UNIVERSIDADE FEDERAL DO MARANHÃO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA DE ELETRICIDADE
CURSO DE PÓS-GRADUAÇÃO EM ENGENHARIA DE ELETRICIDADE

CARLOS CÉSAR GOMES MENDES

FORMALIZAÇÃO DA TRANSFORMAÇÃO DE MODELOS UTILIZANDO A LINGUAGEM Z

São Luís
2011

CARLOS CÉSAR GOMES MENDES

FORMALIZAÇÃO DA TRANSFORMAÇÃO DE MODELOS UTILIZANDO A LINGUAGEM Z

Dissertação apresentada ao Curso de Pós-Graduação em Engenharia de Eletricidade da Universidade Federal do Maranhão para obtenção do título de Mestre em Engenharia de Eletricidade, área de Concentração Ciência da Computação.

Orientador: Ph.D. Zair Abdelouahab
Co-orientador: Dr. Denivaldo Cicero Pavão Lopes

São Luís
2011

Mendes, Carlos César Gomes

Formalização da Transformação de Modelos Utilizando a Linguagem Z. / Carlos César Gomes Mendes. – São Luís, 2011.

136 f.

Orientador: PhD. Zair Abdelouahab
Co-orientado: Dr. Denivaldo Cicero Pavão Lopes
Dissertação (Mestrado) – Programa de Pós-Graduação em Engenharia de Eletricidade, Universidade Federal do Maranhão.

1. MDE. 2. Métodos Formais. 3. Linguagem Formal Z. I. Título.

CDU 004.41

FORMALIZAÇÃO DA TRANSFORMAÇÃO DE MODELOS UTILIZANDO A LINGUAGEM Z

Carlos César Gomes Mendes

Dissertação aprovada em 29 de julho de 2011.

Prof. Zair Abdelouahab, Ph.D.
(Orientador)

Prof. Denivaldo Cicero Pavão Lopes, Dr.
(Co-orientador)

Profa. Daniela Barreiro Claro, Dra.
(Membro da Banca Examinadora)

Prof. Samyr Beliche Vale, Dr.
(Membro da Banca Examinadora)

Ao meu Deus, pela força em todo esse tempo dedicado ao mestrado.

A minha família pelo incentivo de todos.

A minha esposa, Mayna Vanessa, pelo apoio, incentivo, consolo, compreensão e companheirismo.

Ao meu filho Enzo Benjamin.

AGRADECIMENTOS

Em primeiro lugar a Deus, pela vida concedida, pela força, paz, sabedoria e perseverança, que me permitiram concluir este trabalho e superar os desafios encontrados ao longo da minha vida.

À minha família pela audácia e arte de colocar a educação como prioridade em minha vida.

À minha esposa Mayna Vanessa pela paciência, compreensão, felicidade que tem me proporcionado e por ter estado do meu lado nos momentos difíceis, me incentivando a não desistir do mestrado.

Ao meu filho Enzo Benjamin que tem me dado alegria com a sua vinda a nossa família.

Ao Prof. Dr. Zair Abdelouahab pelo empenho, paciência e dedicação na orientação dos trabalhos realizados.

Ao professor Dr. Denivaldo Lopes, pela orientação, pela paciência e apoio para a concretização deste trabalho, pela fé na minha evolução e pelo incentivo em todos os momentos.

A CAPES pelo apoio e contribuição financeira dada durante todo o desenvolvimento do trabalho aqui apresentado.

Aos amigos do mestrado Eduardo Devidson, Raimundo Neto, Lianna, Vladimir, Amélia, Ariel, Frederico, Valéria, Bruna, Berto. Em especial, ao Eduardo pela ajuda e paciência durante os trabalhos das disciplinas e a Lianna pelas contribuições significativas para esta pesquisa. E a todos os amigos novatos que ingressaram no mestrado neste ano de 2011.

Ao meu grande amigo da graduação Freud, que também está terminando o mestrado.

Ao Programa de Pós-Graduação de Engenharia de Eletricidade, representados pelos Prof. Dr. João Viana, pela Profª. Ph.D. Maria da Guia e pelo Alcides Martins Neto, pela ajuda e paciência.

A todos aqueles que direta ou indiretamente contribuíram para a realização deste trabalho.

O temor do Senhor é o princípio da ciência...(Pv 1.7)

O ensino deveria ser assim: quem o receba o recolha como um bem inestimável, nunca como uma obrigação penosa. (Einstein)

RESUMO

Nesta dissertação, apresenta-se uma abordagem baseada na Teoria dos Conjuntos e na Linguagem de Especificação Formal Z para formalizar a Transformação entre Modelos dentro do contexto da Engenharia Dirigida a Modelos (MDE). A motivação desta pesquisa se deu devido a constatação de que a literatura sobre MDE tem apresentado ambiguidades e inconsistências nos modelos utilizados para abstrair o processo de transformação de modelos no contexto da MDE. Esta falta de precisão nestes tipos de modelos leva o usuário a interpretar de forma errada estruturas complexas presentes no mapeamento de elementos do modelo fonte para o modelo alvo. Sendo assim, propõe-se desenvolver uma metodologia formal que elimine as ambiguidades e inconsistências presentes nas representações informais da transformação de modelos da MDE. Para solucionar este problema, desenvolveu-se um *Framework* Conceitual Formal que agrupa os elementos envolvidos no processo de transformação, onde estes são representados através de artefatos matemáticos da Teoria dos Conjuntos e especificados em linguagem Z. Este *Framework* é validado através de um estudo de caso que contêm transformações, testadas na ferramenta de prova matemática Z/EVES, que suporta declarações feitas em linguagem Z.

Palavras-chave: Engenharia Dirigida a Modelos. Arquitetura Dirigida a Modelos. Métodos Formais. Teoria dos Conjuntos. Linguagem Formal Z. Z/EVES.

ABSTRACT

In this thesis, an approach based on Set Theory and on the Z Formal Language Specification is presented to formalize the transformations between models in the context of Model Driven Engineering (MDE). The motivation for this research is justified due the ambiguities and inconsistencies present in the models of transformation used to abstract the model transformation process in the MDE context. The precision absence in these the models lead the user to misinterpret complex structures present in the mapping of the source model elements to the target model elements. In this context, we proposed to develop a formal methodology that eliminates the ambiguities and inconsistencies present in the informal representations of model transformation in MDE. To solve this problem, a Formal and Conceptual *Framework* is developed that groups the elements involved in the process of transformation, represented by mathematical artifacts from the Set Theory and specified on Z language. This *Framework* is validated through a case study that contains complex transformations, tested on the mathematical proof tool Z/EVES, which supports statements made in Z language.

Keywords: Model Driven Engineering. Model Driven Architecture. Formal Methods. Set Theory. Z Formal Language. Z / EVES.

LISTA DE FIGURAS

p.

LISTA DE TABELAS

LISTA DE SIGLAS

ATL	*Atlas Transformation Language*
CSP	*Communicating Sequential Processes*
CWM	*Common Warehouse Metamodel*
DOM_FONTE	Domínio Fonte
ECORE	*Eclipse Core*
EDOC	*Enterprise Distributed Object Computing*
EMF	*Eclipse Modeling Framework*
EVES	*Enviroment for Verifying and Evaluating Software*
IBM	*International Business Machines*
IMA_ALVO	Imagem Alvo
MDA	*Model Driven Architecture*
MDE	*Model Driven Engineering*
MOF	*Meta Object Facility*
OCL	*Object Constraint Language*
OMG	*Object Management Group*
PIM	*Platform Independent Model*
PSM	*Platform Specific Model*
QVT	*Query/View/Transformation*
RMI	*Remote Method Invocation*
SQL	*Structured Query Language*
UML	*Unified Modeling Language*
VMD	*Vienna Development Method*
WSDL	*Web Service Description Language*

SUMÁRIO

1 INTRODUÇÃO

Este capítulo descreve o cenário, a definição do problema, a justificativa, a proposta de solução, o objetivo geral, os objetivos específicos, a metodologia de trabalho e a organização dos capítulos desta dissertação.

1.1 CENÁRIO

Nos últimos anos, a engenharia de software tem se esforçado em melhorar o processo de desenvolvimento de software, propondo abordagens centradas no processo de modelagem. Como resultado desta nova tendência, a Engenharia Dirigida a Modelos (MDE) surgiu [1]. Esta abordagem consiste em ter modelos no centro do processo de desenvolvimento, manutenção e evolução de software.

Entre os exemplos de padrões baseados em MDE tem-se: a MDA (Model Driven Architecture)TM [4] da OMG (Object Management Group), EMF (Eclipse Modeling Framework) do Eclipse Project [41], Software Factories da Microsoft [8].

Produtividade, portabilidade, interoperabilida, manutenção e documentação são benefícios que podem ser alçados através de MDE. Estes benefícios levaram diversas áreas a adotar a MDE para suas aplicações [2]. No entanto, estas diversas áreas não se preocuparam com a padronização da terminologia dos artefatos que formam a base de MDE. De acordo com Favre [3], em cada espaço tecnológico os conceitos de modelo, metamodelo e transformação tomam uma diferente conotação. Por exemplo, o que é chamado de um "metamodelo" em Modelware corresponde ao que é chamado um "esquema" em Documentware e Dataware, e uma gramática em Grammarware [3].

Neste contexto, uma série de conceitos e notações gráficas surgiram ao longo do tempo em torno dos artefatos de MDE, tais como: modelo, metamodelo, metametamodelo, definição de transformação, especificação de transformação, mapeamento, regras de transformação, plataforma e outros [3].

Devido à demanda pela abordagem MDE, vários modelos representativos foram construídos com diferentes objetivos em mente para representar uma situação particular de um sistema a ser modelado.

Deste modo, os elementos de MDE tomaram diferentes denotações e representações informais de acordo com os interesses e as necessidades do sistema particular em estudo.

1.2 DEFINIÇÃO DO PROBLEMA

Com este cenário em mente, os modelos encontrados na literatura utilizados para representar os elementos envolvidos em MDE não garantem interpretações corretas quando as aplicações são complexas. Devido à inconsistência na sua representação, usuários de diferentes áreas poderão interpretar de forma ambígua os elementos existentes em MDE.

Entre os elementos de MDE que sofrem com a falta de consenso em sua terminologia, cita-se a transformação entre modelos [3]. Por exemplo, no "manual da OMG Versão 1.0.1" [4] e no livro "*MDA Explaned"* de *"Kleppe et al"* [5] representam a transformação com denotações diferentes, o que é chamado uma *transformationSpecification* em [4] (ver Figura 1.1) aparece como *transformationDefinition* em [5] (ver Figura 1.2).

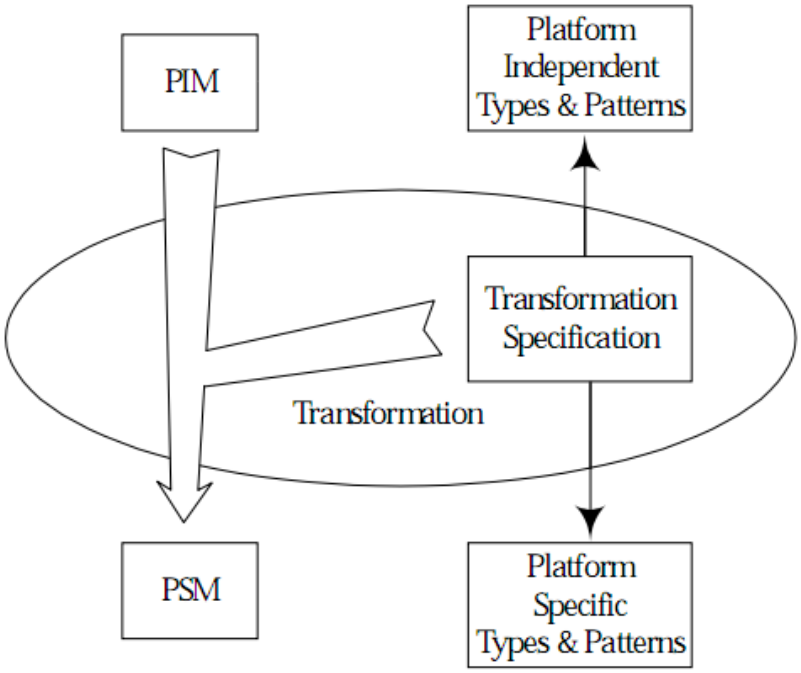

Figura 1.1: Transformação de modelos segundo o *"MDA Guide Version 1.0.1"* [4]

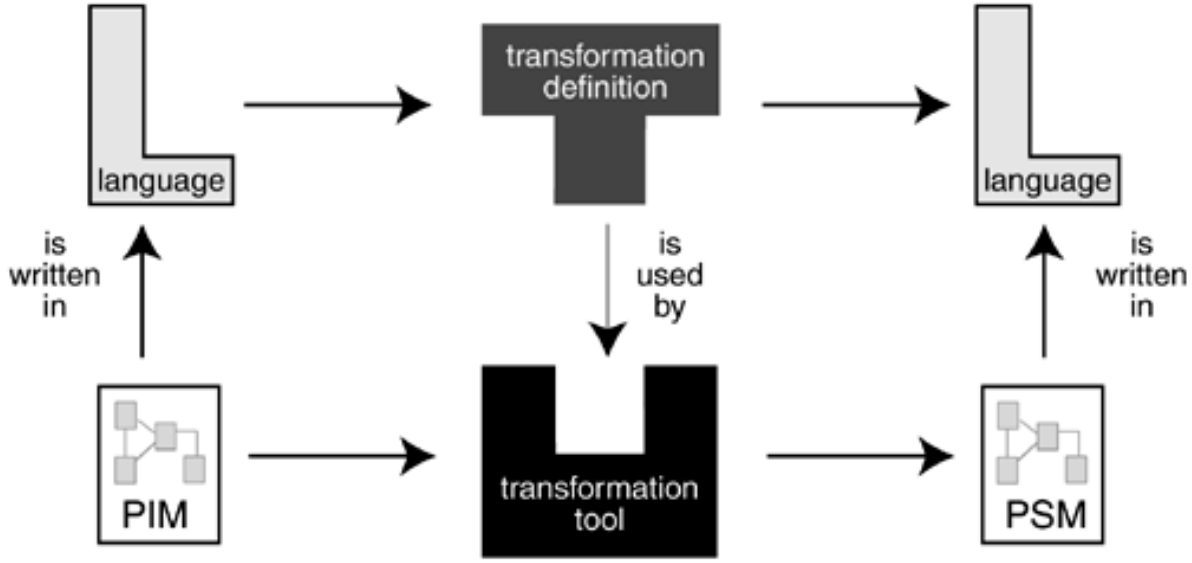

Figura 1.2: Transformação de modelos segundo o *"MDA Explained"* [5]

Esta inconsistência na forma de representar o processo de transformação entre modelos poderá resultar em interpretações ambíguas por parte do usuário em relação à transformação. Sem o apoio preciso da linguagem com que os modelos da Figura 1.1 e da Figura 1.2 estão expressos, não é possível raciocinar sobre a essência dos significados dos conceitos de MDE.

Sabendo que a transformação entre modelos é um processo importante no desenvolvimento de software utilizando MDE, torna-se então necessário uma mudança na representação gráfica informal dos elementos da MDE, principalmente em relação ao processo de transformação do modelo fonte (PIM) para o modelo alvo (PSM).

Assim, propomos uma solução baseada em uma modelagem consistente e estruturada, baseada em uma linguagem formal como, por exemplo, a linguagem Z, construída através dos métodos formais, onde se possa raciocinar de forma completa sobre a transformação entre modelos no contexto de MDE.

1.3 JUSTIFICATIVA E PROPOSTA DE SOLUÇÃO

Em MDE, uma transformação tem um papel de extrema importância no mapeamento [32] entre os elementos dos metamodelos, por isso necessita-se de uma modelagem consistente que aborde o conceito de transformação de forma clara, eliminando possíveis ambiguidades que possam vir a existir em sua interpretação.

A proposta desta pesquisa é baseada em uma abordagem matemática, buscando construir uma metodologia através da Teoria dos Conjuntos e da linguagem de especificação formal Z [6], para expressar a transformação entre modelos da MDE de maneira clara e que evite ambiguidades em sua interpretação.

1.4 OBJETIVOS

1.4.1 OBJETIVO GERAL

Desenvolver *"uma abordagem baseada na Teoria dos Conjuntos e na linguagem de especificação formal Z para formalizar a Transformação entre Modelos".*

1.4.2 OBJETIVOS ESPECÍFICOS

Para atingir o objetivo geral, os seguintes objetivos específicos tiveram que ser contemplados:

- Realizar uma análise formal da transformação de modelo, a fim de estabelecer quais elementos matemáticos serão utilizados para representar os elementos de MDE presentes no processo de transformação;
- Desenvolver uma metodologia baseada na linguagem Z para formalizar a transformação em MDE;
- Propor um *framework* conceitual formal que elimine ambiguidades na interpretação da transformação de modelos no contexto da MDE.

1.5 METODOLOGIA DE TRABALHO

A metodologia que foi adotada para o desenvolvimento e a produção desta pesquisa, primeiramente, se iniciou com a pesquisa bibliográfica, com o intuito, de coletar informações de livros, teses, dissertações, periódicos, anais de congressos e websites para uma total familiarização com a literatura especializada.

Em seguida, um estudo mais aprofundado em Engenharia de Requisitos, Engenharia Dirigida a Modelos (MDE), Métodos Formais, Linguagem de Especificação Formal Z, Teoria dos Conjuntos e Lógica de Predicados de Primeira Ordem foi realizado.

Também um estudo prático da ferramenta de especificação formal Z/EVES [12] foi realizado. O Z/EVES é utilizado para validar as especificações feitas em *esquemas* da linguagem Z, os esquemas [6] são as estruturas utilizadas para especificar formalmente a transformação do modelo fonte para o modelo alvo.

Com esta base adquirida, um estudo de caso foi desenvolvido para testar a metodologia proposta. E, por fim, propor um *framework* conceitual formal baseado na Linguagem Z e na Teoria dos Conjuntos, que articule uma visão global com uma visão específica da transformação entre modelos.

1.6 ORGANIZAÇÃO DA DISSERTAÇÃO

Esta dissertação está organizada em 7 (sete) capítulos.

No Capítulo 1, apresenta-se uma descrição do cenário, da definição do problema, da justificativa, da proposta de solução, do objetivo geral, dos objetivos específicos, da metodologia de trabalho e da organização dos capítulos desta dissertação.

No Capítulo 2, apresenta-se a fundamentação teórica e os principais conceitos e tecnologias que foram estudados e utilizados no desenvolvimento desta pesquisa.

No Capítulo 3, apresentam-se os trabalhos encontrados na literatura sobre a importância de um modelo que representa um determinado sistema em estudo. O uso de métodos formais baseados na Teoria dos Conjuntos como uma alternativa de solução para o problema da ambiguidade presente na representação da transformação entre modelos e por fim é feita uma análise dos trabalhos relacionados.

No Capítulo 4, apresentam-se uma análise matemática da transformação de modelos. Apresentam-se também a metodologia sugerida para formalizar a transformação entre modelos.

No Capítulo 5, descreve-se a proposta de um *framework* conceitual formal para representar o processo de transformação de modelos e descrever os relacionamentos existentes entre os elementos presentes.

No Capítulo 6, apresenta-se um estudo de caso com três exemplos de transformação de modelos para que se possa validar o framework conceitual proposto, bem como a comprovação do axioma que generaliza as transformações como uma relação binária.

No Capítulo 7, apresentam-se as conclusões da pesquisa, os resultados obtidos, as contribuições deste trabalho, as limitações encontradas e os possíveis trabalhos futuros.

2 REFERENCIAL TEÓRICO

Este capítulo descreve os principais conceitos e tecnologias que foram estudados e utilizados no desenvolvimento desta pesquisa.

2.1 MODEL DRIVEN ENGINEERING (MDE)

Como proposta para melhorar o processo de desenvolvimento de software, academias e organizações propuseram abordagens baseadas em modelos, surgindo assim a Model Driven Engineering (MDE) ou Engenharia Dirigida a Modelos. Esta tendência tem modelos no centro do processo de desenvolvimento de software.

A MDE tem como enfoque utilizar modelos para promover benefícios tais como reduzir os custos e o tempo no desenvolvimento do software e melhorar a sua qualidade final.

A relevância dos modelos em MDE não consiste apenas em uma documentação do software desenvolvido, mas estes modelos podem ser compreendidos por computadores, ou seja, esses modelos contêm informações que entram na máquina, são manipuladas e retornam uma saída [7].

Assim, esta abordagem permite harmonizar diferentes tipos de tecnologias e suportar diferentes domínios de aplicação. Tem-se como exemplos de MDE: os padrões MDA (*Model Driven Architecture*)^{TM} [4] da OMG (*Object Management Group*), EMF (*Eclipse Modeling Framework*) do Eclipse Project [41] e *Software Factories* da Microsoft [8].

Esta abordagem não pretende substituir os processos tradicionais de desenvolvimento de software e sim contribuir para o seu aprimoramento, porém, de uma maneira racional para mover informações de uma fase para outra fase do desenvolvimento. MDE permite respostas rápidas e eficientes para atender as necessidades inesperadas de novos requisitos tanto funcionais quanto não-funcionais, apresentando uma melhor e rápida adaptação de novas tecnologias e a integração de software desenvolvido em diversas plataformas [9].

Portanto, os modelos podem ser manipulados e transformados entre si, constituindo artefatos para garantir a longevidade, qualidade e baixo custo das aplicações de software.

Em MDE, a linguagem utilizada para criar modelos é descrita por um metamodelo, ou seja, o que precede o modelo. Um metamodelo é “um modelo que define a linguagem para exprimir um modelo”. Um metamodelo é simplesmente um modelo da linguagem de modelagem, “ele define a estrutura, a semântica e as restrições para uma família de modelos” [11].

O metamodelo é expresso por um metametamodelo. Um metametamodelo é “um modelo que define a linguagem para expressar metamodelos. A relação entre um metametamodelo e um metamodelo é similar à relação entre um metamodelo e um modelo” [10].

A relação entre modelo e metamodelo e a relação entre metamodelo e metametamodelo constituem a base da arquitetura a quatro níveis de modelos da MDE.

Em MDE, uma plataforma é definida como “um conjunto de subsistemas e tecnologias que fornecem um conjunto coerente de funcionalidades através de interfaces e padrões de uso especificado” [4].

O relacionamento entre o sistema e esses elementos é descrito na Figura 2.1.

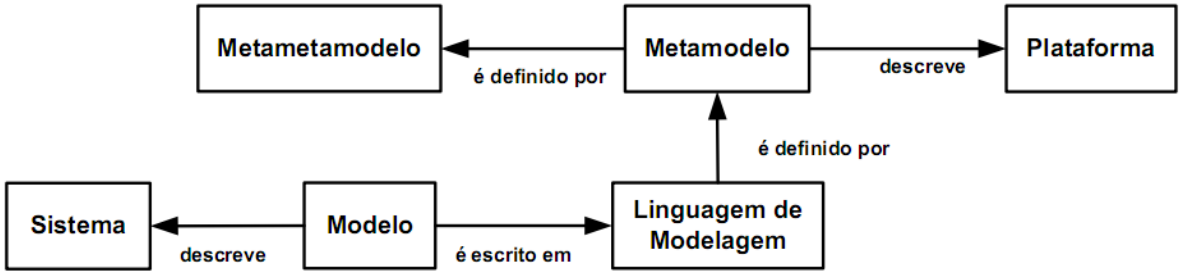

Figura 2.1: Modelos, Metamodelos, Metametamodelo e Plataforma [11]

Na Figura 2.1, verifica-se que estão presentes alguns artefatos da MDE assim como os seus relacionamentos. Porém, nota-se que estão faltando alguns elementos de extrema importância como, por exemplo, a representação da transformação de metamodelos.

A Figura 2.2 mostra a arquitetura de quatro níveis de modelos da MDE. Esta arquitetura é fundamental para toda abordagem dirigida a modelos.

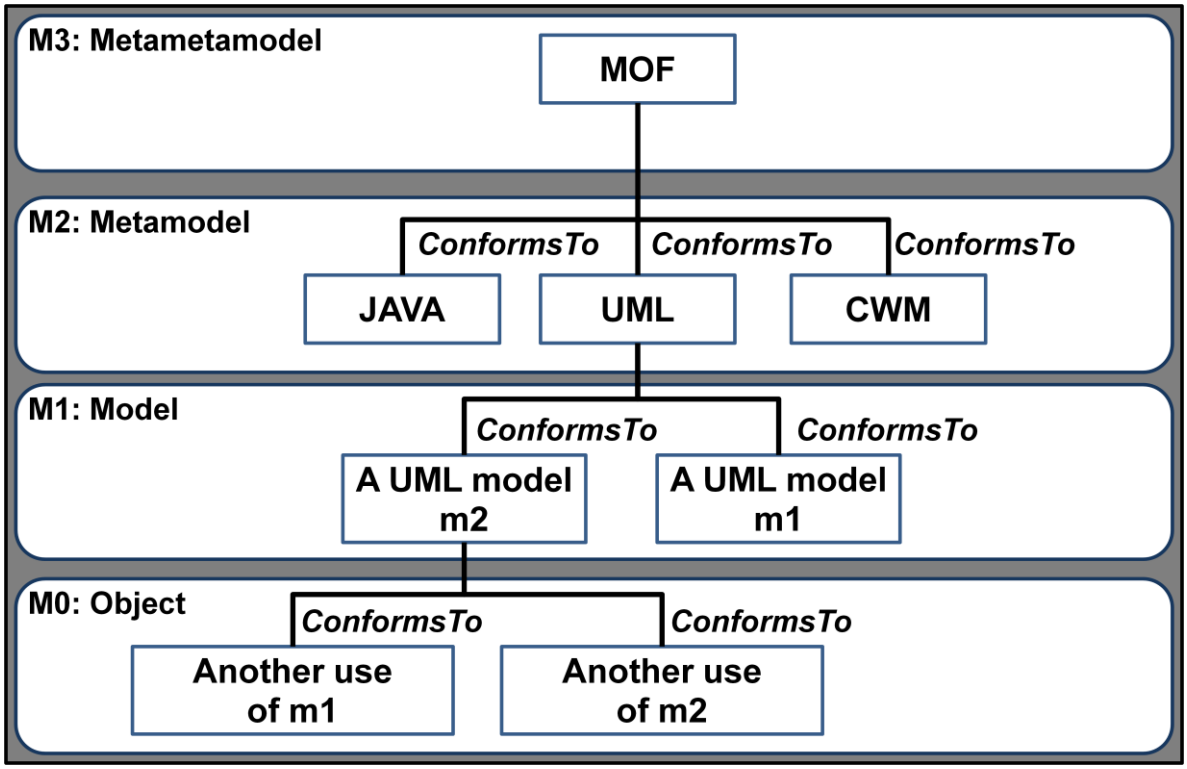

Figura 2.2: Arquitetura de quatro níveis de modelos adaptada de [5]

A MDE tem sua arquitetura distribuída em quatro níveis de modelagem.

De acordo com a Figura 2.2, nota-se que as camadas M3 e M2, M2 e M1, M1 e M0 possuem um relacionamento do tipo "***ConformsTo***". Estas camadas podem ser definidas da seguinte forma [7]:

1. **M3 (metametamodelo):** a camada M3 constitui a base da arquitetura de metamodelagem, tem a função primordial de definir linguagens para especificar metamodelos. Um metametamodelo define um modelo de mais alto nível que o metamodelo, e este primeiro é tipicamente mais compacto que o segundo. Exemplos de metametamodelos: MOF (*Meta Object Facility*) da OMG [10] e EMF (*Elipse Modeling Framework*) do projeto Eclipse [41];
2. **M2 (metamodelo):** é a camada onde estão os metamodelos, tem a função principal de definir uma linguagem para especificar modelos. Os metamodelos são tipicamente mais elaborados que os metametamodelos. A linguagem UML possui um metamodelo que descreve estruturalmente como devem ser criados os modelos UML;
3. **M1 (modelo):** é a camada onde se encontram os modelos do mundo real. Um modelo é definido por um metamodelo. A função principal da camada de

modelos é definir uma linguagem para descrever um domínio da informação. Por exemplo, um modelo de um sistema bancário utilizando a linguagem UML;

4. **M0 (informação):** é a camada onde estão os objetos de usuários que representam as informações finais. A principal função dos objetos de usuários é descrever um domínio específico em uma plataforma final.

Nesta arquitetura, deve-se notar a existência de poucos metametamodelos, por exemplo, MOF (**M3**) e vários metamodelos (**M2**), por exemplo, UML, JAVA e CWM (*Common Warehouse Metamodel)*, um grande número de modelos (**M1**) e uma infinidade de objetos (**M0**) [39].

Assim, devido aos benefícios proporcionados pela abordagem MDE muitos padrões tem surgido nos últimos anos como, por exemplo, a MDA (*Model Driven Architecture*)^{TM} [4] da OMG.

2.2 MODEL DRIVEN ARCHITECTURE (MDA)

A *Model Driven Architecture* (MDA) ou Arquitetura Dirigida a Modelos é um padrão criado e especificado pela OMG (*Object Management Group*) [23], a MDA é considerada uma instância da MDE. A MDA tem como objetivo promover o uso de modelos no desenvolvimento, manutenção e evolução de software e favorecer a interoperabilidade e portabilidade de sistemas.

A MDA é uma abordagem que abrange a geração de código a partir de um modelo, ou seja, é uma abordagem na qual a especificação do sistema é feita de forma independente de plataforma, fornecendo no final do desenvolvimento um esqueleto de código escrito em uma linguagem específica, como por exemplo, JAVA, SQL, PASCAL [4].

Este padrão permite a construção de sistemas computacionais a partir de modelos que podem ser entendidos muito mais rapidamente do que os “códigos” de linguagem de programação [11].

2.2.1 CONCEITOS IMPORTANTES DA MDA

No contexto da Engenharia Dirigida a Modelos (MDA), os elementos envolvidos no processo de transformação podem ser conceituados da seguinte forma:

a) Modelo: Um modelo é uma descrição de um (ou de uma parte do) sistema expresso em uma linguagem bem definida, ou seja, respeitando um formato (uma sintaxe) e um significado (uma semântica) preciso que é apropriado para a interpretação automática por um computador [5].

b) Metamodelo: É um modelo que define a linguagem para exprimir um modelo, ou seja, é simplesmente um modelo de um modelo. O metamodelo define a estrutura, a semântica e as restrições para uma família de modelos [11].

c) Metametamodelo: É um modelo que define a linguagem para expressar metamodelos. A relação entre um metametamodelo e um metamodelo é similar à relação entre um metamodelo e um modelo [10].

d) Especificação de Correspondência (*Mapping Specification*): É um modelo que visa definir os relacionamentos existentes entre os elementos do metamodelo fonte com os elementos do metamodelo alvo. Uma especificação de correspondência pode ser definida como uma função [40]:

$$Match'(M_a, M_b) = C_{M_a \rightarrow M_b} / M_c$$

Onde:

- $\boldsymbol{Match'(M_a, M_b)}$ é a função que recebe dois metamodelos e gera um modelo de correspondência;
- $\boldsymbol{C_{Ma \rightarrow Mb} / M_c}$ é uma especificação de correspondência (i.e. modelo de correspondência). Onde $\boldsymbol{C_{Ma \rightarrow Mb}}$ é um modelo de correspondência entre $\boldsymbol{M_a}$ e $\boldsymbol{M_b}$ conforme o metamodelo $\boldsymbol{M_c}$;

e) Mapeamento (*Mapping*): Conforme é descrito no trabalho de "*Bernstein*" [32], um mapeamento entre dois modelos M_1 e M_2, é um modelo "map_{12}" e dois morfismos (relação binária entre dois elementos), um entre M_1 e "map_{12}" e outro entre "map_{12}" e M_2. O morfismo é uma relação binária, ou seja, é um conjunto de pares $<o_1,o_2>$ onde o_1 e o_2 estão em M_1 e M_2 respectivamente [32].

f) Definição de transformação: Uma definição de transformação é um conjunto de regras de transformação que, juntas, descrevem como um modelo em uma linguagem fonte pode ser transformado em um modelo na linguagem alvo [5]. Descreve em detalhes e implementa os passos na transformação de um modelo em outro modelo, conforme a especificação de correspondência [40].

