma transformação, outras transformações que utilizam este modelo de saída como modelo de entrada também serão impactadas. Além disso, este impacto pode ser causado em cascata quando a transformação impactada com a mudança também impacta a próxima transformação da cadeia e, assim por diante. Dependendo da mudança, toda a cadeia pode ser impactada. Consequentemente, o projeto pode se tornar inconsistente se as transformações impactadas não forem adaptadas com a mudança.

A diferença entre as métricas *ROMTC* e *RCITOM* é que esta última não considera o impacto na cadeia de transformações, mas apenas o impacto no modelo de saída gerado pela própria transformação onde a mudança é aplicada.

Como previamente apresentado na Fig. 5.2 (trecho do metamodelo de mudanças), um módulo de transformação pode conter vários *transformation blocks*. Um módulo de transformação dentro de uma cadeia de transformações é impactado quando pelo menos um de seus *transformation blocks* é impactado, isto é, quando um *transformation block* recebe como entrada um elemento do modelo de saída impactado com a mudança. Por exemplo, suponha que em uma cadeia de transformações um dado *transformation block R1* deve ser removido da transformação *T1*. Portanto, todos os elementos do modelo de saída deste *transformation block* (*R1*) serão impactados. Suponha também que um dado *transformation block R2* da transformação *T2* na cadeia de transformações recebe como entrada um dos elementos do modelo gerado pelo *transformation block R1*. Portanto, como o *transformation block R2* faz referência a um elemento do modelo impactado, ele também é considerado impactado.

Consequentemente, a transformação *T2* na cadeia é impactada. É importante ressaltar que, dentro de uma transformação, mais que um *transformation block* pode fazer referência a um elemento do modelo impactado e, consequentemente, mais que um *transformation block* pode ser impactado.

A métrica ***ROMTC*** é calculada através da função $calcROMTC(e_i)$, onde e_i representa o elemento a ser modificado. Conforme apresentado na Equação 5.6, esta função primeiro calcula a razão entre o número de transformações (da cadeia) impactadas com a mudança e o número total de transformações existentes na cadeia ($u/(r-1)$), exceto a própria transformação a ser modificada. Depois, a função calcula a razão entre o número de *transformation blocks* impactados (h) e o número total de *transformation blocks* (v) para cada transformação impactada, e em seguida, calcula a média para estes resultados ($(\sum_{k=1}^{u} h/v)/u$). Por fim, o resultado do numerador é dividido por 2 para assegurar que o resultado da função seja um valor entre 0 e 1. Sejam:

- $M = \{m_1, ..., m_r\}$ o conjunto de todas as transformações $m_i(i = 1, ..., r)$ que fazem parte de uma cadeia, onde r é o número destas transformações;
- $M' = \{m'_1, ..., m'_u\}$; $u < r$; $M' \subseteq M$ o subconjunto de transformações $m'_i(i = 1, ..., u)$ (que fazem parte de uma cadeia) impactadas com a mudança, onde u é o número destas transformações impactadas;
- $S = \{s_1, ..., s_v\}$ o conjunto de todos os *transformation blocks* (de uma transformação) $s_i(i = 1, ..., v)$ que recebem como entrada um elemento do modelo, onde v é o número total destes *transformation blocks*;
- $S' = \{s'_1, ..., s'_h\}$; $h <= v$; $S' \subseteq S$ o subconjunto de *transformation blocks* (de uma transformação) que recebem como entrada um elemento do modelo e são impactados com a mudança $s'_i(i = 1, ..., h)$, onde h é o número destes *transformation blocks* impactados.

$$calcROMTC(e_i) = \frac{u/(r-1) + (\sum_{k=1}^{u} h/v)/u}{2} \tag{5.6}$$

5.3.7 RTEATM

A métrica *RTEATM (Relevance of the Transformation Element Across Transformation Modules)* determina a relevância do elemento da transformação a ser modificado e_i considerando o seu uso em todas as transformações do projeto, ou seja, quando e_i é referenciado dentro de outros módulos de transformação.

As métricas anteriormente apresentadas consideram o impacto de uma mudança apenas na mesma transformação em que o elemento a ser modificado é especificado. No caso de cadeias, a métrica *ROMTC* considera o impacto da mudança apenas na cadeia que engloba a transformação onde o elemento a ser modificado é especificado. No entanto, também é importante considerar o impacto em outras transformações que fazem referência ao elemento a ser modificado e em suas cadeias (se houver), uma vez que o projeto pode se tornar inconsistente se um elemento é modificado e suas referências não são adaptadas em todo o projeto.

A métrica *RTEATM* considera o impacto da mudança em todas as transformações do projeto: ela calcula as métricas para todas as transformações impactadas, até mesmo para ela própria (*RTEATM*) recursivamente. Como a abordagem proposta é baseada na análise estática, essa métrica considera como impactadas as transformações: (i) que especificam elementos que invocam ou estendem o elemento a ser modificado, mesmo se eles não o invocarem em tempo de execução; e (ii) cujo modelo de saída é impactado com a mudança.

A métrica *RTEATM* é calculada através da função $calcRTEATM(e_i, t_i)$, onde e_i representa o elemento a ser modificado e t_i representa a transformação na qual e_i é especificado. Conforme apresentado na Equação 5.7, esta função calcula as métricas *RTERT*, *RITE*, *RCITOM*, *ROMTC* e *RTEATM* para cada transformação impactada e, em seguida, ela calcula a média do resultado destas 5 métricas. Apenas as métricas *RTEC* e *RCTAE* não foram usadas no cálculo da métrica *RTEATM*: elas calculam a relevância em termos das características do elemento a ser modificado e do tipo de mudança, respectivamente, então seus valores permanecerão os mesmos para e_i em qualquer transformação do projeto. Sejam:

- $T = \{t_1, ..., t_c\}$ o conjunto de todas as transformações $t_i (i = 1, ..., c)$ do projeto, onde c é o número destas transformações;
- $T' = \{t'_1, ..., t'_a\}$; $a < c$; $T' \subseteq T$ o subconjunto de transformações $t'_i (i = 1, ..., a)$ do

projeto impactadas com a mudança, onde a é o número destas transformações impactadas;

- A a função $calcRTERT(e_i, t_i)$ que calcula a métrica ***RTERT***;
- B a função $calcRITE(e_i, t_i)$ que calcula a métrica ***RITE***;
- C a função $calcRCITOM(e_i, t_i)$ que calcula a métrica ***RCITOM***;
- D a função $calcROMTC(e_i, t_i)$ que calcula a métrica ***ROMTC***;
- E a função $calcRTEATM(e_i, t_i)$ que calcula a métrica ***RTEATM***.

$$calcRTEATM(e_i) = \frac{\sum_{k=1}^{a}(A + B + C + D + E)/5}{a} \tag{5.7}$$

Como se pode observar, a função que calcula a métrica ***RTEATM*** recebe também como parâmetro a transformação t_i na qual o elemento a ser modificado é especificado, pois ela pode ser chamada recursivamente. Por este motivo, a ferramenta de suporte oferece também uma versão das funções para calcular as métricas ***RTERT***, ***RITE***, ***RCITOM***, ***ROMTC*** e ***RTEATM*** aceitando o parâmetro t_i.

O número de métricas utilizadas para calcular a métrica ***RTEATM*** pode ser escolhido de acordo com a preferência do usuário. Por exemplo, se no projeto não há nenhuma cadeia de transformações, então a métrica ***ROMTC*** pode ser desconsiderada.

5.3.8 Definição dos Índices de Relevância

As métricas ***RTEC***, ***RTERT*** e ***RCTAE*** consideram índices de relevância para calcular o impacto de mudanças. Portanto, para defini-los um estudo foi conduzido com o objetivo de obter um número que represente o índice de cada uma das relevâncias.

Para participar deste estudo, foram selecionadas 09 pessoas experientes em ATL (com pelo menos um ano de experiência) e integrantes de diferentes grupos de pesquisa. Dentre elas haviam estudantes (de graduação e pós-graduação), professores e pesquisadores. O

estudo era composto por um formulário[5], onde os participantes tinham que atribuir um valor em um intervalo de 0 a 5 (onde 0 significa *nenhuma relevância* e 5 significa *extrema relevância*) para cada:

- Característica que um elemento pode assumir (métrica *RTEC*);
- Tipo de relação que um elemento pode assumir (métrica *RTERT*); e
- Tipo de elemento da transformação (métrica *RCTAE*).

Os participantes do estudo tinham que considerar a relevância que eles acreditavam que cada um destes conceitos representam dentro da transformação, bem como observar o que e como eles podem influenciar o impacto de modificar um elemento da transformação. A tabela completa com as respostas dos participantes e com os cálculos para a definição dos índices de relevância é encontrada no site do projeto [Vieira e Ramalho, 2014b].

Os resultados foram coletados e, em relação à métrica *RTEC*, para cada característica que um elemento pode assumir foi calculada a média dos valores atribuídos pelos participantes. Depois, cada média calculada foi normalizada de forma que a soma dos valores normalizados (índices de relevância) para cada característica de um determinado elemento fosse 1. Por exemplo, a soma dos índices de relevância para as características que um *helper* pode assumir deve ser 1. A Tabela 5.1 apresentada anteriormente mostra os índices de relevância usados na métrica *RTEC* obtidos com este estudo.

Em relação à métrica *RTERT*, para cada tipo de relação foi calculada a média dos valores atribuídos pelos participantes. Depois, cada valor foi normalizado de forma que a soma dos índices de relevância para cada tipo de relação que um elemento pode assumir fosse 1. A Tabela 5.2 apresentada anteriormente mostra os índices de relevância obtidos com este estudo para a métrica *RTERT*.

Já para a definição dos índices de relevância utilizados na métrica *RCTAE*, para cada tipo de elemento da transformação foi calculada a média dos valores atribuídos pelos participantes. Depois, cada valor foi normalizado de forma que cada índice de relevância assumisse um valor entre 0 e 1. Neste caso, a diferença é que a soma dos índices de relevância não deve ser 1 como nas outras métricas. A Tabela 5.3 apresentada anteriormente mostra os elementos de uma transformação ATL e os índices de relevância obtidos a partir do estudo.

[5] http://goo.gl/5zUA9z

Depois que todos os índices de relevância foram obtidos, eles foram definidos no módulo referente à Ferramenta de Suporte da abordagem proposta. Eles são valores fixos fornecidos pela ferramenta de suporte quando as métricas *RTEC*, *RTERT* e *RCTAE* são calculadas. Por exemplo, para calcular a métrica *RTEC* a ferramenta de suporte identifica as características que o elemento a ser modificado assume e soma os seus índices de relevância já pré-definidos na ferramenta.

5.3.9 Discussões Gerais

As métricas definidas são gerais para medir o impacto de mudanças em transformações de modelos especificadas em qualquer linguagem, uma vez que as linguagens de transformação de modelos especificam transformações de maneira similar. Por exemplo, elas especificam regras que podem invocar ou ser invocadas por outras regras, bem como funções e constantes que podem ser utilizadas ao longo da transformação.

As características semelhantes presentes nas linguagens de transformação permitem que as métricas propostas possam ser reutilizadas para medir o impacto de mudanças em transformações especificadas em linguagens diferentes. Por exemplo, os desenvolvedores que especificam suas transformações na linguagem QVT podem reutilizar as métricas, uma vez que as informações necessárias para calculá-las podem ser obtidas a partir de qualquer transformação QVT. Para calcular a métrica *RTERT* para um elemento e_i, por exemplo, é necessário identificar quais os tipos de relações que e_i assume dentro da transformação. Esta informação está disponível em qualquer transformação QVT. Portanto, um analisador estático para QVT poderia ser desenvolvido para ajudar os desenvolvedores na extração de tal informação e, assim, eles poderiam calcular o impacto de mudanças em transformações QVT reutilizando as métricas definidas.

Vale ressaltar que os índices de relevância usados em algumas das métricas, embora tenham sido baseados em um estudo empírico, podem ser ajustados de acordo com a necessidade de quem vai utilizar a abordagem. Portanto, para reutilizar as métricas propostas a fim de analisar o impacto de mudanças em transformações especificadas em qualquer linguagem, é necessário: (i) definir as características que cada elemento da transformação pode assumir; (ii) definir os índices de relevância para tais características, assim como para os tipos de relação que um elemento pode assumir e para os tipos de elementos da transforma-

ção; e (iii) desenvolver um analisador estático para a linguagem desejada ou qualquer outra ferramenta de suporte a fim de recuperar, a partir da transformação, a informação necessária para calcular as métricas.

Além disso, algumas das métricas também podem ser adotadas para calcular o impacto de mudanças em programas especificados em linguagens de propósito geral, tais como Java. Em uma classe Java se definem métodos e atributos, enquanto que em uma transformação se definem *helpers* e regras. Portanto, para reutilizar as métricas os desenvolvedores devem adaptá-las de acordo com o contexto de classes Java. Por exemplo, suponha que se deseja analisar o impacto de remover um método Java de uma classe. A métrica *RTEC* poderia ser reutilizada para calcular a relevância deste método em termos das características que ele assume na classe Java. A métrica *RTERT* poderia ser reutilizada para calcular a relevância deste método em termos de suas dependências, isto é, das relações que ele assume dentro da classe Java. A métrica *RITE* poderia ser reutilizada para calcular a relevância deste método em termos dos elementos impactados quando a mudança é aplicada. A métrica *RCTAE* poderia ser reutilizada para calcular a relevância deste método em termos da dificuldade de se aplicar a mudança. Por fim, a métrica *RTEATM* poderia ser reutilizada para calcular a relevância deste método considerando o seu uso em outras classes Java do projeto. Apenas as métricas *RCITOM* e *ROMTC* não são aplicáveis ao contexto de Java, pois tratam de conceitos específicos para linguagens de transformação. Para a obtenção das informações que as métricas necessitam, pode ser reutilizado um analisar estático já existente para Java.

5.4 Módulo: Ferramenta de Suporte

O módulo *Ferramenta de Suporte* é responsável por processar as informações obtidas dos módulos *Analisador Estatico* e *Metricas* para gerar os elementos impactados e o valor de impacto para uma determinada mudança. A Fig. 5.3 ilustra através de um diagrama de componentes UML [Object Management Group, 2011b] como a ferramenta de suporte usa esses módulos.

O componente *Analisador Estatico* fornece uma interface (*IAnalisadorEstatico*) com métodos para obter informações sobre qualquer elemento especificado em uma transformação, enquanto que o componente *Metricas* fornece uma interface (*IMetricas*) com métodos para

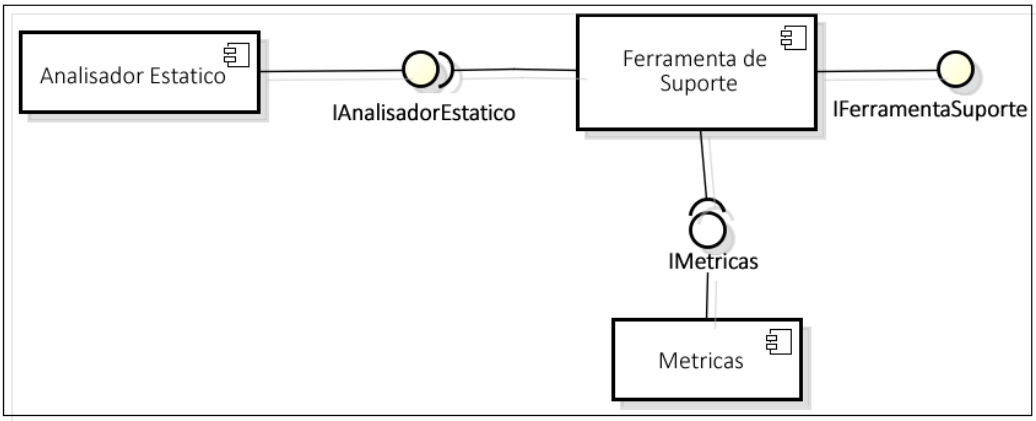

Figura 5.3: Diagrama de componentes para a ferramenta de suporte.

calcular as métricas propostas neste trabalho.

O componente *Ferramenta de Suporte* fornece uma interface (*IFerramentaSuporte*) com dois métodos: (i) `getImpactedElements(Entity elem)` para obter os elementos impactados, onde o parâmetro *elem* é o elemento a ser modificado; e (ii) `calcChangeImpactValue(Entity elem)` para calcular o valor de impacto, onde o parâmetro *elem* é o elemento a ser modificado. Porém, se os usuários desejarem escolher quais métricas devem ser consideradas no cálculo do valor de impacto, eles podem invocar o método `calcChangeImpactValue(Entity elem, Set metrics)`, que recebe como parâmetro o elemento a ser modificado e os nomes das métricas a serem consideradas.

Portanto, para usar a ferramenta de suporte (disponível em http://goo.gl/F9npHb) os usuários apenas têm que invocar os métodos apresentados anteriormente informando o elemento a ser modificado. Desta forma, eles vão obter os elementos impactados e o valor de impacto. Um exemplo de aplicação da abordagem proposta usando o módulo *Ferramenta de Suporte* é apresentado no Capítulo 7.

5.5 Identificação dos Elementos Impactados

Quando um tipo de mudança é aplicado a um determinado elemento da transformação, através do analisador estático é possível identificar o conjunto de elementos impactados com a mudança. Esse conjunto é identificado com base nas dependências (relações) que o elemento a ser modificado possui dentro da transformação.

Os elementos impactados podem ser facilmente identificados com o uso da API fornecida pelo analisador estático: é preciso apenas obter o *lado B* de cada relação do tipo *isInvoked* e *isExtended* que o elemento a ser modificado assume, tendo em vista que o *lado B* apenas destes tipos de relações é que, de fato, são impactados. Vale ressaltar que o *lado A* das relações que um determinado elemento e_i assume é representado por ele próprio, e o *lado B* é representado pelo elemento com o qual e_i se relaciona. Por exemplo, suponha que um *transformation block* chamado `getName()` possui duas relações do tipo *isInvoked*. Nestas duas relações, o *lado A* é representado pelo próprio *transformation block* `getName()` e o *lado B* é representado pelos respectivos elementos que o invocam.

Um pseudo-código para o algoritmo que obtém os elementos impactados por uma mudança é ilustrado no Código Fonte 5.1. Este algoritmo está implementado no protótipo de ferramenta de suporte desenvolvido para a abordagem de análise de impactos de mudanças.