Na verdade, uma especificação de correspondências pode ser visto como um PIM e uma definição de transformação como um PSM. Ambos descrevem a mesma coisa, mas em diferentes níveis [40].

g) Regra de transformação: Uma regra de transformação é uma descrição de como um ou mais elementos de uma linguagem fonte podem ser transformados em um ou mais elementos em uma na linguagem alvo [5].

2.2.2 Abordagens da Transformação de Modelos no Contexto da MDA

No *"MDA Guide"* [4], um mapeamento é um modelo que fornece especificações para a transformação de um PIM em um PSM para uma plataforma especifica. O modelo da plataforma pode determinar a natureza do mapeamento. Nesta dissertação, mapeamento entre dois modelos M_1 e M_2 é uma relação binária entre os elementos (objetos) do modelo M_1 com os elementos (objetos) do modelo M_2.

As abordagens utilizadas para realizar a transformação de modelos são apresentadas a seguir segundo o "*MDA Guide"* [4]:

a) Marcas: Nesta abordagem, uma plataforma em particular é escolhida. Um mapeamento para esta plataforma é disponibilizado ou é preparado. Este mapeamento inclui um conjunto de marcas. As marcas são usadas para identificar os elementos do modelo e em seguida orientar a transformação do PIM para o PSM. O PIM marcado é posteriormente transformado, usando o mapeamento para produzir o PSM [4], como está ilustrado na Figura 2.3.

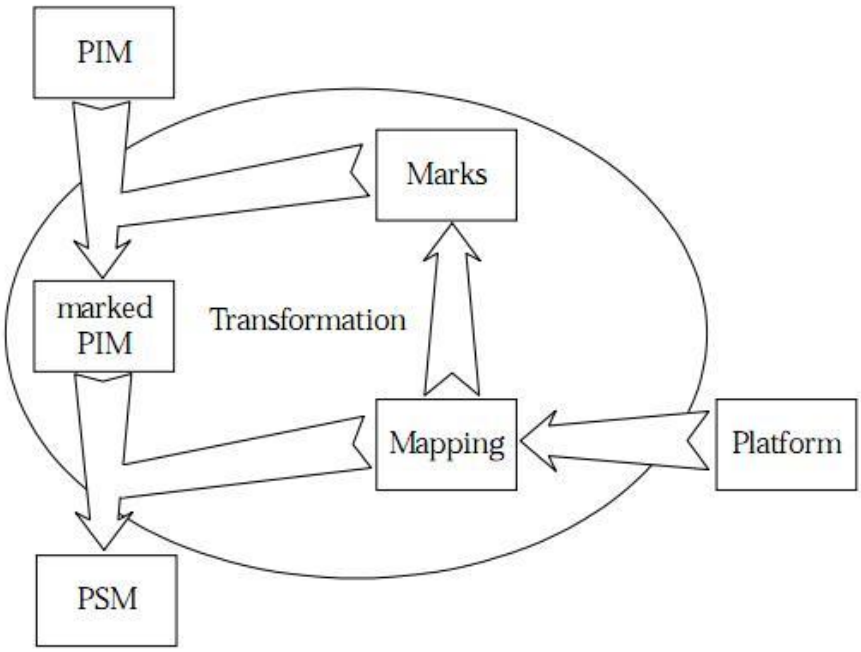

Figura 2.3: Um modelo marcado [4]

b) Transformação de metamodelo (*Metamodel Transformation*): Um modelo é preparado utilizando uma linguagem independente de plataforma, especificada por um metamodelo. Nesta abordagem, uma plataforma em particular é escolhida. A especificação de uma transformação para esta plataforma está disponível ou é preparada. Esta especificação de transformação é definida em termos de um mapeamento entre metamodelos. O mapeamento orienta a transformação do PIM para produzir o PSM [4], conforme está ilustrado na Figura 2.4.

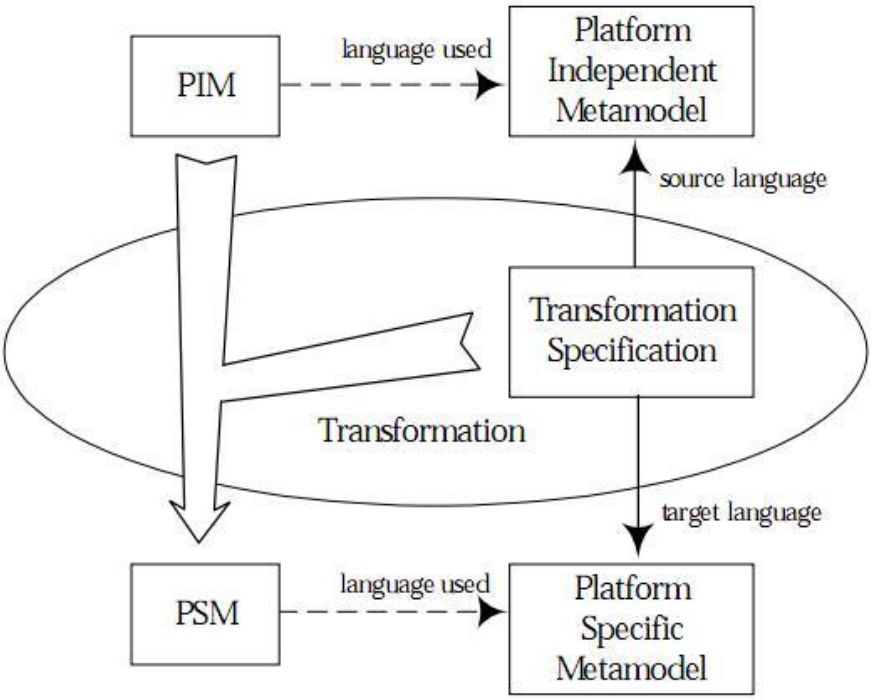

Figura 2.4: Transformação de metamodelos [**4**]

c) Transformação de modelo (*Model Transformation*): Um modelo é preparado usando tipos independentes de plataforma especificados em um modelo. Os tipos podem ser parte de um *framework* de software. Os elementos no PIM são subtipos dos tipos independentes de plataforma [4], de acordo com a Figura 2.5.

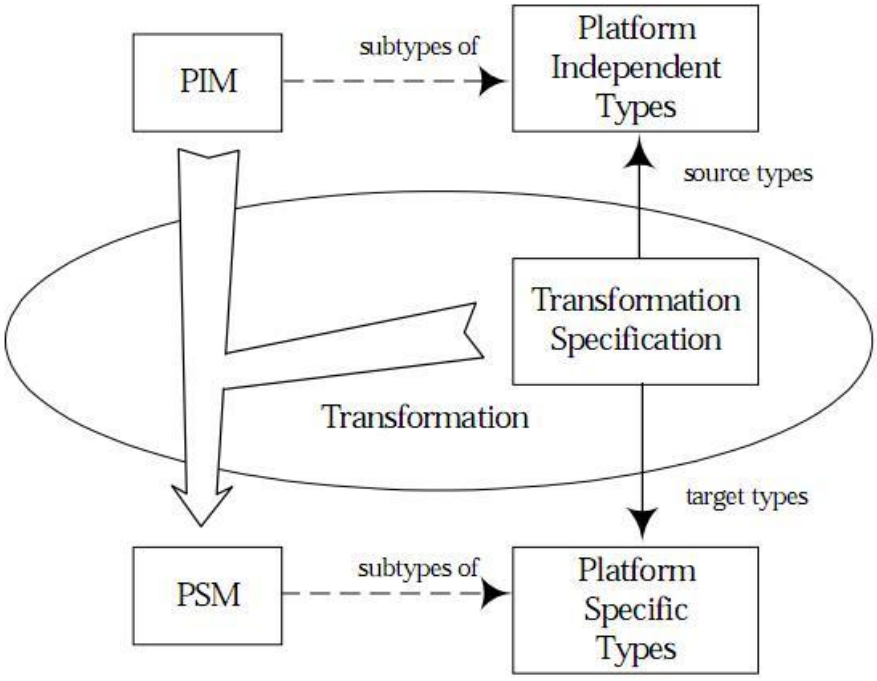

Figura 2.5: Transformação de modelos [**4**]

Nesta abordagem, uma plataforma em particular é escolhida. Uma especificação de uma transformação para esta plataforma está disponível ou é preparada. Esta especificação de transformação está em termos de um mapeamento entre os tipos independentes de plataformas e os tipos dependentes de plataformas. Os elementos no PSM são subtipos dos tipos específicos de plataforma [4].

Esta abordagem difere do mapeamento de metamodelo, pois tipos especificados em um modelo são usados para o mapeamento, em vez de conceitos especificados por um metamodelo [4].

d) Aplicação de Padrões (*Pattern Application*): Esta abordagem é uma extensão das abordagens de mapeamento de modelo e metamodelo, incluindo os padrões juntamente com os tipos ou os conceitos da linguagem de modelagem [4].

Além dos tipos independentes de plataforma, um modelo genérico pode fornecer padrões. Os tipos e os padrões podem ser mapeados para tipos específicos de plataforma e padrões. A abordagem de mapeamento de metamodelo pode usar os padrões da mesma forma [4].

A Figura 2.6 mostra uma outra maneira de usar padrões: como os nomes das marcas específicas de plataforma, ou seja, os nomes dos padrões de projeto que são específicos para uma plataforma [4].

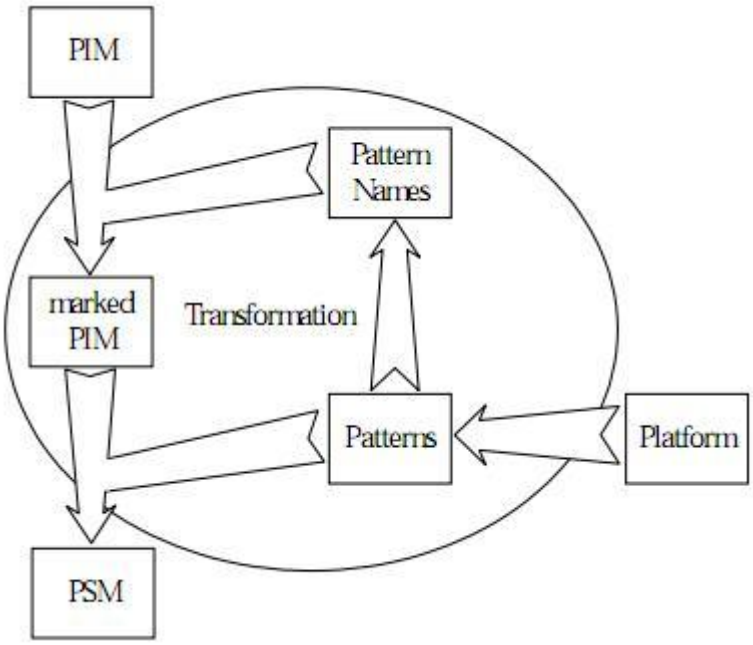

Figura 2.6: Uso de padrões na transformação de modelos [4]

e) Modelo de fusão (*Model Merging*): Nesta abordagem, dois modelos fontes, i.e., PIM e outro modelo, se fundem para gerar um modelo alvo (PSM), como está ilustrado na Figura 2.7. O exemplo anterior mostra o uso de padrões e aplicação de padrões, que é um tipo de modelo de fusão [4].

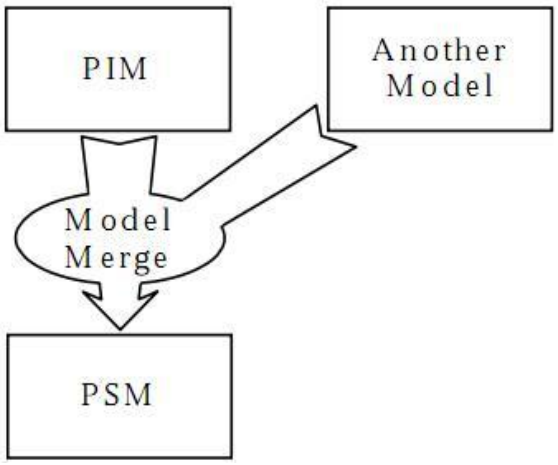

Figura 2.7: Modelos de fusão [4]

Para realizar esta pesquisa, adotou-se a abordagem de Transformação de Metamodelos para fazer a transformação de modelos.

2.2.3 O *FRAMEWORK* BÁSICO DA MDA

Os principais elementos que estão presentes no *framework* básico da MDA são: modelos (PIM's e PSM's), linguagens, transformação, definição de transformação e ferramenta de transformação. Todos esses elementos estão relacionados conforme ilustra a Figura 2.8.

Figura 2.8: *Framework* básico de MDA adaptado de [5]

Neste *framework*, os seguintes elementos da abordagem MDA são destacados [5]:

- **PIM (*Platform Independent Model*)** – É um Modelo Independente de Plataforma. Isto é, independente de qualquer tecnologia de implementação. Neste modelo, todas as funcionalidades do sistema e restrições do negócio são modeladas;
- **PSM (*Platform Specific Model*)** – É um Modelo Específico de Plataforma, feito para uma plataforma específica, ou seja, descreve um sistema com pleno conhecimento sobre uma determinada plataforma. Este modelo descreve detalhes da implementação em uma plataforma e as informações transformadas do PIM. Ele é gerado a partir de um PIM através da transformação de modelos, por isso o PSM representa o modelo alvo;
- **Definição de transformação** – descreve como um modelo fonte (PIM), poderá ser transformar em um modelo alvo (PSM);
- **Ferramenta de transformação** – realiza uma transformação entre modelos de acordo com uma definição de transformação.

A transformação de modelos está dividida em duas etapas: a primeira, o processo de conversão de um modelo fonte (PIM) em um modelo alvo (PSM) (ver Figura 2.9), e a segunda etapa é o processo de transformação de um modelo alvo (PSM) em um código-fonte,

segundo uma definição de transformação (*transformation definition*), tal como é ilustrado na Figura 2.9.

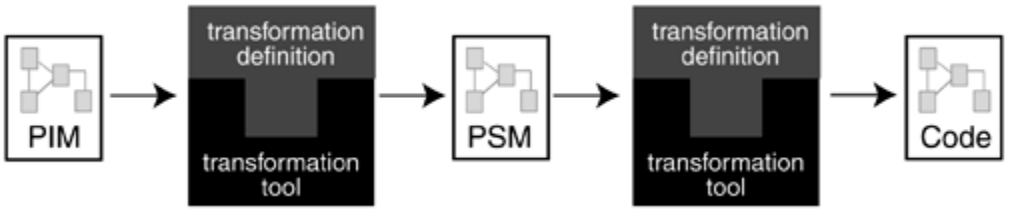

Figura 2.9: Processo de geração de código em MDA [5]

As transformações de modelos podem ser realizadas nos seguintes sentidos [4]:

1. **PIM para PIM:** este tipo de transformação é necessário para o acréscimo ou subtração de informações nos modelos ou mudança da linguagem de modelagem;
2. **PIM para PSM:** este tipo de transformação é necessário quando um PIM está suficientemente enriquecido e deseja-se obter este modelo com detalhes de plataforma;
3. **PSM para PIM:** é útil para separar a lógica de negócio da sua plataforma. Particularmente, este tipo de transformação é mais utilizado no processo de engenharia reversa;
4. **PSM para PSM:** é utilizada nas fases de refinamento da plataforma, de implementação, de otimização, de configuração ou mudança de plataforma.

As aplicações envolvendo MDA trazem muitas melhorias para o processo de desenvolvimento de sistemas. Entre elas tem-se [5]:

- **Produtividade:** a transformação do PIM para o PSM é definida uma única vez e pode ser aplicada na implementação de diversos sistemas. Devido a este fato, tem-se uma redução no tempo de desenvolvimento do software;
- **Portabilidade:** um mesmo PIM pode ser automaticamente transformado em vários PSM's de diferentes plataformas, através de mapeamentos;
- **Interoperabilidade:** diferentes PSM's gerados a partir de um mesmo PIM podem ter relacionamentos entre si;
- **Manutenção e Documentação:** O modelo é uma representação abstrata do código. Assim, o PIM cumpre a função de documentação de alto nível que é necessário para qualquer sistema de software.

2.2.4 CARACTERÍSTICAS DESEJÁVEIS DAS TRANSFORMAÇÕES EM MDA

As transformações em MDA precisam ser mais do que o simples processo de geração de um modelo alvo a partir de um modelo fonte, com isso há uma série de características de um processo de transformação que são muito desejáveis [5]:

- **Ajustabilidade:** significa que dada uma regra geral na definição de transformação, uma aplicação desta regra pode ser ajustável;
- **Rastreabilidade:** as informações da transformação do modelo fonte em um modelo alvo são armazenadas para permitir que os elementos do modelo alvo possam ser usados para retornar aos elementos do modelo fonte;
- **Consistência Incremental:** significa que, quando uma informação específica tiver sido adicionada no modelo alvo e este tenha sido regenerado, a informação extra deve estar presente no novo modelo alvo;
- **Bidirecionalidade:** significa que uma transformação pode ser realizada não apenas do modelo fonte (PIM) para o alvo (PSM), mas também do modelo alvo (PSM) para o fonte (PIM), ou seja, transformar um elemento fonte em um elemento alvo e vice- versa, veja Figura 2.10.

Figura 2.10: Transformação Bidirecional em MDA adaptado de [5]

Estas características garantem que uma transformação em MDA possa ser ajustada durante o processo de mapeamento de elementos e também possibilita manter uma relação de persistência entre os relacionamentos do modelo fonte e do modelo alvo.

2.2.5 EXEMPLO DE UMA TRANSFORMAÇÃO DE MODELOS EM MDE

Na Figura 2.11, um exemplo de uma transformação do metamodelo UML para o metamodelo WSDL é apresentado. Neste exemplo, o modelo de UML é conforme o metamodelo da linguagem UML e o modelo de WSDL é conforme o metamodelo da linguagem WSDL. O modelo fonte (PIM) é o metamodelo UML, e o modelo alvo (PSM) é o metamodelo WSDL.

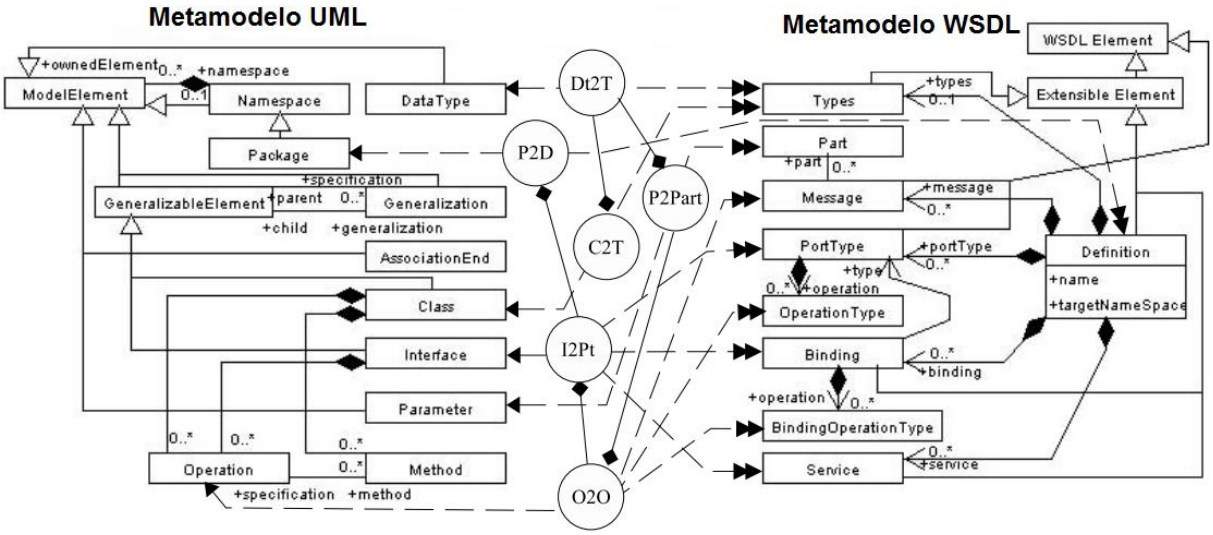

Figura 2.11: Transformação do metamodelo UML para o de WSDL [31]

Na Figura 2.11, o metamodelo UML é constituído de elementos tais como *Package, Class, Interface* e outros que são denominados de elementos fontes, o metamodelo WSDL também é constituído de elementos tais como *Type, Message, Binding* e outros que são os elementos alvos. Os elementos do metamodelo UML são transformados em elementos do metamodelo WSDL.

A relação elemento fonte-alvo é denominada de mapeamento [32] do elemento fonte para o elemento alvo.

Verifica-se na Figura 2.11, a existência de um único elemento no metamodelo UML transformado em um único elemento no metamodelo WSDL, é o caso de mapeamento de elementos de um-para-um. Também existem dois elementos: *DataType* e *Class* no metamodelo UML que são transformados em um mesmo elemento no metamodelo WSDL, é o caso de mapeamento de elementos de muitos-para-um.

E ainda existe um elemento *Operation* no metamodelo UML que é transformado em vários elementos no metamodelo WSDL que são: *Message, OperationType* e

BindingOperationType, é o caso de mapeamento de um-para-muitos. No sentindo inverso (do metamodelo WSDL para o metamodelo UML), também ocorrem estes três casos de mapeamento.

Em uma transformação de modelos, não é obrigatório que todos os elementos do metamodelo fonte sejam mapeados para os elementos do metamodelo alvo. Os elementos envolvidos no processo de transformação de modelos são selecionados de acordo com os requisitos da aplicação. Estes elementos selecionados constituem um conjunto denominado de fragmento de metamodelo.

2.3 DEFICIÊNCIAS DAS ESPECIFICAÇÕES INFORMAIS DE SISTEMAS

Os métodos tradicionais encontrados na literatura para especificação de sistemas fazem forte uso da linguagem natural e de uma variada coleção de notações gráficas. No entanto, apesar de uma aplicação cuidadosa de métodos de análise de projetos acoplados a rigorosas revisões que podem conduzir ao desenvolvimento de software de alta qualidade, um descuido na aplicação desses métodos pode criar uma variedade de problemas no desempenho do sistema. Algumas das deficiências encontradas são [42]:

a) **Contradições:** é uma série de afirmações em desacordo umas com as outras. Por exemplo: "*uma parte de uma determinada especificação do sistema pode declarar que o sistema deve monitorar todas as temperaturas em um reator químico, enquanto a outra, especificada por uma pessoa diferente, pode declarar que apenas temperaturas dentro de um certo intervalo devem ser monitoradas".*

b) **Ambiguidades:** são declarações que podem ser interpretadas de diferentes maneiras. Por exemplo, dada a seguinte declaração em linguagem natural: "*a identidade do operador consiste no nome do operador e na senha; a senha consiste de seis dígitos. Deve ser mostrada no monitor de vídeo de segurança e depositada no arquivo de registro de entradas, quando um operador entra no sistema".* Neste exemplo, a palavra *deve* se refere à senha ou a identidade do operador?

De acordo com estes exemplos, nota-se que o processo tradicional de especificação de sistema através da linguagem natural e informal é suscetível a erros, com

isso a necessidade de uma modelagem mais precisa e concisa é de fundamental importância para reduzir estes tipos de erros.

2.4 MÉTODOS FORMAIS

Métodos formais consistem em um conjunto de técnicas e ferramentas baseadas em modelagem matemática e lógica formal, que são usadas para especificar e verificar requisitos e projetos de sistemas computacionais [13].

O uso de métodos formais em um projeto pode assumir várias formas, variando desde o uso ocasional da notação matemática em uma especificação escrita em linguagem natural, até uma especificação completamente formal usando uma linguagem de especificação formal com uma semântica bem definida [13].

Especificações formais usam notação matemática para descrever de maneira precisa as propriedades que um sistema computacional deve ter, sem necessariamente estabelecer como estas propriedades devem ser implementadas. Uma especificação formal descreve o que o sistema deve fazer sem dizer como deve ser feito [14].

Esta abstração torna especificações formais úteis no processo de modelagem do sistema, pois permite que questões sobre o que o sistema faz sejam respondidas com precisão, sem a necessidade de extrair esta informação do código-fonte [14].

A especificação formal pode também ser utilizada pelo desenvolvedor como um "filtro" dos requisitos do cliente, revelando ambiguidades, incompletudes e inconsistências na definição dos requisitos do sistema [15].

Quando a especificação formal é usada no início do processo de desenvolvimento, as falhas são reveladas mais cedo, evitando altos custos nas fases de teste e depuração do código. Quando usados tardiamente, eles podem ajudar a comprovar a correção de uma implementação e a equivalência de diferentes implementações [15].

As linguagens de especificação formal mais conhecidas são: a Linguagem Z, o Método B, VMD (*Vienna Development Method*) e CSP (*Communicating Sequential Processes*).

Neste trabalho, utiliza-se a Linguagem Z, pois esta linguagem apresenta às especificações em textos informais complementadas com descrições formais [37], isto proporciona um "equilíbrio" entre a especificação formal e a especificação informal, com isso as especificações em Z não se tornam tão cansativas para o desenvolvedor.

2.5 LINGUAGEM DE ESPECIFICAÇÃO FORMAL VS LINGUAGEM DE PROGRAMAÇÃO

Uma linguagem de especificação formal é usada para descrever o que um sistema deve fazer para alcançar os requisitos de uma aplicação, no entanto, sem descrever as instruções de como deve ser feito para alcançar este objetivo [37].

As linguagens de especificação formal são diferentes das linguagens de programação que dão detalhes da implementação de um sistema executável, ou seja, dizem como fazer. Uma linguagem de programação permite que um programador especifique precisamente sobre quais dados um computador vai atuar, como estes dados serão armazenados ou transmitidos e quais ações devem ser tomadas sob várias circunstâncias [37].

As linguagens de especificação são usadas durante as fases de análise de requisitos e parte do processo de desenvolvimento de software. As linguagens de especificações geralmente não são executadas diretamente. Elas são de mais alto nível de abstração em relação as linguagens de programação [37].

Uma linguagem de especificação formal obedece a uma rígida coleção de leis matemáticas que tornam possível raciocinar efetivamente sobre como um sistema irá se comportar e reduzir o número de inconsistências no funcionamento da aplicação [6].

2.6 A LINGUAGEM DE ESPECIFICAÇÃO FORMAL Z

A linguagem ou notação Z foi desenvolvida no fim dos anos 70, por *Jean-Raymond Abrial* com a ajuda de *Steve Schuman* e *Bertrand Meyer* [16], e desenvolveu-se durante os anos 80 através de projetos de colaboração da *Universidade de Oxford* (Inglaterra).

Um padrão internacional da linguagem Z foi concluído em 2002 [17]. A linguagem Z, ainda tem uma versão adaptada para a programação orientada a objeto, denominada de "*Object-Z"* [38], nesta versão polimorfismos e heranças são suportados [18].

A linguagem Z é uma linguagem de especificação formal usada para descrever e modelar o comportamento de sistemas computacionais. Ela possui um alfabeto, uma sintaxe e uma semântica precisa. A Linguagem Z é baseada na Teoria dos Conjuntos e na Lógica de Predicados de Primeira Ordem [6]. Consequentemente, uma especificação escrita em Z herda as propriedades da matemática, assim como o rigor formal da matemática também é incorporado em Z.

2.6.1 SÍMBOLOS BÁSICOS DE UMA ESPECIFICAÇÃO EM LINGUAGEM Z

Na Tabela 2.1, os símbolos dos operadores lógicos em Z são apresentados.

Tabela 2.1: Símbolos dos operadores lógicos em Z [14]

Símbolos	Descrição
$\wedge$	E
$\vee$	Ou
$\Rightarrow$	Implica
$\Leftrightarrow$	Equivalente
$\neg$	Não

Na Tabela 2.2, os símbolos dos operadores relacionais em Z são apresentados.

Tabela 2.2: Símbolos dos operadores relacionais em Z [14]

Símbolos	Descrição
$=$	Igualdade
$\neq$	Negação de Igualdade
$\leqslant$	Menor ou igual
$<$	Menor
$\geqslant$	Maior ou igual
$>$	Maior

Na Tabela 2.3, os símbolos dos operadores sobre conjuntos em Z são apresentados.

Tabela 2.3: Símbolos dos operadores de conjuntos em Z [14]

Símbolos	Descrição
∀	Para todo
∃	Existe (pelo menos um)
∄	Não Existe
∈	Pertence
∉	Não Pertence
⊆	Subconjunto ou igual
⊂	Subconjunto
∪	União
∩	Intersecção

2.6.2 ESQUEMAS

Os esquemas são considerados a espinha dorsal de uma especificação em Z. Eles auxiliam a estruturar e modularizar a especificação, possibilitam descrever estados, operações, tipos, predicados e teoremas.

Os esquemas estão divididos em duas partes, conforme ilustra a Figura 2.12.

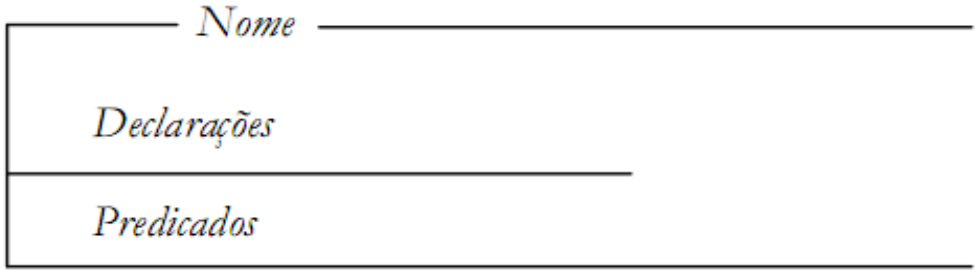

Figura 2.12: Um esquema de prova de teoremas em Z [20]

No esquema genérico da Figura 2.12, tem-se acima da linha divisória a parte declarativa, onde as variáveis de entrada e saída são declaradas. A segunda parte, abaixo da linha divisória do esquema, é a parte predicativa, onde são especificadas as regras em lógica de predicados de primeira ordem. Estas regras estabelecem as relações que devem existir entre as variáveis de entrada e saída do sistema [20]. Na segunda parte, as precondições e pós-

condições estão descritas e estas devem ser satisfeitas pelos valores inicias das variáveis de estado envolvidas na operação [19].

A Figura 2.13 descreve um esquema em Z, denominado de "*Pessoal"*.

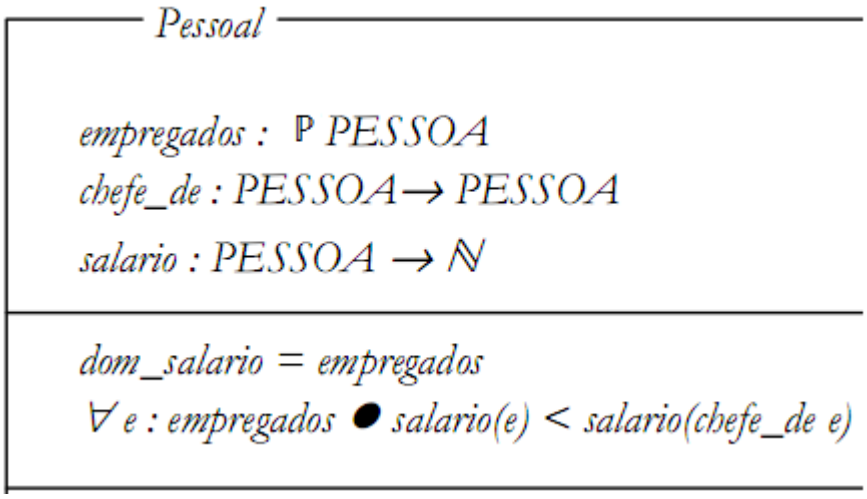

Figura 2.13: Um exemplo prático de um esquema na linguagem Z [20]

No esquema da Figura 2.13, tem-se acima da linha divisória a declaração das variáveis:

- A variável "***empregados***" definida como um conjunto de ***PESSOA***;
- Uma função matemática "*chefe_de"* (**PESSOA** → **PESSOA**) o domínio dessa função é o conjunto **PESSOA** e é constituído pelas pessoas da empresa que são chefe de alguma outra pessoa e o segundo conjunto também chamado de **PESSOA** é o contra-domínio da função e é constituído por pessoas que são empregados e possui algum chefe, esta função associa uma dada pessoa que é chefe a uma lista de pessoas que são seus empregados;
- E por último uma função "*salario"* (***PESSOA*** → ℕ) que associa uma dada ***PESSOA***, seja ela chefe ou empregado, a um valor natural (ℕ) para o seu salário.

Este esquema possui como predicado, a relação de que todos os funcionários que são empregados possuem salário menor do que o do seu chefe.

O predicado detalhado fica: o conjunto de funcionários ***e*** do tipo "***empregados***" deve satisfazer a seguinte condição: ***salario (e) < salario (chefe_de e).*** O símbolo "•" é um separador de estruturas da parte predicativa do esquema, que separa ***e:empregados*** (que é um conjunto) da desigualdade ***salario (e) < salario (chefe_de e).***

2.6.3 AXIOMA MATEMÁTICO DA LINGUAGEM Z

As definições axiomáticas são estruturas de especificação que admitem uma determinada declaração como verdade [19].

A Figura 2.14 apresenta um axioma onde é declarada uma função representada por ***f: veiculos*** $\longrightarrow$ ***motores*** que associa um tipo de ***veículo*** ao seu ***motor***.

$$f : veiculos \rightarrow motores$$

$$\begin{aligned} &\operatorname{dom} f = veiculos = \{carro?, moto?\} \\ &\operatorname{ran} f = motores = \{Ma!, Mb!, Mc!\} \\ &\forall v : veiculos \bullet fcarro? = Ma! \end{aligned}$$

Figura 2.14: Um axioma em linguagem Z [6]

Abaixo da linha divisória do esquema da Figura 2.14, tem-se o predicado, onde:

- **dom *f*** – é um conjunto chamado de domínio da função ***f*** que é constituído da seguinte forma ***veículos = {carro?, moto?}***;
- **ran *f*** – é um conjunto chamado de imagem da função ***f*** que é constituído da seguinte forma ***motores = {Ma!, Mb!, Mc!}*** onde ***Ma!, Mb! e Mc!*** são motores;

Na terceira linha do predicado segue a regra que deve ser admitida como verdadeira, onde: para todo e qualquer que seja ($\forall$) o elemento ***v*** do conjunto veículos a função ***f*** associará o elemento de entrada ***carro?*** pertencente ao conjunto ***veículos*** a somente o elemento ***Ma!*** pertencente ao conjunto ***motores***.

2.6.4 SÍMBOLOS DE CONSTRUÇÃO DE ESQUEMAS DA LINGUAGEM Z

Para utilizar construções em Z é necessário ter conhecimento de algumas das definições dos símbolos utilizados. Estes símbolos também são utilizados nas definições axiomáticas e declarações de esquemas.

Na Figura 2.15, a definição do símbolo de abreviação livre em Z é apresentada.

Símbolo	Nome	Função/Definição	Exemplo
::=	Abreviação livre	Denota um novo símbolo para o termo. Os elementos presentes na declaração são livres, não necessitando estar previamente especificados.	Result ::= Error | Fail | Success

Figura 2.15: Definição de abreviação livres em Z [14]

Na Figura 2.16, a definição do símbolo de separador declarativo em Z é apresentada.

Símbolo	Nome	Função/Definição	Exemplo
|	Separador declarativo	Separa a parte declarativa da parte predicativa. A parte anterior é a parte declarativa e a parte posterior a parte predicativa.	{declarativa | predicativa} $\{x : \mathbb{Z} \mid x \geqslant 0\}$

Figura 2.16: Definição do separador declarativo em Z [14]

Na Figura 2.17, a definição dos símbolos de delimitadores declarativos em Z é apresentada.

Símbolo	Nome	Função/Definição	Exemplo
{}	Delimitadores declarativos	Delimitadores nos quais estão contidas as partes predicativas e declarativas da função definida.	{declarativa | predicativa} $\{x : \mathbb{Z} \mid x \geqslant 0\}$

Figura 2.17: Definição de delimitadores declarativos em Z [14]

Na Figura 2.18, a definição do símbolo de declaração de tipos em Z é apresentada.

Símbolo	Nome	Função/Definição	Exemplo
[ ]	Declaração de tipo	Declaração de tipos em Z. Para quesito de organização e consistência da notação, a utilização de tipos é extremamente pertinente.	[CLIENT, ADDRESS]

Figura 2.18: Definição de declaração de tipos em Z [14]

Na Figura 2.19, a definição de separador em Z é apresentada.

Símbolo	Nome	Função/Definição	Exemplo
•	Separador	Fornecer mecanismo que utilize dos valores da parte declarativa em uma nova expressão. Sendo a função {a : b | c • d }, pode-se definir como: conjunto das expressões `d` que `a` é do tipo `b` e satisfaz as condições em `c`.	{ x : CLIENT | x is blond • phone(x) }

Figura 2.19: Definição de separador em Z [14]

2.6.5 SÍMBOLOS DAS OPERAÇÕES ENTRE CONJUNTOS DA LINGUAGEM Z

Algumas das construções mais relevantes em Z são apresentadas a seguir, sendo utilizadas nas operações com conjuntos.

Na Figura 2.20, a definição do símbolo de mapeamento em Z é apresentada. Tem a mesma função que o mapeamento [32] de elementos em MDE.

Símbolo	Nome	Função/Definição	Exemplo
$\mapsto$	Mapeamento	Relaciona um elemento x a outro elemento y. Possui o mesmo comportamento de relação existente em um par ordenado (x, y).	$x \mapsto y$

Figura 2.20: Definição de Mapeamento em Z [14]

Na Figura 2.21, a definição do símbolo de produto cartesiano em Z é apresentada.

Símbolo	Nome	Função/Definição	Exemplo

$\times$	Produto Cartesiano	Representação de um produto cartesiano entre dois conjuntos.	Singer=={Waters, Gilmour}
			Music=={Dogs, Mother, Money}
			Singer $\times$ Music==
			{
			{Waters,Dogs},{Gilmour,Dogs},
			{Waters,Mother},{Gilmour,Mother},
			{Waters,Money},{Gilmour Money}
			}

Figura 2.21: Definição de Produto Cartesiano em Z [14]

Na Figura 2.22, a definição do símbolo de relação binária em Z é apresentada.

Símbolo	Nome	Função/Definição	Exemplo
$\leftrightarrow$	Relação	Conjunto de todas as relações entre x e y.	A relação é uma abreviação de: $X \leftrightarrow Y == \mathbb{P}\ \{ X \times Y \}$ Se X = {a,b} e Y = {1,2} $X \leftrightarrow Y$ == {∅, {a,1}, {a,2},{b,1}, {b,2}, {{a,1},{a,2}}, {{a,1},{b,1}}, {{a,1},{b,2}}, {{a,2},{b,1}}, {{a,2},{b,2}}, {{b,1},{b,2}}, ...}

Figura 2.22: Definição de Relação Binária em Z [14]

Na Figura 2.23, a definição do símbolo de domínio em Z é apresentada.

Símbolo	Nome	Função/Definição	Exemplo
Dom	Domínio	Domínio de uma função. O domínio de uma função sendo $X \leftrightarrow Y$ é denotado pelos elementos em X associados aos elementos em Y.	dom R = $\{ x : X; y : Y \mid x \mapsto y \in R \bullet x \}$ Se x = {{1↦A},{2↦B},{3↦C}} dom(x) = {1,2,3}

Figura 2.23: Definição de Domínio em Z [14]

Na Figura 2.24, a definição do símbolo de imagem em Z é apresentada.

Símbolo	Nome	Função/Definição	Exemplo
Ran	Imagem	Imagem de uma função. A imagem de uma função sendo $X \leftrightarrow Y$ é denotado pelos elementos em Y associados ao elementos em X.	ran R = $\{ x : X; y : Y \mid x \mapsto y \in R \bullet y \}$ Se x = {{1↦A},{2↦B},{3↦C}} ran(x) = {A,B,C}

Figura 2.24: Definição de Imagem em Z [14]

2.6.6 Símbolos e Definições de Funções da Linguagem Z

A definição de função se dá como: cada objeto de um conjunto deve estar relacionado à no máximo um objeto de outro conjunto [14].

Na Tabela 2.4, os tipos de funções em Z são apresentados.

Tabela 2.4: Tipos de funções em Z [14]

Símbolos	Descrição
$\rightarrow\!\!\!\!\!\!+$	Função Parcial
$\rightarrow$	Função Total
$\rightarrowtail\!\!\!\!\!\!+$	Função Parcial Injetora
$\rightarrowtail$	Função Total Injetora
$\twoheadrightarrow\!\!\!\!\!\!+$	Função Parcial Sobrejetora
$\twoheadrightarrow$	Função Total Sobrejetora
$\rightarrowtail\!\!\!\!\!\!\twoheadrightarrow$	Função Total Bijetora

Na Figura 2.25, a função parcial em Z é apresentada.

Função	Definição
Função parcial	Considerando a função $X \rightarrow\!\!\!\!\!\!+ Y$ é parcial porque nem todos os elementos de X possam estar associados a elementos de Y.

Figura 2.25: Função Parcial em Z [14]

Na Figura 2.26, a função total em Z é apresentada.

Função	Definição
Função total	Considerando a função X → Y é total porque todos os elementos de X estão associados a elementos de Y.

Figura 2.26: Função Total em Z [14]

Na Figura 2.27, a função parcial injetora em Z é apresentada.

Função	Definição
Função parcial injetora	Considerando a função X ⤔ Y é injetora parcial porque os elementos de X que possuam associação estão associados a apenas um elemento distinto em Y. Pode haver elementos em X que não estão associados a elementos de Y.

Figura 2.27: Função Parcial Injetora em Z [14]

Na Figura 2.28, a função total injetora em Z é apresentada.

Função	Definição
Função total injetora	Considerando a função X ↣ Y é injetora total porque todos os elementos de X estão associados a apenas um elemento distinto em Y.

Figura 2.28: Função Total Injetora em Z [14]

Na Figura 2.29, a função parcial sobrejetora em Z é apresentada.

Função	Definição
Função parcial sobrejetora	Considerando a função X ⤖ Y é sobrejetora parcial porque todos os elementos em Y que possuam associação estão associados a apenas um elemento distinto em X. Pode haver elementos em Y que não estão relacionados com elementos em X.

Figura 2.29: Função Parcial Sobrejetora em Z [14]

Na Figura 2.30, a função total sobrejetora em Z é apresentada.

Função	Definição
Função total sobrejetora	Considerando a função X → Y é sobrejetora total porque todos os elementos em Y estão associados a um elemento distinto em X.

Figura 2.30: Função Total Sobrejetora em Z [14]

Na Figura 2.31, a função total bijetora em Z é apresentada.

Função	Definição
Função total bijetora	Considerando a função X ⤖Y é bijetora total porque todos os elementos de X possuem apenas um elemento distinto associado em Y.

Figura 2.31: Função Total Bijetora em Z [14]

A linguagem Z ainda contém uma "caixa" de ferramentas matemáticas que é uma biblioteca padrão ou ainda uma espécie de "*toolkit*" de definições matemáticas. Este "*toolkit*" é uma ferramenta que permite muitas estruturas serem descritas de forma resumida, pois os tipos de dados que ele contém são orientados em direção a simplicidade da representação matemática [6].

O "*toolkit*" é constituído de operações básicas de conjuntos, relação binária, pares ordenados, função como um tipo especial de relação, os números naturais, as seqüências que são como funções cujos domínios são determinados por segmentos de números naturais e por fim os "*bag's"* que são um tipo especial de conjuntos [6].

Podem-se reunir os benefícios da linguagem Z da seguinte forma:

- Geralmente as especificações formais são muitas vezes cansativas, os projetistas de Z minimizaram este problema desenvolvendo sua estrutura de especificação apresentada em textos informais e complementada com descrições formais [38];
- Melhora a prevenção de ambiguidades na definição de conceitos comuns;
- Ajuda a obter uma especificação mais consistente e segura de uma aplicação.

2.7 A FERRAMENTA DE VALIDAÇÃO FORMAL Z/EVES

A última versão do Z/EVES é a 2.4.1 e foi lançada em 2005 [12]. Entretanto, o mecanismo das provas do Z/EVES está estável há mais de 10 anos, isto é, não se tem relato de nenhuma falha no provador.

Optou-se por utilizar o Z/EVES, devido a fácil manipulação das suas funcionalidades, pois o usuário não precisa ser um especialista em métodos formais para utilizá-lo, os conceitos abordados nesta ferramenta são elementares da teoria dos conjuntos.

O Z/EVES é baseado na teoria dos conjuntos e na lógica de predicados de primeira ordem. Esta ferramenta é utilizada para criar, checar e analisar os esquemas especificados em linguagem Z [12].

Z/EVES é um software baseado no sistema EVES (*Enviroment for Verifying and Evaluating Software*) [21,22], que utiliza o provador do EVES para efetuar provas matemáticas. Contudo, não é necessário ter um conhecimento aprofundado sobre o sistema EVES para usar o Z/EVES, pois o Z/EVES realiza automaticamente as conversões internas. O especificador lida somente com a notação em linguagem Z e os comandos de prova disponibilizados pelo provador [12].

As especificações podem ser inseridas e verificadas em um parágrafo no Z/EVES. Um parágrafo é o local na janela principal de especificação do Z/EVES onde estão as regras escritas em linguagem Z indicando as etapas da especificação. O Z/EVES ainda pode ser usado para realizar: checagem de sintaxe, tipos, expansão de esquemas, cálculo de pré-

condições, checagem de domínio, provas de refinamento, execução simbólica e prova de teoremas.

Para construir as provas usando o Z/EVES, o especificador dispõe de um conjunto de definições matemáticas e teoremas baseados no *toolkit* de definições matemáticas da linguagem Z [12].

O Z/EVES também é capaz de ler e importar arquivos inteiros de especificações que tenham sido previamente preparados no formato LATEX, através de um pacote chamado `z-eves.sty` que define os comandos necessários para a especificação formal da linguagem Z no LATEX.

2.7.1 JANELA PRINCIPAL DE ESPECIFICAÇÃO DE PARÁGRAFOS DO Z/EVES

A janela de especificação é a tela principal do Z/EVES, onde várias funções importantes (tal como salvar e sair) estão disponíveis. Nesta janela a parte formal de uma especificação é exibida, ou seja, os parágrafos contendo as regras escritas em linguagem Z através dos esquemas. A Figura 2.32 ilustra a janela principal de especificação em Z do Z/EVES.

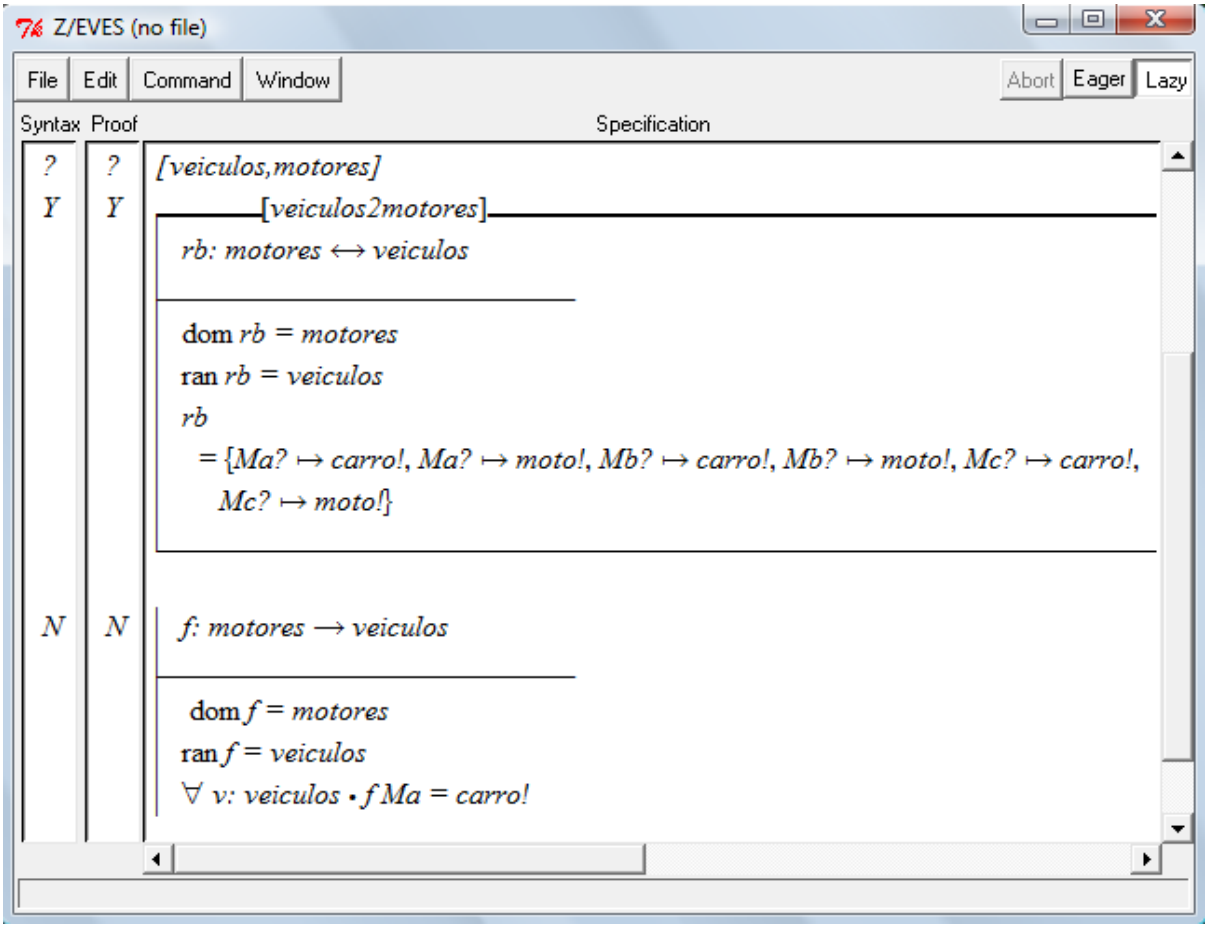

Figura 2.32: Janela de especificação de parágrafos [12]

O Z/EVES oferece dois módulos de operação, **"***Eager***"** e **"***Lazy***"**. No módulo **"***Eager***",** as regras de checagem são rígidas, ou seja, um parágrafo pode somente ser checado se todos os outros anteriores a ele já tiverem sido checados.

No módulo **"***Lazy***",** um parágrafo pode ser checado individualmente, ou seja, a sua verificação é independente da checagem dos parágrafos anteriores a ele. Este módulo é útil na experimentação, por exemplo, uma especificação pode conter duas versões alternativas de um parágrafo, para serem analisados isoladamente.

O estado de verificação da validade do parágrafo é exibido em duas colunas à esquerda de cada parágrafo.

A primeira coluna, da esquerda para a direita, exibe o estado da sintaxe do parágrafo através de três símbolos [12]:

- "?" indica que o parágrafo ainda não foi checado;
- "Y" indica que o parágrafo foi checado e não tem erros de sintaxe;
- "N" indica que o parágrafo foi checado e tem erros de sintaxe.

Na segunda coluna, da esquerda para a direita, é exibido o estado da prova matemática, do parágrafo usando um dos três símbolos [12]:

- “?” indica que o parágrafo ainda não foi checado, e o estado da prova matemática ainda não pode ser determinado;
- “Y” indica que o parágrafo foi checado e não tem erros de sintaxe ou tipo;
- “N” indica que o parágrafo tem um objetivo associado que ainda não foi provado.

Estas colunas ajudam o usuário a manter o controle das provas matemáticas da especificação, mostrando o que foi e o que ainda não foi verificado.

2.8 RESUMO

Neste capítulo, apresentou-se uma revisão dos principais conceitos das tecnologias utilizadas para o desenvolvimento deste trabalho. A MDE foi apresentada destacando suas principais características, seus casos particulares, como por exemplo, o padrão MDA, as tecnologias envolvidas e os benefícios proporcionados pela utilização desta abordagem.

Uma visão geral do uso dos métodos formais no desenvolvimento de software foi apresentada. Também as características da linguagem de especificação formal Z com as suas funcionalidades necessárias para a produção desta pesquisa e por fim uma descrição da ferramenta de prova formal Z/EVES.

3 ESTADO DA ARTE

Este capítulo descreve os trabalhos encontrados na literatura sobre modelagem de sistemas, as deficiências presentes nos modelos informais que representam a transformação de modelos.

Apresenta-se também, abordagens que propõem o uso de métodos formais baseados na Teoria dos Conjuntos como uma alternativa de solução para o problema da ambiguidade presente na representação da transformação de modelos. E por fim é feito uma análise dos trabalhos relacionados.

3.1 MODELAGEM DE SISTEMAS DE SOFTWARE

Os modelos são visões do software em um determinado nível de abstração. Segundo "*Mellor"* [11], modelo "é uma abstração de um sistema físico que distingue o que é pertinente do que não é, com o intuito de simplificar a realidade". Um modelo contém todos os elementos necessários à representação de um sistema real [11].

De acordo com "*Kleppe et al"* [5] um modelo é "uma descrição de um (ou de uma parte do) sistema expresso em uma linguagem bem definida, ou seja, respeitando um formato (uma sintaxe) e um significado (uma semântica) preciso que é apropriado para a interpretação automática por um computador" [5].

A criação de modelos é feita utilizando uma linguagem de modelagem bem definida que apresente uma sintaxe e uma semântica para regulamentar a criação dos elementos e suas relações. Uma linguagem de modelagem é uma especificação formal bem definida que contém os elementos de base para construir modelos. Além disto, uma linguagem de modelagem é concebida dentro de um domínio limitado e com objetivos específicos [9].

Assim as linguagens naturais não são adequadas para descrever o funcionamento de um sistema de software, porque elas não podem ser interpretadas por computadores e podem permitir falhas na sua interpretação.

3.2 INCONSISTÊNCIAS NOS MODELOS REPRESENTATIVOS DA TRANSFORMAÇÃO DE MODELOS

Os elementos presentes na arquitetura de MDE tais como modelo, metamodelo, metametamodelo, transformação e outros devem ser definidos através de "modelos representativos" precisos, construídos através de uma linguagem sólida e bem definida que evite erros de interpretação.

Porém os modelos representativos encontrados na literatura de MDE apenas descrevem os elementos da arquitetura de MDE através da linguagem natural (geralmente o inglês) e desenhos informais [3]. Como exemplos, têm-se os modelos representativos ilustrados na Figura 3.1, Figura 3.2 e Figura 3.3 que foram retiradas do MDA Guide Version 1.0.1[4].

Na Figura 3.1, a transformação do PIM (modelo fonte) para o PSM (modelo alvo) é abstraída como *transformationSpecification*.

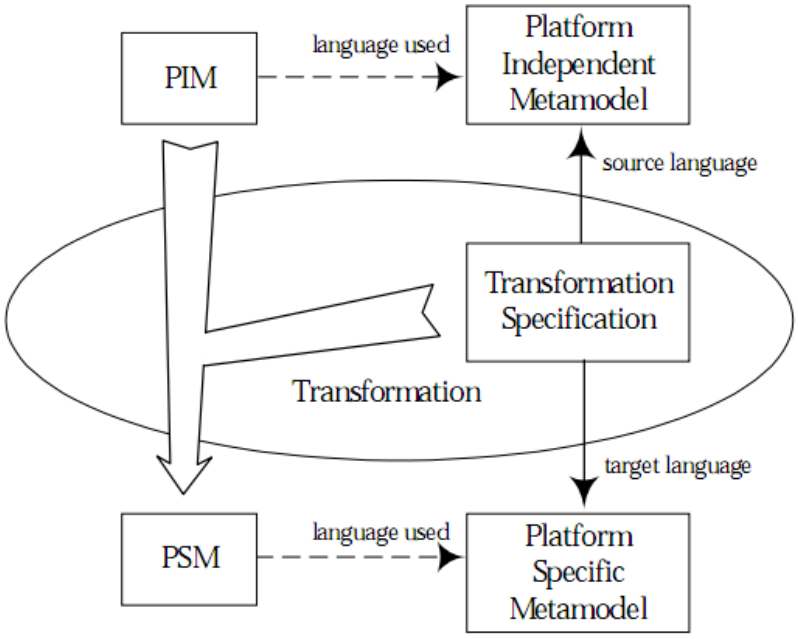

Figura 3.1: Transformação entre metamodelos no *"MDA Guide Version 1.0.1"* [4]

A Figura 3.2 representa o processo de transformação através de um modelo parcialmente diferente do modelo da Figura 3.1. Ao comparar-se as duas figuras, percebe-se que elas apresentam formas e denotações diferentes para a representação da transformação.

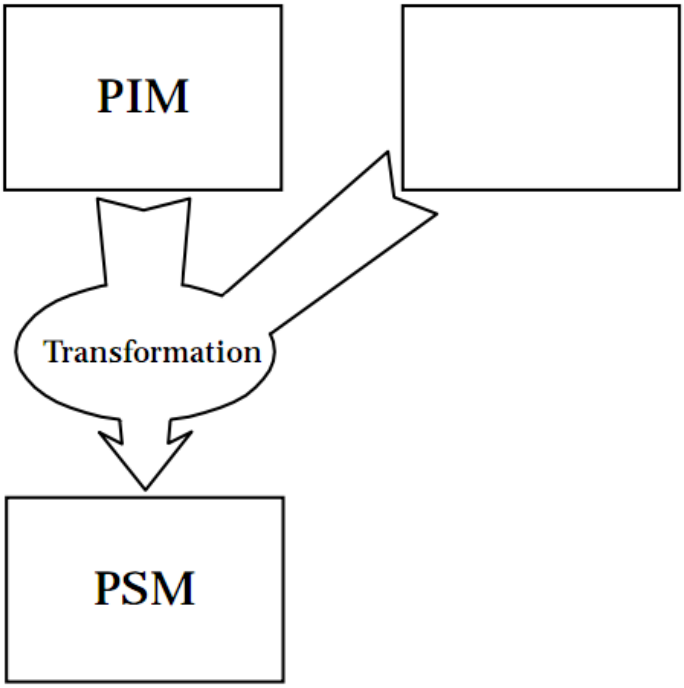

Figura 3.2: Transformação padrão segundo *"MDA Guide Version 1.0.1"* [4]

Na Figura 3.3, extraída do mesmo material, no lugar de *Transformation* foi colocado *ModelMerge*.

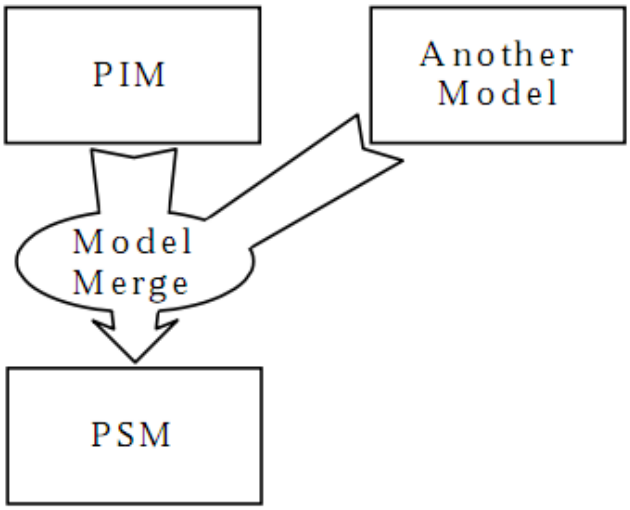

Figura 3.3: Transformação de modelos através da técnica de fusão de modelos [4]

No *"MDA Guide 1.0.1",* o texto que se refere à Figura 3.3, explica que é uma maneira diferente de mostrar mais detalhes de outras maneiras de como uma transformação pode ser feita. Porém no mesmo parágrafo o autor destaca que a Figura 3.3 é destinada a ser

sugestiva e que é apenas um modelo genérico, e nada mais. Com isso, a versão do *MDA Guide* (Versão 1.0.1) não usa qualquer linguagem bem definida.

Uma outra falha desses modelos representativos é que eles não abordam a característica da *bidirecionalidade,* característica esta que garante a uma transformação ser realizada em ambas as direções do PIM (modelo fonte) para o PSM (modelo alvo) e do PSM para o PIM. Este processo pode ser também denominado de transformação reversa.

Em uma versão anterior de um documento da OMG [23], um modelo representativo foi proposto para tentar formalizar a bidirecionalidade e outros elementos da MDE. A Figura 3.4 ilustra este modelo representativo.

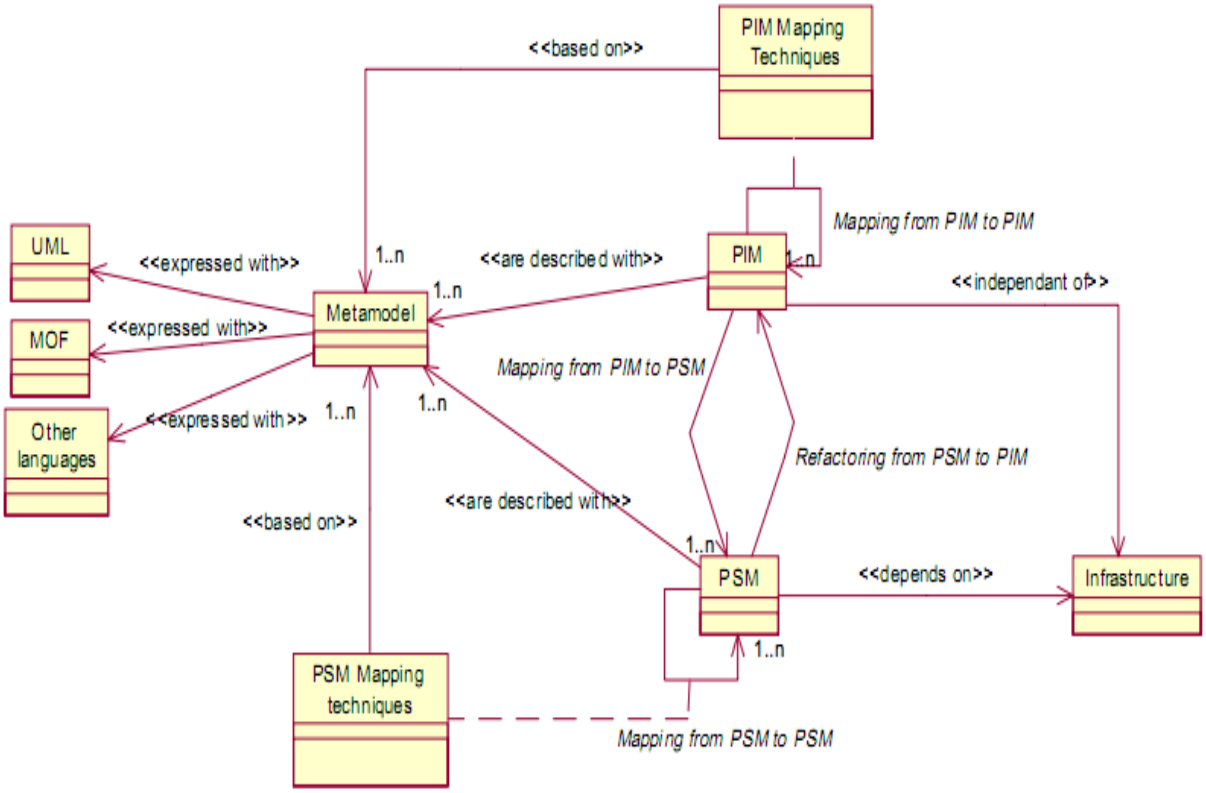

Figura 3.4: Descrição de um metamodelo [23]

No entanto, conforme ilustrado na Figura 3.4, verifica-se que ela não é formalizada em uma linguagem bem definida, ou seja, com uma semântica e uma sintaxe precisa, como por exemplo, a linguagem UML. Com isso o modelo representativo da Figura 3.4, não está completamente coerente.