Código Fonte 5.1: Pseudo-código do algoritmo que obtém os elementos impactados.

```
1  Set getImpactedElements(elementName)
2  var:
3      Element elem,
4      Set impactedElements,
5      Set relations
6  elem <= AtlAnalyzerFacade.getModuleElement(elementName)
7  relations <= elem.getRelations()
8  FOR EACH rel IN relations DO
9    IF rel.getType() == 'isInvoked' OR rel.getType() == 'isExtended'
10   THEN impactedElements.add(rel.getEntityB())
11 ENDFOR
12 RETURN impactedElements
```

Como se pode observar no Código 5.1, o algoritmo recebe o parâmetro `elementName` (linha 1), referente ao nome do elemento a ser modificado. As linhas 2-5 mostram as variáveis que serão manipuladas ao longo do algoritmo: `elem` (linha 3) - referente ao elemento a ser modificado; `impactedElements` (linha 4) - referente ao conjunto de elementos impactados por uma mudança aplicada em `elem`; e `relations` (linha 5) - referente ao conjunto de relações que `elem` assume. Inicialmente, o método `getModuleElement(...)`

definido na fachada `AtlAnalyzerFacade` [6] é invocado para obter o elemento a ser modificado (linha 6). Para isto, o método recebe como parâmetro o nome do elemento. Em seguida, o método `getRelations()` (linha 7) é invocado para obter o conjunto de relações que `elem` assume. Este conjunto de relações é percorrido (linha 8) e, para cada relação, o algoritmo verifica se seu tipo é `isInvoked` ou `isExtended` (linha 9) e, caso ele seja, adiciona ao conjunto de elementos impactados o *lado B* da relação (linha 10). Por fim, o algoritmo retorna o conjunto de elementos impactados como resultado (linha 12).

5.6 Cálculo do Valor de Impacto

Para calcular o valor de impacto com base nas sete métricas anteriormente apresentadas em (5.1), (5.2), (5.3), (5.4), (5.5), (5.6) e (5.7), foi definida a função $calcChangeImpactValue(e_i)$ ilustrada em (5.8), onde e_i é o elemento a ser modificado. Seja:

- $M = \{m_1, ..., m_y\}$ o conjunto das métricas propostas $m_i (i = 1, ..., 7)$, onde y é o número de métricas.

$$calcChangeImpactValue(e_i) = \frac{\sum_{k=1}^{y} m_k}{y} \tag{5.8}$$

Vale ressaltar que, quando calculado, o valor de impacto por si só não pode ser interpretado. No entanto, se o valor de impacto for calculado para diversas mudanças a serem aplicadas em uma transformação, então é possível comparar os impactos de cada mudança.

A abordagem de análise de impacto de mudanças proposta permite aos usuários customizar a função $calcChangeImpactValue(e_i)$ para calcular o impacto considerando apenas as métricas que eles estão interessados em analisar. Por exemplo, os usuários podem remover a métrica *RTEC* se eles não desejarem considerar as características do elemento a ser modificado. Além disso, os usuários podem adicionar novas métricas que julgarem importantes. A abordagem proposta oferece tal flexibilidade para ajudar os usuários a calcular o valor de

[6] Vale ressaltar que, neste pseudo-código, o analisador estático de ATL é usado através da fachada `AtlAnalyzerFacade`.

impacto de uma mudança de acordo com suas preferências. Por esta razão, cada métrica calcula a relevância do elemento a ser modificado considerando diferentes aspectos.

5.7 Classificação do Impacto das Mudanças

Um estudo foi elaborado para definir como os valores de impacto calculados pela abordagem proposta deveriam ser classificados como um impacto *baixo* ou *alto*. Para este estudo, foram selecionados os três participantes mais experientes em transformações ATL que estavam disponíveis, de diferentes grupos de pesquisa e universidades. Foi atribuído aos participantes um guia de execução (disponível em: *http://goo.gl/b5coUH*) e foi solicitado para eles classificarem um conjunto de mudanças como um impacto *baixo* ou *alto* usando seus próprios critérios. Para evitar viés, as transformações utilizadas neste estudo foram diferentes das utilizadas no estudo de caso a ser apresentado na Capítulo 8. Foram selecionadas três transformações ATL reais disponíveis em diferentes repositórios, como seguem:

- *ATLWFR* [7] (projeto EMFTVM);
- *SimpleOCLtoEMFTVM* [8] (projeto SimpleOcl);
- *PIM2PITM* [9] (projeto Mobit).

Para cada uma destas transformações foram selecionadas três mudanças a serem analisadas e classificadas. Devido ao longo tempo estimado para os participantes analisarem as transformações, foi decidido considerar apenas este pequeno número de mudanças. Foram selecionadas as mudanças que poderiam afetar diferentes partes da transformação. Os participantes (P1, P2 e P3) classificaram cada mudança como apresentado na Tabela 5.4. Apesar dos participantes serem experientes em ATL, pode-se observar na Tabela 5.4 que algumas de suas classificações foram divergentes.

Depois que os participantes classificaram as mudanças (como impacto baixo ou alto) e os valores de impacto para cada uma delas foi calculado usando a abordagem proposta, foi definido um intervalo de valores para as classificações *baixo* e *alto*. Para isso, foi calculada

[7] http://soft.vub.ac.be/viewvc/EMFTVM/trunk/emftvm.compiler/transformations
[8] https://code.google.com/a/eclipselabs.org/p/simpleocl
[9] https://sites.google.com/a/computacao.ufcg.edu.br/mobit

Tabela 5.4: Visão geral da classificação feita pelos participantes.

Tipo de Mudança	**P1**	**P2**	**P3**
changeConstantType	Alto	Baixo	Baixo
addTransformationBlockParameter	Baixo	Baixo	Baixo
removeConstant	Baixo	Baixo	Alto
changeFunctionReturnType	Alto	Alto	Baixo
changeFunctionBody	Baixo	Baixo	Baixo
changeConstantName	Alto	Baixo	Baixo
removeTransformationBlock	Alto	Alto	Alto
changeTransformationBlockName	Baixo	Baixo	Baixo
removeTransformationBlockInheritance	Alto	Alto	Alto

a média dos valores de impacto correspondentes às mudanças que foram classificadas pelos participantes como impacto *baixo*. Por exemplo, suponha que no estudo apenas os tipos de mudança *changeConstantName*, *removeOutputModelElementBinding* e *changeOutputModelElementType* foram classificados como *baixo* pelos participantes e com a abordagem proposta foram obtidos *0.36*, *0.14* e *0.28* como seus valores de impacto, respectivamente. Então, a média (*0.26*) destes valores de impacto foi calculada para classificar um impacto como *baixo*. Neste exemplo, mudanças com valores de impacto até *0.26* seriam classificadas como de baixo impacto e mudanças com valores de impacto maior que *0.26* seriam classificadas como de alto impacto.

Como resultado do estudo, foram obtidos os valores para representar o impacto de uma mudança como apresentado na Tabela 5.5.

Tabela 5.5: Intervalo de valores para representar a classificação de um impacto.

Intervalo de Valores	**Classificação**
Valor de Impacto ≤ 0.37	Impacto Baixo
Valor de Impacto ≥ 0.38	Impacto Alto

Como será apresentado no estudo de caso do Capítulo 8, com o uso da abordagem proposta foi calculado o impacto de cada mudança e ele foi representado através de um número, enquanto que os participantes classificaram os impactos como *baixo* e *alto*. Portanto, o intervalo de valores e suas respectivas classificações foram usadas para que se possa comparar e analisar os resultados da avaliação da análise do impacto de mudanças usando a abordagem proposta e um processo manual (Seção 8.4 do Capítulo 8).

Capítulo 6

Analisador Estático para ATL

Este capítulo tem a finalidade de apresentar o analisador estático para ATL desenvolvido por Vieira e Ramalho [Vieira e Ramalho, 2011a] como um suporte para a abordagem amparada na análise estática que foi abordada no capítulo 5. Primeiramente, será ilustrada uma visão geral do analisador estático (Seção 6.1). Depois, será apresentada a classificação dos tipos de relações definidos (Seção 6.2). Em seguida, questões de design para a construção de um analisador estático em outras linguagens serão apresentadas (6.3). Por fim, dois exemplos de aplicação do analisador estático são apresentados (Seção 6.4).

6.1 Visão Geral

Tendo em vista que para a aplicação da abordagem proposta neste trabalho foi escolhida ATL como linguagem de transformação de modelos, se fez necessário desenvolver um analisador estático específico para esta linguagem. O analisador estático para ATL desenvolvido provê uma API com métodos para manipular os elementos de uma transformação ATL. A API encontra-se disponível sob a licença GPL (*General Public License*) em [Vieira e Ramalho, 2011b].

Uma visão geral do analisador estático é apresentada na Figura 6.1. Inicialmente, o analisador estático recebe como entrada o arquivo de transformação ATL a ser analisado. O código fonte da transformação é inspecionado através do método `parse(InputStream in)` disponível na API de ATL [Eclipse Foundation, 2010]. Como resultado, um elemento do tipo `EObject` é gerado, o qual pode ser explorado para extrair informações sobre os

elementos da transformação em análise, tais como regras e *helpers*. Dado que o analisador estático foi implementado em Java, estas informações são extraídas do elemento `EObject` para instanciar classes Java de acordo com o metamodelo de ATL [Jouault, 2007]. Essas classes Java devem ser instanciadas como objetos Java com o objetivo de guardar as informações sobre a transformação ATL de entrada. Depois que todas as classes Java tenham sido instanciadas com informações obtidas do `EObject`, a API com vários métodos provida pelo analisador estático pode ser usada.

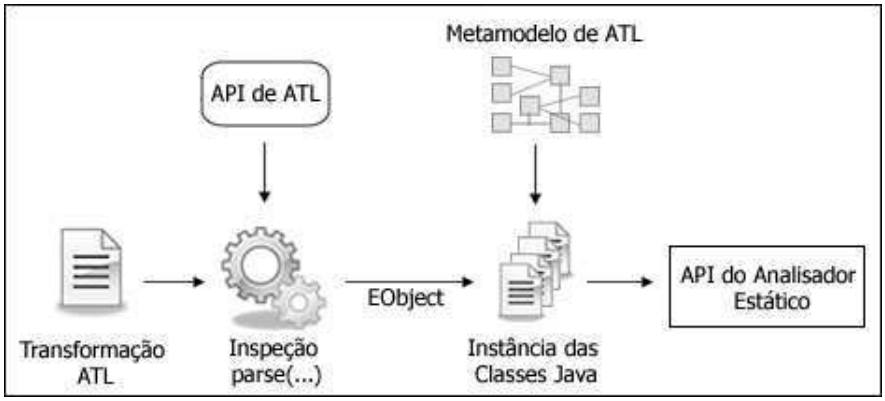

Figura 6.1: Visão geral do analisador estático para ATL.

O analisador estático foi implementado de acordo com o metamodelo de ATL no intuito de definir as mesmas características e relacionamentos existentes em uma transformação ATL. Portanto, cada metaclasse do metamodelo de ATL foi implementada como uma classe Java e cada propriedade (incluindo seus relacionamentos) de uma metaclasse foi implementada como um atributo de uma classe Java. Por exemplo, de acordo com a Figura 6.2 que ilustra um trecho do metamodelo de ATL, foi necessário criar, dentre outros elementos: (i) uma classe Java abstrata chamada *`Rule`* contendo um atributo `name`; e (ii) duas classes Java concretas chamadas `MatchedRule` e `CalledRule` contendo seus respectivos atributos.

Como se pode observar na Figura 6.2, a linguagem ATL permite que seus usuários definam três tipos de unidades (`Unit`): `Module`, `Query` e `Library`. Um `Module` pode ser composto por zero ou mais elementos do tipo *`ModuleElement`*, os quais podem ser um `Helper` ou uma regra (`Rule`). Uma regra pode ser imperativa (`CalledRule`) ou declarativa (`MatchedRule`), e uma `LazyMatchedRule` é um tipo de `MatchedRule`.

Dado que todas as informações sobre a transformação ATL em análise tenham sido instanciadas como objetos Java, os métodos do analisador estático podem ser usados para ma-

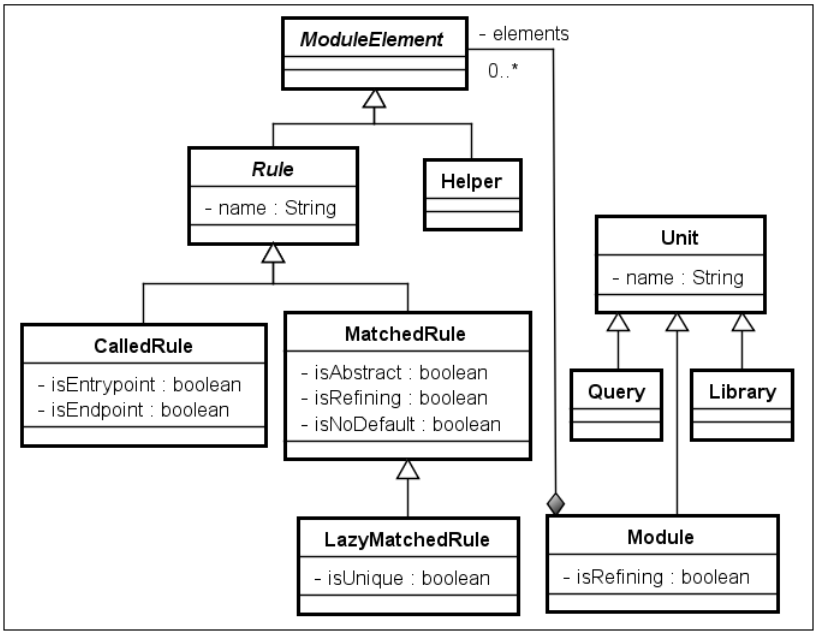

Figura 6.2: Trecho do metamodelo de ATL.

nipular uma série de elementos da transformação, tais como regras, *helpers*, modelos e metamodelos. Por exemplo, pode-se obter todas as *matched rules* da transformação através do método `getMatchedRules()`. A Tabela 6.1 mostra alguns dos métodos oferecidos pelo analisador estático e uma breve descrição deles.

6.2 Classificação dos Tipos de Relações

A abordagem para análise de impacto de mudanças proposta neste trabalho requer informações sobre as dependências de uma transformação em termos de: (i) herança entre as regras; e (ii) invocação de regras/*helpers*, ou seja, as chamadas entre estes elemento. Para prover estas informações, o analisador estático define os conceitos de `Entidade` e `Relação`, que são amplamente adotados no contexto geral de análise de impacto de software [Bohner e Arnold, 1996]. No contexto deste trabalho, uma `Entidade` é uma generalização para todas as metaclasses do metamodelo de ATL, sendo assim, cada elemento definido em uma transformação ATL é uma entidade. Já uma `Relação` é um relacionamento entre duas metaclasses (entidades) do metamodelo de ATL.

Em termos de implementação, uma classe Java abstrata chamada *`Entity`* foi criada para representar qualquer metaclasse do metamodelo de ATL e uma classe Java chamada `Relation` foi criada para representar qualquer relacionamento entre duas metaclasses.

Tabela 6.1: Alguns métodos providos pelo analisador estático de ATL.

Assinatura do Método	Descrição
public Set<TransfElement> getInvokerTransfElem()	Obtém um conjunto de todos os elementos do módulo que invocam um elemento específico;
public int getNumberOfRelations()	Obtém o número de relações que um elemento específico assume;
public Set<MatchedRule> getChildren()	Obtém um conjunto de *matched rules* que estendem uma específica *matched rule*;
public MatchedRule getMatchedRules()	Obtém um conjunto de todas as *matched rules* de uma transformação;
public Rule getRule (String ruleName)	Obtém uma regra específica (*lazy*, *matched* ou *called*) a partir do nome informado;
public Helper getHelper (String helperName)	Obtém um *helper* específico a partir do nome informado;
public Set<OclModel> getInputMetamodels()	Obtém um conjunto contendo o metamodelo de cada modelo de entrada da transformação;
public Set<OclModel> getOutputMetamodels()	Obtém um conjunto contendo o metamodelo de cada modelo de saída da transformação.

O Código Fonte 6.1 ilustra um trecho de implementação da classe Java abstrata `Entity`, que define dois atributos: (i) `entityType` - o tipo da entidade, e (ii) `relations` - o mapeamento de todas as relações que a entidade possui, onde a chave é o tipo da relação e o valor é o conjunto de relações que possuem este tipo.

Código Fonte 6.1: Trecho de implementação da classe Java Entity.

```
1  public abstract class Entity {
2      protected String entityType;
3      protected Map<RelationType, Set<Relation>> relations =
4          new HashMap<RelationType, Set<Relation>>();
5      ...
6  }
```

Por outro lado, o Código Fonte 6.2 ilustra um trecho de implementação da classe Java `Relation`, que define três atributos: (i) `entityA` - o lado A da relação, (ii) `entityB` - o lado B da relação, e (iii) `type` [1] - o tipo da relação entre a entidade A e a entidade B. Ao

[1] É importante ressaltar que o atributo `type` definido na classe Java `Relation` não se refere aos tipos de relacionamentos da UML, tais como generalização, associação, etc.

longo deste documento, qualquer metaclasse ou elemento de uma transformação será referido como uma "entidade" e qualquer relacionamento entre duas metaclasses será referido como uma "relação entre duas entidades".

Código Fonte 6.2: Trecho de implementação da classe Java Relation.

```
1  public class Relation {
2      private Entity entityA;
3      private Entity entityB;
4      private RelationType type;
5      ...
6  }
```

Foi definido um tipo de relação para cada relacionamento do metamodelo de ATL. A classificação de alguns destes tipos de relação é apresentada na Tabela 6.2, onde cada linha corresponde a uma instância da classe Java `Relation`, isto é, uma relação entre duas entidades. Por exemplo, na linha 1, `owns_outpattern` (coluna Tipo) representa o tipo de relação indicando que uma `Rule` possui um `OutPattern` em uma determinada transformação. Inversamente, na linha 2, `outpattern_owned_by` (coluna Tipo) representa o tipo de relação indicando que um `OutPattern` é possuído por uma `Rule` em uma determinada transformação. Por outro lado, na linha 7, `invokes` (coluna Tipo) representa o tipo de relação indicando que uma `Rule` invoca um `Helper`. Inversamente, na linha 8, `isInvoked` (coluna Tipo) representa o tipo de relação indicando que um `Helper` é invocado por uma `Rule` em uma determinada transformação. A lista completa de todos os tipos de relações que foram definidas para ATL é apresentada no Apêndice A deste documento.