No modelo representativo da Figura 3.4, também não está claro como se classifica uma transformação do PIM para o PSM em bidirecional, apesar de haver uma indicação implícita na Figura 3.4 da existência desta transformação entre eles, veja na Figura 3.5.

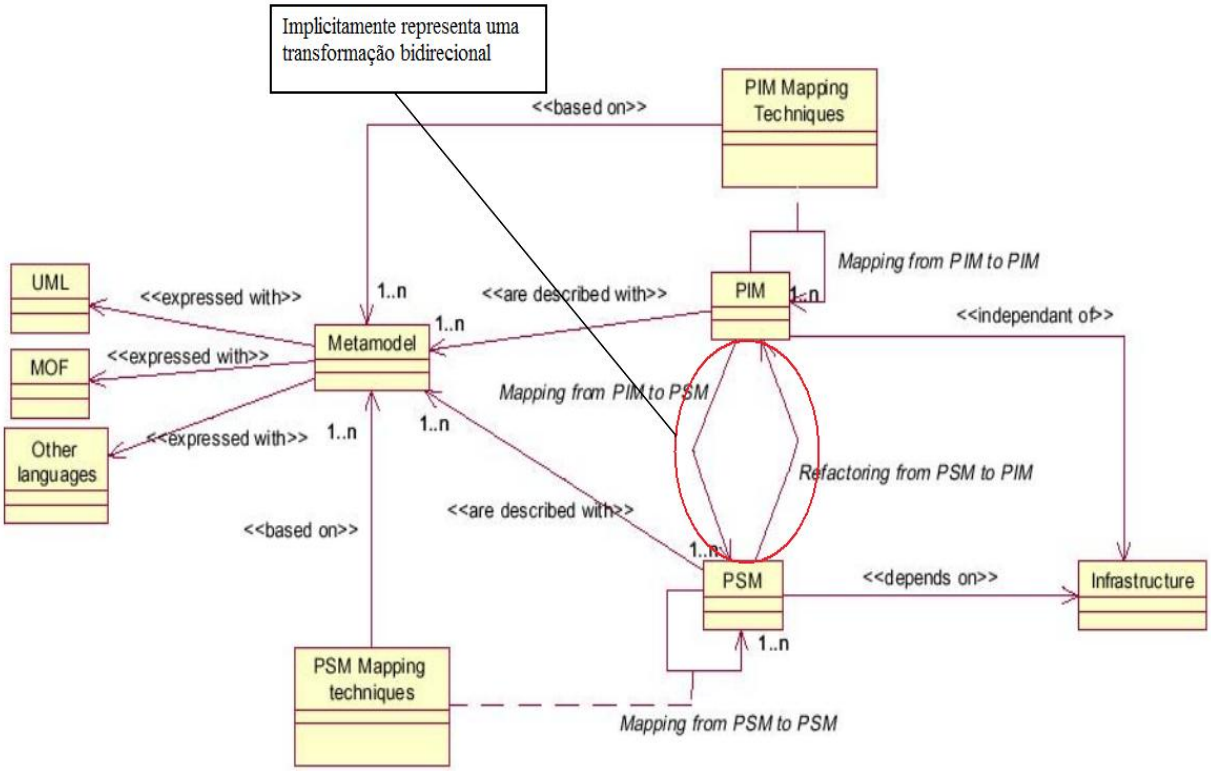

Figura 3.5: Bidirecionalidade Implícita [23]

Infelizmente, o modelo representativo da Figura 3.5 não está totalmente coerente, pois não contêm de forma clara todos os elementos de MDE presentes em uma transformação, o que pode resultar em confusão ao usuário e erros em sua interpretação.

No trabalho de "*Bézivin"* [24], a transformação é representada apenas por uma seta no sentido do modelo fonte para o modelo alvo como ilustrado a Figura 3.6. Isto ocorre devido a transformação ser realizada entre modelo fonte e modelo alvo, onde estes últimos são criados respectivamente conforme a um metamodelo da linguagem fonte e a um metamodelo da linguagem alvo.

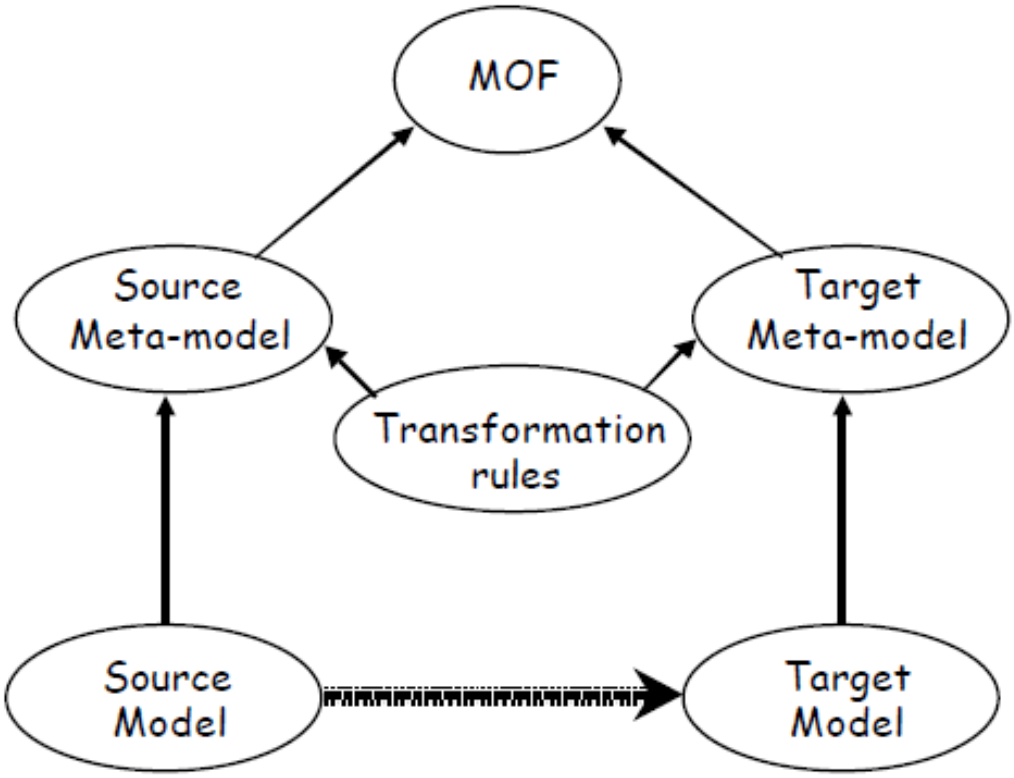

Figura 3.6: Transformação de modelos segundo *"Bézivin"* [24]

No modelo da Figura 3.6 verifica-se a existência de alguns elementos presentes no processo de transformação, tais como metamodelo fonte e alvo, metametamodelo MOF, as regras de transformação. Porém, verifica-se que este modelo não descreve nenhum tipo de mapeamento entre os elementos do metamodelo fonte com os elementos do metamodelo alvo.

Na Figura 3.6, verifica-se a existência da *Transformation rules* (regras de transformação), mas não existe um elemento como *transformation tool* (ferramenta de transformação) nem algo como *transformation definition* (definição de transformação).

Ao comparar a Figura 3.6 com a Figura 3.7 extraída do livro "*MDA Explained"* [5] nota-se a ausência desses elementos no processo de transformação, mais uma vez comprova-se a falta de consenso na terminologia dos modelos representativos que tentam formalizar o processo de transformação no contexto da MDE.

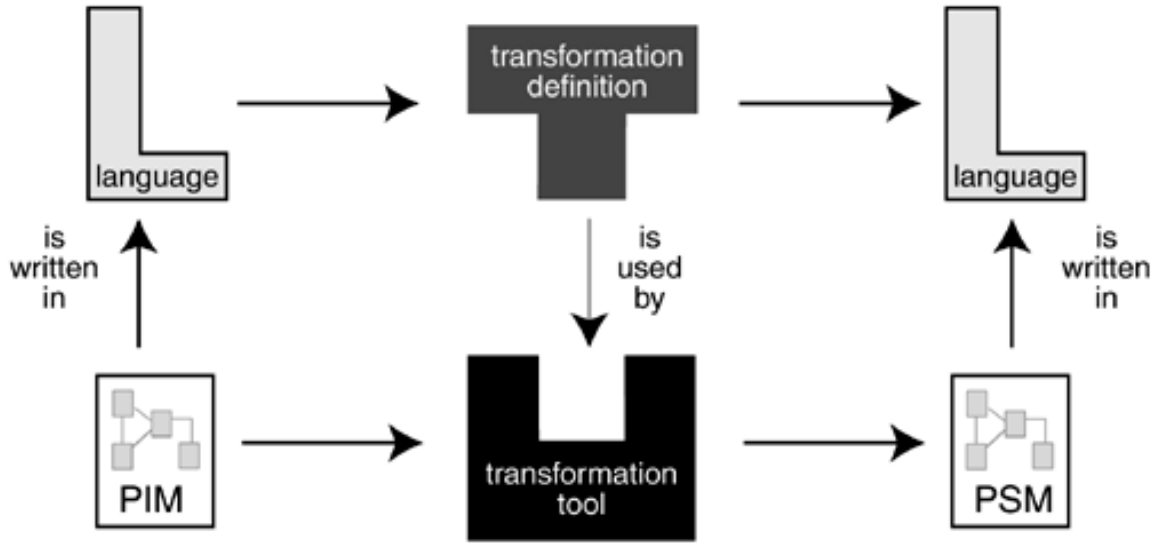

Figura 3.7: Transformação de modelos segundo o *"MDA Explained"* [5]

Também ao comparar-se a Figura 3.6 com a Figura 3.7, concluí-se que a transformação está sendo representada por termos diferentes, o que é uma *Transformation rules* (regras de transformação) na Figura 3.6, na Figura 3.7 se chama *transformation definition* (definição de transformação). Concluímos novamente a inconsistência na terminologia que abstrai a ideia de transformação

Se estes modelos representativos da MDA forem levados para se raciocinar sobre estruturas complexas, eles não serão suficientemente satisfatórios para dar respostas corretas sobre uma dada aplicação, como por exemplo, eles não abordam às transformações que são realizadas em ambas as direções, ou seja, as transformações bidirecionais. Logo, esses modelos não são totalmente suficientes para se interpretar o processo de transformação com precisão.

O livro *"MDA destilada"* [11] fornece um simples modelo representativo de MDE expresso em UML (ver Figura 3.8). Este modelo constitui uma das contribuições fortes deste livro. Ele ajuda muito o leitor a compreender quais são os conceitos importantes de MDE e alguns de seus relacionamentos.

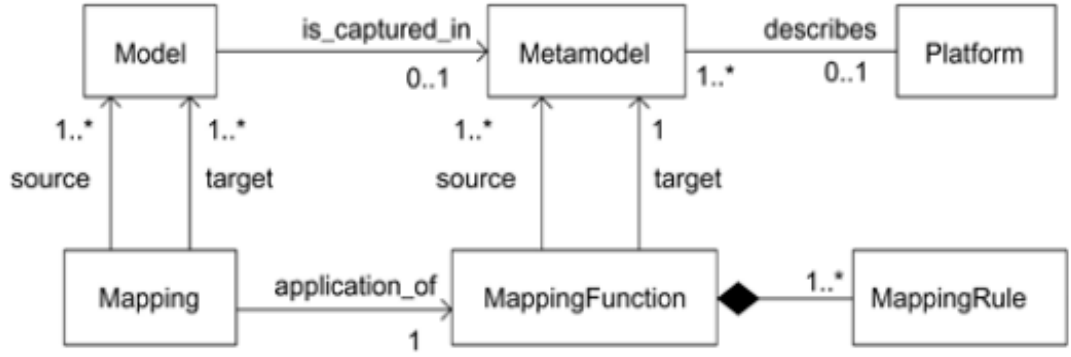

Figura 3.8: Transformação de modelos segundo o *"MDA Destilada"* [11]

Infelizmente, este modelo representativo não é inteiramente coerente, pois não abrange todos os elementos de uma aplicação em MDE. Por exemplo, de acordo com o modelo da Figura 3.8, não há um objeto como um metametamodelo (um modelo que define a linguagem para expressar metamodelos).

Os mapeamentos entre os elementos do metamodelo alvo com os do metamodelo fonte não são considerados neste modelo, não está claro se a transformação é entre modelo ou metamodelo e a transformação é representada pelo termo "*MappingFunction"* outra denotação diferente.

De acordo com "*Kleppe et al"* [5] os modelos representativos da Figura 3.1 à Figura 3.8, apresentadas nesta seção não podem ser classificadas como modelos. Pois, elas não possuem uma linguagem bem definida, ou seja, não respeita um formato (uma sintaxe) e um significado (uma semântica) preciso [5].

Assim, elas estão sujeitas a erros de interpretação e, portanto dependendo da complexidade da aplicação elas não podem ser usadas para raciocinar sobre estruturas complexas de forma precisa.

3.3 TRABALHOS RELACIONADOS

Enquanto os modelos representativos encontrados na literatura de MDE foram construídos com diferentes objetivos em mente, eles são úteis apenas para situações específicas em particular. Com isso vários modelos representativos distintos surgiram em casos particulares, resultando em uma infinidade de representações diferentes para os elementos de MDE.

Assim, os conceitos de modelo, metamodelo e de transformação podem ser confundidos, não apenas no espaço tecnológico de MDA, mas também no espaço tecnológico de *Grammarware*, de *Documentware*, de *Dataware*, etc [25]. E esta é a situação em que uma "modelagem consistente" é necessária.

Favre [3] propõe o uso da Teoria dos Conjuntos para aumentar a compreensão dos elementos de MDE e os relacionamentos existentes entre eles. Favre sugere construir um "modelo representativo" dos conceitos de MDE denominado de *megamodelo*, desde que este *megamodelo* inclua o conceito de modelo e metamodelo em sua representação [3].

A Teoria dos Conjuntos usada por Favre [3] é a teoria presente nas especificações formais da linguagem Z [6], Favre [3] ainda sugere usar os elementos matemáticos da

linguagem Z como notação para expressar os elementos existentes no processo de transformação de modelos.

Em seguida, os megamodelos de Favre [3] que tratam da transformação de modelos são apresentados.

No primeiro megamodelo de Favre [3] o processo de transformação é representado em diagramas de objetos, como está ilustrado na Figura 3.9.

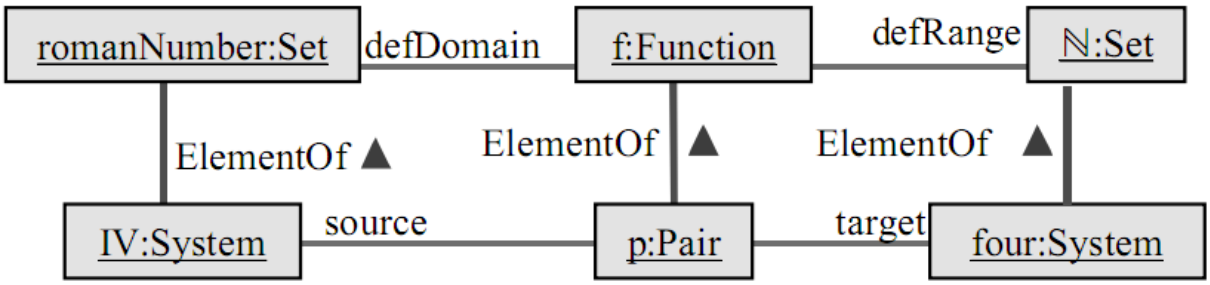

Figura 3.9: Transformação de modelos em Diagramas de Objetos [3]

O megamodelo da Figura 3.9 descreve que uma função chamada *f* transforma o número romano *IV* elemento do conjunto *defDomain* (domínio de definição) para o número inteiro *four* elemento do conjunto *defRange* (conjunto-imagem).

Ao comparar o megamodelo de Favre [3] da Figura 3.9 com os modelos apresentados nas figuras anteriores (ver Figura 3.1, Figura 3.6 e Figura 3.7) verifica-se que de acordo com Favre [3], a transformação é formalizada como uma "*função matemática"*, o metamodelo fonte e o metamodelo alvo são respectivamente a "*definição do domínio (defDomain")* e a *"definição da imagem (defRange)"* da função (ver Figura 3.9) e a relação entre um elemento de um conjunto e o conjunto é definida como *ElementOf* .

De acordo com Favre [3], o conceito de transformação é naturalmente definido na teoria dos conjuntos como uma função (Favre [3] não descreve qual o tipo de função é usada), pois o conceito de relação binária (neste caso de acordo com a Figura 3.9 a relação binária aqui tratada é uma função matemática) é claramente uma boa abstração para as transformações conforme os diagramas de objetos da Figura 3.9 [3].

O segundo megamodelo é ilustrado na Figura 3.10. Este megamodelo é o *megamodelo padrão* de Favre [3], onde os principais elementos da arquitetura de MDE e os tipos de relacionamentos existentes no processo de transformação são representados.

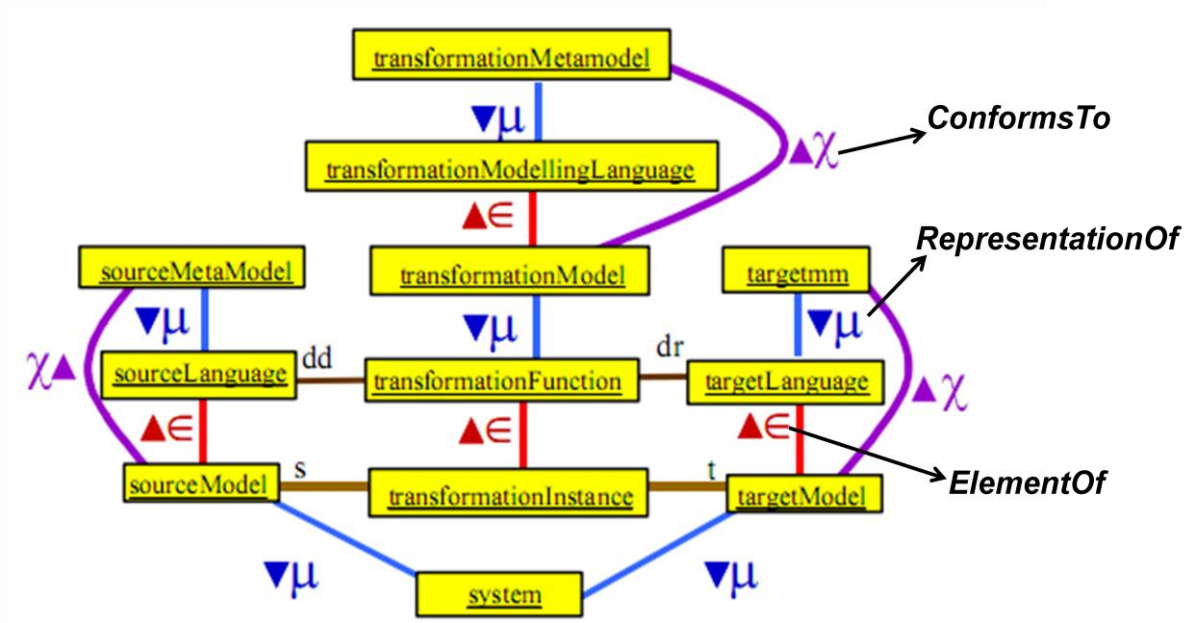

Figura 3.10: Transformação de modelos segundo o Megamodelo Padrão de "*Favre*" [3]

No megamodelo padrão de Favre [3] apresentado na Figura 3.10, a transformação é formalizada como uma "função matemática*"* através do termo *transformationFunction,* o metamodelo fonte (*sourceMetaModel*) é formalizado como o "domínio da função" e o metamodelo alvo (*targetmm*) como a *"*imagem da função*"*.

Na Figura 3.10, as letras "dd" (*definition domain)* representam a definição do domínio e as letras "dr" (*definition range*) representam a definição da imagem. As letras "s" (*source*) e "t" (*target*) representam respectivamente o modelo fonte e o modelo alvo.

Nesse megamodelo padrão, o símbolo μ representa a relação *RepresentationOf*, uma relação muito importante definida entre um modelo e um sistema em estudo representado por este modelo, esta relação foi identificada por muitos outros autores em [26] e [27].

Os outros símbolos χ e ∈ representam respectivamente as relações *ConformsTo* **(χ)** e *ElementOf* (∈). *ConformsTo* **(χ)** define a relação entre um modelo e um metamodelo, ou seja, entre modelos, pois um metamodelo também é um modelo. E a relação *ElementOf* (∈) estabelece a relação entre os elementos (por ex.: classes, interfaces) e o seu conjunto (metamodelo).

No megamodelo da Figura 3.10 verifica-se a existência de um elemento denominado de *transformationFunction* (que representa uma transformação) e que tem o papel de transformar um elemento no metamodelo fonte em um elemento no metamodelo alvo (ver Figura 3.10).

Neste megamodelo, observa-se ainda que o metamodelos fonte (*sourceMetaModel*) e o metamodelo alvo (*targetmm*) são representados respectivamente por uma linguagem fonte *(sourceLanguage)* e uma linguagem alvo (*targetLanguage*) conforme a relação *RepresentationOf (*μ**)**, ver a Figura 3.10.

A função de transformação (*transformationFunction)* é representada pelo modelo de transformação (*transformationModel)* de acordo com a relação *RepresentationOf (*μ**),** ver a Figura 3.10.

Os conceitos da teoria dos conjuntos utilizados por Favre [3] para construir o megamodelo estão descritos no pacote da Figura 3.11.

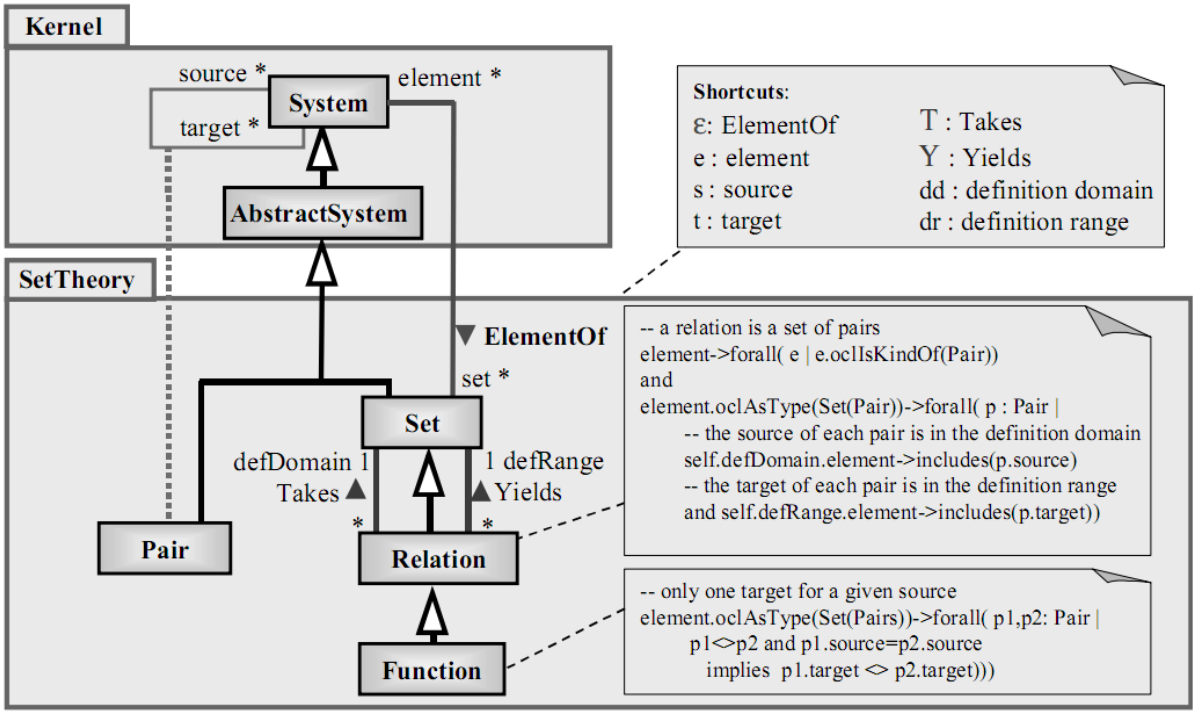

Figura 3.11: Pacote da Teoria dos Conjuntos [3]

Este pacote apresenta os conceitos da teoria dos conjuntos através da notação UML. As declarações em OCL são usadas para definir os elementos da teoria dos conjuntos, como pode ser verificado a Relação (*Relation*) e a Função (*Function*) são definidos em OCL nos retângulos ao lado da Figura 3.11.

Os elementos Pares (*Pair*), Conjunto (*Set*), Relação (*Relation*) e Função (*Function*) da teoria dos conjuntos são utilizados para representar os elementos da MDE presentes no processo de transformação de modelos.

3.4 ANÁLISE DOS TRABALHOS RELACIONADOS

Dos trabalhos relacionados, pode-se concluir que na literatura vigente os modelos representativos que apresentam os elementos de MDE não estão completamente coerentes quanto à representação dos elementos da arquitetura de MDE, principalmente quanto à representação da transformação entre modelos.

Os modelos fornecidos pela OMG no *"MDA Guide Version 1.0.1"*, apresentam varias denotações para representar um único conceito da MDE relacionado com a ideia de transformação.

Além do manual da OMG, outras propostas também são incompletas ao tentarem padronizar a ideia de transformação, pois devidos aos diferentes objetivos com que os seus modelos foram desenvolvidos, os seus autores não levaram em consideração uma representação genérica associada aos casos específicos em que a transformação pode ocorre.

Assim, esses modelos se tornaram inconsistentes ao utilizar diferentes nomenclaturas para representar um único conceito de MDE, resultando em ambiguidades em suas representações.

Estas falhas na representação conduzem o usuário a cometer falhas em sua interpretação, uma vez que, a maior parte dos modelos discutidos neste capítulo tratam a transformação apenas como unidirecional e os modelos que tentam ilustrar a existência de uma transformação bidirecional, não deixam claro como é feito para se diferenciar uma transformação bidirecional de uma transformação unidirecional.

Além de que alguns desses modelos não apresentam todos os elementos presentes no processo de transformação de modelos e nem os relacionamentos entre estes elementos.

Em meio a este cenário, Favre [3] propõe a utilização da Teoria dos Conjuntos associada à linguagem de especificação formal Z para solucionar este problema.

Favre [3] propõe utilizar um megamodelo padrão para representar os artefatos da MDE presentes na transformação de modelos.

Neste megamodelo, verificou-se que a transformação é representada como uma função matemática da teoria dos conjuntos. No entanto, Favre [3] não especificar qual o tipo de função está sendo utilizada por ele para representar a transformação. Ele também considera o metamodelo fonte como o domínio e o metamodelo alvo como o conjunto-imagem dessa função.

Porém, o megamodelo de Favre não está totalmente correto, pois uma transformação em MDE nem sempre irá ter características de uma função matemática, pois existem transformações com mapeamentos de elementos de um-para-muitos e de muitos-para-muitos que contraria a definição matemática de função.

Com isso, conclui-se que o megamodelo de Favre [3] é incompleto, pois é inadequado utilizar a definição de função para generalizar a transformação no contexto da MDE, uma vez que a ideia de função não atende a todos os casos de mapeamento de elementos.

3.5 RESUMO

Neste Capítulo, apresentou-se os principais modelos representativos encontrados na literatura de MDE, bem como a falta de consenso nestes modelos sobre como representar o processo de transformação da MDE. Sendo assim, há ambiguidades nos termos utilizados para representar a transformação de modelos. Também, apresentou-se o trabalho de Favre [3] que propõe o uso da Teoria dos Conjuntos e da linguagem Z para resolver o problema da falta de precisão nos modelos informais.

4 FORMALIZAÇÃO DA TRANSFORMAÇÃO DE MODELOS

Neste capítulo, uma abordagem desenvolvida para formalizar matematicamente o processo de transformação de modelos da MDE é apresentada, bem como uma análise matemática da transformação e os resultados obtidos com esta análise. E a metodologia sugerida para provar formalmente a transformação é apresentada.

4.1 ESTUDO FORMAL DA TRANSFORMAÇÃO ENTRE MODELOS

Um modelo é uma simplificação de um sistema construído com um determinado objetivo em mente, estando escrito em uma linguagem com uma sintaxe e semântica precisa, além de evitar ambiguidades e inconsistências em sua interpretação [5].

Por isso, realizou-se uma análise formal da transformação de modelos com o objetivo de estabelecer qual elemento matemático da teoria dos conjuntos melhor representa os elementos da MDE envolvidos no processo de transformação.

4.1.1 ANÁLISE FORMAL DA TRANSFORMAÇÃO DE MODELOS

Ao realizar uma análise formal da transformação de modelos, os elementos matemáticos utilizados para representar os elementos de MDE são: Conjuntos [6], Relação Binária [6], Função, Função Bijetiva [6] e Pares Ordenados [6]. Conforme o "*toolkit*" de definições matemáticas do Z/EVES [12] estes elementos tem as seguintes características:

a) Conjuntos – O conceito de conjunto é especificado em Z de acordo com o esquema da Figura 4.1. No esquema da Figura 4.1, denominado de "X", dois conjuntos distintos X e X são declarados. Uma relação binária, representada pelo símbolo $\leftrightarrow$, existe entre os conjuntos.

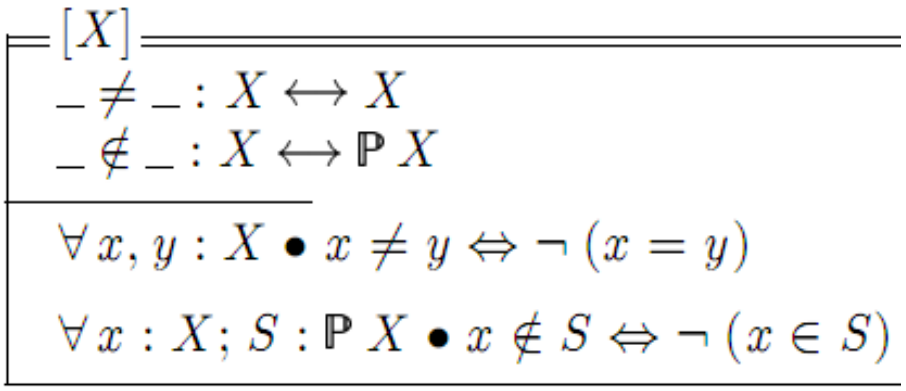

Figura 4.1: Conjuntos em linguagem Z [6]

No esquema da Figura 4.1, a regra que assegura a definição de conjunto em Z está declarada em lógica de predicados de 1º ordem.

O símbolo "•" é um separador de estruturas da parte predicativa. O símbolo $\mathbb{P}$X é chamado de conjunto das partes de um conjunto e representa todos os subconjuntos de X, como por exemplo, se A = {x, y} então $\mathbb{P}$A = {∅, {x}, {y}, {x, y}}.

Os conjuntos definidos na linguagem Z são recursos importantes para formalizar a transformação de modelos, com isto nesta dissertação uma transformação possui um domínio e um conjunto-imagem. Estes dois conjuntos são criados respectivamente baseados no metamodelos fonte e no metamodelo alvo da transformação.

Porém, nesta dissertação o metamodelo fonte não é o domínio da transformação e o metamodelo alvo não é o conjunto-imagem da transformação, ou seja, é diferente da abordagem apresentada no trabalho do Favre [3].

Pois no trabalho de Favre [3] os megamodelos (ver Figuras 3.9 e Figura 3.10) representam a transformação como uma função, isto implica que de acordo com a definição matemática de função, entende-se que **todos** os elementos do domínio (no caso do megamodelo de Favre [3] o metamodelo fonte) deverão está associados a algum elemento do conjunto-imagem (metamodelo alvo) para que a transformação tenha características de uma função matemática.

No entanto, na prática em um processo de transformação de modelos nem sempre **todos** os elementos do metamodelo fonte serão mapeados para os elementos do metamodelo alvo.