Tabela 6.2: Classificação de alguns tipos de relações.

Tipo	Entidade A	Entidade B
owns_outpattern	Rule	OutPattern
outpattern_owned_by	OutPattern	Rule
owns_outpattern_elements	OutPattern	OutPatternElement
outpattern_element_owned_by	OutPatternElement	OutPattern
extends	MatchedRule	MatchedRule
has_inmodels	Module	OclModel
invokes	Rule	Helper
isInvoked	Helper	Rule
isExtended	MatchedRule	MatchedRule
has_outmodels	Module	OclModel

Apenas dois dos tipos de relações não correspondem a um relacionamento especificado no metamodelo de ATL: `invokes` e `isInvoked`, como apresentados nas linhas 7 e 8 da Tabela 6.2. Estes tipos de relações adicionais foram definidos para se obter não somente as dependências em termos de relacionamentos entre as entidades de uma transformação, mas também as dependências em termos de invocação de regras e *helpers*.

6.3 Questões de Design para a Construção de um Analisador Estático

Diretrizes foram estabelecidas neste trabalho para o desenvolvimento (em Java) de um analisador estático para ATL. Estas mesmas diretrizes também podem ser seguidas para se desenvolver um analisador estático para qualquer outra linguagem de transformação de modelos. A seguir, são apresentadas estas diretrizes:

- *Recebimento de um arquivo de entrada* - o analisador estático irá inspecionar o código fonte da transformação, portanto, ele deve receber como entrada o arquivo de transformação na linguagem a ser analisada;

- *Concordância com o metamodelo da linguagem desejada* - o analisador deve ser desenvolvido de acordo com o metamodelo da linguagem que se deseja realizar a análise estática no intuito de definir as mesmas características e relacionamentos existentes na linguagem. Para isto, cada metaclasse do metamodelo deve ser implementada como uma classe Java (se o analisador for implementado em Java) e cada propriedade de uma metaclasse deve ser implementada como um atributo de sua classe Java correspondente. A implementação destas classes Java irá refletir toda a estrutura da linguagem de transformação desejada, facilitando a implementação de métodos para obter qualquer tipo de informação da transformação;

- *Definição dos conceitos de* `Entidade` *e* `Relação` - para prover informações sobre as dependências e relacionamentos entre os elementos da transformação, o analisador estático deve definir os conceitos de `Entidade` e `Relação`[2]. Portanto, deve ser

[2]Uma `Entidade` é uma generalização para todas as metaclasses do metamodelo, sendo assim, cada elemento definido em uma transformação de determinada linguagem é uma entidade. Já uma `Relação` é um relacionamento entre duas metaclasses (entidades) do metamodelo.

criada uma classe Java chamada `Entity` para representar qualquer metaclasse do metamodelo e deve ser criada uma classe Java chamada `Relation` para representar qualquer relacionamento entre duas metaclasses;

- *Implementação da API* - funções devem ser implementadas para manipular os elementos da transformação e obter as informações necessárias para a análise estática, tais como regras, *helpers*, modelos e metamodelos de entrada e de saída, assim como informações sobre a dependência entre estes elementos, como por exemplo, as regras que estendem determinada regra e os *helpers* que invocam determinado *helper*.

Vale ressaltar que este guia foi apresentado com base na metodologia utilizada para o desenvolvimento do analisador estático para ATL. Possivelmente, devem existir outras maneiras e técnicas de se desenvolver um analisador estático no escopo de transformações de modelos.

6.4 Exemplos de Aplicação

Nesta seção, são apresentados dois exemplos concretos ilustrando como o analisador estático de ATL deve ser utilizado.

No primeiro exemplo, o analisador estático foi utilizado para obter o conjunto de *matched rules* que estendem uma específica *matched rule* dentro de um módulo de transformação ATL. O Código Fonte 6.3 mostra um trecho deste módulo de transformação disponível em [Eclipse Foundation, 2014a], que é um simples copiador: cada elemento do modelo de origem é copiado para o modelo de destino.

Suponha que no módulo de transformação ATL chamado `Copy` não seja mais necessário copiar *classifiers*, de forma que seja necessário remover a *matched rule* `CopyClassifier` (linhas 31-34). No entanto, existem três regras (`CopyDataType`, `CopyEnumeration` e `CopyClass`) na transformação `Copy` que estendem esta *matched rule*. Então, é necessário identificá-las antes de remover a regra `CopyClassifier` para adaptá-las com a mudança e eliminar as dependências. A utilização do analisador estático na transformação ATL `Copy` é ilustrada no Código Fonte 6.4, que apresenta uma classe Java *main* que deve ser implementada pelo desenvolvedor para recuperar informações de uma determinada transformação.

Código Fonte 6.3: Trecho da transformação Copy.atl

```
1  module Copy;
2  create OUT : MM from IN : MM;
3  rule CopyDataType extends CopyClassifier {
4     from s : MM!DataType
5     to t : MM!DataType
6  }
7  rule CopyEnumeration extends CopyClassifier {
8     from s : MM!Enumeration
9     to t : MM!Enumeration (literals <- s.literals)
10 }
11 abstract rule CopyModelElement extends CopyLocatedElement {
12    from s : MM!ModelElement
13    to t : MM!ModelElement (name <- s.name)
14 }
15 abstract rule CopyLocatedElement {
16    from s : MM!LocatedElement
17    to t : MM!LocatedElement (location <- s.location)
18 }
19 rule CopyPackage extends CopyModelElement {
20    from s : MM!Package
21    to t : MM!Package (contents <- s.contents)
22 }
23 rule CopyClass extends CopyClassifier {
24    from s : MM!Class
25    to t : MM!Class (
26          isAbstract <- s.isAbstract,
27          supertypes <- s.supertypes,
28          structuralFeatures <- s.structuralFeatures,
29          operations <- s.operations)
30 }
31 rule CopyClassifier extends CopyModelElement {
32    from s : MM!Classifier
33    to t : MM!Classifier
34 }
```

Código Fonte 6.4: Exemplo 1 da utilização do analisador estático.

```
1  package org.atlanalyzer.main;
2  import java.io.IOException;
3  import java.util.Set;
4  import org.atlanalyzer.entities.AtlAnalyzerFacade;
5  import org.atlanalyzer.entities.MatchedRule;
6  import org.atlanalyzer.entities.Module;
7  import org.eclipse.m2m.atl.core.ATLCoreException;
8
9  public class Main {
10       public static void main(String[] args)
11       throws ATLCoreException, IOException {
12            AtlAnalyzerFacade facade = AtlAnalyzerFacade.getInstance(
13                 "../ATL_analyzer/atl/Copy.atl");
14            Module unit = (Module) facade.getUnit();
15            MatchedRule rule =
16                 (MatchedRule) unit.getRule("CopyClassifier");
17            Set<MatchedRule> children = rule.getChildren();
18       }
19 }
```

Como mostrado no Código Fonte 6.4, primeiramente, foi obtida uma instância da fachada `AtlAnalyzerFacade` passando como parâmetro a transformação ATL a ser analisada (linhas 12-13). O analisador estático foi utilizado a partir desta fachada, pois ela permite o acesso ao módulo de transformação a ser analisado, a partir do qual qualquer informação da transformação pode ser obtida. Na linha 14, a partir da fachada foi obtida a unidade ATL a ser analisada, neste caso, um módulo ATL. Em seguida, a partir deste módulo foi obtida a regra a ser removida passando como parâmetro o seu nome (linhas 15-16). Por fim, a partir desta regra foram obtidas as *matched rules* que a estendem (linha 17).

No segundo exemplo, ilustrado no Código Fonte 6.5, o analisador estático foi utilizado para detectar todos os elementos do módulo (regras e *helpers*) da transformação que invocam um determinado *helper*. Quando se pretende modificar ou remover um determinado *helper* de uma transformação, é importante identificar as regras e os demais *helpers* que o invocam.

O Código Fonte 6.5 mostra um trecho da transformação ATL `Class2Relational` disponível em [Eclipse Foundation, 2005a], que gera um modelo relacional a partir de um modelo de classe.

Suponha que se deseja modificar o tipo de retorno do *helper* `objectIdType` (linhas 3-5). Então, todos os elementos do módulo que invocam este *helper* devem ser identificados e analisados antes de qualquer mudança. O Código Fonte 6.6 ilustra como usar o analisador

Código Fonte 6.5: Trecho da transformação Class2Relational.atl

```
1  module Class2Relational;
2  create OUT : Relational from IN : Class;
3  helper def: objectIdType : Relational!Type =
4     Class!DataType.allInstances()->select(
5     e | e.name = 'Integer')->first();
6  rule Class2Table {
7     from c : Class!Class
8     to out : Relational!Table (
9           name <- c.name,
10          key <- Set {key}
11          col <- Sequence {key}->union(c.attr->select(
12             e | not e.multiValued)),
13          ),
14          key : Relational!Column (
15          name <- 'objectId',
16          type <- thisModule.objectIdType)
17 }
18 rule DataType2Type {
19    from dt : Class!DataType
20    to out : Relational!Type (name <- dt.name)
21 }
22 rule ClassAttribute2Column {
23    from a : Class!Attribute (a.type.oclIsKindOf(
24          Class!Class) and not a.multiValued)
25    to foreignKey : Relational!Column (
26          name <- a.name + 'Id',
27          type <- thisModule.objectIdType)
28 }
```

estático de ATL para obter tais informações. Como mencionado anteriormente, o desenvolvedor deve implementar uma classe Java *main* e utilizar o analisador estático a partir da fachada `AtlAnalyzerFacade`.

No Código Fonte 6.6, as linhas 1-15 são semelhantes ao exemplo anterior (Código Fonte 6.4): algumas classes da seção de *import* foram substituídas e o nome do arquivo da transformação ATL a ser analisada na linha 14 foi modificado. Na linha 16, foi obtido o *helper* a ser modificado. Por fim, foram obtidos todos os elementos do módulo que invocam o *helper* `objectIdType` (linhas 17-18).

Código Fonte 6.6: Exemplo 2 da utilização do analisador estático.

```
1  package org.atlanalyzer.main;
2  import java.io.IOException;
3  import java.util.Set;
4  import org.atlanalyzer.entities.AtlAnalyzerFacade;
5  import org.atlanalyzer.entities.Helper;
6  import org.atlanalyzer.entities.Module;
7  import org.atlanalyzer.entities.ModuleElement;
8  import org.eclipse.m2m.atl.core.ATLCoreException;
9
10 public class Main {
11       public static void main(String[] args)
12       throws ATLCoreException, IOException {
13            AtlAnalyzerFacade facade = AtlAnalyzerFacade.getInstance(
14                 "../ATL_analyzer/atl/Class2Relational.atl");
15            Module unit = (Module) facade.getUnit();
16            Helper helper = unit.getHelper("objectIdType");
17            Set<ModuleElement> moduleElements =
18                 helper.getInvokerModElems();
19       }
20 }
```

Embora tenham sido ilustrados exemplos simples, é importante ressaltar que o analisador estático é muito útil para auxiliar os desenvolvedores na realização de suas tarefas de desenvolvimento, tais como manutenção e *debugging*, uma vez que: (i) ele não está sujeito a erros, diferentemente de uma análise realizada manualmente; (ii) ele pode analisar transformações muito extensas sem exigir esforço adicional; (iii) a sua API é fácil de entender e manipular;

e (iv) os desenvolvedores economizam esforço e tempo de desenvolvimento, pois automaticamente identificam as dependências e as relações do elemento do módulo a ser modificado em uma transformação.

Capítulo 7

Exemplo de Aplicação

Este capítulo ilustra como usar a abordagem proposta neste trabalho para identificar os elementos impactados e para calcular o valor de impacto causado por três tipos de mudança a serem aplicadas à transformação ATL KM32OWL [Eclipse Foundation, 2014b], que é um mapeamento de KM3 (*Kernel MetaMetaModel*) para OWL (*Web Ontology Language*) e contempla um total de 20 elementos em sua especificação.

Para este exemplo de aplicação, foi calculado o impacto das mudanças considerando as características do elemento a ser modificado, suas relações e os elementos impactados pela mudança, ou seja, as métricas *RTEC*, *RTERT* e *RITE*. Portanto, o método `calcChangeImpactValue(Entity elem)` invoca três outros métodos para calcular as métricas consideradas neste exemplo. Não foram consideradas todas as métricas no intuito de simplificar o entendimento deste exemplo e ao mesmo tempo para mostrar como é possível customizar a abordagem para o uso de métricas específicas.

A fim de ilustrar a aplicação da abordagem, foram aplicados três tipos de mudança que são apresentados nas próximas seções: *changeTransformationBlockParameterType*, *changeFunctionContextType* e *removeTransformationBlock*.

7.1 ChangeTransformationBlockParameterType

Primeiramente, será analisado o impacto de modificar o tipo de parâmetro da *called rule* `addCardinalityRestriction` apresentada no Código Fonte 7.1. Depois de invo-

car o método `getImpactedElements(Entity elem)`[1], foram identificados os seguintes elementos impactados por essa mudança: `Attribute2DataTypeProperty`, `EnumerationProperty2ObjectProperty` e `Reference2ObjectProperty`, que são *matched rules* que invocam a *called rule* a ser modificada. Estes elementos devem ser adaptados depois que a mudança for aplicada.

Código Fonte 7.1: Called rule addCardinalityRestriction da transformação KM32OWL

```
1  rule addCardinalityRestriction(s: KM3!StructuralFeature) {
2    to
3       c : OWL!CardinalityRestriction (
4           superClass <- s.owner,
5           OWLCardinality <- literal,
6           OWLOnProperty <- s
7       ),
8       literal : OWL!TypedLiteral (
9           lexicalForm <- s.upper.toString(),
10          datatypeURI <- u
11      ),
12      u : OWL!URIReference (
13          uri <- uri
14      ),
15      uri : OWL!UniformResourceIdentifier (
16          name <- thisModule.primitiveTypeMap.get('Integer'))
17 }
```

O elemento a ser modificado assume duas características: *hasParameter*, porque ele recebe um parâmetro do tipo `KM3!StructuralFeature` (linha 1 do Código Fonte 7.1); e *hasToBlock*, porque ele especifica um bloco "to" que gera elementos no modelo de saída (linhas 2-16 do Código Fonte 7.1). Além disso, o elemento a ser modificado assume dois tipos de relação: *isInvoked*, porque ele é invocado por outros elementos da transformação; e *invokes*, porque ele invoca um elemento da transformação (linha 16 do Código Fonte 7.1).

Depois de calcular as métricas *RTEC*, *RTERT* e *RITE* foram obtidos os resultados 0.31, 1 e 0.15, respectivamente. Portanto, depois de invocar o método

[1]Para identificar o conjunto de elementos impactados por cada tipo de mudança, o método `getImpactedElements(Entity elem)` foi invocado a partir do módulo *Ferramenta de Suporte*, onde $elem$ é o elemento a ser modificado.

`calcChangeImpactValue(Entity elem)`, foi obtido o valor de impacto *0.49*. Até o momento não há parâmetros para dizer se este valor indica um impacto baixo ou alto, o que se pode considerar aqui é que: (i) três elementos impactados devem ser adaptados para invocar a *called rule* `addCardinalityRestriction` com um tipo de parâmetro diferente; e (ii) o elemento a ser modificado assume duas características e dois tipos de relações.

7.2 ChangeFunctionContextType

Será analisado agora o impacto de modificar o tipo de contexto do *helper* `isInverseFunctional` apresentado no Código Fonte 7.2. Depois de invocar o método `getImpactedElements(Entity elem)`, foi identificado apenas um elemento impactado por esta mudança: `Reference2ObjectProperty`, que é o único que invoca o *helper* a ser modificado e deve ser adaptado depois que a mudança for aplicada.

Código Fonte 7.2: Helper isInverseFunctional da transformação KM32OWL

```
1  helper context KM3!Reference def: isInverseFunctional(): Boolean =
2    if self.opposite.oclIsUndefined()
3       then false
4       else self.opposite.upper = 1 and self.opposite.lower = 1
5    endif;
```

O elemento a ser modificado assume duas características: *hasNotPrimitiveContextType*, porque o tipo do contexto dele é `KM3!Reference` (linha 1 do Código Fonte 7.2) e não um tipo primitivo; e *hasIfExpression*, porque ele faz uso de uma expressão "if" (linhas 2-5 do Código Fonte 7.2). Além disso, o elemento a ser modificado assume o tipo de relação *isInvoked*, pois ele é invocado por outro elemento da transformação.

Depois de calcular as métricas *RTEC*, *RTERT* e *RITE* foram obtidos os resultados 0.33, 0.51 e 0.05, respectivamente. Portanto, depois de invocar o método `calcChangeImpactValue(Entity elem)`, foi obtido o valor de impacto de *0.30*. Comparado com o valor de impacto da mudança anterior (*ChangeTransformationBlockParameterType*), nota-se que ele tem um impacto menor. De fato, faz sentido que este valor seja menor, uma vez que a mudança impacta apenas um elemento e o elemento a ser modificado assume duas características e apenas um tipo de relação. Se, por exemplo, a mudança

tivesse impactado 10 elementos e se o *helper* `isInverseFunctional` assumisse quatro características (*hasNotPrimitiveContextType*, *hasIfExpression*, *hasParameter* e *hasCollectionExpression*) e dois tipos de relações (*invokes* e *isInvoked*), o valor de impacto poderia ser bem mais alto: *0.74*. Como se pode observar neste exemplo, três variáveis (o número de elementos impactados, as características e os tipos de relações) influenciaram o valor de impacto.

7.3 RemoveTransformationBlock

Neste exemplo será analisado o impacto de remover a *lazy matched rule* `SymmetricProperty` apresentada no Código Fonte 7.3. Depois de invocar o método `getImpactedElements(Entity elem)`, foi identificado apenas um elemento impactado por esta mudança: `Reference2ObjectProperty`, que é o único que invoca a *lazy matched rule* a ser removida e ele deve ser adaptado após a remoção.

Código Fonte 7.3: Lazy matched rule SymmetricProperty da transformação KM32OWL

```
1  lazy rule SymmetricProperty {
2    from
3      r : KM3!Reference
4    to
5      o : OWL!SymmetricProperty (
6        isDefinedBy <- r
7      )
8  }
```

O elemento a ser removido não assume nenhuma característica. Por outro lado, ele assume o tipo de relação *isInvoked*, pois ele é invocado por outro elemento da transformação.

Depois de calcular as métricas *RTEC*, *RTERT* e *RITE* foram obtidos os resultados de 0, 0.27 e 0.05, respectivamente. Portanto, depois de invocar o método `calcChangeImpactValue(Entity elem)`, foi obtido o valor de impacto *0.11*. Comparado com o valor de impacto das mudanças anteriores (*ChangeTransformationBlockParameterType* e *ChangeFunctionContextType*), pode-se notar que ele tem o menor impacto, uma vez que a mudança impacta apenas um elemento e o elemento a ser modificado não assume nenhuma característica e apenas um tipo de relação.

7.4 Discussões

A Tabela 7.1 apresenta os valores de impacto calculados para as mudanças *changeTransformationBlockParameterType*, *changeFunctionContextType* e *removeTransformationBlock* anteriormente apresentadas neste exemplo de aplicação. Considerando que um gerente de projeto tem que aplicar na transformação KM32OWL um conjunto de mudanças (coluna 1 da tabela), o valor de impacto calculado para cada mudança poderia ser usado como referência para ajudá-lo a escalonar e priorizar as mudanças. Por exemplo, a mudança `RemoveTransformationBlock` (linha 3) a ser aplicada à *lazy matched rule* `SymmetricProperty` poderia ser a primeira a ser escalonada, enquanto que a mudança *ChangeTransformationBlockParameterType* (linha 1) poderia ser a última. Além disso, os desenvolvedores poderão mais facilmente adaptar as partes da transformação afetadas com cada mudança, uma vez que os elementos impactados são automaticamente obtidos com o auxílio da ferramenta de suporte.

Tabela 7.1: Valores de impacto calculados no exemplo de aplicação.

Tipo de Mudança	**Elemento da Transf. ATL**	**Valor de Impacto**
changeTransformationBlockParameterType	addCardinalityRestriction (called rule)	0.49
changeFunctionContextType	isInverseFunctional (helper)	0.30
removeTransformationBlock	SymmetricProperty (lazy matched rule)	0.11

De fato, os valores de impacto ilustrados na Tabela 7.1 (coluna 3) estão de acordo com a complexidade de aplicá-las. Por exemplo, é menos custoso remover a *lazy matched rule* `SymmetricProperty` (linha 3) do que modificar o tipo de parâmetro da *called rule* `addCardinalityRestriction` (linha 1) porque: (i) a primeira é invocada por apenas uma regra, enquanto a segunda é invocada por três regras que devem ser adaptadas para invocar a regra modificada de acordo com o seu novo tipo de parâmetro e cada adaptação exige esforço e tempo de desenvolvimento; e (ii) a primeira não assume nenhuma característica, enquanto a segunda assume duas (*hasParameter* e *hasToBlock*), indicando a relevância da *called rule* uma vez que ela recebe um parâmetro a ser utilizado ao longo da regra e define um bloco “to” que é um importante construtor ATL para especificar como os elementos

devem ser gerados no modelo de saída.

É mais evidente analisar o impacto de mudanças nestes três exemplos simples, pois a abordagem proposta identifica automaticamente os elementos impactados e estão sendo consideradas mudanças simples e transformações pequenas. No entanto, para transformações maiores e diversificadas não é trivial identificar os elementos impactados, nem mensurar o impacto da mudança de um elemento.

Capítulo 8

Estudo de Caso

Este capítulo descreve o estudo de caso conduzido neste trabalho com o intuito de identificar métricas para análise de impacto de mudanças em transformações de modelos e de avaliar a abordagem proposta. Ele foi realizado de acordo com a metodologia para elaboração de um estudo de caso proposta por Yin [Yin, 2003] que define as seguintes fases: Definição do Estudo (Seção 8.1), Planejamento do Estudo (Seção 8.2), Execução do Estudo (Seção 8.3) e Análise e Interpretação dos Resultados (Seção 8.4). Este capítulo também apresenta algumas ameaças à validade do estudo de caso e como elas foram tratadas (Seção 8.5). Todo o material do estudo está disponível online no site do projeto [Vieira e Ramalho, 2014b].

8.1 Definição do Estudo

O estudo de caso conduzido neste trabalho teve dois grandes objetivos. O primeiro deles foi o de servir como alicerce e embasamento para a *identificação de métricas* para medir o impacto de mudanças em transformações de modelos. O segundo objetivo foi o de *avaliar a abordagem proposta* neste trabalho.

É importante enfatizar que as métricas propostas neste trabalho não foram definidas exclusivamente com base nos resultados do estudo de caso realizado. O estudo serviu de alicerce mas, antes mesmo de iniciá-lo, muito tempo foi dedicado ao estudo de métricas existentes em outros escopos e ao estudo, principalmente, do funcionamento das transformações de modelos a fim de se entender o impacto que mudanças podem causar. Portanto, o esboço de algumas métricas (*RTEC*, *RTERT* e *RITE*) já tinha sido definido.

Para que o primeiro objetivo do estudo fosse alcançado, foi necessário investigar e entender como diferentes desenvolvedores analisam manualmente o impacto de mudanças em transformações de modelos antes de aplicá-las. Além disso, foram investigados os critérios que eles adotam para analisar os impactos.

Nesse contexto, a primeira questão de pesquisa foi elaborada para o estudo: *Quais critérios os usuários adotam para identificar os elementos impactados e medir o impacto de mudanças em transformações de modelos?* Para responder esta questão, as respostas de cada participante foram analisadas para uma melhor compreensão dos critérios que eles adotaram para classificar os impactos das mudanças e identificar os elementos impactados.

No contexto de MDD, não foi encontrado nenhum trabalho semelhante ao proposto nesta tese. Portanto, não foi possível avaliar a abordagem proposta comparando sua eficiência com a eficiência de uma outra abordagem. Atualmente, quando os desenvolvedores precisam aplicar mudanças em suas transformações eles adotam um processo manual para analisar os impactos: para cada tipo de mudança a ser aplicada, eles identificam dentro da transformação o elemento a ser modificado e procuram por possíveis dependências deste elemento.

Portanto, para que o segundo objetivo deste estudo fosse alcançado, os resultados da abordagem proposta foram comparados com os resultados da análise manual do impacto de mudanças. Para isto, foi solicitado aos participantes que analisassem manualmente alguns tipos de mudança a fim de identificar os elementos impactados e classificar o impacto para cada mudança. Em seguida, a abordagem proposta foi usada para realizar essas mesmas tarefas de forma automática. Dessa forma, foi possível comparar os resultados dos participantes com os resultados obtidos com a aplicação da abordagem proposta. Neste contexto, a seguinte questão de pesquisa foi elaborada para o estudo: *Quão divergentes são os resultados da análise de impacto de mudanças realizada por nossa abordagem e pela análise manual?*

Para responder esta questão, as respostas dos participantes foram analisadas e comparadas com os resultados obtidos com a abordagem proposta. Uma vez que os participantes não tinham conhecimento prévio de como calcular o valor de impacto para as mudanças, para simplificar este estudo foi solicitado que eles classificassem os impactos como *baixo* ou *alto* ao invés de calcular o valor do impacto. Para evitar viés, não foi definido para os participantes o que seria um impacto baixo ou alto, eles tinham que usar seus próprios critérios. A descrição de como as classificações de um impacto (baixo ou alto) foram definidas para a

abordagem proposta encontra-se na Subseção 5.7.

8.2 Planejamento do Estudo

Durante a fase de planejamento foi definida toda a infra-estrutura necessária para a condução deste estudo: o contexto em que o estudo foi realizado, os participantes, os casos (projetos ATL) utilizados no estudo e a instrumentação. Foram selecionados projetos especificados em ATL para este estudo tendo em vista que nesta tese a abordagem proposta foi aplicada no escopo de ATL.

O estudo foi realizado no contexto de *estudantes vs profissionais*, onde os estudantes assumiram o papel de profissionais para analisar o impacto de uma mudança na transformação. Foram selecionados 14 usuários de ATL para participar deste estudo. Eles eram estudantes de graduação e pós-graduação (mestrado ou doutorado) em Ciência da Computação e tinham no mínimo um ano de experiência com transformações de modelos especificadas em ATL. No intuito de se obter um grupo representativo de participantes, eles foram selecionados de diferentes universidades e grupos de pesquisa.

Para a seleção dos projetos utilizados neste estudo, foram considerados como critério projetos que: (i) mantinham um repositório com as mudanças das transformações; (ii) foram desenvolvidos por diferentes grupos de pesquisa; (iii) tinham o código fonte aberto; e (iv) eram de pequeno porte. Como resultado, foram escolhidos quatro projetos de transformações ATL e, para cada um deles, foi selecionada uma transformação, como seguem.

- *PITM2PSTM* [1] (projeto Mobit);
- *SimpleGTtoEMFTVM* [2] (projeto SimpleGT);
- *emig2EMFTVM* [3] (projeto Emf Migrate);
- *BPMN_para_Atividades* [4] (projeto Transformação-bpm).

Para cada uma destas transformações foram selecionadas três mudanças a serem analisadas neste estudo, as quais são reais e foram retiradas de diferentes versões do repositório.

[1] https://sites.google.com/a/computacao.ufcg.edu.br/mobit
[2] https://code.google.com/a/eclipselabs.org/p/simplegt
[3] https://code.google.com/a/eclipselabs.org/p/emfmigrate
[4] https://code.google.com/p/transformacao-bpm

Portanto, foram analisados no total 12 tipos de mudança. Para a seleção das transformações foram seguidos os seguintes critérios: (i) elas não deveriam ser extensas (até 1.400 LOC), dado que os participantes tinham que analisar manualmente cada mudança e esta tarefa poderia exigir muito tempo para análise de transformações extensas; e (ii) os tipos de mudança deveriam ser diferentes para que a avaliação fosse realizada com mudanças diversificadas.

Para o contexto de ATL, os tipos de mudança devem ser aplicados aos seus elementos (*called rules*, *matched rules*, *lazy matched rules*, *attributes* e *helpers*). Neste estudo, os tipos de mudança utilizados foram os seguintes: *changeFunctionContextType*, *removeOutputModelElement*, *addTransformationBlock*, *changeConstantType*, *changeConstantName*, *changeOutputModelElementType*, *removeOutputModelElementBinding*, *removeTransformationBlockInheritance*, *changeConstantContextType*, *removeTransformationBlockFilter*, *changeTransformationBlockName* e *changeFunctionParameterType*.

Foram selecionados apenas quatro transformações ATL e três tipos de mudança para cada uma delas a ser analisada devido ao tempo total que os participantes iriam gastar para executar o estudo. Se fosse selecionado um grande número de transformações, os resultados do estudo poderiam ter sido prejudicados devido à fadiga dos participantes. Além disso, os participantes poderiam se sentir desencorajados a realizar o estudo.

Para a execução deste estudo, foi fornecida toda a instrumentação necessária para os participantes, como seguem:

- *Treinamento em transformações de modelo ATL*, onde o condutor do estudo mostrou aos participantes os conceitos de ATL necessários para o estudo. Este treinamento teve o intuito de nivelar o conhecimento dos participantes, tendo em vista que alguns deles, apesar da experiência com ATL, já não trabalhavam com a linguagem há algum tempo;

- *Apresentação do estudo de caso*, onde o condutor apresentou o contexto do estudo, objetivos, material a ser utilizado, os tipos de mudanças a serem analisados e as tarefas a serem executadas;

- *Guia de execução*, um formulário onde os participantes seguiram um conjunto de instruções para executar o estudo.

No intuito de coletar todos os dados deste estudo, foi aplicado o guia de execução para cada participante, onde eles seguiram as instruções para analisar cada tipo de mudança e,

em seguida, eles responderam às perguntas necessárias relativas à mudança sendo analisada. Para servir de auxílio durante a fase de análise e interpretação, foi elaborado um banco de dados que contém todos os dados coletados.

8.3 Execução do Estudo

Antes da execução deste estudo de caso, primeiramente, foi necessário realizar um estudo piloto para testar o estudo de caso que foi planejado (na fase anterior) e corrigir quaisquer problemas ocorridos ao longo de sua execução. Para conduzir o estudo piloto foi selecionado aleatoriamente um participante. O condutor do estudo fez uma apresentação para ele sobre os conceitos de ATL necessários para o estudo, bem como sobre o estudo de caso a ser realizado. Em seguida, o participante seguiu o guia de execução e respondeu as perguntas necessárias no formulário.

Como *feedback* para este estudo piloto, o participante indicou algumas questões do formulário que não deixavam claro o que deveria ser analisado na transformação. Portanto, quando a execução foi concluída, os ajustes foram feitos no guia de execução de acordo com esse *feedback* obtido e, por fim, o estudo de caso foi conduzido.

Como os participantes estavam localizados em lugares diferentes, o estudo foi executado através de vídeo-conferências, onde o condutor do estudo primeiramente apresentou os conceitos sobre transformações ATL necessários para o estudo e, em seguida, ele apresentou o estudo a ser executado. Depois, o condutor atribuiu a cada participante um guia de execução (um formulário online disponível em: *http://goo.gl/40BUvz*) com uma ordem diferente das mudanças a serem analisadas, que foram definidas aleatoriamente para evitar que a análise das últimas mudanças sejam afetadas devido ao cansaço dos participantes. Eles seguiram o guia de execução e após a análise de cada mudança, eles responderam um questionário com as seguintes perguntas:

- *Q1:* Quais elementos você identifica como impactados por a mudança (se houver)?
- *Q2:* Justifique. Que critérios você adotou para identificar os elementos impactados?
- *Q3:* Como você classifica o impacto desta mudança (baixo ou alto)?
- *Q4:* Justifique. Que critérios você adotou para classificar o impacto?

É importante ressaltar que nas questões *Q2* e *Q4* não foram apresentados aos participantes do estudo exemplos de critérios que eles poderiam adotar, os critérios foram inteiramente fornecidos (respondidos) pelos próprios participantes.

As respostas das questões *Q2* e *Q4* foram úteis para que se pudesse alcançar o primeiro objetivo deste estudo, que foi o de servir de embasamento na identificação de métricas a partir de uma análise dos critérios adotados pelos participantes [Vieira e Ramalho, 2014a]. Já as respostas das questões *Q1* e *Q3* ajudaram a avaliar a abordagem proposta, comparando-as com os resultados obtidos pela abordagem.

Cada participante gastou cerca de três horas para executar este estudo de caso. Depois que a execução foi concluída, os dados coletados foram organizados em um banco de dados e analisados para a verificação de sua validade. Foi detectado que apenas dois dos participantes não haviam concluído a análise de todas as mudanças e foi necessário eliminá-los do estudo. Portanto, as respostas de 12 participantes foram usadas para análise.

Para finalizar a execução do estudo, a abordagem proposta foi utilizada para calcular o valor de impacto de cada mudança e para obter os elementos impactados. Em seguida, os resultados obtidos com a abordagem foram comparados com os obtidos pela análise manual realizada pelos participantes.

8.4 Análise e Interpretação dos Resultados

Como este estudo de caso foi dividido em dois grandes objetivos, nesta seção será apresentada a análise e interpretação dos resultados obtidos para alcançar cada um deles. Primeiramente, será discutido sobre a identificação de novas métricas para medir o impacto de mudanças em transformações de modelos (Subseção 8.4.1). Em seguida, será discutida a avaliação da abordagem proposta neste trabalho (Subseção 8.4.2). Por fim, as conclusões sobre as questões de pesquisa elaboradas no estudo são discutidas (Subseção 8.4.3).

8.4.1 Identificação de Métricas

As respostas de todos os participantes foram analisadas a fim de se entender os critérios que eles adotaram para identificar os elementos impactados pela mudança e classificar os impactos. Foi possível notar que os participantes não tinham um consenso sobre os critérios

adotados, mas a maioria deles classificou os impactos considerando o número de elementos afetados com a mudança. Segue uma compilação de todos os critérios adotados pelos participantes para analisar o impacto das mudanças. Em parênteses, é informado o número de participantes que adotou cada um dos critérios.

- *Papel do elemento a ser modificado* (02) - cada elemento desempenha um determinado papel na transformação e possui as suas próprias características. Por exemplo, uma *matched rule* pode ser uma regra abstrata e estender outra regra. Por outro lado, um *helper* pode ser invocado por outros elementos da transformação e aceitar parâmetros. Portanto, os participantes indicaram que quanto mais relevantes forem as características de um elemento, maior será o impacto da mudança;

- *Relações entre elementos* (11) - dentro de uma transformação podem existir várias relações entre seus elementos. Por exemplo, se uma regra *R1* estender outra regra *R2* pode-se dizer que existe uma relação de herança entre *R1* e *R2*. Portanto, os participantes indicaram que quanto mais relações um elemento assumir, maior será a sua relevância dentro da transformação e maior será o impacto da mudança;

- *Número de elementos impactados* (09) - para cada mudança a ser aplicada, os participantes identificaram o elemento a ser modificado dentro da transformação e eles procuraram as possíveis dependências deste elemento. Por exemplo, se um *helper* que é invocado por duas regras é removido da transformação, essas regras serão os elementos impactados pela mudança. Portanto, os participantes indicaram que quanto maior for o número de elementos impactados pela mudança, maior será o impacto da mudança;

- *Tipo da mudança* (04) - o impacto em uma transformação depende também do tipo de mudança a ser aplicada. Alguns tipos de mudança podem impactar mais do que outros. Por exemplo, a mudança *addTransformationBlockParameter* geralmente causa um impacto maior do que a mudança *changeTransformationBlockName*, pois a primeira mudança exige que as demais regras que invocam a regra a ser modificada sejam adaptadas para obter e informar o novo parâmetro adicionado, enquanto que a segunda mudança é apenas uma mudança de nome da regra. Portanto, os participantes

indicaram que quanto mais complexo for o tipo de mudança, maior será o impacto da mudança;

- *Impacto no modelo de saída* (04) - algumas mudanças podem afetar o modelo de saída gerado pelo módulo de transformação. Por exemplo, se uma regra vai ser removida da transformação, então o modelo de saída gerado por esta transformação será afetado. Portanto, os participantes indicaram que quanto mais o modelo de saída for afetado, maior será o impacto da mudança.