Nesta dissertação criou-se o domínio e o conjunto-imagem, respectivamente distintos do metamodelo fonte e do metamodelo alvo, para que estes conjuntos contenham apenas os elementos dos metamodelos fonte e do metamodelo alvo que estejam envolvidos no processo de transformação. Estes conjuntos, domínio e conjunto-imagem, representam o fragmento de metamodelo da transformação.

Os elementos que serão fornecidos para o domínio e para o conjunto-imagem serão selecionados de acordo com os requisitos estabelecidos na aplicação.

b) Relação Binária – A relação binária é especificada em Z, de acordo com a seguinte regra:

$$X \leftrightarrow Y == \mathbb{P}(X \times Y)$$

Sendo X e Y conjuntos, então $X \leftrightarrow Y$ é uma relação binária entre X (domínio fonte) e Y (imagem alvo). $\mathbb{P}(X \times Y)$ é um conjunto que contêm todos os subconjuntos do produto cartesiano $X \times Y$. Assim, a expressão $X \leftrightarrow Y == \mathbb{P}(X \times Y)$ indica que a relação binária é igual a todos os subconjuntos do produto cartesiano $X \times Y$.

Toda relação binária realiza um tipo de mapeamento entre os elementos fonte do domínio, representados por *x*, com os elementos alvo do conjunto-imagem, representados por *y*. Um mapeamento pode ser representado em linguagem Z por $x \mapsto y$ e indica um par ordenado do tipo (*x*, *y*). Este mapeamento é o mesmo definido na letra "e" do item 2.2.1 do Capítulo 2 desta dissertação, onde a definição foi extraída de "*Bernstein*" [32].

Na Figura 4.2, tem-se um esquema com um predicado que formaliza em linguagem Z o mapeamento [32] entre os elementos *x* e *y* representado por um par ordenado (*x*, *y*).

$$[X, Y]$$
$$\_ \mapsto \_ : X \times Y \longrightarrow X \times Y$$
$$\forall x : X; y : Y \bullet$$
$$x \mapsto y = (x, y)$$

Figura 4.2: A Relação Binária em linguagem Z [6]

O predicado do esquema da Figura 4.2 indica que para todo e qualquer *x* do tipo X e *y* do tipo de Y existe um mapeamento [32] de *x* para *y* que é igual a um par ordenado (x, y).

A **relação binária** é o elemento matemático ideal para representar uma transformação em MDE, pois ela é mais abrangente do que a definição matemática de função utilizada por Favre [3]. Uma vez que a relação binária é compatível com os casos de mapeamento de elementos de um-para-um, um-para-muitos, muitos-para-um e muitos-para-muitos da transformação unidirecional e bidirecional.

Já a função utilizada por Favre [3] não é compatível quando em uma transformação um elemento fonte estiver mapeado [32] para mais de um elemento alvo ou quando ocorrer uma transformação reversa e um elemento alvo for mapeado para mais de um elemento fonte.

c) Função – A função em Z é um tipo especial de relação binária.

A relação binária torna-se uma função quando transforma um elemento do conjunto fonte (domínio) a um e somente um elemento do conjunto alvo (contra-domínio). Os principais tipos de funções são a função parcial, a função total e a função bijetiva.

c_1) Função Parcial – Na Figura 4.3, o predicado que estabelece a condição de existência de uma função parcial é descrito.

$$X \pfun Y == \{ f : X \rel Y \mid (\forall x : X; y_1, y_2 : Y \bullet (x \mapsto y_1) \in f \land (x \mapsto y_2) \in f \Rightarrow y_1 = y_2) \}$$

Figura 4.3: Função Parcial [6]

A regra da Figura 4.3 estabelece a condição de existência de uma função parcial, entre um conjunto *X* e *Y*, onde uma relação binária *f* entre *X* e *Y* tal que (|) para todo e qualquer que seja ($\forall$) um *x* do tipo *X* e um y_1 e y_2 do tipo *Y* existe um mapeamento [32] de *x* para y_1 ($x \mapsto y_1$) pertencente a *f* e (^) um mapeamento de *x* para y_2 ($x \mapsto y_2$) também pertencente a *f* implicando que y_1 é igual a y_2, ou

seja, que a relação binária *f* será uma função se um elemento fonte do domínio estiver associado a um e somente um elemento alvo do contra-domínio da função.

A função parcial é utilizada quando se necessita relacionar apenas uma parte dos elementos do domínio com os elementos do contra-domínio da função, sem que a relação binária perca a característica de uma função matemática.

c_2) Função Total – Na Figura 4.4, a função total é ilustrada.

$$X \longrightarrow Y == \{ f : X \pfun Y \mid \mathrm{dom}\, f = X \}$$

Figura 4.4: Função Total [6]

Esta regra estabelece a existência de uma função total ($X \longrightarrow Y$) quando uma função parcial (f : X $\pfun$ Y) f de um conjunto X para um conjunto Y tal que (|) o domínio (dom f) desta função parcial for igual ao conjunto X da função parcial, ou seja, dom f = X.

c_3) Função Bijetiva – Outra função importante da teoria dos conjuntos é a função bijetiva. A Figura 4.5 descreve o esquema que especifica a função bijetiva em linguagem Z.

$$[X, Y]$$

$$f : X \fun Y$$

$$X \bij Y = \{ f : X \fun Y \mid \mathrm{ran}\, f = Y \}$$

Figura 4.5: O conceito de Função Bijetiva em linguagem Z [6]

O predicado abaixo da linha divisória do esquema da Figura 4.5, garante a existência da função bijetiva entre dois conjuntos distintos X e Y, onde o símbolo $\bij$ representa a função bijetiva em Z e o símbolo $\longrightarrow$ representa a função total.

O predicado abaixo da linha divisória do esquema da Figura 4.5 indica que uma função bijetiva de X em Y (X $\bij$ Y) é estabelecida quando tem-se uma função

total de X em Y ($X \longrightarrow Y$) tal que se a imagem de *f* é igual a Y (*ran f* = Y), ou seja, quando o conjunto imagem (*ran f*) da função for igual ao contra-domínio Y da função.

A função bijetiva é o artefato matemático ideal para formalizar uma transformação bidirecional, pois este tipo de função pode ser realizada em ambos os sentidos semelhante à transformação bidirecional. Por isso, nesta dissertação, a transformação bidirecional é tratada como uma **função bijetiva**.

4.1.2 Hierarquia de Artefatos Matemáticos de uma Transformação de Modelos

Com a análise formal da transformação de modelos, estabeleceu-se qual elemento matemático da teoria dos conjuntos serão utilizados para representar cada elemento de MDE envolvidos no processo de transformação. Estes elementos matemáticos denominaram-se de Artefatos Matemáticos.

Os artefatos matemáticos utilizados no processo de transformação são: Pares Ordenados, Conjunto, Relação Binária, Função, Função Bijetiva [6], Sistema [28] e Sistema Abstrato [28]. Estes artefatos matemáticos estão organizados em uma **Hierarquia de Artefatos Matemáticos** conforme é ilustrado na Figura 4.6.

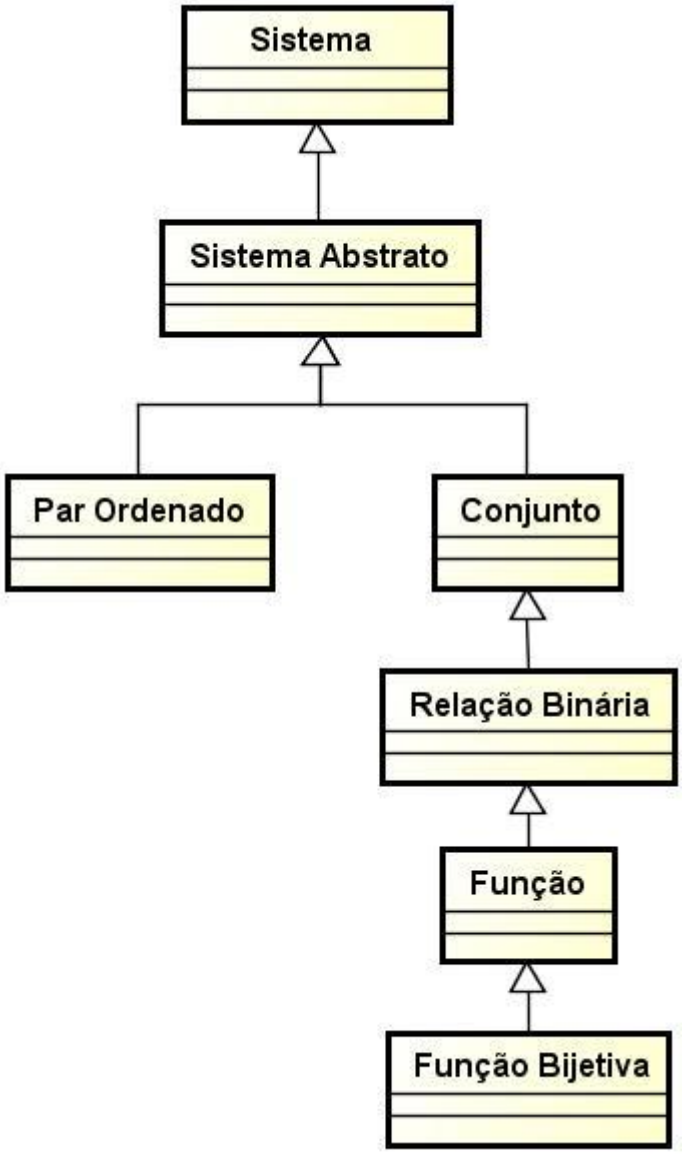

Figura 4.6: Hierarquia dos Artefatos Matemáticos Adaptada de [3]

A hierarquia da Figura 4.6 é baseada na hierarquia de Favre [3] apresentada na Figura 3.11. Entretanto, a diferença entre essas duas hierarquias é a inclusão do artefato matemático **função bijetiva**. A inclusão desse artefato é devido à função bijetiva ser mais específica para representar a transformação bidirecional, pois ambas podem ser realizadas em ambos os sentidos.

Além da função bijetiva uma função pode ser classificada também como injetiva e sobrejetiva. Na hierarquia da Figura 4.6 não há necessidade de se incluir estas outras duas funções, elas podem ser representadas de forma genérica como uma função, conforme é apresentada na Figura 4.6.

Na hierarquia da Figura 4.6 classifica-se a transformação bidirecional como uma função bijetiva que é um conceito mais específico para a bidirecionalidade, pois ambas podem ser realizadas em ambos os sentidos.

Como uma função bijetiva é um tipo de relação binária a transformação bidirecional também pode ser representada por uma relação binária.

A relação binária é um artefato matemático ideal para representar também a transformação unidirecional com características de função parcial, total, injetora ou sobrejetora, pois todas estas funções são um tipo de relação binária.

Com isso, pode-se representar de forma geral uma transformação de modelos, seja unidirecional ou bidirecional, por uma relação binária.

Cada elemento da hierarquia da Figura 4.6 pode ser conceituado, como segue:

- **Sistema:** sistema é um conjunto de partes interagentes e interdependentes que, conjuntamente, formam um todo unitário com determinado objetivo e efetuam determinada função [43].
- **Sistema Abstrato:** Consiste em ideias e conceitos que, eventualmente, residem na mente humana para serem processados pelo cérebro humano e computadores [28];
- **Pares Ordenados:** Um par ordenado é formado por dois elementos de um conjunto, no qual o primeiro é o elemento fonte e o segundo é o elemento alvo [6];
- **Conjuntos:** Consiste em uma coleção de elementos (objetos) distintos [6];
- **Relação Binária:** Uma relação binária é um conjunto de pares ordenados, onde o primeiro elemento é o fonte e o segundo é o alvo [6];
- **Função:** Uma função é um tipo especial de relação binária, que transforma um elemento fonte do domínio em um, e somente um elemento alvo do contra-domínio [19].
- **Função Bijetiva:** Uma função bijetiva é um tipo especial de função que se aplica no sentido do domínio para o conjunto-imagem e também no sentido inverso do conjunto-imagem para o domínio através de uma função inversa que também é bijetiva. Na função bijetiva, o conjunto-imagem é igual ao contra-domínio da função [6].

4.2 RESULTADOS DA ANÁLISE FORMAL DA TRANSFORMAÇÃO

De acordo com o estudo formal da transformação entre modelos apresentado no tópico anterior, foi possível estabelecer qual elemento da teoria dos conjuntos melhor representa os elementos da MDE que estão presentes no processo de transformação.

A análise formal da transformação possibilitou a construção da Tabela 4.1, onde a primeira coluna tem o elemento da MDE existente no processo de transformação, na segunda coluna o artefato matemático correspondente da teoria dos conjuntos e a terceira coluna o símbolo representativo na linguagem Z:

Tabela 4.1: Resumo da Análise Formal da Transformação

MDE	Artefato Matemático	Símbolo em Z
Metamodelo	Conjunto	**X, Y, Z,...**
Elemento Fonte	Elemento do Domínio	***x***
Elemento Alvo	Elemento do Conjunto-Imagem	***y***
Mapeamento	Par Ordenado	***(x,y)*** **ou** ↦
Transformação de modelos	Relação Binária	$\leftrightarrow$
Transformação Bidirecional	Função Bijetiva	$\rightarrowtail\!\!\!\!\!\!\rightarrow$
Transformação Reversa	Função Inversa	$\mathbf{T}^{\sim}$

Na primeira linha da Tabela 4.1, o metamodelo é associado ao artefato matemático conjunto, isto devido o metamodelo reunir elementos comuns em sua estrutura, assim como os conjuntos reúnem também elementos comuns. Na última coluna da primeira linha tem-se uma letra maiúscula qualquer sendo o símbolo utilizado para representar o metamodelo em linguagem Z.

Na segunda linha da Tabela 4.1, o elemento fonte é associado a um elemento do domínio da transformação (representada de forma genérica por uma relação binária) e na segunda linha e última coluna, tem-se uma letra *x* minúscula utilizada para representar um elemento fonte em linguagem Z.

Na terceira linha da Tabela 4.1, o elemento alvo é associado a um elemento do conjunto-imagem da transformação, na última coluna e terceira linha tem-se uma letra *y* minúscula utilizada para representar um elemento alvo em linguagem Z.

Na quarta linha da Tabela 4.1, o mapeamento [32] do elemento fonte para um elemento alvo é associado a um par ordenado *(x,y),* na última coluna da quarta linha tem-se o símbolo *(x,y)* ou ↦ para representar um mapeamento em linguagem Z.

Na quinta linha da Tabela 4.1, a transformação de modelos é associada ao artefato matemático relação binária, pois as transformações do tipo unidirecional e bidirecional podem ser representadas por uma relação binária, como descrito no item 4.1.2. Na última coluna da quinta linha, tem-se o símbolo que pode ser utilizado para representar uma transformação de modelos em linguagem Z.

Na sexta linha da Tabela 4.1, a transformação bidirecional é associada a uma função bijetiva, pois esta última é um artefato matemático mais específico para representar uma transformação bidirecional. Uma vez que a função bijetiva é aplicável em ambas as direções, assim como a transformação bidirecional. Na última coluna da sexta linha o símbolo em linguagem Z utilizado para representar uma transformação bidirecional é apresentado.

Na sétima linha da Tabela 4.1, a transformação reversa é associada a uma função inversa, pois ambas são aplicáveis no sentido do metamodelo alvo para o metamodelo fonte. Na última coluna da sétima linha o símbolo em linguagem Z utilizado para representar uma transformação reversa é apresentado.

4.3 METODOLOGIA

A Figura 4.7 apresenta a metodologia usada para formalizar uma transformação de modelos. A metodologia está organizada em um fluxograma.

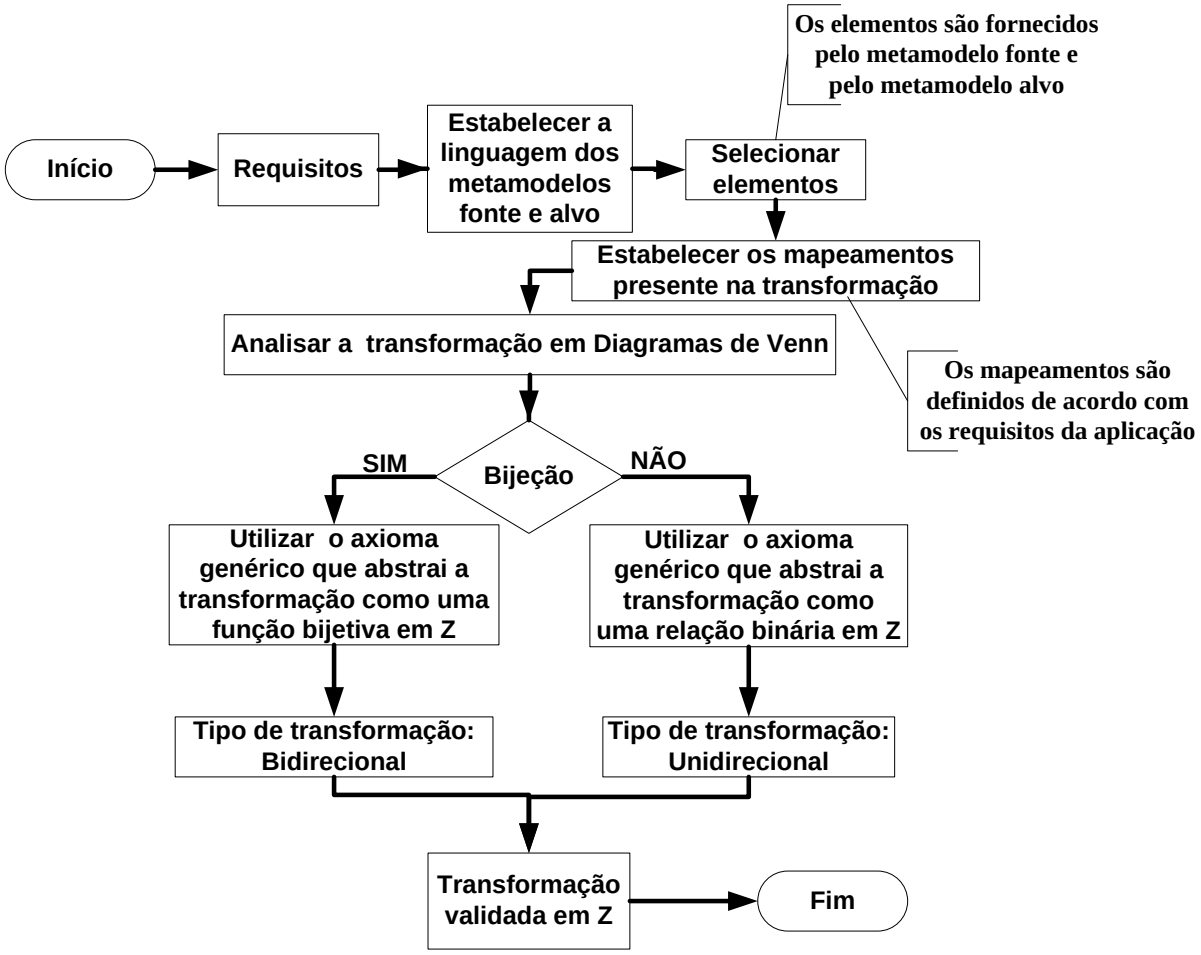

Figura 4.7: Metodologia para a formalização da transformação

Nesta metodologia, o passo inicial é definir os requisitos da aplicação, estes requisitos são estabelecidos de acordo com o estudo de caso em questão.

O segundo passo consiste em estabelecer a linguagem fonte (por exemplo: UML) para o metamodelo fonte (PIM) e a linguagem alvo (por exemplo: JAVA) para o metamodelo alvo (PSM). Estas linguagens são definidas de acordo com o problema a ser resolvido, se o usuário quer obter o esqueleto do código final em JAVA, então a linguagem alvo será a do metamodelo de JAVA.

O terceiro passo é selecionar os elementos do metamodelo fonte e do metamodelo alvo que estarão contidos respectivamente no domínio e no conjunto-imagem da transformação que pode ser uma função bijetiva ou uma relação binária. Este passo é

importante para que sejam definidos quais elementos do metamodelo fonte e alvo farão parte da transformação.

No quarto passo, os mapeamentos entre os elementos do metamodelo fonte com os do metamodelo alvo são estabelecidas de acordo com a necessidade de se atender aos requisitos da aplicação.

No quinto passo, os mapeamentos são estabelecidos e analisados matematicamente através dos diagramas de Venn [33] da teoria dos conjuntos. Nesta etapa, dois diagramas de Venn são feitos, um representando o domínio e o outro representando o contra-domínio (contra-domínio contém o conjunto-imagem) e então mapeia-se os elementos do domínio para os elementos do contra-domínio.

Após representar os mapeamentos entre os elementos em diagramas de Venn [33], verificamos se os mapeamentos existentes têm características de uma **relação binária** ou uma **função bijetiva**.

Se a transformação for igual a uma bijeção, então utiliza-se o axioma genérico que abstrai a transformação como uma função bijetiva em Z e concluí-se que a transformação é bidirecional.

Se a transformação não for igual a uma bijeção, então utilizar-se o axioma genérico que abstrai a transformação como uma relação binária em Z e concluí-se que a transformação é unidirecional.

No final da metodologia, obtem-se a transformação validada em Z e classificada em uma transformação com características de uma função bijetiva ou uma relação binária.

Ao se identificar dois artefatos matemáticos que classifica a transformação como unidirecional ou bidirecional, conseguiu-se eliminar a ambiguidade presente nos modelos apresentados anteriormente, que não fazem distinção entre a transformação unidirecional e bidirecional.

4.4 RESUMO

Todo este estudo formal da transformação entre modelos em MDE foi importante para se compreender com mais clareza o sentido próprio dos elementos envolvidos em uma transformação, bem como também identificar quais artefatos matemáticos são mais adequados para representar os elementos presentes no processo de transformação de modelos.

Os resultados deste estudo também foram fundamentais para se concluir que os modelos presentes na literatura expressam a ideia de transformação como uma “função”, uma vez que as palavras "mapeamento", "mapear", "transformar", “correspondência” são geralmente usadas como termos equivalentes a ideia de "função" da Teoria dos Conjuntos, conduzindo o usuário a pensar que todas as transformações são funções e que somente ocorrem em uma única direção (fonte-para-alvo).

Concluiu-se também que os modelos (ou megamodelos) propostos no trabalho de Favre [3] para representar a transformação, não são completamente satisfatórios, uma vez que o autor não faz distinção entre a transformação unidirecional e bidirecional. O megamodelo de Favre [3] não detalha quando se utiliza a relação binária no lugar de função e também não menciona se a função utilizada para representar a transformação é uma função simples ou sobrejetora ou injetora ou bijetora.

5 PROPOSTA DE UM *FRAMEWORK* PARA FORMALIZAR A TRANSFORMAÇÃO ENTRE MODELOS

O *framework* apresentado neste capítulo foi desenvolvido de acordo com os resultados obtidos na análise formal da transformação descrita no capítulo anterior. O *framework* também é baseado no megamodelo de Favre [3].

Este *framework* tem como objetivo formalizar a transformação entre modelos da MDE, bem como definir os elementos presentes na transformação de modelos e os relacionamentos existentes entre os elementos de seus metamodelos.

5.1 *FRAMEWORK* CONCEITUAL FORMAL

O tipo de *framework* desenvolvido nesta pesquisa é um *framework* conceitual. Um *framework* conceitual consiste em um conjunto de conceitos representados através da linguagem UML usados para resolver um problema de um domínio específico [36].

Este *framework* não se trata de um software executável, mas sim de um modelo construído com elementos extraídos da Teoria dos Conjuntos e da Engenharia Dirigida a Modelos e tem como objetivo representar o processo de transformação de modelos da MDE de forma clara e consistente.

A Figura 5.1 ilustra o *framework* proposto nesta dissertação modelado em diagramas de pacotes da UML.

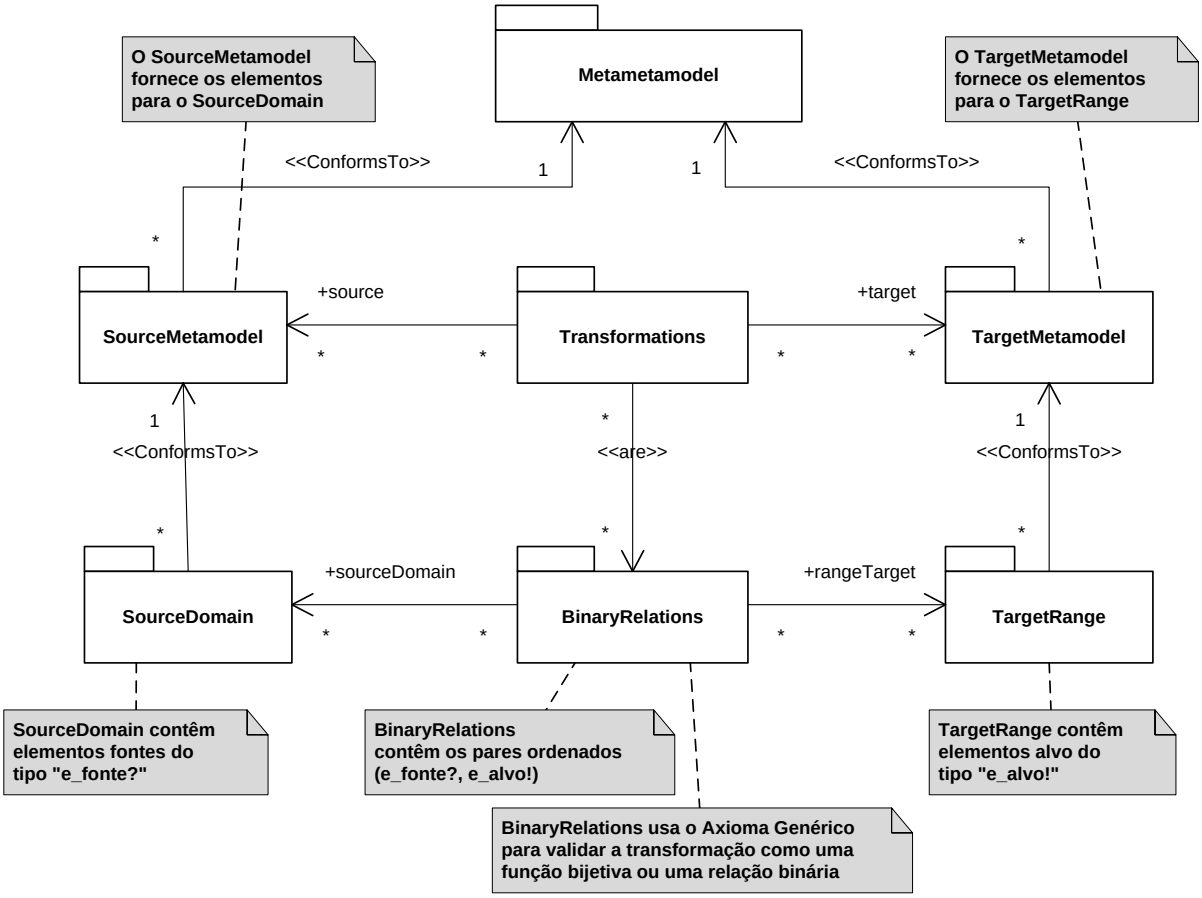

Figura 5.1: *Framework* Conceitual Formal em Diagramas de Pacotes

Na Figura 5.1, o *framework* é constituído dos pacotes *Metametamodel* (Metametamodelo), *SourceMetamodel (*metamodelo fonte*), Transformations (*Transformações*), TargetMetamodel (*metamodelo alvo*), SourceDomain (*Domínio Fonte*), BinaryRelations (*Relação Binária*), TargetRarge (*Imagem Alvo*).*

O pacote *Metametamodel* (Metametamodelo) define a linguagem para expressar o *SourceMetamodel* (metamodelo fonte) e o *TargetMetamodel* (metamodelo alvo).

O *SourceMetamodel* (metamodelo fonte) e o *TargetMetamodel* (metamodelo alvo) estão conformes o *Metametamodel* (Metametamodelo) de acordo com a relação *ConformsTo*, com isso os metamodelos fonte e alvo são criados conformes o metametamodelo estabelecido.

O *SourceMetamodel* (metamodelo fonte) fornece os elementos para o *SourceDomain (*Domínio Fonte*).* O *TargetMetamodel* (metamodelo alvo) fornece os elementos para o *TargetRange (*Imagem Alvo*).*

No pacote *Transformations (*transformações*),* as regras de transformação estão contidas. Estas regras realizam a transformação dos elementos (objetos) do metamodelo fonte (*SourceMetamodel*) em elementos (objetos) do metamodelo alvo (*TargetMetamodel*).

O pacote *BinaryRelations (*Relação Binária*)* contêm os pares ordenados do tipo *(e_fonte?, e_alvo!).* A *BinaryRelations (*Relação Binária*)* usa o axioma genérico da Figura 5.5 para validar a transformação como uma função bijetiva ou uma relação binária.

A Figura 5.2 ilustra o *framework* formalizado em diagramas de Venn [33], em seguida a Tabela 5.1 compara este *framework* com o megamodelo de Favre [3].

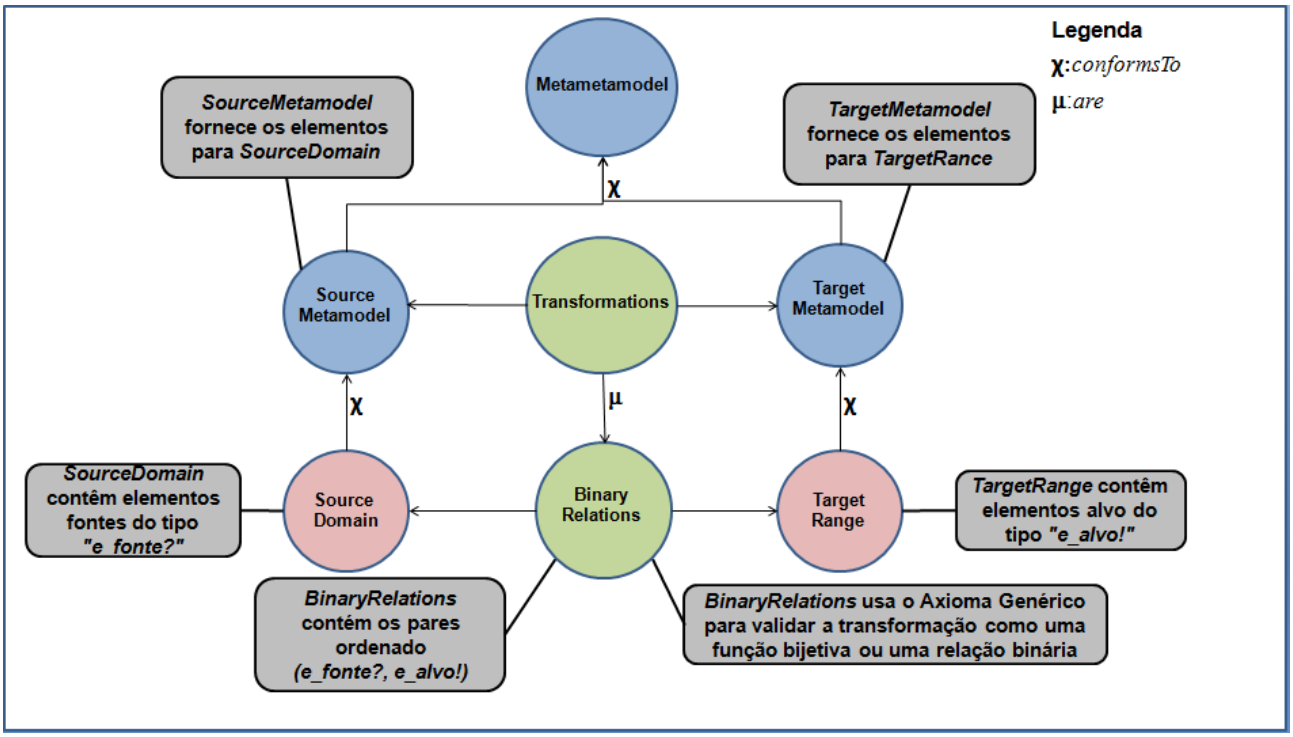

Figura 5.2: *Framework* Conceitual Formal em Diagramas de Venn

Na Figura 5.2, os pacotes são representados pelos diagramas de Venn, estes diagramas tratam os pacotes da Figura 5.1 como conjuntos de elementos.