Este estudo de caso foi realizado quando tinha sido definido apenas um esboço de três métricas (*RTEC*, *RTERT* e *RITE*). Então, com o *feedback* obtido a partir do estudo e com novos *insigths* foi possível definir neste trabalho um total de sete métricas: *RTEC*, *RTERT*, *RITE*, *RCTAE*, *RCITOM*, *ROMTC* e *RTEATM*, ratificando as três métricas previamente esboçadas.

Depois que todas as métricas foram definidas, a abordagem proposta foi utilizada para calcular os valores de impacto para cada mudança deste estudo e todos os resultados foram analisados, como apresentado a seguir.

8.4.2 Avaliação da Abordagem

Uma vez que os impactos das mudanças já foram calculados pela abordagem proposta e classificados pelos participantes do estudo, nesta seção será feita uma análise da comparação destes resultados.

Vale ressaltar que para esta avaliação não foram consideradas as métricas *ROMTC* e *RTEATM*, pois durante as pesquisas por repositórios de projetos ATL reais não foi encontrada uma quantidade significativa de transformações envolvendo cadeias e dependências entre módulos que viabilizasse a condução deste estudo.

A Tabela 8.1 apresenta as classificações para 5 dos 12 tipos de mudança analisadas neste estudo de caso[5] (a tabela completa está disponível no Apêndice C deste documento), onde: (i) A coluna 1 mostra os tipos de mudança; (ii) a coluna 2 mostra os impactos classificados pela abordagem e seus respectivos valores de impacto informados entre parênteses; (iii) a co-

[5]Para não tornar a leitura exaustiva, são discutidas 5 mudanças, que são suficientes para representar a análise realizada.

luna 3 mostra os impactos classificados pelos participantes, onde o número de participantes que indicaram aquele valor é informado entre parênteses; (iv) a coluna 4 mostra o número de elementos impactados identificados pela abordagem; e (v) a coluna 5 mostra o número de elementos impactados identificados pelos participantes, onde o número de participantes que indicaram aquele valor é informado entre parênteses. Cada um dos tipos de mudança e suas respectivas análises são discutidas a seguir.

Tabela 8.1: Classificação dos impactos e dos elementos impactados.

Tipo de Mudança	**Classificação (abordagem)**	**Classificação (participantes)**	**N° Elem. Imp. (abordagem)**	**N° Elem. Imp. (participantes)**
removeOutputModelElementBinding	Baixo (0.24)	Baixo (11) Alto (01)	01	01 (08) 02 (02) 05 (02)
changeConstantName	Baixo (0.26)	Baixo (12) High (0)	04	01 (01) 03 (01) 04 (08) 05 (02)
changeConstantContextType	Alto (0.48)	Baixo (01) Alto (11)	08	05 (01) 06 (01) 07 (01) 08 (03) 09 (05) 10 (01)
changeFunctionContextType	Baixo (0.34)	Baixo (09) Alto (03)	03	02 (01) 03 (06) 04 (05)
changeConstantType	Baixo (0.37)	Baixo (06) Alto (06)	10	08 (02) 09 (01) 10 (04) 11 (05)

RemoveOutputModelElementBinding - mudança referente à "remoção de um determinado *binding* de um elemento de saída especificado na *matched rule* `OutputBinding` da transformação *SimpleGTtoEMFTVM*". A maioria dos participantes (11) classificou o impacto desta mudança da mesma forma que a abordagem proposta (*baixo*), dado que eles acreditam que poucos elementos (um, dois ou cinco) serão impactados por esta mudança e eles consideraram os elementos impactados como critério para classificar o impacto. Por outro lado, apenas um participante classificou o impacto dessa mudança como *alto* porque ele acredita que a mudança na *matched rule* `OutputBinding` também impactará o modelo

de saída gerado pela transformação. No período em que este estudo foi conduzido, ainda não havia sido definida uma métrica que considerasse os impactos nos modelos de saída. Portanto, a fim de melhorar a abordagem proposta, com o *feedback* desse participante foi definida uma nova métrica chamada *RCITOM* (anteriormente apresentada na Seção 5.3.5 do Capítulo 5) para considerar os impactos no modelo de saída das transformações;

ChangeConstantName - mudança referente à "alteração do nome do atributo `InheritedTypes` especificado na transformação *PITM2PSTM*". Todos os participantes classificaram o impacto desta mudança da mesma forma que a abordagem proposta (*baixo*), pois eles acreditam que ela não vai afetar a semântica do atributo e é fácil de ser aplicada: é necessário apenas procurar o nome do atributo dentro da transformação e substituí-lo pelo novo nome. Dado que a abordagem considera como impactados todos os elementos que invocam o elemento a ser modificado, o impacto vai aumentar à medida que o número de elementos impactados aumenta, mesmo quando a mudança é simples. Tendo em vista que os participantes consideraram o nível de dificuldade para aplicar a mudança como critério, para melhorar a abordagem proposta foi definida uma nova métrica chamada *RCTAE* (anteriormente apresentada na Seção 5.3.4 do Capítulo 5) para considerar a dificuldade de aplicar uma mudança;

ChangeConstantContextType - mudança referente à "alteração do tipo de contexto do atributo `matcherPatterns` especificado na transformação *SimpleGTtoEMFTVM*". A maioria dos participantes (11) classificou o impacto desta mudança da mesma forma que a abordagem proposta (*alto*) devido ao grande número de elementos impactados pela mudança. Por outro lado, apenas um participante classificou o impacto desta mudança como *baixo*, uma vez que ele acredita que a mudança do tipo de contexto de um atributo não afetará a sua lógica. Para uma melhor análise dessa mudança, nota-se mais uma vez a importância da métrica *RCTAE* que foi definida, visto que ela considera a dificuldade de se aplicar a mudança. Portanto, caso a mudança afete a lógica do elemento a ser modificado, maior será a dificuldade de aplicá-la e maior será o seu impacto;

ChangeFunctionContextType - mudança referente à "alteração do tipo de contexto do *helper* `isMultipleIncoming` especificado na transformação *BPMN_para_Atividades*". A maioria dos participantes (09) classificou o impacto desta mudança da mesma forma que a abordagem proposta (*baixo*), pois eles acreditam que um pequeno número de elementos são

impactados pela mudança. Por outro lado, três participantes classificaram o impacto como *alto*, uma vez que eles acreditam que a mudança vai afetar a lógica do *helper* e por isso o impacto deve ser alto. Além disso, alguns deles consideraram o papel do elemento a ser modificado na transformação. A partir do *feedback* desses participantes, foi considerado um índice de relevância diferente para cada tipo de elemento na métrica *RCTAE*, dependendo do papel dele na transformação;

ChangeConstantType - mudança referente à "alteração do tipo do atributo `voidType` especificado na transformação *PITM2PSTM*". Como se pode ver na Tabela 8.1, o valor de impacto (*0.37*) calculado pela abordagem proposta refere a um impacto *baixo*, mas esse valor foi exatamente o limite que separa a classificação de um impacto como baixo ou alto. Pelas respostas dos participantes nota-se que eles ficaram divididos: metade classificou o impacto como *baixo* e metade como *alto*. Portanto, pode-se dizer que a classificação deles foi bastante próxima da abordagem proposta. No geral, eles consideraram os elementos impactados como critério para definir o impacto.

Este estudo de caso exploratório foi útil para investigar como os usuários manualmente analisavam o impacto de mudanças em transformações de modelo. Os *insights* significativos encontrados foram introduzidos na abordagem proposta, a fim de melhorá-la. Por exemplo, considerar nas métricas: (i) o impacto da mudança no modelo de saída da transformação; (ii) o tipo de mudança a ser aplicada; e (iii) o nível de dificuldade de aplicar a mudança.

Com este estudo não se pode concluir que os impactos classificados pela abordagem proposta estão corretos e os classificados pelos participantes estão errados, ou vice-versa. No entanto, pode-se afirmar que, mesmo os participantes não tendo ideia de como a abordagem calculava os impactos, os resultados obtidos a partir da análise de impacto de mudanças manual foram muito próximos dos resultados da abordagem. Isto indica que as métricas definidas neste trabalho estão seguindo a mesma direção que os participantes para calcular o impacto das mudanças.

Em geral, considerando também o esforço e o tempo de análise, pode-se observar que os resultados obtidos com a abordagem proposta foram muito melhores do que com a análise manual, pois há uma redução do esforço e tempo de análise dado que o valor de impacto e os elementos da transformação afetados com a mudança são detectados automaticamente. Por exemplo, os participantes gastaram cerca de três horas para classificar o impacto das mu-

danças e identificar os elementos impactados no estudo de caso, enquanto que a abordagem realizou estas mesmas tarefas automaticamente. Além disso, a abordagem ajuda os gerentes de projeto a melhor priorizar as mudanças de acordo com o valor do impacto de cada delas em toda a transformação.

8.4.3 Discussões Sobre as Questões de Pesquisa

A primeira questão de pesquisa elaborada para este estudo de caso foi a seguinte: *Quais critérios os usuários adotam para identificar os elementos impactados e medir o impacto de mudanças em transformações de modelos?*

Após a análise das respostas de cada participante do estudo, foi possível entender como eles analisam manualmente o impacto em uma transformação antes mesmo de aplicar uma mudança. Portanto, respondendo à primeira questão de pesquisa, os seguintes critérios foram identificados (a Seção 8.4.1 apresentou mais detalhes sobre cada um deles):

- O papel, dentro da transformação, do elemento a ser modificado;
- As relações entre o elemento a ser modificado e os demais elementos da transformação;
- O número de elementos impactados pela mudança;
- O tipo da mudança a ser aplicada;
- O impacto da mudança no modelo de saída da transformação.

A segunda questão de pesquisa elaborada para este estudo de caso foi a seguinte: *Quão divergentes são os resultados da análise de impacto de mudanças realizada por nossa abordagem e pela análise manual?*

Em relação ao número de elementos impactados, como mostram as colunas 4 e 5 da Tabela 8.1 previamente ilustrada, pode-se observar que os resultados da abordagem proposta foram muito próximos dos resultados dos participantes. Em geral, a diferença observada em alguns tipos de mudança é devido ao fato de que alguns participantes consideraram como impactado o próprio elemento da transformação a ser modificado. É importante mencionar que, para algumas mudanças, embora os participantes tenham identificado o mesmo número de elementos impactados que a abordagem proposta, os elementos que eles identificaram não

foram exatamente os mesmos identificados pela abordagem. Além disso, como os participantes identificaram manualmente os elementos impactados, esta tarefa é propensa a erros.

Os critérios que os participantes adotaram para classificar os impactos das mudanças não contemplaram em sua essência todos os critérios (métricas) usados pela abordagem proposta, tais como as características e os tipos de relações que o elemento a ser modificado assume. Mesmo os participantes não tendo conhecimento prévio sobre como a abordagem calculou os impactos, os resultados obtidos a partir da análise manual foram muito próximos dos resultados da abordagem proposta.

Para a compreensão do quão divergentes foram as respostas (em relação à classificação dos impactos e ao número de elementos impactados) entre os próprios participantes deste estudo, foi utilizado o método chamado ICC (*Intraclass Correlation*) [Uebersax, 2007]. Trata-se de uma medida geral de concordância ou consenso entre dois ou mais avaliadores (*raters*) sobre o mesmo conjunto de assuntos (*subjects*). Neste estudo, os avaliadores foram os participantes e os assuntos foram os tipos de mudança. Como entrada, o método ICC requer uma tabela onde as linhas são os assuntos e as colunas são os avaliadores. Portanto, cada célula da tabela corresponde à resposta de um avaliador sobre um assunto.

O resultado do método ICC é um número entre *0* e *1*, onde *1* representa perfeita concordância e *0* representa nenhuma concordância. Para a utilização do ICC foi necessário adaptar os dados coletados durante o estudo para o formato exigido: uma tabela onde as linhas representavam as mudanças e as colunas representavam os participantes. Então, foram definidas duas tabelas: a primeira, onde cada célula correspondia à classificação que um participante atribuía a uma mudança; e a segunda, onde cada célula correspondia ao número de elementos impactados que um participante identificava para uma mudança. A ferramenta online chamada StatsToDo [Chang, 2013] foi usada para calcular o ICC para as duas tabelas.

Existem cinco níveis de concordância para o ICC (pobre, razoável, moderada, forte e quase perfeita). Em relação ao número de elementos impactados, como resultado foi obtido o ICC de *0.83*, que corresponde a uma *concordância quase perfeita*. Isto significa que a grande maioria dos participantes concordaram entre si sobre o número de elementos impactados para cada tipo de mudança, apesar de haver divergências, elas foram poucas. Vale ressaltar que, embora os participantes tenham identificado o mesmo número de elementos impactados para algumas mudanças, alguns dos elementos identificados não foram exata-

mente os mesmos. Já em relação à classificação dos impactos, como resultado foi obtido o ICC de *0.40*, que corresponde a uma *concordância razoável* entre os participantes. Embora apenas analisando os dados já seja possível observar a divergência entre os próprios participantes na classificação dos impactos, o valor do ICC foi usado para confirmar e quantificar tal divergência. Como já era esperado, com este estudo foi possível observar que os participantes não têm um consenso comum para classificar o impacto das mudanças, que se torna uma prática bem subjetiva. Esta divergência pode ser explicada pela pouca experiência deles com análise de impacto de mudanças em transformações.

8.5 Ameaças à Validade

Nesta seção são discutidas as ameaças à validade que foram identificadas neste estudo e como seus efeitos foram controlados ou mitigados.

Validade Interna: Neste estudo, os participantes tiveram que analisar 12 tipos de mudança e responder quatro perguntas para cada uma delas, assim, gastando cerca de três horas para a conclusão do estudo. Considerando isto, é possível que a análise das últimas mudanças seja afetada pela fadiga dos participantes ou pela pressa em terminar o estudo. Por outro lado, a análise das últimas mudanças também poderia ser afetada pelo conhecimento adquirido ao longo do estudo, de forma que os participantes poderiam responder as últimas perguntas com mais confiança do que as primeiras. Para atenuar essas ameaças, foram elaboradas diferentes versões do guia de execução e, para cada uma delas, a ordem das mudanças a serem analisadas foi randomizada. Além disso, os guias de execução foram aleatoriamente atribuídos aos participantes.

Outra ameaça à validade interna é a experiência dos participantes. Uma vez que a análise de impactos é subjetiva, eles podem adotar critérios diferentes para classificar as mudanças e identificar os elementos impactados. Além disso, a experiência dos participantes pode influenciar os resultados de nosso estudo. Para atenuar esta ameaça, foi oferecido a todos os participantes um treinamento sobre transformações de modelo ATL, incluindo os conceitos necessários para a execução do estudo.

Validade Externa: Como o estudo de caso foi usado para a avaliação empírica deste trabalho, algumas condições não permitem a generalização dos resultados. Primeiramente,

devido ao limite de pessoas disponíveis para participar do estudo, pois os resultados foram obtidos a partir da análise de apenas 12 participantes. Além disso, embora as transformações utilizadas neste estudo sejam reais e não-triviais, foi analisado um pequeno número de tipos de mudança (12).

Capítulo 9

Trabalhos Relacionados

O trabalho desenvolvido nesta tese está inserido em três contextos principais: análise estática, análise de impacto de mudanças e definição de métricas. Foi investigado e estudado o estado da arte de cada um deles com o intuito de descobrir o que já foi realizado no escopo de transformações de modelos. Os principais trabalhos relacionados encontrados são discutidos nas seções seguintes.

9.1 Análise Estática

Atualmente, o campo da Engenharia de Software oferece uma série de ferramentas que propõem a análise estática de programas escritos em diversas linguagens de programação. Algumas destas ferramentas são apresentadas a seguir.

FindBugs [FindBugs, 2013], *Checkstyle* [Checkstyle, 2014] e *Design Wizard* [Brunet, 2009] são softwares livres que propõem a análise estática de programas implementados em Java. O *FindBugs* é uma ferramenta bastante popular originalmente desenvolvida por Bill Pugh e David Hovemeyer. Ela inspeciona o *bytecode* de classes Java à procura de potenciais problemas. Já existem *plug-ins* do *FindBugs* disponíveis para várias IDEs de desenvolvimento, tais como o Eclipse [Loskutov, 2011], o Netbeans [Prox, 2007] e o IntelliJ IDEA [Pfeiler, 2012]. Por outro lado, o *Checkstyle* é uma ferramenta originalmente desenvolvida por Oliver Burn com o objetivo de verificar se código fonte de programas Java adere a um padrão de codificação. Ele é recomendado para projetos que pretendem forçar os desenvolvedores a seguir um estilo de programação específico. Por padrão, o *Checkstyle* oferece

suporte às convenções de código da Sun [Oracle, 1999], mas ele é altamente configurável. Além disso, esta ferramenta fornece algumas funcionalidades para verificar erros, tais como a verificação de código duplicado. O *Design Wizard*, por sua vez, é uma API desenvolvida por João Brunet que permite extrair informações do *bytecode* de programas Java e oferece diversos métodos de consulta às informações extraídas. Ele é bastante útil para permitir revisões de código automatizadas, bem como para escrever testes de design [Brunet et al., 2009] de uma forma fácil e limpa. Vale ressaltar que a API proposta por João Brunet foi uma das principais fontes de inspiração para o desenvolvimento do analisador estático de ATL desenvolvido neste trabalho.

MOPS (MOdelchecking Programs for Security properties) [Chen e Wagner, 2002] e *UNO* [Holzmann, 2002] são softwares para análise estática de programas implementados em C. O *MOPS* é um software livre originalmente desenvolvido por Hao Chen e David Wagner. Ele é usado para encontrar falhas de segurança (aquelas que violam propriedades de segurança temporais[1]) e para verificar a conformidade com regras de programação defensiva. O *MOPS* é focado em identificar regras de práticas de programação segura. Então, ele verifica se as propriedades de segurança estão sendo obedecidas dentro de um programa. Por outro lado, o *UNO* é um projeto de pesquisa acadêmico originalmente desenvolvido por Gerard Holzmann. Ele é capaz de detectar os três tipos mais comuns de defeitos de software: uso de variáveis não inicializadas, referências a ponteiros nulos e indexação fora dos limites do array (*out-of-bounds array indexing*). O *UNO* analisa um software baseado em um conjunto pré-definido de propriedades, mas ele também permite aos usuários especificar suas próprias propriedades.

Adicionalmente, existem alguns softwares que são multilinguagens, isto é, eles oferecem suporte à análise estática de programas escritos em diferentes linguagens. Exemplos destes softwares são o *Yasca (Yet Another Source Code Analyzer)* [Scovetta, 2010] e o *Coverity Prevent* [Coverity, 2012]. *Yasca* é um software livre originalmente desenvolvido por Michael Scovetta. Ele oferece suporte à análise estática de código fonte escrito em Java, C, C++, HTML, JavaScript, ASP, ColdFusion, PHP, COBOL, .NET, dentre outras linguagens. Por outro lado, o *Coverity Prevent* é um software comercial desenvolvido pela empresa Coverity. Ele oferece suporte à análise estática de código fonte escrito em C, C++, C# e Java.

[1] Propriedades de segurança temporais simplesmente descrevem a ordem de uma sequência de operações.

No contexto de MDD, há uma carência muito grande de ferramentas que permitam a análise estática das transformações. Nesse sentido, foi encontrado apenas o trabalho desenvolvido por Ujhelyi et al. [Ujhelyi et al., 2009], que propõe uma abordagem para análise estática. Contudo, o analisador estático proposto serve para detectar erros comuns em transformações especificadas apenas em VIATRA2 VTCL (*Viatra Textual Command Language*) [Eclipse Foundation, 2011], que é uma linguagem de transformação de grafos (*Graph Transformation Languages*) que descreve transformações de modelos como uma série de transformações de grafo. Para definir estas séries algum tipo de estrutura de controle é usada, como máquinas de estado abstratas [Börger, 2003]. A abordagem proposta por Ujhelyi et al. diferencia-se do trabalho de tese proposto tendo em vista que ela é focada na identificação de erros comuns em transformações e não na análise de impacto de mudanças. Além disso, ela não é aplicada à transformações ATL e não provê uma API com métodos que permitem o acesso a qualquer elemento especificado na transformação.

Por outro lado, Tisi et al. [Tisi et al., 2010] apresentam uma série de propostas para facilitar a definição de *HOT* (*Higher-Order Transformations*)[2] em ATL, onde cada proposta está focada em uma classe de transformação específica. Estas propostas são baseadas na análise de um conjunto de 42 transformações em ATL disponíveis. Por exemplo, este trabalho propõe uma biblioteca de HOTs composta pelos *helpers* que foram recorrentemente utilizados dentro do conjunto dos 42 HOTs analisados. Apesar do trabalho proposto por Tisi et al. oferecer, dentre outras propostas, uma API com vários *helpers*, ele não está focado na análise estática das transformações, não permitindo aos desenvolvedores obterem informações sobre qualquer elemento especificado em uma transformação ATL.

Como visto nos trabalhos citados, não foi encontrada nenhuma técnica existente de análise estática para a linguagem de transformação de modelos ATL. Além disso, as técnicas já existentes no campo da engenharia de software não puderam ser reutilizadas porque elas são específicas para determinadas linguagens de programação, tendo seus próprios conceitos e estruturas.

[2]HOT é um tipo de transformação que tem outra transformação como modelo de entrada e/ou saída. Ele pode ser especificado em qualquer linguagem de transformação.

9.2 Análise de Impacto de Mudanças

Vários trabalhos no escopo de análise de impacto de mudanças foram encontrados. A seguir, os principais deles são discutidos de acordo com três contextos diferentes: (i) Contexto de Transformações (Subseção 9.2.1), onde se enquadra a abordagem proposta neste trabalho; (ii) Contexto de Metamodelos (Subseção 9.2.2); e Contexto de Linguagens de Programação (Subseção 9.2.3).

9.2.1 Contexto de Transformações

A análise de impacto de mudanças é um campo de pesquisa muito incipiente no contexto de MDD, principalmente, relacionado a transformações. Poucos trabalhos relacionados ao proposto nesta tese foram encontrados.

O trabalho proposto por Wimmer et al. [Wimmer et al., 2012] define um catálogo de refatorações que visam melhorar os atributos de qualidade relacionados à manutenção de transformações de modelo (tais como a legibilidade, reusabilidade e extensibilidade), assim como o desempenho destas transformações. O catálogo foi definido com base na experiência dos autores em MDD, assim como na análise de transformações ATL existentes vindas de diferentes fontes. Como resultado, as refatorações foram definidas em quatro categorias: renomeação, reetruturação, herança e otimização de expressões OCL. O catálogo de refatorações também pode ser aplicado a outras linguagens de transformação de modelo que seguem o paradigma baseado em regras, tais como QVT. A aplicação destas refatorações pode ser semiautomática através da utilização de HOTs.

O trabalho proposto por Wimmer et al. é estritamente focado em mudanças para transformações de modelos. No entanto, as mudanças adotadas nele são apenas de refatorações e, além do mais, o trabalho proposto por Wimmer et al. não identifica os elementos impactados nem mensura o impacto da mudança (refatoração) causado na transformação.

Por outro lado, Marzullo et al. [Marzullo et al., 2010] propõem utilizar técnicas de MDA juntamente com anotações no modelo dos projetos para automatizar a abordagem de análise de impacto de mudanças, a qual é um dos módulos do projeto MDA4Eclipse [Object Management Group, 2012]. A ideia apresentada por Marzullo et al. é anotar (rotular) os elementos usuais de um modelo (classes, pacotes, métodos, relacionamentos, etc.) para

permitir ao engenho MDA identificar, a partir de tais anotações, quais requisitos determinado elemento representa. Desta forma, sendo possível a geração de código fonte anotado e útil para análise, isto é, a anotação servirá para rastrear qualquer código fonte gerado para um determinado requisito e vice-versa. Apesar do trabalho de Marzullo et al. envolver MDA e análise de impacto de mudanças, ele usa técnicas de MDA para auxiliar a análise de impacto de mudanças em projetos de software. O seu foco é mais voltado para o rastreamento dos requisitos de um software para seu código fonte e vice-versa, assim, facilitando a análise de impacto de mudanças no projeto.

Conforme foi discutido, atualmente não há nenhum trabalho focado na análise de impacto de mudanças em transformações de modelos. Isso significa que se trata de um campo de pesquisa incipiente e com um grande potencial, no qual o trabalho desta tese é promissor dado que é o único que propõe uma abordagem, amparada na análise estática, para a análise de impacto de mudanças em transformações de modelos.

9.2.2 Contexto de Metamodelos

Apesar da análise de impacto de mudanças ser um campo de pesquisa muito incipiente no contexto de MDD, alguns trabalhos no contexto de metamodelos foram encontrados.

Garcia e Díaz [Garcia e Díaz, 2010] propõem uma abordagem para auxiliar os desenvolvedores a adaptar suas transformações de modelo às mudanças realizadas no metamodelo. A abordagem proposta utiliza *Higher-Order Transformations* para gerar *transformações anotadas*, isto é, transformações que incluem sugestões sobre como as mudanças no metamodelo impactam as regras e quais regras são impactadas. Esta abordagem é semiautomática, pois apenas algumas transformações podem ser completamente geradas, dependendo do tipo de mudança aplicada. O trabalho proposto por Garcia e Díaz não permite aos desenvolvedores analisar nem mensurar o impacto em todo o projeto causado por uma mudança aplicada em uma transformação, seu foco é na análise de mudanças apenas em metamodelos.

Já o trabalho proposto por Mendez et al. [Mendez et al., 2010] define uma metodologia para adaptar transformações de modelos depois que os metamodelos referenciados por elas são modificados. Ele propõe um conjunto de tarefas que devem ser realizadas para reestabelecer a consistência depois da mudança do metamodelo, são elas: (i) detecção do impacto, onde as inconsistências causadas pela mudança do metamodelo são identificadas; (ii) análise

do impacto, onde as alterações necessárias para deixar a transformação consistente com o metamodelo modificado são obtidas através de uma assistência humana; e (iii) adaptação da transformação, onde as alterações encontradas na tarefa anterior são aplicadas. Porém, esta última tarefa é apenas citada como trabalho futuro.

Nessa mesma direção, o trabalho proposto por Garcés et al. [Garcés et al., 2012] também propõe uma técnica para adaptar transformações de modelos depois que um metamodelo é modificado. Trata-se de uma técnica semiautomática que gera as transformações adaptadas com as mudanças do metamodelo através de *composição de transformação externa*: uma cadeia de transformações simula o comportamento das transformações impactadas, onde cada cadeia é apropriada para um dado cenário que é caracterizado pelo tipo de mudança aplicada ao metamodelo de entrada ou/e de saída. As mudanças consideradas por Garcés et al. estão no contexto de refatoração (alterações de elementos), construção (adição de novos elementos) e destruição (remoção de elementos). Esta abordagem gera as transformações através da linguagem de comparação chamada AtlanMod (AML) [Garcés et al., 2009], uma DSL para definir estratégias de comparação de modelos que computa diferenças e equivalências entre modelos.

Ainda em relação a evolução de metamodelos, Iovino et al. [Iovino et al., 2012] aborda a identificação, previsão e avaliação da importância que o impacto de mudanças no metamodelo tem sobre os artefatos existentes. Ele propõe uma abordagem que permite aos desenvolvedores estabelecer relações entre o metamodelo e seus artefatos relacionados, bem como detectar automaticamente esses elementos dentro dos artefatos pelas mudanças no metamodelo. Além disso, existem alguns trabalhos que propõem automatizar a migração de modelos existentes de acordo com o metamodelo evoluído [Garcés et al., 2009], [Herrmannsdoerfer et al., 2009], [Paternostro e Hussey, 2006] e [Rose et al., 2010b]. Uma comparação entre eles é apresentada por Rose et al. [Rose et al., 2010a].

Embora Garcia e Díaz, Mendez et al., Garcés et al. (2012), Iovino et al., Garcés et al. (2009), Herrmannsdoerfer et al., Paternosrto e Hussey, e Rose et al. sejam bem relacionados à abordagem de impacto de mudanças proposta nesta tese, eles estão focados em mudanças (evolução) de metamodelos. Além do mais, eles não permitem que os desenvolvedores mensurem o impacto causado na transformação por uma mudança aplicada a qualquer elemento da própria transformação.

9.2.3 Contexto Geral de Engenharia de Software

Dentro da Engenharia de Software, a análise de impacto de mudanças de software já é um tema bastante amadurecido e já existem vários trabalhos que propõem abordagens para esta finalidade [Hattori et al., 2008] [Maia et al., 2010] [Zhang et al., 2008] [Ren et al., 2004]. Nesta seção serão discutidos apenas dois dos trabalhos encontrados que também fazem uso da análise estática e serviram como referência no desenvolvimento desta tese.

O trabalho proposto por Hattori et al. [Hattori et al., 2008] é voltado para a análise de impacto de programas escritos em Java. Ele propõe e avalia uma técnica probabilística que permite a ordenação dos impactos por probabilidade, bem como a redução do número de falso-positivos, aumentando assim a precisão da análise de impacto. Para isto, a técnica proposta: (i) identifica os impactos de uma mudança antes mesmo que ela seja realizada, ou seja, utilizando a análise estática; e (ii) atribui probabilidades de ocorrência aos impactos através do uso de informação sobre o histórico de mudanças do software analisado. A técnica foi implementada em uma ferramenta chamada *Impala*, que utiliza o *Design Wizard* proposto por Brunet [Brunet, 2009] para gerar, em forma de entidades e relacionamentos, a representação do sistema a ser analisado.

Similarmente ao trabalho de Hattori et al., o trabalho proposto por Maia et al. [Maia et al., 2010] é voltado para a análise de impacto de programas escritos em Java. Ele propõe e avalia uma técnica híbrida para análise de impacto que tem por objetivo reduzir o número de falso-negativos[3] encontrados e ordenar os impactos de acordo com sua relevância para a mudança. Baseada tanto na análise estática quanto na análise dinâmica, a técnica proposta: (i) usa análise estática para identificar dependências estruturais entre entidades de código; (ii) usa análise dinâmica para identificar dependências baseado nas sequências de execução dos rastros do software; e (iii) ordena os resultados de ambas as análises de acordo com a sua relevância. A técnica híbrida proposta por Maia et al. foi implementada como um protótipo de ferramenta chamada *SD-Impala*, que é uma extensão da *Impala* [Hattori et al., 2008] adicionando a análise dinâmica e ordenando os resultados em termos da provável importância de uma entidade no conjunto de impactos.

Além destes trabalhos apresentados não estarem no escopo de transformação de modelos, eles não oferecem uma técnica para calcular o valor de impacto para a aplicação da mudança.

[3] São impactos que ocorrem de fato, mas não são identificados pela técnica de análise de impacto

9.3 Métricas

Antes das métricas serem definidas neste trabalho de tese no intuito de calcular o valor de impacto para a aplicação das mudanças, as métricas já existentes foram investigadas na literatura tanto no contexto de MDD quanto no contexto geral da engenharia de software. O intuito era reutilizar métricas já existentes ou usá-las de inspiração para a definição das métricas da abordagem proposta nesta tese. As métricas de software são amplamente utilizadas para avaliar a qualidade de software em geral, mas no contexto de MDD poucos trabalhos foram realizados nesse sentido.

Dentro do contexto MDD, mais especificamente focado em transformações de modelos ATL, o trabalho proposto por Vignaga [Vignaga, 2009] define um conjunto de 81 métricas para medir transformações de modelos e permitir avaliar a sua qualidade. As métricas são específicas para a linguagem ATL e são organizadas de acordo com a metaclasse (do metamodelo de ATL) para a qual elas se aplicam. Estas métricas foram derivadas do metamodelo de ATL com base na intuição de quais propriedades de uma transformação ATL seriam interessantes medir.

Nessa mesma direção, Amstel et al. [van Amstel et al., 2008] propõem um conjunto de 27 métricas para permitir avaliar a qualidade de transformações de modelos. No entanto, as métricas são aplicadas às transformações especificadas utilizando o sistema de reescrita de termo ASF+SDF [van den Brand et al., 2001], que é muito usado para especificar transformações entre linguagens. Em uma versão mais recente deste mesmo trabalho, Amstel et al. [van Amstel et al., 2010] propõem três conjuntos de métricas, onde cada conjunto corresponde à métricas específicas para determinada linguagem de transformação: ATL, QVT Operational Mappings [Object Management Group, 2011a] e Xtend [Efftinge, 2006].

Conforme foi discutido, as métricas propostas por Vignaga, Amstel et al. (2008) e Amstel et al. (2010) são específicas para determinadas linguagens, enquanto as métricas propostas neste trabalho de tese podem ser aplicadas a qualquer transformação de modelos, independente de linguagem. Além disso, as métricas propostas nestes trabalhos são focadas em medir a qualidade das transformações e não o impacto de mudanças.

Capítulo 10

Considerações Finais

Neste trabalho, foi proposta uma abordagem para análise de impacto de mudanças em transformações de modelos que permite aos usuários identificar todos os elementos do módulo de uma transformação impactados por alguma mudança, bem como calcular o valor de impacto para a mudança. A abordagem é formada por três módulos: (i) o *analisador estático* desenvolvido para inspecionar o código fonte de transformações a fim de permitir aos desenvolvedores obter informações sobre qualquer um de seus elementos através de uma API; (ii) as *métricas* definidas para calcular o valor de impacto de uma mudança; e (iii) a *ferramenta de suporte* desenvolvida para calcular automaticamente o valor de impacto de uma determinada mudança e obter o conjunto de elementos impactados por esta mudança. A abordagem é independente de linguagem de transformação, podendo ser utilizada para analisar o impacto de mudanças em outras linguagens, como QVT, por exemplo. Para isto, o módulo *Analisador Estático* deverá ser provido para a manipulação de elementos dentro de uma transformação QVT.

Um estudo de caso foi conduzido neste trabalho com dois principais objetivos: servir como alicerce e embasamento para a identificação de métricas a partir do *feedback* dos participantes e avaliar a abordagem proposta comparando os seus resultados com uma abordagem manual de análise de impacto de mudanças. Atingindo o primeiro objetivo deste estudo, como resultado foi identificado e definido um conjunto de sete métricas: *RTEC*, *RTERT*, *RITE*, *RCTAE*, *RCITOM*, *ROMTC* e *RTEATM*. Além deste estudo, as métricas foram definidas com base também em análises minuciosas de várias transformações de modelos. Atingindo o segundo objetivo do estudo, a abordagem proposta foi avaliada e os resulta-

dos obtidos com a abordagem foram bem similares aos obtidos com a análise manual dos impactos pelos participantes. Porém, considerando o esforço e tempo de análise, a abordagem proposta torna-se melhor, pois o cálculo do impacto e a identificação dos elementos impactados são automáticos, reduzindo o esforço e tempo gasto do usuário. Além disso, por não ser uma tarefa manual ela não está sujeita a erros de análise. Os resultados obtidos com esta avaliação foram bastante satisfatórios, sendo possível observar que, de fato, os valores calculados para o impacto de cada mudança foram coerentes com a realidade.

Uma grande dificuldade que ocorreu durante a condução do estudo foi encontrar pessoas com experiência e bom conhecimento em transformações de modelos ATL dispostas a participar do estudo. Os participantes foram buscados em vários grupos de pesquisa e nas comunidades, mas apenas 12 se disponibilizaram. Outra dificuldade foi encontrar repositórios com projetos reais de transformações de modelos. Muitos projetos eram privados e outros, em sua maioria, não possuíam um repositório de versões onde fosse possível coletar os tipos de mudanças que já foram aplicados. Por outros lado, muitos repositórios encontrados tinham mudanças simples e não contemplavam cadeias de transformações. Foi possível selecionar três módulos de transformações, considerando, para cada um deles, quatro diferentes tipos de mudanças oriundas do repositório.

Considerando a carência de técnicas para análise de impacto de mudanças na área de MDD, a abordagem proposta neste trabalho servirá para auxiliar os envolvidos em um projeto baseado em MDD ao longo do seu processo de desenvolvimento. Os gerentes de projeto poderão usar a abordagem como suporte para tomar decisões de projeto. Por exemplo, eles poderão priorizar um conjunto de mudanças a serem aplicadas em um módulo de transformação de acordo com o número de elementos impactados ou de acordo com o valor de impacto calculado para cada mudança. Por outro lado, os desenvolvedores poderão usar a abordagem proposta para identificar, automaticamente, todos os elementos da transformação impactados por uma mudança a ser aplicada. Desta forma, eles poderão facilmente detectar partes da transformação que devem ser adaptadas depois que a mudança for aplicada.