O conjunto *SourceDomain* representa o fragmento do metamodelo fonte e o conjunto *TargetRange* representa o fragmento do metamodelo alvo. Os fragmentos de metamodelos são conjuntos formados pelos elementos selecionados do metamodelo fonte (*SourceMetamodel*) e do metamodelo alvo (*TargetMetamodel*) que estão envolvidos na transformação.

A relação binária entre os fragmentos *SourceDomain* e *TargetRange*, ver Figura 5.2, é compatível com os casos de mapeamento de elementos de um-para-um, de um-para-muitos, de muitos-para-um e de muitos-para-muitos.

O conjunto *Transformations* entre o metamodelo fonte (*SourceMetamodel*) e o metamodelo alvo (*TargetMetamodel*) contêm todas as regras de transformação entre os elementos fonte e alvo.

De acordo com a Figura 5.2, os metamodelos fonte (*SourceMetamodel*) e alvo (*TargetMetamodel*) estão relacionados com o metametamodelo pela relação ***ConformsTo*** representado pelo símbolo **χ**. Esta relação indica que os metamodelos fonte (*SourceMetamodel*) e o alvo (*TargetMetamodel*) são definidos conforme um metametamodelo estabelecido (por ex.: *MOF* ou *ECORE*). A relação ***ConformsTo*** **(χ)** foi adotada dos trabalhos de Favre em [28] e [29] da série *"From Ancient Egypt to Model Driven Engineering"* [30].

A criação dos conjuntos *SourceDomain* e *TargetRange* ocorre devido a não obrigatoriedade de se considerar em uma dada transformação todos os mapeamentos existentes entre o metamodelo fonte e o metamodelo alvo, pois o *SourceDomain* e o *TargetRange* reúnem apenas os elementos que estão envolvidos na transformação de modelos.

O conjunto *SourceDomain* (fragmento do *SourceMetamodel*) é um subconjunto do metamodelo fonte, isto implica que o metamodelo fonte contêm os elementos do conjunto *SourceDomain.* O conjunto *TargetRange* (fragmento do *TargetMetamodel*) é um subconjunto do metamodelo alvo, isto implica que o metamodelo alvo contêm os elementos do conjunto *TargetRange*.

No *framework* da Figura 5.2, o conjunto *SourceDomain* possui apenas elementos fonte (representados por *e_fonte?*) que serão transformados em elementos alvo (*e_alvo!*) no conjunto *TargetRange*, e o conjunto *TargetRange* possui apenas os elementos alvo que possuem correspondentes no conjunto *SourceDomain*. O sinal de interrogação "?" indica um elemento fonte (entrada) e o sinal de exclamação "!" indica um elemento alvo (saída).

Cada mapeamento do elemento fonte do tipo "*e_fonte?"* do metamodelo fonte para um elemento alvo do tipo "*e_alvo!"* do metamodelo alvo é tratado no *framework* como um par ordenado (*e_fonte?, e_alvo!*) conforme o símbolo **μ (*are*)** entre *Mappings* e *BinaryRelations*.

A relação binária é formada por um conjunto de pares ordenados do tipo *(e_fonte?,e_alvo!),* e cada par ordenado *(e_fonte?, e_alvo!)* pertence a relação binária. Assim, mesmo que a transformação de modelos contenha mapeamentos com características de uma função, ela continuará sendo uma relação binária, pois uma função é um tipo especial de relação binária.

A relação binária da Teoria dos Conjuntos é o artefato matemático ideal para representar a transformação, ver Figura 5.2.

Ainda na Figura 5.2 é descrito que a relação binária utiliza um Axioma Genérico validado no Z/EVES para especificar a transformação como uma função bijetiva ou uma relação binária, este axioma será apresentado na Figura 5.5 e explicado no Capítulo 6.

A Tabela 5.1 descreve os elementos envolvidos no processo de transformação de modelos, que estão presentes no *framework* conceitual formal e não estão presentes no megamodelo de Favre [3].

Tabela 5.1: Comparação entre o *Framework* Proposto e o Megamodelo de *"Favre"*

MDE	***Framework* Conceitual Formal**	**Megamodelo de Favre [3]**
Transformação de modelos	*Relação Binária*	*transformationFuction*
PIM	*MODELO FONTE*	*sourceModel*
PSM	*MODELO ALVO*	*targetModel*
Metametamodelo	*METAMETAMODELO*	*Não Tem*
Relação "ConformsTo"	*χ : ConformsTo*	*Parcialmente – apenas entre o modelo e o metamodelo*
Um elemento do PIM	*Elemento Fonte: e_fonte?*	*Não Tem*
UM elemento do PSM	*Elemento Alvo: e_alvo!*	*Não Tem*
Mapeamento	*Par Ordenado*	*Não Tem*
Fragmento de metamodelo do PIM	*SourceDomain*	*Não Tem*
Fragmento de metamodelo do PSM	*TargetRange*	*Não Tem*

A principal diferença entre o *framework* proposto nesta dissertação e o megamodelo de Favre [3] é a forma de representar a transformação entre os modelos. No *framework* apresentado nesta dissertação, a transformação é representada como uma relação binária, ver Figura 5.2, e no trabalho de Favre é representada como uma função (***transformationFunction***) como é ilustrado no diagrama de objetos da Figura 5.3 e na Figura 5.4, ambas extraídas do trabalho de Favre [3].

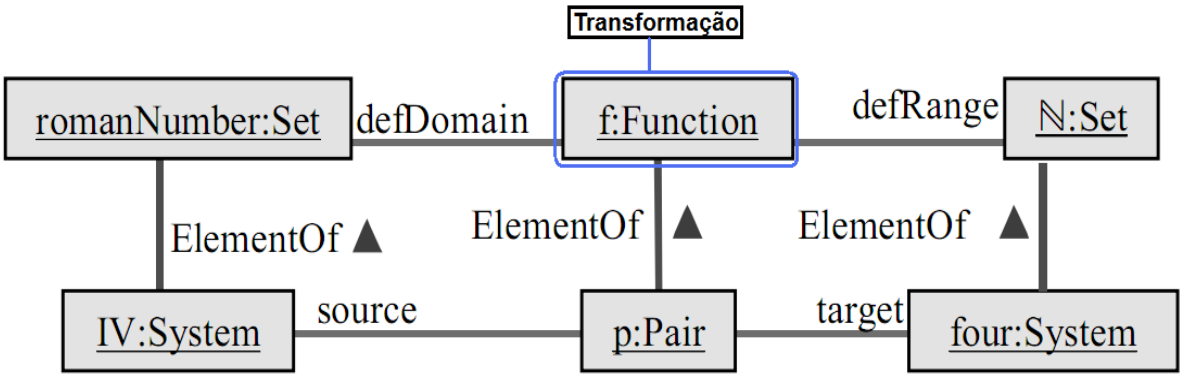

Figura 5.3: Transformação representada por uma função no Diagramas de Objetos [3]

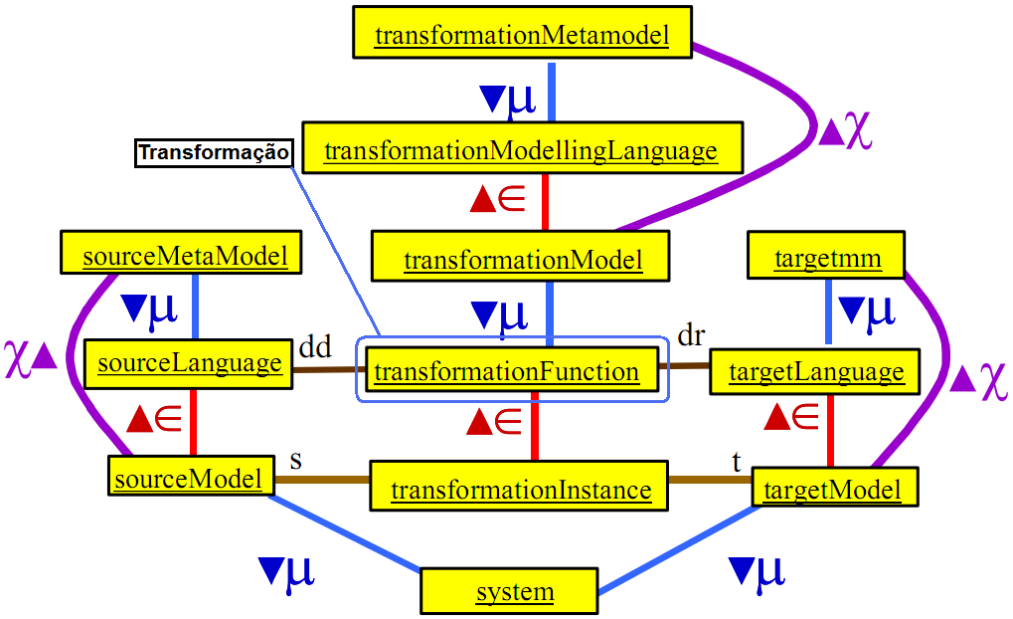

Figura 5.4: Transformação no Megamodelo Padrão de "*Favre*" [3]

A relação binária foi adotada para representar a transformação no *Framework* Conceitual Formal (ver Tabela 5.1) por ser mais abrangente que a função. Pois o conceito de função é adequado para representar apenas as transformações que apresentam os casos de mapeamento de elementos de um-para-um e de muitos-para-um, os outros casos de mapeamento de elementos de um-para-muitos e de muitos-para-muitos contraria o conceito matemático de função. Este fato é comprovado no estudo de caso do Capítulo 6.

De acordo com a Tabela 5.1, verifica-se também que no megamodelo de Favre não aparece nenhum elemento que represente um metametamodelo, nem a relação ***ConformsTo (χ)*** entre o metametamodelo e o metamodelo fonte (*SourceDomain*) e o metamodelo alvo (*TargetRange*). Estes elementos são ilustrados no *framework* da Figura 5.2.

O megamodelo de Favre também não utiliza nenhuma representação para: um único elemento do PIM (*e_fonte?*), um único elemento do PSM (*e_alvo!*), mapeamento,

fragmento do metamodelo fonte (*SourceDomain*), fragmento do metamodelo alvo (*TargetRange*). Estes elementos são representados no *framework* conceitual formal da seguinte forma:

- **Elemento Fonte:** *e_fonte?* - representa um único elemento do PIM;
- **Elemento Alvo:** *e_alvo!* - representa um único elemento do PSM*;*
- **Par Ordenado***:* representa o mapeamento entre o elemento fonte e o alvo;
- **SourceDomain:** representa o fragmento do metamodelo fonte, que consiste em um conjunto de elementos do PIM que serão transformados para o PSM;
- **TargetRange:** representa o fragmento do metamodelo alvo, que consiste em um conjunto de elementos do PSM que possui elementos fonte correspondentes no PIM.

Assim, verificamos que o megamodelo de Favre [3] não é completamente satisfatório para generalizar uma transformação em MDE, pois o termo função utilizado para representar uma transformação não é compatível quando a transformação apresentar os casos de mapeamento de elementos de um-para-muitos e muitos-para-muitos.

5.2 AXIOMA QUE GENERALIZA A TRANSFORMAÇÃO COMO UMA RELAÇÃO BINÁRIA

No *framework* da Figura 5.2, é mencionado que a relação binária usa um axioma genérico validado no Z/EVES para especificar a transformação como uma função bijetiva ou uma relação binária, este axioma é exibido na Figura 5.5.

$$
\begin{array}{|l}
T : DOM\_FONTE \leftrightarrow IMA\_ALVO \\
\hline
\mathrm{dom}\ T = DOM\_FONTE \\
\mathrm{ran}\ T = IMA\_ALVO \\
\land\ T \in DOM\_FONTE \leftrightarrow IMA\_ALVO \\
\land\ e\_fonte? \in DOM\_FONTE \\
\land\ e\_alvo! \in IMA\_ALVO \\
\land\ T\ applies\$\ to\ e\_fonte? \\
\forall\ T : DOM\_FONTE \leftrightarrow IMA\_ALVO;\ e\_fonte? : DOM\_FONTE;\ e\_alvo! : IMA\_ALVO \bullet \\
T\ applies\$\ to\ e\_fonte? \Rightarrow T(e\_fonte?) = e\_alvo! \lor (T\ applies\$\ to\ e\_fonte? \Rightarrow T(e\_fonte?) = e\_alvo! \\
\Leftrightarrow T^{\sim}\ applies\$\ to\ e\_alvo! \Rightarrow T^{\sim}(e\_alvo!) = e\_fonte?)
\end{array}
$$

Figura 5.5: Axioma que generaliza a transformação como uma Relação Binária

Este axioma não é baseado em nenhum trabalho existente na área, ele é o produto dos resultados alcançados nesta pesquisa. O axioma da Figura 5.5 foi criado a partir da análise dos resultados obtidos nos testes apresentados no estudo de caso do Capítulo 6, onde se especificou em linguagem Z três tipos de transformação de modelos: de um metamodelo

UML para um de JAVA; de um metamodelo UML para um de SQL e de um metamodelo UML para um de WSDL.

O objetivo com que criamos este axioma consiste em fornecer um predicado que seja admitido como verdadeiro para especificar as transformações como uma relação binária, quando a transformação apresentar os casos de mapeamento de elementos de um-para-um, de um-para-muitos, de muitos-para-um e de muitos-para-muitos.

Este axioma associado com a metodologia descrita no Capítulo 4, dá suporte para que se possa especificar em linguagem Z uma transformação do tipo unidirecional como uma relação binária e do tipo bidirecional como uma função bijetiva.

De acordo com este axioma, quando a transformação for do tipo bidirecional a relação binária pode ser considerada como uma função bijetiva, isto ocorre devido uma função bijetiva ser considerada como um tipo especial de relação binária.

No capítulo 6, este axioma é explicado com mais detalhes e também é apresentado um exemplo para comprovar que é mais adequado representar uma transformação de modelos como uma relação binária do que uma função.

5.3 RESUMO

Neste capítulo, apresentou-se o *framework* conceitual que formaliza o processo de transformação em MDE. As diferenças entre este *framework* conceitual e o megamodelo de Favre [3] foram reunidas em uma tabela (ver seção 5.1, pag. 78), a principal diferença é o termo utilizado para representar a transformação entre modelos. E, por fim, apresentou-se um axioma genérico que generaliza as transformações em MDE como uma relação binária da teoria dos conjuntos.

6 ESTUDO DE CASO

Neste capítulo, apresenta-se um estudo de caso para que se possa validar o *framework* conceitual proposto, bem como a veracidade do axioma que generaliza as transformações como uma relação binária.

Utiliza-se a ferramenta de prova formal Z/EVES [12] para checar e validar os esquemas e o axioma genérico que formalizam as transformações apresentadas neste capítulo.

6.1 APRESENTAÇÃO DO ESTUDO DE CASO

Este estudo de caso é constituído de três transformações de modelos: a primeira é feita de um metamodelo UML para um metamodelo JAVA, a segunda é uma transformação de um metamodelo UML para um metamodelo SQL e a terceira é uma transformação de um metamodelo UML para um metamodelo WSDL.

6.2 TRANSFORMAÇÃO DO METAMODELO UML PARA O METAMODELO JAVA

As transformações em MDE podem ser de dois tipos: bidirecionais ou unidirecionais. Quando se tratando das transformações bidirecionais, o mapeamento pode ser realizado em ambos os sentidos, do metamodelo fonte para o alvo e vice-versa.

A função bijetiva admite um mapeamento entre os elementos do domínio para o conjunto-imagem e vice-versa. Neste caso, por se tratar de uma função bijetiva, o conjunto-imagem é igual ao contra-domínio da função.

De acordo com essas características, pode-se representar a transformação bidirecional por uma função bijetiva em linguagem Z.

A fim de comprovar que a transformação bidirecional pode ser abstraída como uma função bijetiva, propõe-se realizar um estudo de caso com uma transformação do fragmento de metamodelo UML para um fragmento de metamodelo JAVA, envolvendo a tecnologia de objetos distribuídos do JAVA RMI.

Para esta transformação de modelos, estabeleceu-se como metamodelo fonte um metamodelo UML e como metamodelo alvo um metamodelo JAVA, conforme ilustra a Figura 6.1.

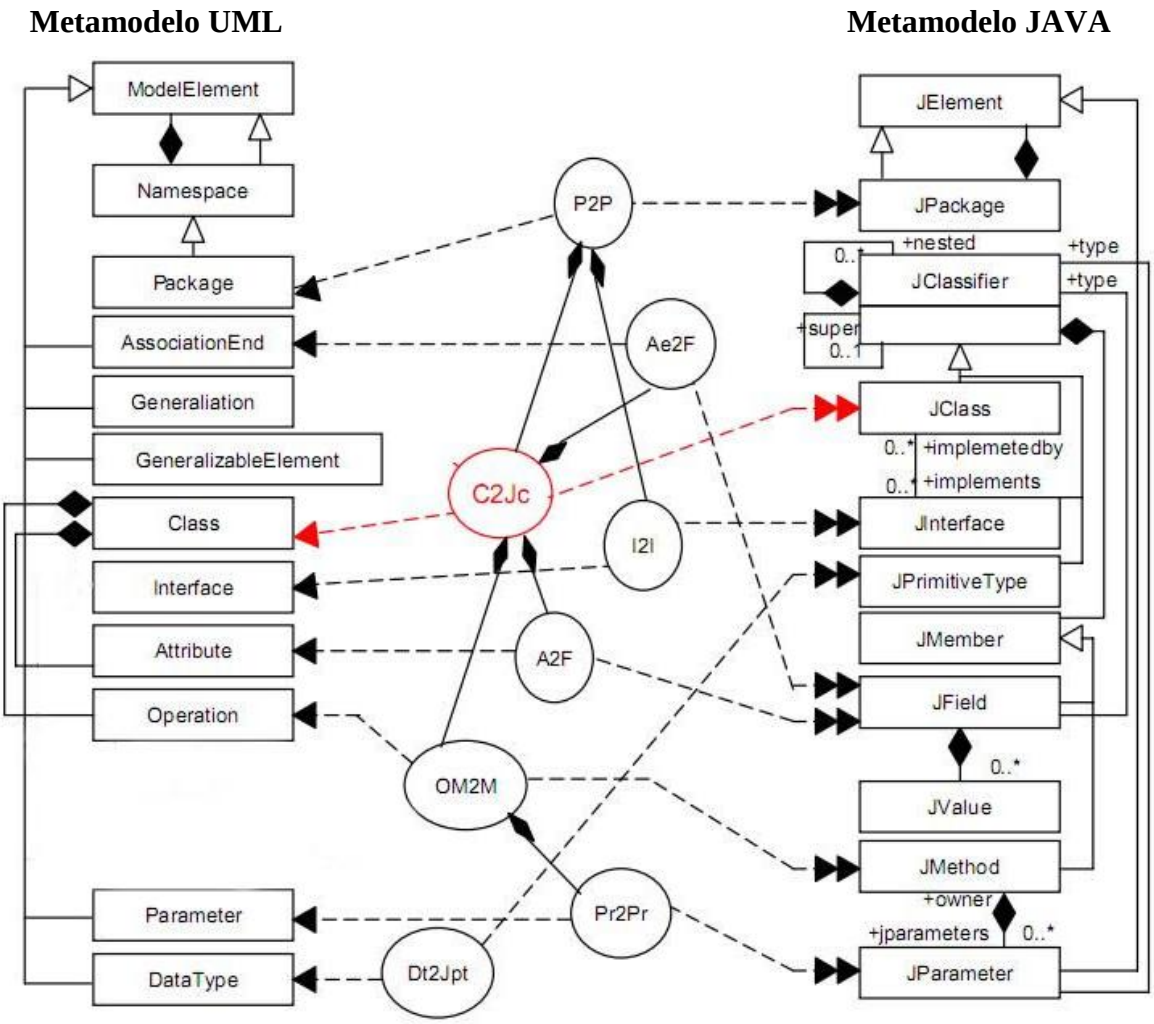

Figura 6.1: Transformação do metamodelo UML para o metamodelo JAVA [31]

Os possíveis mapeamentos existentes entre os elementos do metamodelo UML com os do metamodelo JAVA são ilustradas na Figura 6.1. No entanto, para este exemplo de transformação de modelos, não serão considerados todos os mapeamentos de elementos entre os metamodelos da Figura 6.1, apenas os necessários para se atingir os requisitos da aplicação que serão estabelecidos adiante.

Os elementos dos metamodelos que farão parte da transformação constituirão os fragmentos dos metamodelos UML e JAVA.

O estudo de caso envolvendo a tecnologia de objetos distribuídos do Java RMI realiza a troca de mensagens entre Cliente e Servido, como ilustra a Figura 6.2.

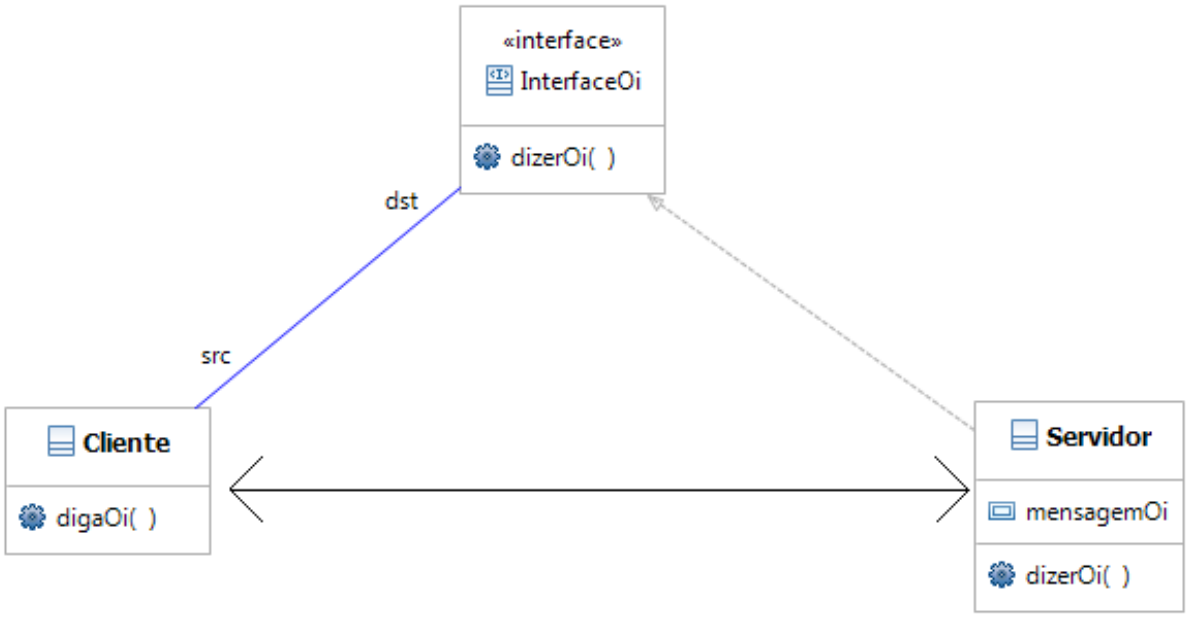

Figura 6.2: Estudo de Caso do processo de troca de mensagens

Neste tipo de aplicação, o cliente faz uma requisição ao servidor, e o servidor por sua vez implementa os métodos que estão definidos na interface e envia a resposta ao cliente.

Os requisitos desta aplicação são:

1. Definir uma interface remota que descreve como o cliente e o servidor se comunicam um com o outro;
2. Definir o aplicativo servidor que implementa a interface remota;
3. Definir o aplicativo cliente que utiliza uma referência de interface remota para interagir com o servidor (isto é, um objeto da classe que implementa a interface remota);
4. Compilar e executar o servidor e o cliente.

Em JAVA RMI, o papel da interface é apenas definir os métodos utilizados, não implementá-los, quem fica responsável por implementar os métodos é o servidor.

Para se atingir os requisitos desta aplicação, foram estabelecidos os mapeamentos entre os elementos dos metamodelos necessários para se atingir os requisitos estabelecidos, ver Tabela 6.1.

Tabela 6.1: Mapeamento entre os elementos do metamodelo UML com os do JAVA

UML	JAVA
Package?	*JPackage!*
Class?	*JClass!*
Interface?	*JInterface!*
Operation?	*JMethod!*
Parameter?	*JParameter!*
Atribute?	*JField!*
DataType?	*JPrimitiveType!*

Comparando-se a Figura 6.1 com a Tabela 6.1, observa-se que nem todos os elementos do metamodelo UML e do metamodelo JAVA estão envolvidos na transformação, isto é devido a sua seleção ocorrer de acordo com a necessidade de se atingir os requisitos da aplicação. Estes elementos constituem o fragmento de metamodelo UML e o fragmento de metamodelo JAVA.

Após esta etapa, analisou-se a transformação do fragmento de metamodelo UML para o fragmento de metamodelo JAVA por meio de Diagramas de Venn [33].

O domínio é chamado de DOM_UML, onde este é constituído dos elementos selecionados do metamodelo UML e o conjunto-imagem é IMA_JAVA (que neste caso tratando-se de uma função bijetiva, o conjunto-imagem é igual ao contra-domínio da função) sendo constituído dos elementos selecionados do metamodelo JAVA, como ilustra a Figura 6.3.

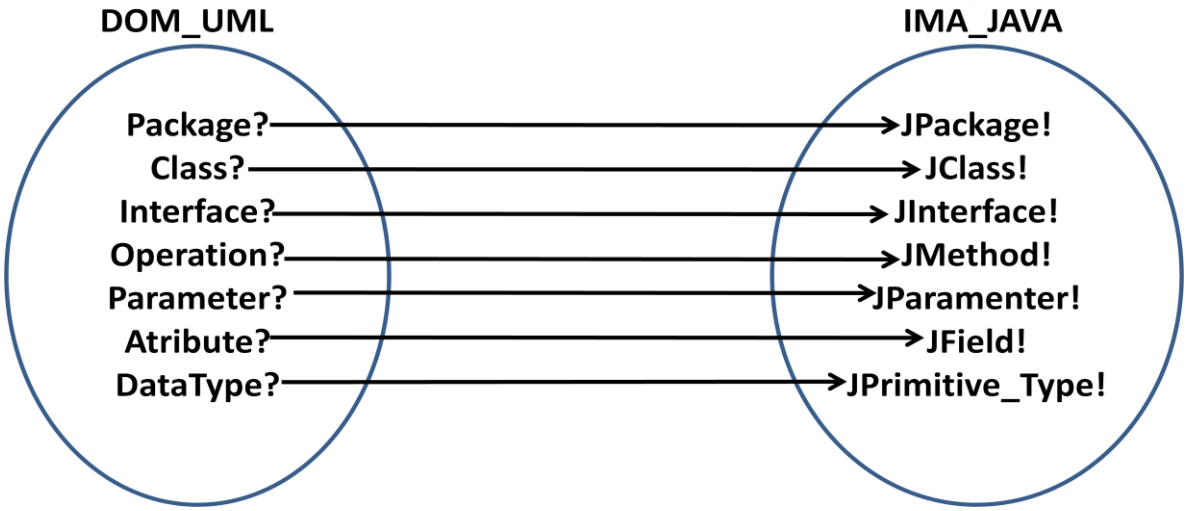

Figura 6.3: Mapeamento de UML para JAVA em Diagramas de Venn

Verifica-se através dos diagramas de Venn, que esta transformação de UML para JAVA é associada a uma função bijetiva (uma função só é bijetiva se admitir uma inversa), pois, pode-se obter uma função inversa no sentido do conjunto-imagem para o domínio da função (que em MDE é caracterizada como uma transformação reversa no sentido do metamodelo alvo para o fonte), como ilustrado na Figura 6.4.

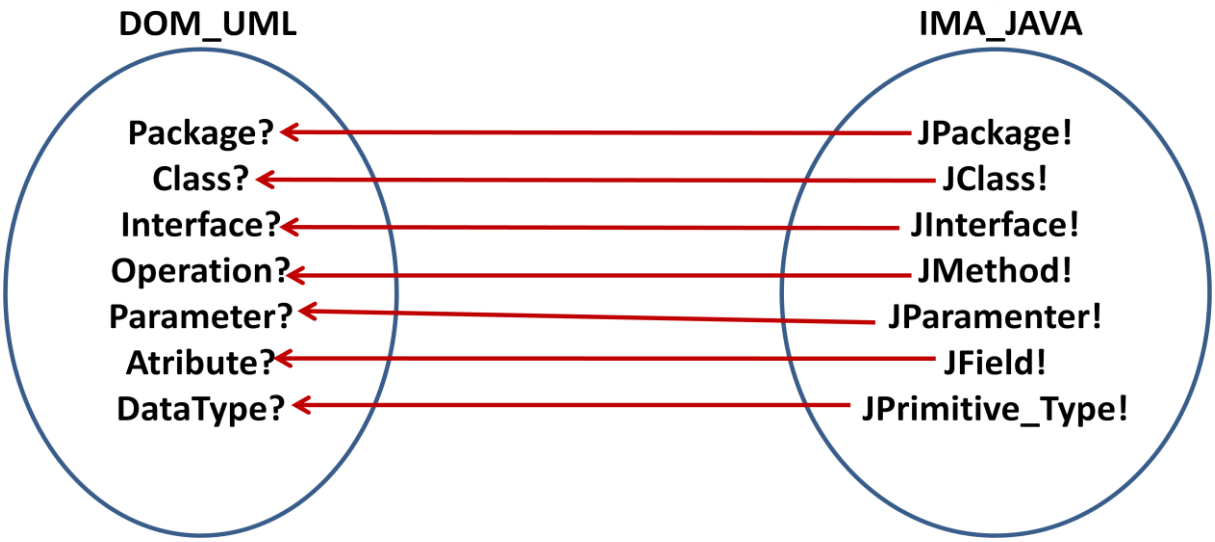

Figura 6.4: Transformação Reversa de JAVA para o UML

Logo se esta transformação do fragmento de metamodelo UML para o fragmento de metamodelo JAVA é bijetiva, então é do tipo bidirecional.

Assim na Hierarquia de Artefatos Matemáticos esta transformação é classificada como uma função bijetiva, conforme é indicado na Figura 6.5.