10.1 Contribuições

No contexto de transformações de modelos dentro da abordagem de MDD, este trabalho oferece diversas contribuições conforme ressaltadas a seguir:

- *Auxílio aos gerentes de projeto*. A abordagem proposta permitirá aos gerentes de projeto calcular o valor de impacto causado por qualquer mudança e identificar os elementos impactados. Portanto, eles poderão tomar uma decisão mais segura sobre quais mudanças realizar tomando também como critério o impacto que será causado por cada uma delas no projeto como um todo, isto é, eles poderão melhor priorizar as mudanças;

- *Auxílio aos desenvolvedores*. Com a abordagem proposta, os desenvolvedores poderão identificar, automaticamente, todas as partes da transformação que serão afetadas com uma mudança. Dessa forma, eles poderão mais facilmente aplicar as mudanças, economizando esforço e tempo de desenvolvimento;

- *Desenvolvimento de um analisador estático*. Como nesta tese a abordagem proposta foi aplicada à transformações de modelos especificadas em ATL, foi necessário desenvolver um analisador estático que manipulasse todos os elementos especificados dentro de uma transformação. Além do uso na própria abordagem, este analisador estático pode ser utilizado para outras finalidades, como depuração do código das transformações e detecção de *bugs*;

- *Metamodelo para análise de impacto de mudanças*. No intuito de formalizar o conceito de *mudança* e a definição de métricas para calcular o impacto de mudanças em transformações de modelos, foi construído um metamodelo genérico para qualquer mudança aplicada a transformações especificadas em qualquer linguagem. Ele pode ser utilizado para a aplicação da abordagem proposta em outras linguagens de transformação e para futuras extensões da abordagem;

- *Métricas para mensurar o impacto de mudanças*. Embora já existam algumas métricas no escopo de MDD, elas servem para medir atributos de qualidade de transformações. As métricas propostas neste trabalho contribuem para o avanço do estado da arte no que tange a análise de impacto de mudanças em transformações de modelos;

- *Melhoria no processo de desenvolvimento de softwares baseados em MDD.* Uma vez que a fase de manutenção de transformações de modelos será facilitada com o apoio da abordagem proposta, haverá também uma contribuição significativa para o processo de desenvolvimento de projetos baseado em MDD;

- *Avaliação da abordagem*. A abordagem foi avaliada através de um estudo de caso, que contou com a participação de pessoas experientes no desenvolvimento de transformações de modelos. Os resultados deste estudo indicaram que a abordagem proposta está seguindo a mesma direção que os desenvolvedores para mensurar o impacto de mudanças e identificar os elementos impactados;

- *Ferramenta de suporte*. Para a utilização da abordagem proposta de forma automática, um protótipo de ferramenta de suporte foi desenvolvido (por meio de uma licença GPL), facilitando o seu uso.

Além disso, este trabalho também oferece contribuições no contexto da análise de impacto de mudanças, no qual nenhum trabalho que permita calcular o impacto associado a uma mudança foi encontrado na literatura. Geralmente, as técnicas já existentes detectam apenas os elementos impactados com uma mudança. Como a abordagem proposta nesta tese permite calcular o impacto de mudanças (além de detectar os elementos impactados), toda a metodologia criada para a definição das métricas pode ser reutilizada para a definição de novas métricas em diferentes contextos. Dessa forma, a análise de impacto de mudanças pode se tornar mais eficiente, permitindo ao gerente de projetos usar informações tanto dos elementos impactados quanto do valor de impacto causado pela mudança para a tomada de decisões.

10.2 Trabalhos Futuros

Com a conclusão deste trabalho, algumas ideias surgiram no sentido tanto de dar continuidade ao que já foi desenvolvido quanto de propor outras direções de pesquisa. A seguir, alguns destes trabalhos propostos como futuros são apresentados.

- *Proposta de novas métricas*. A abordagem proposta poderá ser aprimorada com a

definição de novas métricas, considerando aspectos ainda não cobertos pelas métricas já definidas;

- *Analisador estático para QVT*. Assim como ATL, QVT também é uma linguagem de transformação de modelos bastante utilizada pelos desenvolvedores. A fim de expandir a abordagem proposta para a aplicação em QVT, um trabalho que pode ser feito é o desenvolvimento de um analisador estático para a manipulação dos elementos dentro de uma transformação QVT;

- *Análise híbrida de impacto de mudanças*. Para que a análise de impacto de mudanças seja completa, seria interessante incorporar na abordagem proposta a análise de impacto de mudanças amparada na análise dinâmica. Com uma abordagem híbrida, seria possível combinar técnicas de análise estática para analisar os impactos antes de aplicar a mudança e técnicas de análise dinâmica para analisar os impactos depois de aplicar a mudança. Dessa forma, seria possível analisar os impactos de forma mais eficiente;

- *Integração da abordagem no plugin do Eclipse*. Atualmente, já existe um plugin do Eclipse para os usuários desenvolverem transformações em ATL. Uma ideia interessante seria fazer uma integração da abordagem proposta com este plugin, que poderia ser mais facilmente manipulado pelos usuários.

Bibliografia

[ATLAS Group e LINA & INRIA e Nantes, 2005] ATLAS Group e LINA & INRIA e Nantes (2005). The Kernel MetaMetaModel (KM3) Manual 0.3. http://www.eclipse.org/gmt/am3/km3/doc/KernelMetaMetaModel%5Bv00.06%5D.pdf.

[Bohner e Arnold, 1996] Bohner, S. e Arnold, R. (1996). *Software Change Impact Analysis*. IEEE Computer Society Press, Los Alamitos, CA, USA.

[Börger, 2003] Börger, E. (2003). Abstract State Machines: A Method for High-Level System Design and Analysis.

[Brunet et al., 2009] Brunet, J., Guerrero, D., e Figueiredo, J. (2009). Design Tests: An Approach to Programmatically Check Your Code Against Design Rules. Em *Software Engineering - Companion Volume, 2009. ICSE-Companion 2009. 31st International Conference*, p. 255–258.

[Brunet, 2009] Brunet, J. A. (2009). Design Wizard Project. www.designwizard.org.

[Chang, 2013] Chang, A. (2013). StatsToDo - Intraclass Correlation Program. http://www.statstodo.com/ICC_Pgm.php.

[Checkstyle, 2014] Checkstyle (2014). Checkstyle: A Static Analyzer Tool for Java. http://checkstyle.sourceforge.net.

[Chen e Wagner, 2002] Chen, H. e Wagner, D. (2002). MOdelchecking Programs for Security properties (MOPS): A Static Analyzer Tool for C Code. http://www.cs.berkeley.edu/~daw/mops.

[Coverity, 2012] Coverity (2012). Coverity Prevent: A Static Analyzer Tool for C/C++, C# and Java Languages. `http://www.coverity.com/products/static-analysis.html`.

[Czarnecki e Helsen, 2003] Czarnecki, K. e Helsen, S. (2003). Classification of Model Transformation Approaches. Em *OOPSLA'03 Workshop on Generative Techniques in the Context of Model-Driven Architecture*.

[Eclipse Foundation, 2005a] Eclipse Foundation (2005a). Class to Relational ATL Transformation Example. `http://www.eclipse.org/atl/atlTransformations/#Class2Relational`.

[Eclipse Foundation, 2005b] Eclipse Foundation (2005b). UML to Java Example. `http://www.eclipse.org/atl/atlTransformations/#UML2Java`.

[Eclipse Foundation, 2008] Eclipse Foundation (2008). Xpand Project. http://www.eclipse.org/modeling/m2t/?project=xpand. 2008.

[Eclipse Foundation, 2009] Eclipse Foundation (2009). Acceleo Project. http://www.eclipse.org/modeling/m2t/?project=acceleo.

[Eclipse Foundation, 2010] Eclipse Foundation (2010). ATLAS Transformation Language API. `http://git.eclipse.org/c/mmt/org.eclipse.atl.git`.

[Eclipse Foundation, 2011] Eclipse Foundation (2011). VIATRA2 Framework. An Eclipse GMT Subproject. `http://www.eclipse.org/gmt/VIATRA2`.

[Eclipse Foundation, 2012] Eclipse Foundation (2012). ATL Transformations. `http://www.eclipse.org/atl/atlTransformations`.

[Eclipse Foundation, 2014a] Eclipse Foundation (2014a). ATL User Guide. `http://wiki.eclipse.org/ATL/User_Guide_-_The_ATL_Language`.

[Eclipse Foundation, 2014b] Eclipse Foundation (2014b). KM32OWL Example. `http://www.eclipse.org/atl/atlTransformations/#KM32OWL`.

[Eclipse Foundation, 2014c] Eclipse Foundation (2014c). Plug-in do Eclipse para ATL. `http://www.eclipse.org/atl`.

[Efftinge, 2006] Efftinge, S. (2006). openArchitectureWare 4.1 Xtend Language Reference.

[FindBugs, 2013] FindBugs (2013). FindBugs: A Static Analyzer Tool for Java. `http://findbugs.sourceforge.net`.

[Garcés et al., 2009] Garcés, K., Jouault, F., Cointe, P., e Bézivin, J. (2009). Managing Model Adaptation by Precise Detection of Metamodel Changes. Em *Proceedings of the 5th European Conference on Model Driven Architecture - Foundations and Applications*, ECMDA-FA '09, p. 34–49, Berlin, Heidelberg. Springer-Verlag.

[Garcés et al., 2012] Garcés, K., Vara, J., Jouault, F., e Marcos, E. (2012). Adapting Transformations to Metamodel Changes via External Transformation Composition. *Software and Systems Modeling*.

[Garcia e Díaz, 2010] Garcia, J. e Díaz, O. (2010). Adaptation of Transformations to Metamodel Changes. *Library*, 2:1–9.

[Hattori et al., 2008] Hattori, L., Santos, G. D., Cardoso, F., e Sampaio, M. (2008). Mining Software Repositories for Software Change Impact Analysis: A Case Study. Em *SBBD '08: Proceedings of the 23rd Brazilian Symposium on Databases*, p. 210–223, Porto Alegre, Brazil, Brazil. Sociedade Brasileira de Computação.

[Herrmannsdoerfer et al., 2009] Herrmannsdoerfer, M., Benz, S., e Juergens, E. (2009). COPE - Automating Coupled Evolution of Metamodels and Models. Em *Proceedings of the 23rd European Conference on ECOOP 2009 — Object-Oriented Programming*, Genoa, p. 52–76, Berlin, Heidelberg. Springer-Verlag.

[Holzmann, 2002] Holzmann, G. J. (2002). UNO: Static Source Code Checking for User Defined Properties. Em *6th World Conf. on Integrated Design and Process Technology, IDPT '02*.

[Iovino et al., 2012] Iovino, L., Pierantonio, A., e Malavolta, I. (2012). On the Impact Significance of Metamodel Evolution in MDE. *Journal of Object Technology*, 11(3):3: 1–33.

[Jouault, 2007] Jouault, F. (2007). ATL Metamodel 2.0. `http://www.emn.fr/z-info/atlanmod/index.php/Atlantic#ATL_2.0`.

[Jouault e Kurtev, 2006] Jouault, F. e Kurtev, I. (2006). Transforming Models with ATL. Em *Proceedings of the 2005 International Conference on Satellite Events at the MoDELS*, MoDELS'05, p. 128–138, Berlin, Heidelberg. Springer-Verlag.

[Lehnert, 2011] Lehnert, S. (2011). A Taxonomy for Software Change Impact Analysis. Em *Proceedings of the 12th International Workshop on Principles of Software Evolution and the 7th Annual ERCIM Workshop on Software Evolution*, IWPSE-EVOL '11, p. 41–50, New York, NY, USA. ACM.

[Loskutov, 2011] Loskutov, A. (2011). FindBugs Plugin for Eclipse. `http://marketplace.eclipse.org/content/findbugs-eclipse-plugin`.

[Maia et al., 2010] Maia, M. C. O., Bittencourt, R. A., Figueiredo, J. C. A. d., e Guerrero, D. D. S. (2010). The Hybrid Technique for Object-Oriented Software Change Impact Analysis. Em *Proceedings of the 2010 14th European Conference on Software Maintenance and Reengineering*, CSMR '10, p. 252–255, Washington, DC, USA. IEEE Computer Society.

[Marzullo et al., 2010] Marzullo, F. P., Mario, V. F. D., da Silva, J. P., Nunes, L. S., e de Souza, J. M. (2010). A Model-Driven Development (MDD) Approach to Change Impact Analysis. Em Sabherwal, R. e Sumner, M., editors, *ICIS (International Conference on Information Systems)*, p. 3. Association for Information Systems.

[Mendez et al., 2010] Mendez, D., Etien, A., Muller, A., e Casallas, R. (2010). Towards Transformation Migration After Metamodel Evolution. Em *Model and Evolution Workshop*, Olso, Norway.

[Object Management Group, 2007] Object Management Group (2007). XML Metadata Interchange. `http://www.omg.org/spec/XMI/2.1.1`.

[Object Management Group, 2008] Object Management Group (2008). MOF Model to Text Transformation Language 1.0. `http://www.omg.org/spec/MOFM2T/1.0`.

[Object Management Group, 2011a] Object Management Group (2011a). Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification. `http://www.omg.org/spec/QVT/1.1`.

[Object Management Group, 2011b] Object Management Group (2011b). Unified Modeling Language (UML) Infrastructure 2.4.1. `http://www.omg.org/spec/UML/2.4.1/Infrastructure/PDF`.

[Object Management Group, 2012] Object Management Group (2012). MDA4eclipse Research Project. `www.mda4eclipse.com.br`.

[Oldevik, 2006] Oldevik, J. (2006). MOFScript Eclipse Plug-In: Metamodel-Based Code Generation. Em *Eclipse Technology Workshop (EtX) at ECOOP*, volume 2006.

[Oracle, 1999] Oracle (1999). Code Conventions for the Java Programming Language. `http://www.oracle.com/technetwork/java/codeconvtoc-136057.html`.

[Paternostro e Hussey, 2006] Paternostro, M. e Hussey, K. (2006). Advanced Features of EMF. Tutorial at EclipseCon 2006. `http://www.eclipse.org/modeling/emf/docs/presentations/EclipseCon/EclipseCon2006_EMF_Advanced.pdf`.

[Pfeiler, 2012] Pfeiler, A. (2012). FindBugs Plugin for IntelliJ IDEA. `http://plugins.intellij.net/plugin/?id=3847`.

[Pfleeger e Bohner, 1990] Pfleeger, S. L. e Bohner, S. A. (1990). A Framework for Software Maintenance Metrics. Em *Proc. Int'l Conf. Software Maintenance (ICSM)*.

[Pressman, 1995] Pressman, R. S. (1995). *Engenharia de Software*. Makron Books do Brasil, São Paulo, 3 edition.

[Prox, 2007] Prox, J. (2007). FindBugs Plugin for NetBeans. `http://plugins.netbeans.org/plugin/912/findbugs-tm-plugin`.

[Ren et al., 2004] Ren, X., Shah, F., Tip, F., Ryder, B. G., e Chesley, O. (2004). Chianti: a Tool for Change Impact Analysis of Java Programs. Em Vlissides, J. M. e Schmidt, D. C., editors, *OOPSLA*, p. 432–448. ACM.

[Rose et al., 2010a] Rose, L. M., Herrmannsdoerfer, M., Williams, J. R., Kolovos, D. S., Garcés, K., Paige, R. F., e Polack, F. A. C. (2010a). A Comparison of Model Migration Tools. Em Petriu, D. C., Rouquette, N., e Øystein Haugen, editors, *Model Driven Engineering Languages and Systems - 13th International Conference, MODELS 2010, Oslo, Norway, October 3-8, 2010, Proceedings, Part I*, volume 6394 of *Lecture Notes in Computer Science*, p. 61–75. Springer.

[Rose et al., 2010b] Rose, L. M., Kolovos, D. S., Paige, R. F., e Polack, F. A. C. (2010b). Model Migration with Epsilon Flock. Em *Proceedings of the Third international conference on Theory and practice of model transformations*, ICMT'10, p. 184–198, Berlin, Heidelberg. Springer-Verlag.

[Sarantakos, 1998] Sarantakos, S. (1998). *Social Research*. Macmillan.

[Scovetta, 2010] Scovetta, M. V. (2010). Yet Another Source Code Analyzer (Yasca): A Static Analyzer Tool for Several Languages. `http://www.scovetta.com/yasca.html`.

[Selic, 2003] Selic, B. (2003). The Pragmatics of Model-Driven Development. *IEEE Softw.*, 20(5):19–25.

[Sommerville, 2003] Sommerville, I. (2003). *Engenharia de Software*. Addison Wesley, São Paulo, SP, 6 edition. Tradução André Maurício de Andrade Ribeiro; Revisão técnica Kechi Hirama.

[Tisi et al., 2010] Tisi, M., Cabot, J., e Jouault, F. (2010). Improving Higher-Order Transformations Support in ATL. Em *ICMT'10: Proceedings of the International Conference on Model Transformation*, p. 215–229, Malaga, Spain.

[Turver e Munro, 1994] Turver, R. e Munro, M. (1994). An Early Impact Analysis Technique for Software Maintenance. *Journal of Software Maintenance*, 6(1):35–52.

[Uebersax, 2007] Uebersax, J. (2007). Intraclass Correlation and Related Methods. `http://www.john-uebersax.com/stat/icc.htm`.

[Ujhelyi et al., 2009] Ujhelyi, Z., Horváth, A., e Varró, D. (2009). A Generic Static Analysis Framework for Model Transformation Programs. Technical Report TUB-TR-09-EE19, Budapest University of Technology and Economics.

[van Amstel e van den Brand, 2011] van Amstel, M. e van den Brand, M. (2011). Using Metrics for Assessing the Quality of ATL Model Transformations. Em *3rd International Workshop on Model Transformation with ATL*, Zürich, Switzerland. CEUR-Workshop Proceedings.

[van Amstel et al., 2010] van Amstel, M. F., , van den Brand, M. G. J., e Nguyen (2010). Metrics for Model Transformations. Em *Proceedings of the Ninth Belgian-Netherlands Software Evolution Workshop (BENEVOL 2010)*.