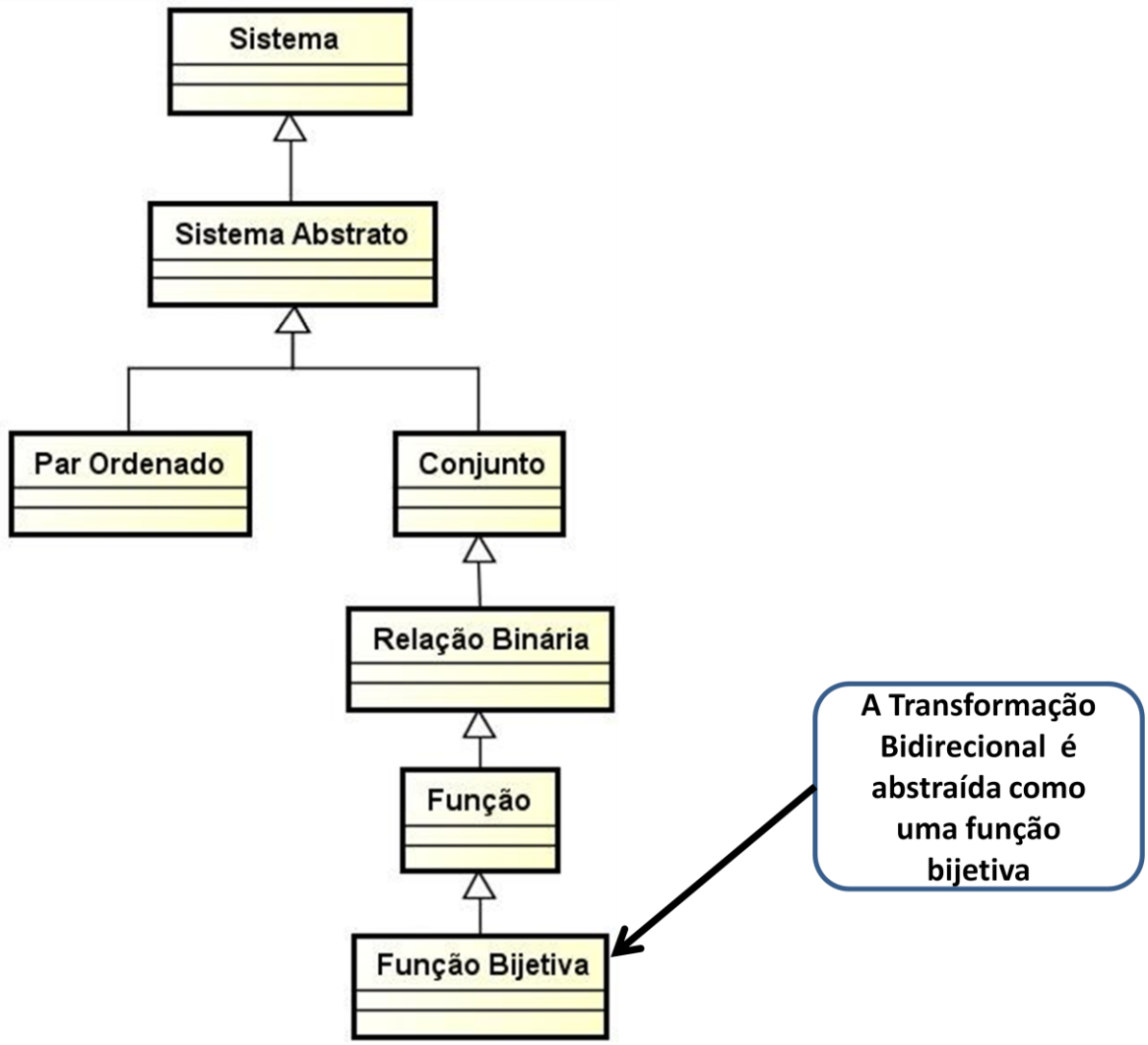

Figura 6.5: Função Bijetiva na Hierarquia de Artefatos Matemáticos

Esta transformação pode ser provada em especificações formais na linguagem Z através de esquemas validados no Z/EVES.

Antes de iniciar a especificação do esquema, é necessário fazer a declaração de tipos livres do domínio (representado por **DOM_UML)** e do conjunto-imagem (representado por **IMA_JAVA)** como ilustrado na Figura 6.6.

O domínio é constituído pelos elementos do metamodelo UML e o conjunto-imagem constituído pelos elementos do metamodelo JAVA. Os elementos são selecionados a partir dos metamodelos para formarem o domínio e o conjunto-imagem da transformação.

$$
\begin{array}{ll}
DOM\_UML ::= package? & IMA\_JAVA ::= jpackage! \\
\mid class? & \mid jclass! \\
\mid operation? & \mid jmethod! \\
\mid interface? & \mid jinterface! \\
\mid parameter? & \mid jparameter! \\
\mid atribute? & \mid jfield! \\
\mid data\_type? & \mid jprimitive\_type!
\end{array}
$$

Figura 6.6: Declaração de Tipos Livres do Domínio e do Conjunto-Imagem

Este passo é necessário ser feito antes de especificar os conjuntos **DOM_UML** e **IMA_JAVA,** porque os seus elementos deverão existir antes de sua declaração nos conjuntos **DOM_UML** e **IMA_JAVA**. No *framework* conceitual formal proposto no Capítulo 5, o **DOM_UML** é representado por ***SourceDomain*** e o **IMA_JAVA** é representado por ***TargetRange***.

Após ser declarado o tipo livre do domínio e do conjunto-imagem da transformação, são definidos os elementos do domínio e do conjunto-imagem entre chaves, ver Figura 6.7.

$$DOM\_UML = \{package?, class?, operation?, interface?, parameter?, atribute?, data\_type?\}$$

$$IMA\_JAVA = \{jpackage!, jclass!, jinterface!, jmethod!, jparameter!, jfield!, jprimitive\_type!\}$$

Figura 6.7: Definição dos elementos do Domínio e do Conjunto-Imagem

Em seguida, os pares ordenados são estabelecidos, ver Figura 6.8. Os pares ordenados representam os mapeamentos entre os elementos dos metamodelos fonte e do metamodelo alvo necessários para se atingir os requisitos da aplicação.

$$DOM\_UML \times IMA\_JAVA = \{(package?, jpackage!), (class?, jclass!), (interface?, jinterface!), (operation?, jmethod!), (parameter?, jparameter!), (atribute?, jfield!), (data\_type?, jprimitive\_type!)\}$$

Figura 6.8: Especificação dos Pares Ordenados de DOM_UMLxIMA_JAVA

Em seguida, é feita a prova formal dos mapeamentos entre os elementos dos metamodelos em um tipo de esquema denominado de "*Schema Box",* selecionado na janela de edição (*Mini Editor*) do Z/EVES, ver Figura 6.9.

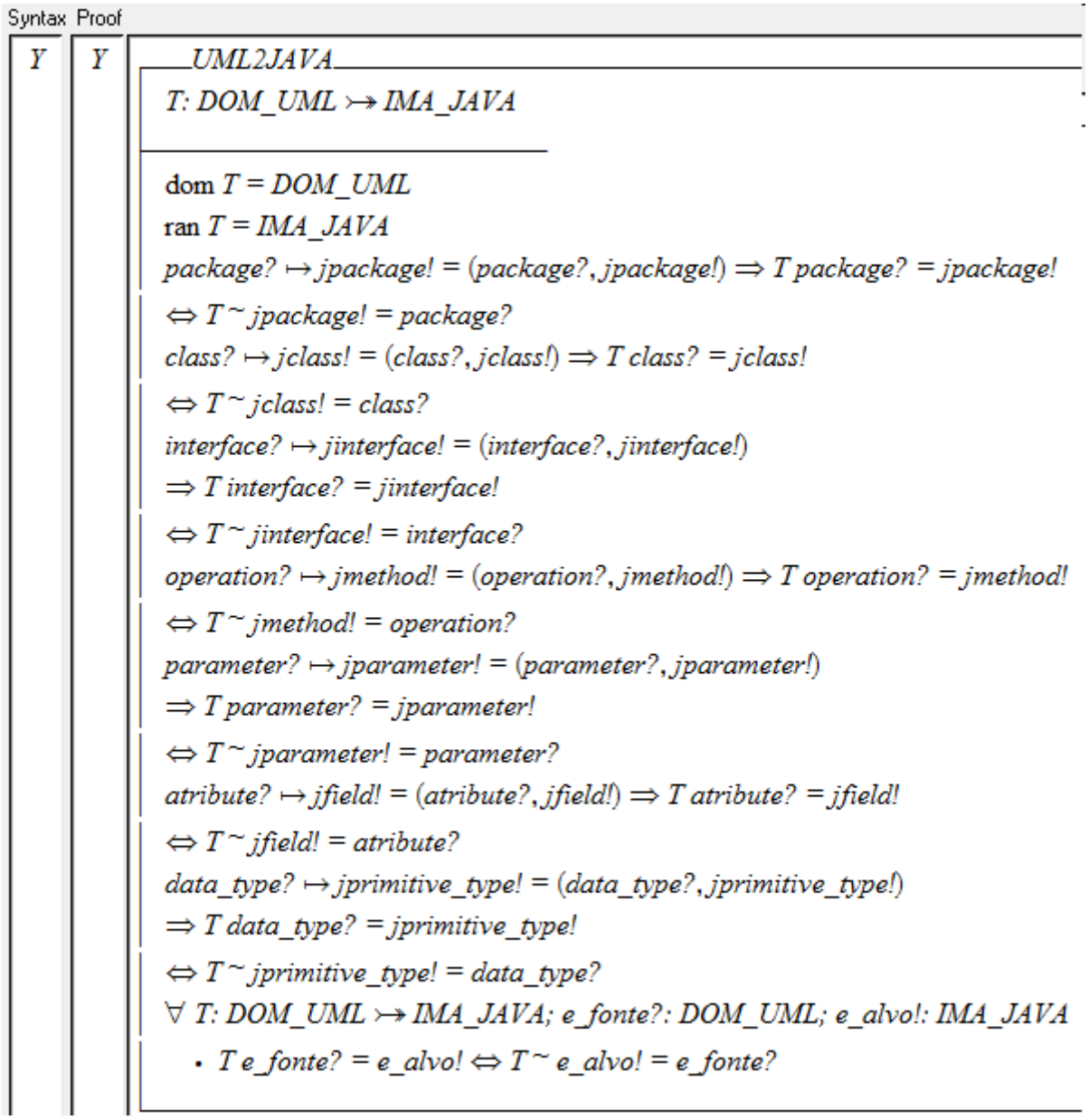

Figura 6.9: Transformação de UML para JAVA especificada no Z/EVES

Na Figura 6.9, a tela principal de especificação do Z/EVES é ilustrada contendo o esquema que valida a transformação do fragmento de metamodelo UML para o de JAVA como uma função bijetiva. Nas duas colunas do lado esquerdo da tela é exibido o resultado da checagem do esquema, a letra Y para a sintaxe (*Syntax*) e para a prova (*Proof*) indica que o predicado não apresenta erros na sintaxe e nem inconsistências na prova.

Na Figura 6.10, o esquema da Figura 6.9 é apresentado no formato LATEX. O formato LATEX é uma forma mais organizada de escrever o esquema da Figura 6.9.

$$
\begin{array}{|l}
\textit{UML2JAVA} \\
T : DOM\_UML \rightarrowtail IMA\_JAVA \\
\hline
\text{dom } T = DOM\_UML \\
\text{ran } T = IMA\_JAVA \\
\\
package? \mapsto jpackage! = (package?, jpackage!) \Rightarrow T(package?) = jpackage! \\
\Leftrightarrow T^{\sim}(jpackage!) = package? \\
\\
class? \mapsto jclass! = (class?, jclass!) \Rightarrow T(class?) = jclass! \\
\Leftrightarrow T^{\sim}(jclass!) = class? \\
\\
interface? \mapsto jinterface! = (interface?, jinterface!) \Rightarrow T(interface?) = jinterface! \\
\Leftrightarrow T^{\sim}(jinterface!) = interface? \\
\\
operation? \mapsto jmethod! = (operation?, jmethod!) \Rightarrow T(operation?) = jmethod! \\
\Leftrightarrow T^{\sim}(jmethod!) = operation? \\
\\
parameter? \mapsto jparameter! = (parameter?, jparameter!) \Rightarrow T(parameter?) = jparameter! \\
\Leftrightarrow T^{\sim}(jparameter!) = parameter? \\
\\
atribute? \mapsto jfield! = (atribute?, jfield!) \Rightarrow T(atribute?) = jfield! \\
\Leftrightarrow T^{\sim}(jfield!) = atribute? \\
\\
data\_type? \mapsto jprimitive\_type! = (data\_type?, jprimitive\_type!) \Rightarrow T(data\_type?) = jprimitive\_type! \\
\Leftrightarrow T^{\sim}(jprimitive\_type!) = data\_type? \\
\\
\forall T : DOM\_UML \rightarrowtail IMA\_JAVA;\ e\_fonte? : DOM\_UML;\ e\_alvo! : IMA\_JAVA \bullet \\
T(e\_fonte?) = e\_alvo! \Leftrightarrow T^{\sim}(e\_alvo!) = e\_fonte? \\
\hline
\end{array}
$$

Figura 6.10: Esquema da transformação de UML para JAVA (formato LATEX)

De acordo com o esquema da Figura 6.10, tem-se que a transformação do fragmento de metamodelo UML para o de JAVA é validada em Z.

A explicação do esquema da Figura 6.10 é dada da seguinte forma: observa-se que acima da linha divisória tem-se a declaração da transformação *T* como uma função bijetiva, representada como $DOM\_UML \rightarrowtail IMA\_JAVA$, o símbolo $\rightarrowtail$ representa uma bijetividade entre conjuntos na linguagem Z.

Abaixo da linha divisória, é descrito o predicado que garante que a transformação do fragmento de UML para o de JAVA é bidirecional e pode ser associada a uma função bijetiva.

Neste predicado tem-se:

- *dom T* = DOM_UML: especificando que o domínio da transformação é DOM_UML;
- *ram T* = IMA_JAVA: especificando que o conjunto-imagem da transformação é IMA_JAVA.

Em seguida vêm à especificação dos relacionamentos entre os elementos fonte e alvo, cada relacionamento é representado pelo símbolo $\mapsto$ que indica um mapeamento entre esses elementos, que também pode ser representado em Z como um par ordenado, por exemplo:

$$package? \mapsto jpackage! = (package?, jpackage!)$$

representa o mapeamento do elemento de entrada *package?* para o elemento de saída *jpackage!* que pode ser representado em Z como o par ordenado (*package?, jpackage!*).

Em seguida vem o símbolo $\Rightarrow$ (implicação) implicando que a transformação *T(package?) = jpackage!*, toma um elemento *package*? ∈ DOM_UML e transforma em um elemento *jpackage*! ∈ IMA_JAVA se, e somente se ($\Leftrightarrow$) a transformação reversa $T^{\sim}$ realizar a seguinte transformação reversa: $T^{\sim}$ *(jpackage!) = package?* que toma um elemento *jpackage*! ∈ IMA_JAVA e transforma em um elemento *package*? ∈ DOM_UML, pois a transformação *T* é associada a uma função bijetiva em linguagem Z, o que garante existir uma função inversa que em MDE é a transformação reversa $T^{\sim}$.

E assim, todos os outros mapeamentos são especificados de acordo com este predicado.

Por fim, na última linha do predicado, tem-se a generalização desta prova, formalizada como segue:

Para toda e qualquer que seja ($\forall$) a transformação *T* do tipo bijetiva ($\rightarrowtail\!\!\!\!\!\twoheadrightarrow$) do DOM_UML para o conjunto IMA_JAVA, e qualquer o elemento fonte do tipo DOM_UML (*e_fonte?:*DOM_UML) e qualquer elemento alvo do tipo IMA_JAVA (*e_alvo!:* IMA_JAVA) a função bijetiva *T* transforma um elemento fonte *(e_fonte?)* em um elemento alvo *(e_alvo!)* se, e somente se ($\Leftrightarrow$) existir uma função inversa que abstrai formalmente a transformação reversa $T^{\sim}$, que transforma um elemento alvo (*e_alvo!)* em um elemento fonte (*e_fonte?*).

De acordo com esta prova formal, a transformação do fragmento de metamodelo UML para o fragmento de metamodelo JAVA deve ser associada a uma função bijetiva, quando for uma transformação bidirecional.

Porém, caso fossem considerados todos os mapeamentos entre os elementos do metamodelo UML com o de JAVA, a transformação não teria características de uma função bijetiva, mas sim de uma relação binária.

Assim, com esta prova verificamos que o modelo (ou megamodelo) de Favre [3] não está totalmente coerente ao generalizar a transformação unicamente como uma função,

uma vez que a transformação pode ser bidirecional e assim possuir características de um tipo específico de função, denominada de função bijetiva.

Além de que, se forem considerados todos os elementos do metamodelo UML e todos os elementos do metamodelo JAVA a transformação não irar possuir características de uma função, mas sim de uma relação binária, contrariando o megamodelo de Favre [3] que generaliza todas as transformações como uma função matemática.

6.3 TRANSFORMAÇÃO DO METAMODELO UML PARA O METAMODELO SQL

A transformação unidirecional é feita em uma única direção (fonte-alvo), com isso não é adequado representar este tipo de transformação como uma função bijetiva. Pois uma função bijetiva permite um mapeamento do elemento fonte para o alvo e vice-versa.

Também não é adequado utilizar nenhum outro tipo de função para representar a transformação unidirecional, pois este tipo de transformação estabelece casos de mapeamento de elementos que contrariam a definição de função, são os casos de mapeamento de um-para-muitos e de muitos-para-muitos.

Para representar a transformação unidirecional o artefato matemático mais adequado é a relação binária, pois este artefato matemático possui um conceito mais abrangente do que o das funções. Conforme ilustra a hierarquia da Figura 6.11, a *função* herda de *relação binária.*

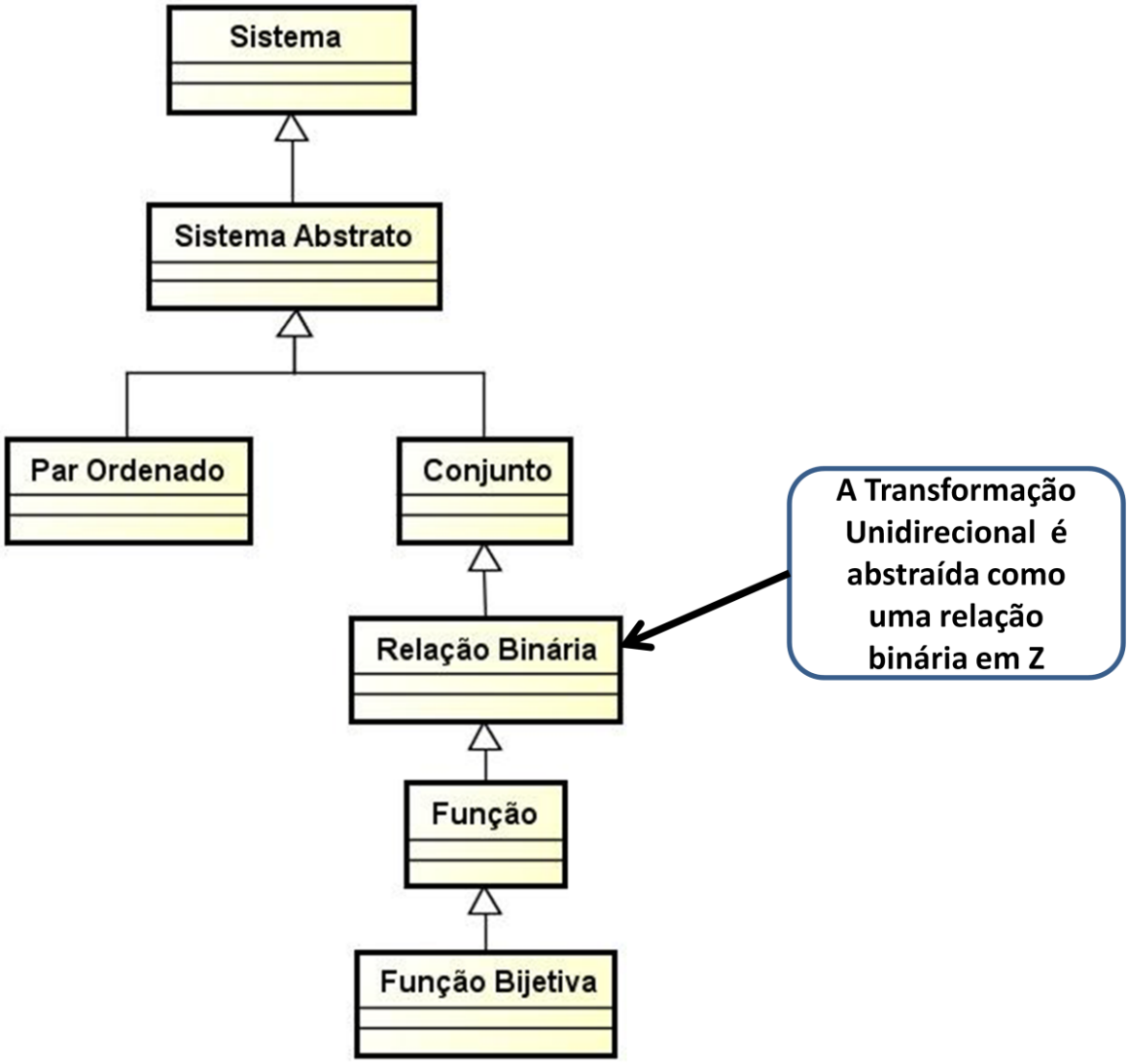

Figura 6.11: Relação Binária na Hierarquia de Artefatos Matemáticos

A relação binária realiza mapeamentos dos tipos: um-para-um, um-para-muitos, muitos-para-um e muitos-para-muitos. Os exemplos da Figura 6.12 e da Figura 6.13 descrevem estes casos de mapeamento através de diagramas de Venn [33]. Na Figura 6.12 e Figura 6.13, nota-se a diferença entre **função, função bijetora e relação binária**.

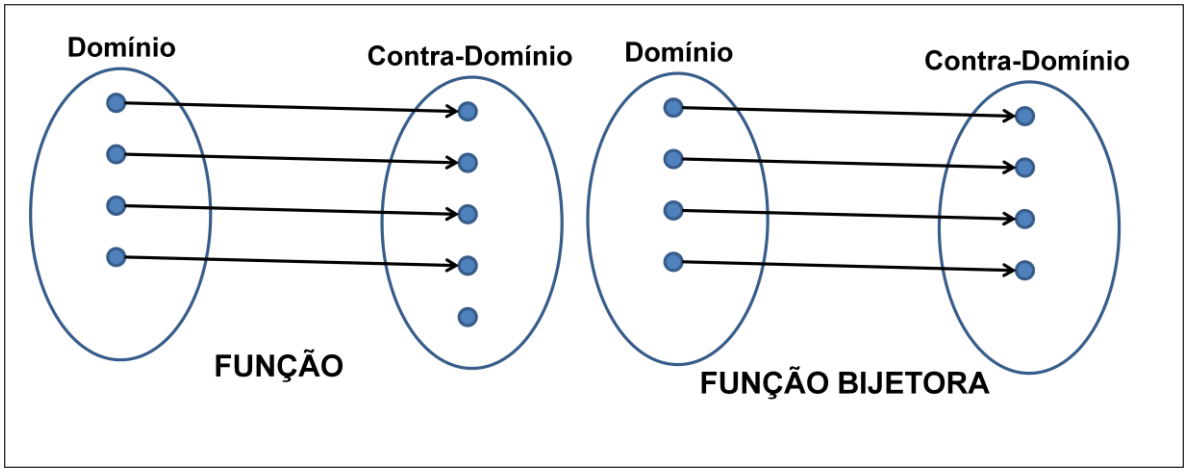

Figura 6.12: Função e Função Bijetora

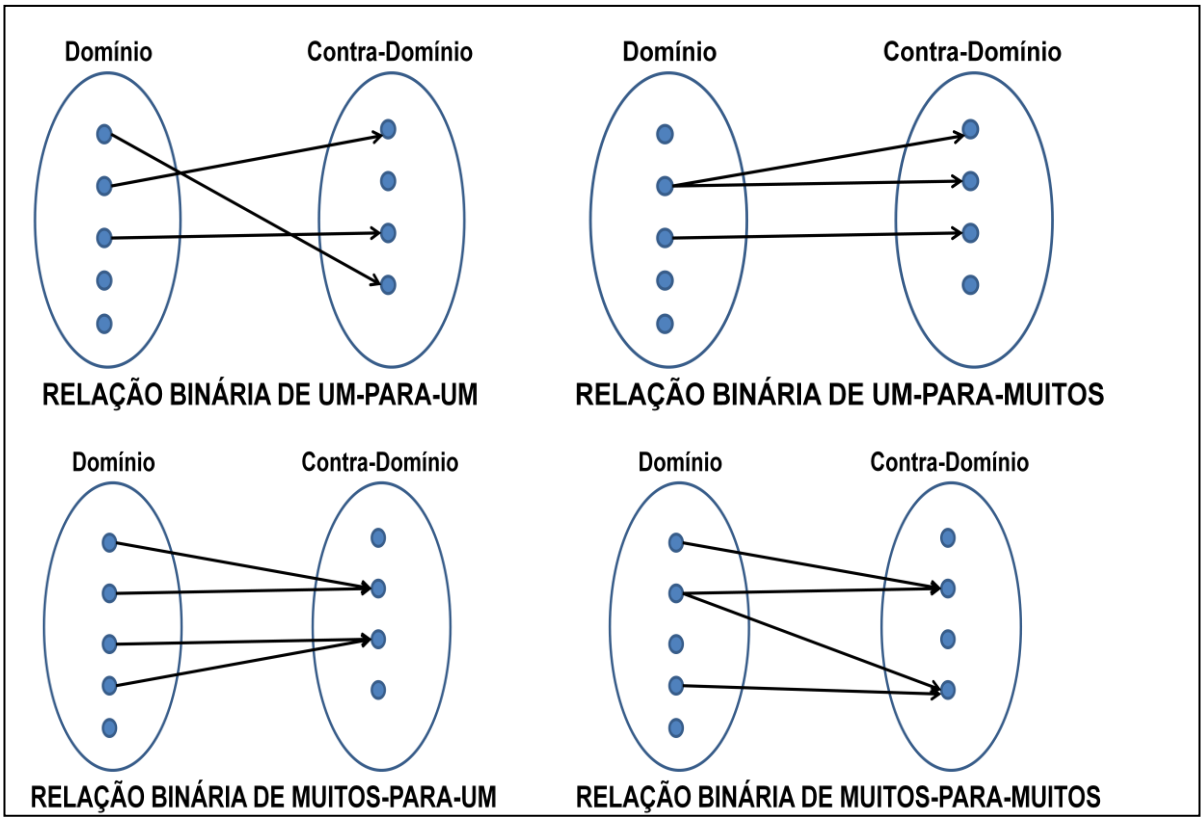

Figura 6.13: Casos de mapeamento suportados pela Relação Binária

Para a relação binária, os casos de associações entre os conjuntos estão de acordo com os casos de mapeamento de um-para-um, de um-para-muitos, de muitos-para-um e de muitos-para-muitos.

Nesta segunda transformação, propõe-se provar que a transformação unidirecional contraria a definição de função quando realiza um mapeamento de elementos de um-para-muitos. Para se alcançar esse objetivo, uma transformação de um fragmento de metamodelo

UML (fonte) para um de SQL (alvo) foi feita através de um estudo de caso de um banco de dados.

Na Figura 6.14, o estudo de caso trivial desta aplicação está representado e consiste de duas classes, uma chamada *Family*, representando uma família, e a outra, *Person*, representando as pessoas que são membros de uma família. As classes que compõem o modelo do estudo de caso do PIM apresentam atributos e relacionamentos.

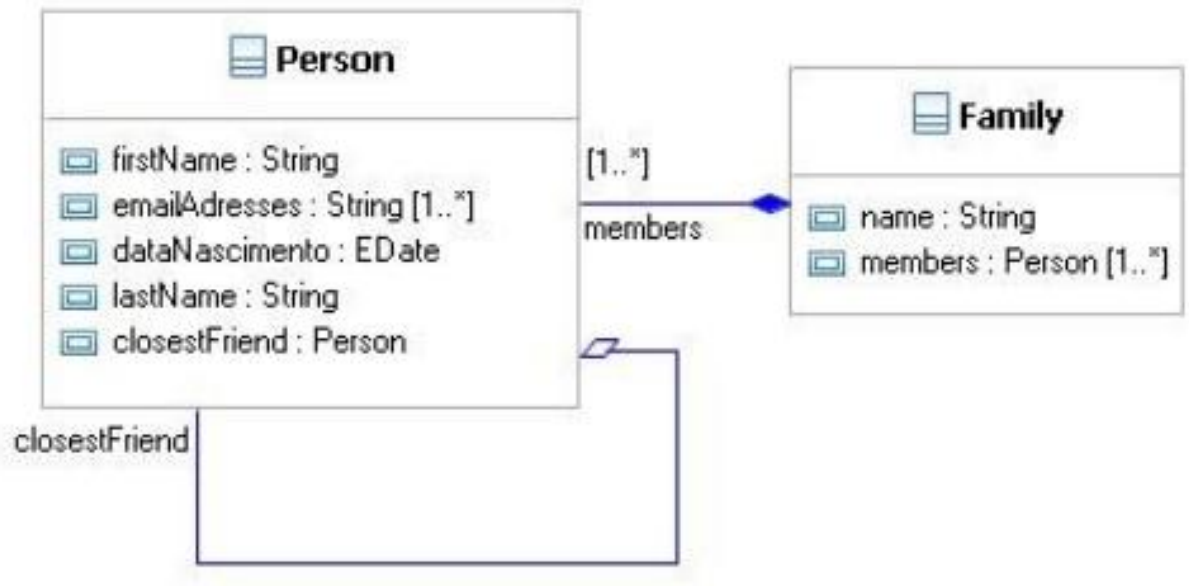

Figura 6.14: Estudo de Caso

Para realizar esta transformação, duas classes foram consideradas do metamodelo da linguagem SQL (ver a Figura 6.15,), uma representando a tabela e outra representando a coluna, sendo que a tabela pode conter várias colunas (*col*), várias chaves estrangeiras (*fkey*). Uma chave primária (*key*) pode ser representada por uma ou várias colunas.

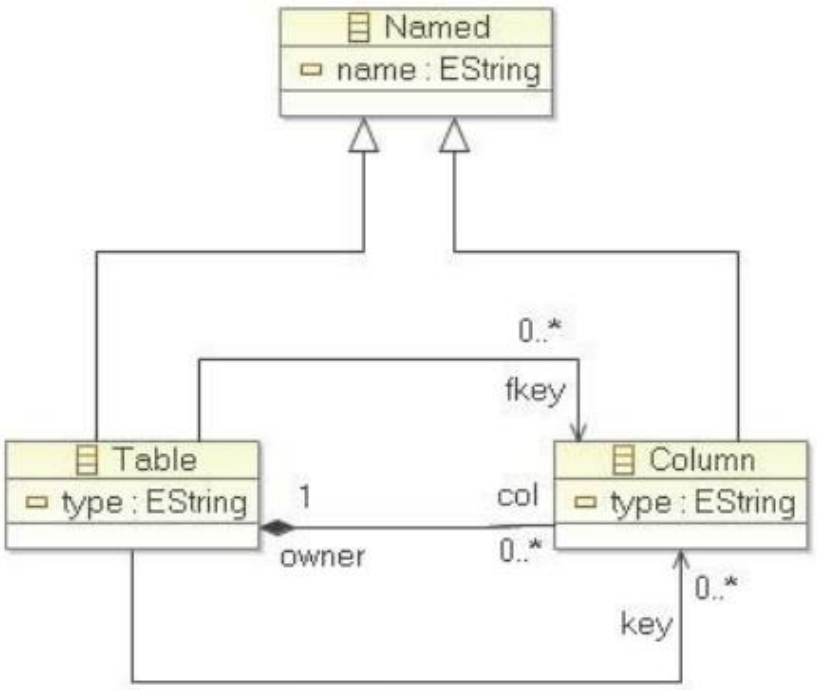

Figura 6.15: Metamodelo simplificado de SQL

Uma vez que as classes apresentam atributos **simples** e **multivalorados**, com tipos *DataType* e *Class*, uma regra para o mapeamento de Classe (*Class?*) para Tabela (*Table!*) foi necessário e 4 regras para tratar o mapeamento de atributos (*Property?*) para colunas (*Column!*), ver Tabela 6.2.

Tabela 6.2: Mapeamento entre os elementos do metamodelo UML com os de SQL

METAMODELO UML	METAMODELO SQL
Class?	*Table!*
Single_ValuedData_TypeProperty?	*Column!*
Multi_ValuedData_TypeProperty?	*Column!*
Class_Property?	*Column!*
Multi_ValuedClass_Property?	*Column!*

Na Tabela 6.2, tem-se o primeiro mapeamento entre uma Classe (*Class?*) em UML para uma Tabela (*Table!*) em SQL. O segundo mapeamento é entre uma Propriedade

Simples do tipo *DataType (Single_ValuedData_TypeProperty?)* para uma coluna (*Column!).* O terceiro mapeamento é entre uma Propriedade Multivalorada do tipo *DataType* (*Multi_ValuedData_TypeProperty?)* para uma coluna (*Column!).*

O quarto mapeamento é entre uma Propriedade do tipo *Class (Class_Property?),* para uma coluna (*Column!).* O quinto mapeamento é entre uma Propriedade do tipo *Class* e Multivalorada *(Multi_ValuedClass_Property?)* para uma Coluna (*Column!*).