[van Amstel et al., 2008] van Amstel, M. F., Lange, C. F. J., e van den Brand, M. G. J. (2008). Metrics for Analyzing the Quality of Model Transformations. Em *Proceedings of the 12th ECOOP Workshop on Quantitative Approaches on Object Oriented Software Engineering*.

[van den Brand et al., 2001] van den Brand, M., van Deursen, A., Heering, J., de Jong, H., de Jonge, M. T. K., Klint, P., Moonen, L., Olivier, P., Scheerder, J., Vinju, J., Visser, E., e Visser, J. (2001). The ASF+SDF Meta-Environment: A component-based language development environment. volume 2027, p. 365–370.

[Vieira e Ramalho, 2011a] Vieira, A. e Ramalho, F. (2011a). A Static Analyzer for Model Transformations. Em *3rd International Workshop on Model Transformation with ATL*, p. 75–88, Zürich, Switzerland. CEUR-Workshop Proceedings.

[Vieira e Ramalho, 2011b] Vieira, A. e Ramalho, F. (2011b). ATL Static Analyzer Source Code. `http://code.google.com/p/atl-static-analyzer/downloads/list`.

[Vieira e Ramalho, 2014a] Vieira, A. e Ramalho, F. (2014a). Metrics to Measure the Change Impact in ATL Model Transformations. Em *15th International Conference on Product-Focused Software Process Improvement (PROFES)*, p. 254–268, Helsinki, Finland. Springer.

[Vieira e Ramalho, 2014b] Vieira, A. e Ramalho, F. (2014b). Model Transformation Impact Analysis Project. `https://sites.google.com/a/computacao.ufcg.edu.br/mtia`.

[Vignaga, 2009] Vignaga, A. (2009). Metrics for Measuring ATL Model Transformations. Technical report, Universidad de Chile.

[Wimmer et al., 2012] Wimmer, M., Martínez, S., Jouault, F., e Cabot, J. (2012). A Catalogue of Refactorings for Model-to-Model Transformations. *Journal of Object Technology*, 11(2):2:1–40.

[World Wide Web Consortium, 2004] World Wide Web Consortium (2004). OWL Web Ontology Language Overview. `http://www.w3.org/TR/owl-features`.

[Yin, 2003] Yin, R. K. (2003). *Case Study Research : Design and Methods*. Sage Publications, 3rd edition.

[Zhang et al., 2008] Zhang, S., Gu, Z., Lin, Y., e Zhao, J. (2008). Celadon: a Change Impact Analysis Tool for Aspect-Oriented Programs. Em *Companion of the 30th International Conference on Software Engineering*, ICSE Companion '08, p. 913–914, New York, NY, USA. ACM.

Apêndice A

Classificação dos Tipos de Relações

Este apêndice apresenta, nas Tabelas A.1, A.2 e A.3, a classificação completa de todos os tipos de relações definidos, isto é, são apresentadas as possíveis relações que podem existir dentro de uma transformação ATL, dependendo dos elementos que ela especifica.

Tabela A.1: Classificação de todos os tipos de relação.

Tipo	Entidade A	Entidade B
owns_parameter	CalledRule	Parameter
owns_outpattern	Rule	OutPattern
owns_inpattern	MatchedRule	InPattern
owns_action_block	Rule	Action Block
owns_rule_variable	Rule	RuleVariableDeclaration
owns_library	Unit	LibraryRef
owns_definition	Helper	OclFeatureDefinition
owns_feature	OclFeatureDefinition	OclFeature
owns_context_definition	OclFeatureDefinition	OclContextDefinition
owns_module_element	Module	ModuleElement
owns_statement	ActionBlock	Statement
	ForStat	Statement
owns_expression	ExpressionStat	OclExpression
owns_source	BindingStat	OclExpression
owns_value	BindingStat	OclExpression
	Binding	OclExpression
owns_condition	IfStat	OclExpression
owns_then_statement	IfStat	Statement
owns_else_statement	IfStat	Statement
owns_iterator	ForStat	Iterator
	LoppExp	Iterator
	ForEachOutPatternElement	Iterator
owns_collection	ForStat	OclExpression
	ForEachOutPatternElement	OclExpression
owns_helper	Library	Helper
	Query	Helper
owns_body	Query	OclExpression

Tabela A.2: Classificação de todos os tipos de relação (continuação).

Tipo	Entidade A	Entidade B
owns_filter	InPattern	OclExpression
owns_inpattern_element	InPattern	InPatternElement
owns_outpattern_element	OutPattern	OutPatternElement
owns_binding	OutPatternElement	Binding
owns_reverse_binding	SimpleOutPatternElement	OclExpression
owns_return_type	Operation	OclType
owns_type	Attribute	OclType
	VariableDeclaration	OclType
	TupleTypeAttribute	OclType
owns_init_expression	Attribute	OclExpression
	VariableDeclaration	OclExpression
owns_ocl_type	OclContextDefinition	OclType
	OclExpression	OclType
	MapType	OclType(valueType)
	MapType	OclType(keyType)
owns_attribute	TupleType	TupleTypeAttribute
owns_variable_declaration	LetExp	VariableDeclaration
	IterateExp	VariableDeclaration
owns_ocl_expression	LetExp	OclExpression
	LoopExp	OclExpression
	PropertyCallExp	OclExpression
	OperationCallExp	OclExpression
	IfExp	OclExpression(condition)
	IfExp	OclExpression(thenExpr)
	IfExp	OclExpression(elseExpr)
	CollectionExp	OclExpression
	MapElement	OclExpression(key)
	MapElement	OclExpression(value)
owns_tuple_part	TupleExp	TuplePart
owns_map_element	MapExp	MapElement
owns_element_type	CollectionType	OclType
library_owned_by	LibraryRef	Unit
helper_owned_by	Helper	Query
	Helper	Library
module_element_owned_by	ModuleElement	Module
rule_variable_owned_by	RuleVariableDeclaration	Rule
action_block_owned_by	ActionBlock	Rule
inpattern_owned_by	InPattern	MatchedRule
outpattern_owned_by	OutPattern	Rule
inpattern_element_owned_by	InPatternElement	InPattern
outpattern_element_owned_by	OutPatternElement	OutPattern
parameter_owned_by	Parameter	Operation
feature_owned_by	OclFeature	OclFeatureDefinition

Tabela A.3: Classificação de todos os tipos de relação (continuação).

Tipo	Entidade A	Entidade B
ocl_expression_owned_by	OclExpression	Operation
	OclExpression	Attribute
	OclExpression	LetExp
	OclExpression	PropertyCallExp
	OclExpression	CollectionExp
	OclExpression	LoopExp
	OclExpression	IfExp(condition)
	OclExpression	IfExp(thenExpr)
	OclExpression	IfExp(elseExpr)
	OclExpression	OperationCallExp
	OclExpression	VariableDeclaration
ocl_type_owned_by	OclType	Operation
	OclType	Attribute
	OclType	CollectionType
	OclType	MapType(keyType)
	OclType	MapType(valueType)
	OclType	OclExpression
	OclType	OclContextDefinition
	OclType	TupleTypeAttribute
	OclType	VariableDeclaration
context_definition_owned_by	OclContextDefinition	OclFeatureDefinition
attribute_owned_by	TupleTypeAttribute	TupleType
variable_declaration_owned_by	VariableDeclaration	LetExp
	VariableDeclaration	IterateExp
tuple_part_owned_by	TuplePart	TupleExp
iterator_owned_by	Iterator	LoopExp
map_element_owned_by	MapElement	MapExp
extends	MatchedRule	MatchedRule
isExtended	MatchedRule	MatchedRule
has_inmodels	Module	OclModel
has_outmodels	Module	OclModel
has_mapsto	InPatternElement	OutPatternElement
has_source_element	OutPatternElement	InPatternElement
has_referred_variable	VariableExp	VariableDeclaration
has_variable_exp	VariableDeclaration	VariableExp
has_metamodel	OclModel	OclModel
has_model	OclModel	OclModel
	OclModelElement	OclModel
has_model_element	OclModel	OclModelElement
invokes	Rule	LazyMatchedRule
	Rule	CalledRule
	Rule	Helper
	Helper	Helper
isInvoked	CalledRule	Rule
	LazyMatchedRule	Rule
	Helper	Helper
	Helper	Rule

Apêndice B

Classificação das Possíveis Ações

Este apêndice apresenta todas as ações possíveis a serem executadas para aplicar cada tipo de mudança considerada nesta tese. Dado que e_i é o elemento a ser modificado, seguem as Tabelas B.1, B.2, B.3 e B.4 com as possíveis ações para modificá-lo de acordo com cada mudança.

Tabela B.1: Possíveis ações para a aplicação de cada tipo de mudança.

Tipo de Mudança	**Ações Possíveis**
addTransformationBlock	1. Adicionar a nova regra e_i.
addTransformationBlockParameter	1. Adicionar o novo parâmetro na regra e_i; 2. Verificar se algum elemento da própria transformação ou de outro módulo de transformação do projeto invoca e_i; 3. (Se invocar) Adaptar o elemento para obter o novo parâmetro; 4. (Se invocar) Incluir o novo parâmetro na invocação da regra e_i; 5. Adaptar a regra e_i para a manipulação do novo parâmetro.
changeConstantContextType	1. Modificar o tipo do contexto de e_i; 2. Verificar se algum elemento da própria transformação ou de outro módulo de transformação do projeto invoca a constante e_i; 3. (Se invocar) Adaptar a invocação de acordo com o novo tipo de contexto.
changeConstantName	1. Modificar o nome da constante e_i; 2. Substituir o antigo nome da constante pelo novo em todos os elementos que a invocam.

Tabela B.2: Possíveis ações para a aplicação de cada tipo de mudança (continuação).

Tipo de Mudança	Ações Possíveis
changeConstantType	1. Modificar o tipo de e_i; 2. Verificar se algum elemento da própria transformação ou de outro módulo de transformação do projeto invoca a constante e_i; 3. (Se invocar) Adaptar a invocação de acordo com o novo tipo da constante.
changeFunctionBody	1. Modificar o corpo da função e_i; 2. Verificar se algum elemento da própria transformação ou de outro módulo de transformação do projeto é afetado com a mudança do corpo da função e_i; 3. (Se for) Adaptar a invocação, se necessário.
changeFunctionContextType	1. Modificar o tipo do contexto de e_i; 2. Verificar se algum elemento da própria transformação ou de outro módulo de transformação do projeto invoca e_i; 3. (Se invocar) Adaptar a invocação de acordo com o novo tipo de contexto.
changeFunctionParameterType	1. Modificar o tipo do parâmetro de e_i; 2. Verificar se algum elemento da própria transformação ou de outro módulo de transformação do projeto invoca a função e_i; 3. (Se invocar) Adaptar o elemento para obter um parâmetro com o novo tipo; 4. (Se invocar) Ajustar a invocação de acordo com o novo tipo de parâmetro.
changeFunctionReturnType	1. Modificar o tipo de retorno da função e_i; 2. Verificar se algum elemento da própria transformação ou de outro módulo de transformação do projeto invoca e_i; 3. (Se invocar) Adaptar o elemento de acordo com o novo tipo de retorno.
changeTransformationBlockName	1. Modificar o nome da regra e_i; 2. Substituir o antigo nome da regra e_i pelo novo em todos os elementos que a invocam.
changeTransformationBlockParameterType	1. Modificar o tipo do parâmetro da regra e_i; 2. Verificar se algum elemento da própria transformação ou de outro módulo de transformação do projeto invoca e_i; 3. (Se invocar) Adaptar o elemento para obter um parâmetro com o novo tipo; 4. (Se invocar) Ajustar a invocação de acordo com o novo tipo de parâmetro.

Tabela B.3: Possíveis ações para a aplicação de cada tipo de mudança (continuação).

Tipo de Mudança	Ações Possíveis
changeOutputModelElementType	1. Modificar o tipo do elemento de saída da regra e_i; 2. Verificar se na regra e_i (cujo tipo do elemento de saída será modificado) existe algum outro elemento de saída que referencia o elemento cujo tipo será modificado; 3. (Se existir) Adaptar este elemento de acordo com o novo tipo; 4. Verificar se a regra e_i é invocada ou estendida por outra regra da própria transformação ou de outro módulo de transformação do projeto; 5. (Se for) Adaptar a regra, se necessário; 6. Verificar se a regra e_i faz parte de uma cadeia de transformações; 7. (Se fizer) Adaptar a cadeia após a modificação, se necessário.
removeConstant	1. Remover a constante e_i; 2. Verificar se algum elemento da própria transformação ou de outro módulo de transformação do projeto invoca a constante e_i; 3. (Se invocar) Remover a invocação de e_i; 4. (Se invocar) Adaptar o elemento que o invocava após a remoção, se necessário.
removeTransformationBlock	1. Remover a regra e_i; 2. Verificar se a regra e_i é invocada ou estendida por outra regra da própria transformação ou de outro módulo de transformação do projeto; 3. (Se for) Remover a invocação ou extensão de e_i; 4. (Se for) Adaptar o elemento após a remoção de e_i, se necessário; 5. Verificar se a regra e_i faz parte de uma cadeia de transformações; 6. (Se fizer) Remover referências à e_i na cadeia; 7. (Se fizer) Adaptar a cadeia após a remoção, se necessário.
removeTransformationBlockFilter	1. Remover o filtro da regra e_i; 2. Verificar se a regra e_i (cujo filtro será removido) é invocada ou estendida por outra regra da própria transformação ou de outro módulo de transformação do projeto; 3. (Se for) Adaptar esta regra após a remoção do filtro, se necessário; 4. Verificar se a regra e_i faz parte de uma cadeia de transformações; 5. (Se fizer) Adaptar a cadeia após a remoção do filtro, se necessário.

Tabela B.4: Possíveis ações para a aplicação de cada tipo de mudança (continuação).

Tipo de Mudança	Ações Possíveis
removeTransformationBlockInheritance	1. Remover herança da regra e_i; 2. Verificar se a regra e_i (cuja herança será removida) é invocada ou estendida por outra regra da própria transformação ou de outro módulo de transformação do projeto; 3. (Se for) Adaptar esta regra, se necessário; 4. Verificar se a regra e_i faz parte de uma cadeia de transformações; 5. (Se fizer) Adaptar a cadeia após a remoção da herança, se necessário.
removeOutputModelElement	1. Remover o elemento de saída da regra e_i; 2. Verificar se na regra e_i (cujo elemento de saída será removido) existe algum outro elemento de saída que referencia o elemento que será removido; 3. (Se existir) Remover a referência; 4. (Se existir) Adaptar o elemento após a remoção; 5. Verificar se a regra e_i é estendida ou invocada por outra regra da própria transformação ou de outro módulo de transformação do projeto; 6. (Se for) Adaptar esta regra após a remoção do elemento de saída de e_i, se necessário; 7. Verificar se a regra e_i faz parte de uma cadeia de transformações; 8. (Se fizer) Adaptar a cadeia após a remoção do elemento de saída de e_i, se necessário.
removeOutputModelElementBinding	1. Remover o *binding* do elemento de saída da regra e_i; 2. Verificar se na regra e_i (cujo *binding* do elemento de saída será removido) existe algum outro elemento de saída que referencia o elemento cujo *binding* será removido; 3. (Se existir) Adaptar este elemento após a remoção do *binding*, se necessário; 4. Verificar se a regra e_i é invocada ou estendida por outra regra da própria transformação ou de outro módulo de transformação do projeto; 5. (Se for) Adaptar esta regra, se necessário; 6. Verificar se a regra e_i faz parte de uma cadeia de transformações; 7. (Se fizer) Adaptar a cadeia após a modificação, se necessário.

Apêndice C

Classificação dos Impactos e dos Elementos Impactados

Este apêndice apresenta a tabela completa que mostra a classificação dos impactos (de cada mudança analisada no estudo de caso) feita tanto pela abordagem proposta quanto pelos participantes do estudo. Além disso, as Tabelas C.1 e C.2 apresentam também o número de elementos impactados com a mudança detectados pela abordagem e pelos participantes do estudo.

Tabela C.1: Classificação dos impactos e dos elementos impactados.

Tipo de Mudança	**Classificação (abordagem)**	**Classificação (participantes)**	**Nº Elem. Imp. (abordagem)**	**Nº Elem. Imp. (participantes)**
removeOutputModelElementBinding	Baixo (0.24)	Baixo (11) Alto (01)	01	01 (08) 02 (02) 05 (02)
changeConstantName	Baixo (0.26)	Baixo (12) High (0)	04	01 (01) 03 (01) 04 (08) 05 (02)
changeConstantContextType	Alto (0.48)	Baixo (01) Alto (11)	08	05 (01) 06 (01) 07 (01) 08 (03) 09 (05) 10 (01)
changeFunctionContextType	Baixo (0.34)	Baixo (09) Alto (03)	03	02 (01) 03 (06) 04 (05)

Tabela C.2: Classificação dos impactos e dos elementos impactados (continuação).

Tipo de Mudança	**Classificação (abordagem)**	**Classificação (participantes)**	**N° Elem. Imp. (abordagem)**	**N° Elem. Imp. (participantes)**
removeOutputModelElement	Baixo (0.36)	Baixo (12) Alto (0)	00	00 (04) 01 (06) 02 (02)
changeConstantType	Baixo (0.37)	Baixo (06) Alto (06)	10	08 (02) 09 (01) 10 (04) 11 (05)
removeTransformationBlockInheritance	Baixo (0.36)	Baixo (04) Alto (08)	05	04 (01) 05 (05) 06 (04) 09 (01) 10 (01)
addTransformationBlock	Baixo (0.28)	Baixo (11) Alto (01)	00	00 (08) 01 (04)
removeTransformationBlockFilter	Baixo (0.20)	Baixo (11) Alto (01)	00	00 (02) 01 (09) 05 (01)
changeTransformationBlockName	Baixo (0.35)	Baixo (09) Alto (03)	04	04 (07) 05 (04) 11 (01)
changeFunctionParameterType	Alto (0.47)	Baixo (03) Alto (09)	05	02 (01) 04 (01) 05 (06) 06 (04)
changeOutputModelElementType	Alto (0.46)	Baixo (07) Alto (05)	01	00 (03) 01 (06) 02 (02) 08 (01)