Após esta etapa, analisa-se a transformação de UML para SQL por meio de Diagramas de Venn, criando um conjunto constituído dos elementos selecionados do metamodelo UML, denominado de DOM_UML, um outro conjunto constituído dos elementos selecionados do metamodelo SQL, denominado de IMA_SQL, como ilustra a Figura 6.16.

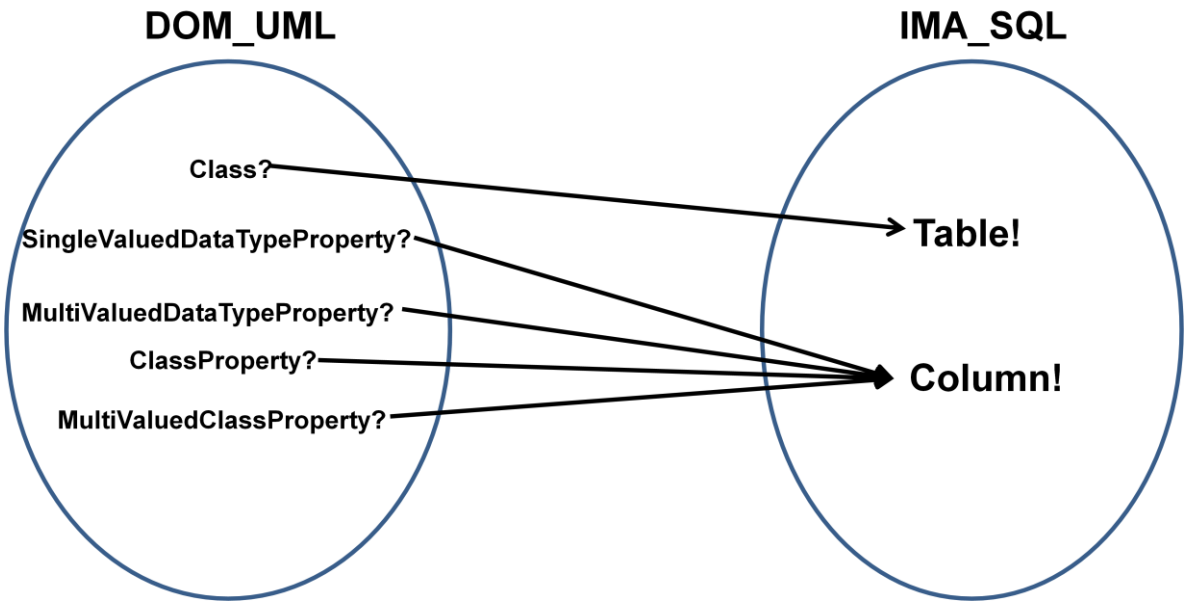

Figura 6.16: Mapeamento de UML para SQL em Diagramas de Venn

Através dos diagramas de Venn, concluímos que a transformação não pode ser representada por uma função bijetiva em Z.

Pois, suponha que a transformação do fragmento de metamodelo UML para o de SQL seja bidirecional, de acordo com a Figura 6.17 a transformação reversa $T^{\sim}$ partindo do conjunto IMA_SQL para o conjunto DOM_UML associa o elemento *Column* a mais de um elemento no domínio (fonte), isto contraria o conceito de função na teoria dos conjuntos,

onde: “um objeto no conjunto de partida deve estar associado a um e somente um objeto no conjunto de chegada” [19].

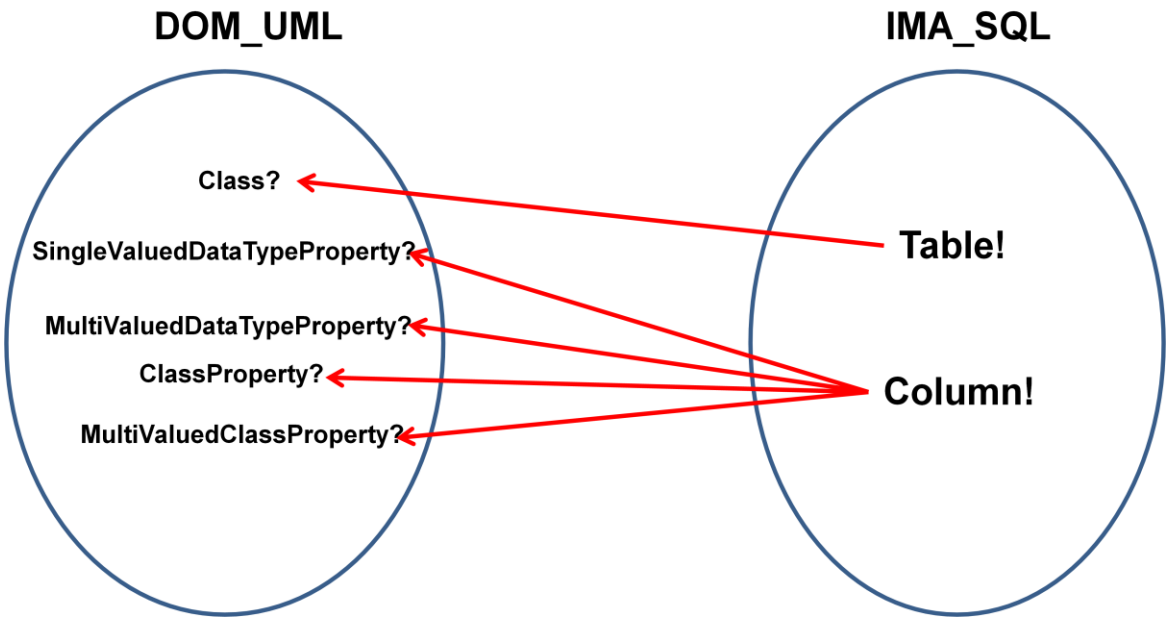

Figura 6.17: Transformação Reversa de SQL para UML

Esta contradição é identificada quando se especifica no Z/EVES a transformação do fragmento de metamodelo UML para o fragmento de SQL como uma função, tomando como referência a orientação do megamodelo de Favre [3] que representa a transformação como uma função.

Antes de iniciar a especificação do esquema, é necessário fazer a declaração de tipos livres do domínio (representado por **DOM_UML)** e do conjunto-imagem (representado por **IMA_SQL**) da função, ver Figura 6.18, bem como definir os elementos dos conjuntos DOM_UML e IMA_SQL entre chaves, ver Figura 6.19.

$$DOM\_UML ::= Class? \mid SingleValuedDataTypeProperty? \mid MultiValuedDataTypeProperty? \mid ClassProperty? \mid MultiValuedClassProperty?$$

$$IMA\_SQL ::= Table! \mid Column!$$

Figura 6.18: Tipos livres do DOM_UML e IMA_SQL

$$DOM\_UML = \{Class?, SingleValuedDataTypeProperty?, MultiValuedDataTypeProperty?, ClassProperty?, MultiValuedClassProperty?\}$$

$$IMA\_SQL = \{Table!, Column!\}$$

Figura 6.19: Descrição dos conjuntos DOM_UML e IMA_SQL

Os elementos são selecionados para os conjuntos **DOM_UML** e **IMA_SQL** conforme os requisitos da aplicação, estes conjuntos representam respectivamente o fragmento de metamodelo UML e do fragmento de metamodelo SQL. No *framework* conceitual proposto, os conjuntos **DOM_UML** e **IMA_SQL** são respectivamente os conjuntos ***SourceDomain*** e ***TargetRange***, ver Figura 5.2 no Capítulo 5.

Em seguida, os pares ordenados são estabelecidos e ilustrados na Figura 6.20. Estes pares ordenados representam os mapeamentos existentes entre os elementos do metamodelo fonte UML com os elementos do metamodelo alvo SQL.

$$DOM\_UML \times IMA\_SQL = \{(Class?, Table!), (SingleValuedDataTypeProperty?, Column!), (MultiValuedDataTypeProperty?, Column!), (ClassProperty?, Column!), (MultiValuedClassProperty?, Column!)\}$$

Figura 6.20: Descrição dos pares ordenados de DOM_UMLxIMA_SQL

E por fim, é feita a prova matemática dos mapeamentos entre os elementos dos metamodelos em um esquema denominado de "*Schema Box"*, definido na janela de edição (*Mini Editor*) do Z/EVES.

Para provar que uma transformação do fragmento de metamodelo UML para o fragmento de metamodelo SQL não pode ser especificada como uma função bijetiva, validam-se no Z/EVES os mapeamentos como se fosse uma função bijetiva e checa-se o parágrafo na janela principal do Z/EVES, conforme é ilustrado na Figura 6.21.

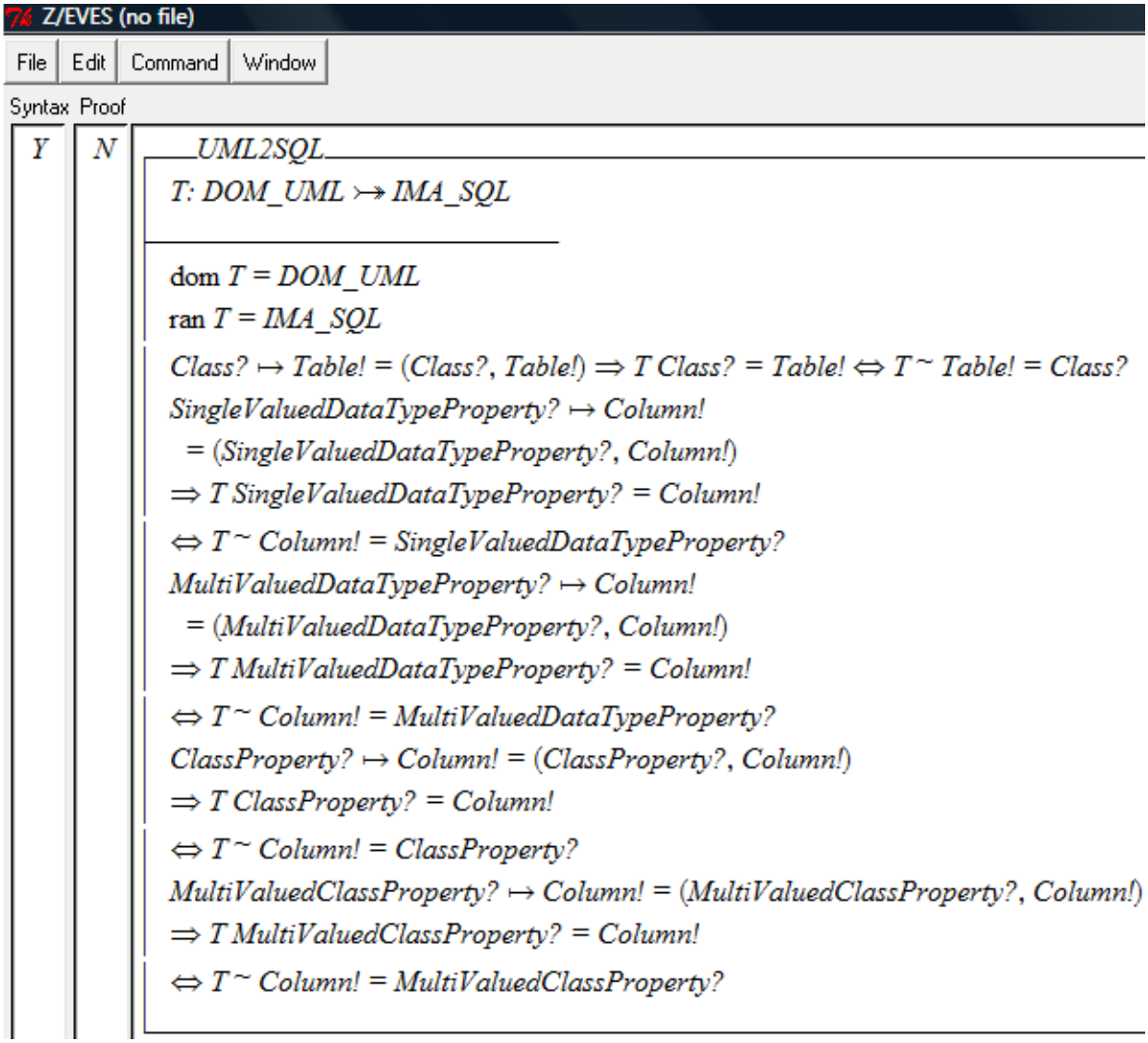

Figura 6.21: O Z/EVES identifica inconsistência na transformação de UML para SQL

O predicado abaixo da linha divisória do esquema da Figura 6.21, é o mesmo utilizado no esquema da Figura 6.9, que validou a transformação do fragmento de metamodelo UML para o de JAVA como uma função bijetiva.

Verifica-se, conforme a Figura 6.21, que na janela principal de especificação do Z/EVES, a prova formal do predicado apresenta um erro expressado pela letra N no lado esquerdo da Figura 6.21.

A letra N indica que existe uma inconsistência na prova do predicado, devido a transformação de UML para SQL ter sido especificada como se fosse uma função bijetiva, conforme indica o símbolo $\rightarrowtail\!\!\!\!\twoheadrightarrow$, que representa uma bijetividade em Z .

Ao se especificar a transformação de UML para SQL como uma função bijetiva, estaria se admitindo que existe uma transformação reversa $T^{\sim}$ do conjunto IMA_SQL para o conjunto DOM_UML, então o elemento *Colunm!* do conjunto IMA_ALVO estaria associado

a mais de um elemento no conjunto DOM_UML pela transformação reversa $T^{\sim}$ que realiza o retorno, conforme está descrito em seguida:

- $T^{\sim}$ *Column!* = *SingleValuedDataType?*
- $T^{\sim}$ *Column!* = *MuliValuedDataTypeProperty?*
- $T^{\sim}$ *Column!* = *ClassProperty?*
- $T^{\sim}$ *Column!* = *MultiValuedClassPrperty?*

Estes quatro mapeamentos contrariam a definição de função, pois eles fazem parte do caso de mapeamento de um-para-muitos. A inconsistência é identificada no retorno da transformação do conjunto IMA_SQL para o conjunto DOM_UML, representada por $T^{\sim}$. Esta transformação reversa não é uma função inversa de IMA_SQL para DOM_UML, pois suas características contrariam a definição de função.

Para resolver esta contradição, deve-se especificar a transformação de UML para SQL como uma *relação binária*. Isto é alcançado ao se substituir no Z/EVES a função bijetiva pela *relação binária* e estabelecer o predicado que garante mapear os elementos do fragmento de metamodelo UML para os de SQL como uma *relação binária*.

Na Figura 6.22, tem-se a prova do esquema que especifica a transformação do conjunto DOM_UML para o conjunto IMA_SQL como uma relação binária entre seus elementos.

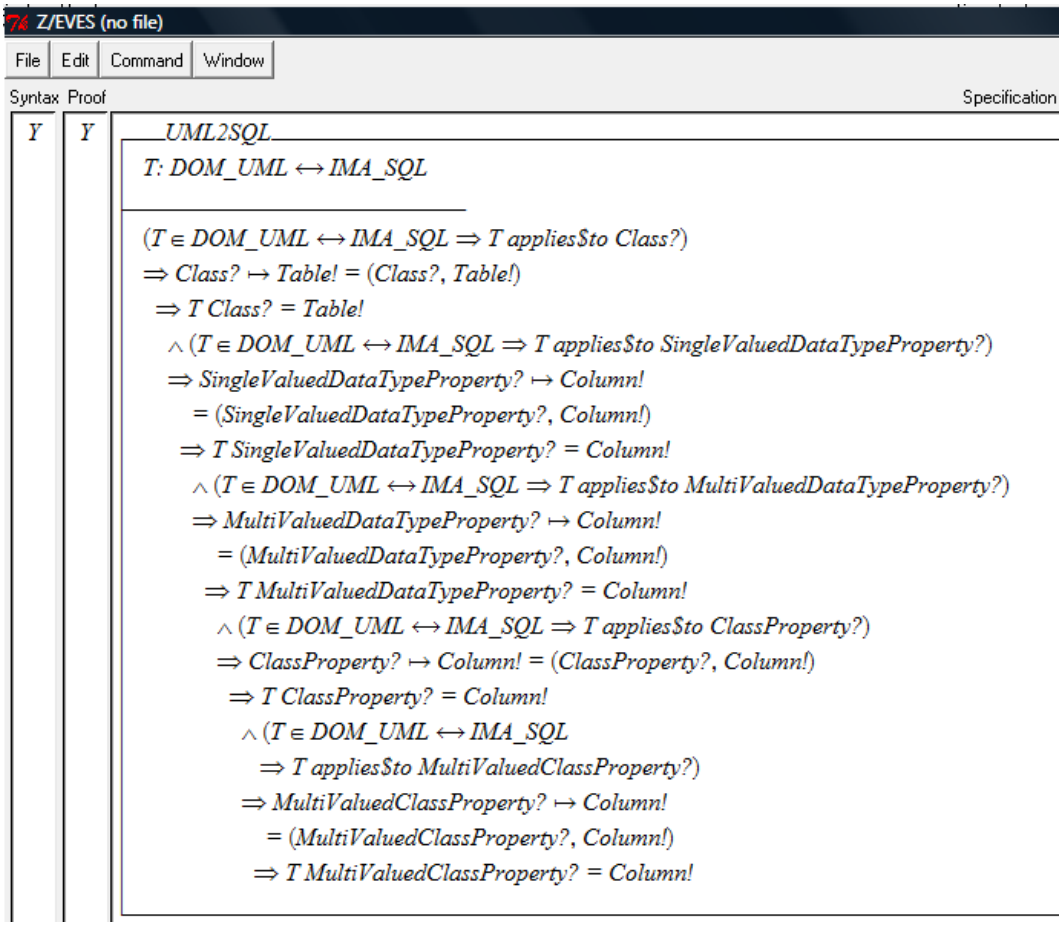

Figura 6.22: Esquema da Transformação de UML para SQL validada no Z/EVES

Ao especificar a transformação de UML para SQL como uma relação binária, não é obrigatório estabelecer no predicado da Figura 6.22 uma transformação reversa $T^{\sim}$ do conjunto IMA_SQL para o DOM_UML, pois não se trata de uma função bijetiva, com isso elimina-se a contradição identificado nesta transformação.

A validação é realizada com sucesso no Z/EVES, onde pode-se verificar a exibição da letra Y para a sintaxe (*Syntax*) e para a prova (*Proof*) nas duas colunas à esquerda da Figura 6.22.

Na Figura 6.23, o predicado do esquema da Figura 6.22 é apresentado de forma mais organizada no formato LATEX.

$$
\begin{array}{|l}
\textit{UML2SQL} \\
T : DOM\_UML \leftrightarrow IMA\_SQL \\
\hline
(T \in DOM\_UML \leftrightarrow IMA\_SQL \Rightarrow T\ applies\$\ to\ Class?) \\
\Rightarrow Class? \mapsto Table! = (Class?, Table!) \\
\Rightarrow T(Class?) = Table! \\
\\
\wedge\ (T \in DOM\_UML \leftrightarrow IMA\_SQL \Rightarrow T\ applies\$\ to\ SingleValuedDataTypeProperty?) \\
\Rightarrow SingleValuedDataTypeProperty? \mapsto Column! = (SingleValuedDataTypeProperty?, Column!) \\
\Rightarrow T(SingleValuedDataTypeProperty?) = Column! \\
\\
\wedge\ (T \in DOM\_UML \leftrightarrow IMA\_SQL \Rightarrow T\ applies\$\ to\ MultiValuedDataTypeProperty?) \\
\Rightarrow MultiValuedDataTypeProperty? \mapsto Column! = (MultiValuedDataTypeProperty?, Column!) \\
\Rightarrow T(MultiValuedDataTypeProperty?) = Column! \\
\\
\wedge\ (T \in DOM\_UML \leftrightarrow IMA\_SQL \Rightarrow T\ applies\$to\ ClassProperty?) \\
\Rightarrow ClassProperty? \mapsto Column! = (ClassProperty?, Column!) \\
\Rightarrow T(ClassProperty?) = Column! \\
\\
\wedge\ (T \in DOM\_UML \leftrightarrow IMA\_SQL \Rightarrow T\ applies\$to\ MultiValuedClassProperty?) \\
\Rightarrow MultiValuedClassProperty? \mapsto Column! = (MultiValuedClassProperty?, Column!) \\
\Rightarrow T(MultiValuedClassProperty?) = Column! \\
\hline
\end{array}
$$

Figura 6.23: Transformação de UML para SQL (formato LATEX)

A explicação do predicado do esquema da Figura 6.23 é dada da seguinte forma: na primeira linha do predicado, tem-se *T* pertencente (∈) a relação binária (↔) entre *DOM_UML* e *IMA_SQL*. Isso implica (⇒) que *T* aplica-se (*applies$to*) ao elemento *Class?* o que implica (⇒) em um mapeamento (representado por $Class? \mapsto Table!$) para o elemento *Table*! pertencente (∈) ao conjunto *IMA_SQL*. Este mapeamento é igual ao par ordenado *(Class?,Table!),* implicando (⇒) na transformação do elemento fonte *Class?* no elemento alvo *Table! (T(Class?) = Table!)* através da transformação *T* que é uma relação binária em Z.

Esta mesma regra é aplicada para os elementos restantes, conforme o símbolo (∧) que indica "*e*" dando continuidade na realização das transformações resultando nas seguintes transformações de elementos fonte em elementos alvo: *T(Single_ValuedData_TypeProperty?) = Column!, T(Multi_ValuedData_TypeProperty?) = Column!, T(Class_Property?)=Column!* e *T(Multi_ValuedClass_Property?) = Column!*.

Após estabelecer estes mapeamentos, checa-se no provador do Z/EVES. A ferramenta retorna a validação do predicado através da exibição da letra Y do lado esquerdo da Figura 6.22 indicando que a sintaxe (*Syntax*) e a prova (*Proof*) do parágrafo estão corretos,

provando assim que a transformação do fragmento de metamodelo UML para o de SQL é uma transformação representada em Z como uma relação binária.

No predicado do esquema da Figura 6.23, observa-se que não existe uma função inversa $T^{\sim}$ indicando uma transformação reversa, pois a transformação do fragmento de metamodelo UML para o de SQL é unidirecional e possível apenas em uma única direção, fato pelo qual esta transformação não pode ser representada por uma função bijetiva em linguagem Z.

6.4 ESPECIFICAÇÕES GENÉRICAS DA TRANSFORMAÇÃO

Os dois exemplos do estudo de caso podem ser generalizados para outras aplicações através de dois axiomas. O primeiro especifica apenas a transformação bidirecional e outro axioma mais abrangente reúne em um único predicado a generalização da transformação unidirecional e bidirecional.

6.4.1 AXIOMA DA TRANSFORMAÇÃO BIDIRECIONAL

O primeiro axioma é o que possui um predicado que formaliza e generaliza a transformação bidirecional em linguagem Z. Como por exemplo, a transformação do fragmento do metamodelo UML para o de JAVA do estudo de caso apresentado no início deste capítulo, a Figura 6.24 ilustra este axioma.

O axioma da Figura 6.24 é criado no Z/EVES, selecionando no *"Mini Edit"* o comando *"Axiom Box".* De acordo com a Figura 6.24, verifica-se que o Z/EVES valida com sucesso o axioma ao exibir nas duas colunas da esquerda a letra Y para a sintaxe (*Syntax)* e para a prova *(Proof),* indicando que não foi encontrado nenhum erro na validação.

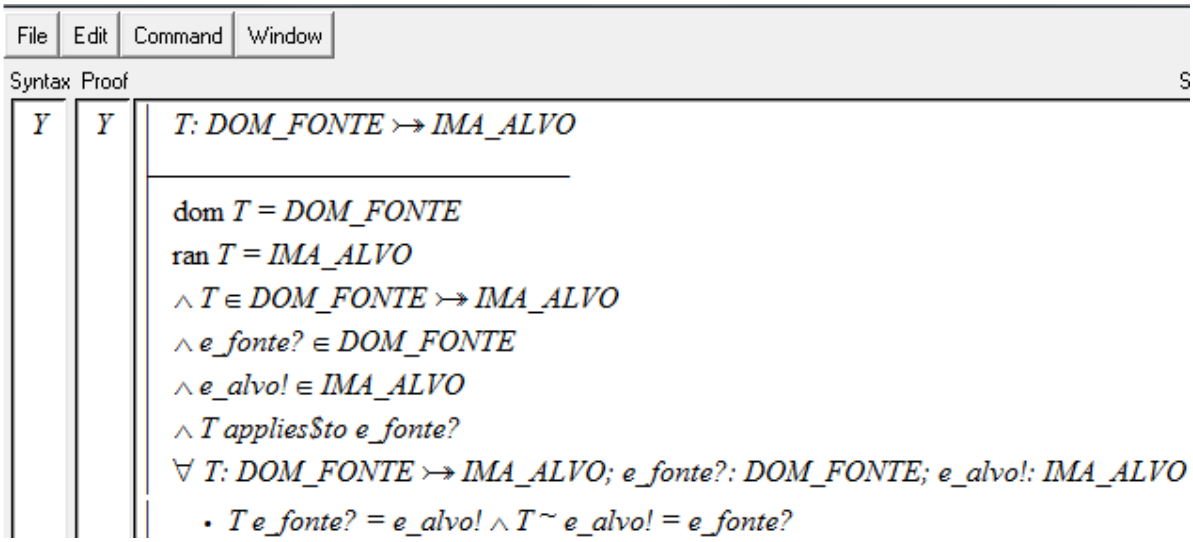

Figura 6.24: Axioma da Transformação Bidirecional validado no Z/EVES

Na Figura 6.25, o predicado do esquema da Figura 6.24 é apresentado de forma mais organizada no formato LATEX.

$$
\begin{array}{l}
T : DOM\_FONTE \rightarrowtail IMA\_ALVO \\
\hline
\mathrm{dom}\ T = DOM\_FONTE \\
\mathrm{ran}\ T = IMA\_ALVO \\
\land\ T \in DOM\_FONTE \rightarrowtail IMA\_ALVO \\
\land\ e\_fonte? \in DOM\_FONTE \\
\land\ e\_alvo! \in IMA\_ALVO \\
\land\ T\ applies\$\ to\ e\_fonte? \\
\forall\ T : DOM\_FONTE \rightarrowtail IMA\_ALVO;\ e\_fonte? : DOM\_FONTE;\ e\_alvo! : IMA\_ALVO \bullet \\
T(e\_fonte?) = e\_alvo! \land T^{\sim}(e\_alvo!) = e\_fonte?
\end{array}
$$

Figura 6.25: Axioma que formaliza a Transformação Bidirecional (formato LATEX)

A explicação do predicado do axioma da Figura 6.25 é dada da seguinte forma: abaixo da linha divisória é indica que o *dom T* (domínio de *T*) é igual ao *DOM_FONTE* e o *ran T* (imagem de *T*) é igual a *IMA_ALVO* e (∧) a transformação *T* pertencente (∈) a função bijetiva do *DOM_FONTE* para o *IMA_ALVO*.

Neste predicado, o elemento fonte (*e_fonte*?) pertence (∈) ao conjunto *DOM_FONTE* e (∧) o elemento alvo (*e_alvo*!) pertence (∈) ao conjunto *IMA_ALVO* e (∧) a transformação *T*, que é uma função bijetiva do *DOM_FONTE* para o *IMA_ALVO*, se aplica (*applies$to*) ao elemento fonte *e_fonte*? .

Generalizamos que para todo ($\forall$) e qualquer que seja uma transformação *T* aplicada do *DOM_FONTE* para o *IMA_ALVO*, tem-se um elemento *e_fonte*? pertencente ($\in$) ao *DOM_FONTE* e um elemento *e_alvo*! pertencente ($\in$) a *IMA_ALVO*, onde a função *T* transforma um elemento *e_fonte?* em um elemento *e_alvo!* e uma função inversa $T^{\sim}$ (que é uma transformação reversa) transformar um elemento alvo (*e_alvo!*) em um elemento fonte (*e_fonte*?).

Este axioma é a regra geral que serve como base para criar e validar outros esquemas que tratam a transformação como uma função bijetiva, quando esta transformação for do tipo bidirecional.

6.4.2 Axioma que Generaliza as Transformações em MDE como uma Relação Binária

O segundo axioma foi apresentado no Capítulo 5 na Figura 5.5. Este axioma generaliza todos os tipos de transformação de modelos como uma *relação binária*, seja ela bidirecional ou unidirecional. Este segundo axioma uniu em uma única regra um predicado que aborda casos de mapeamento de elementos de um-para-um, de um-para-muitos, de muitos-para-um e de muitos-para-muitos.

Na Figura 6.26, a verificação deste axioma no Z/EVES é ilustrada.

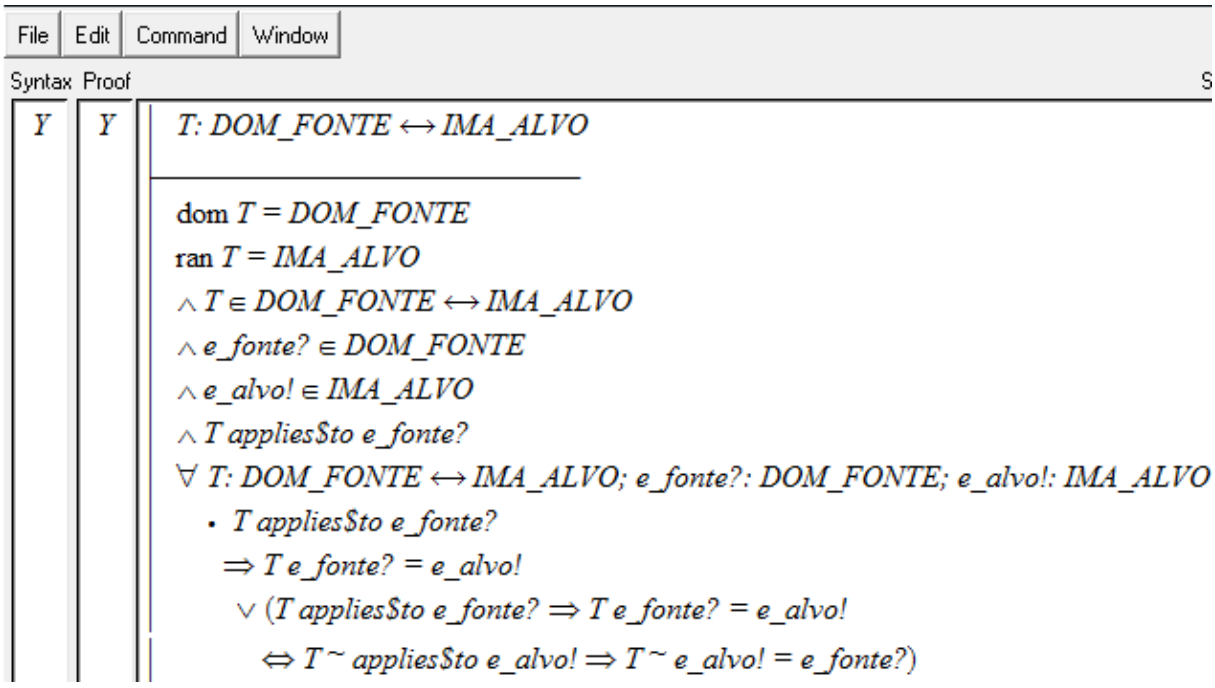

Figura 6.26: Axioma Genérico da transformação de modelos validado no Z/EVES

Criou-se este axioma no Z/EVES, selecionando no *"Mini Edit"* o comando *"Axiom Box"*. De acordo com a Figura 6.26, verifica-se que o Z/EVES valida com sucesso o axioma ao exibir nas duas colunas da esquerda a letra Y, indicando que não foi encontrado nenhum um erro na validação.

Este tipo de axioma é um resultado das especificações feitas no estudo de caso, e a partir dele poderá se concluir que a transformação de modelos em MDE, pode ser representada no *framework* apenas como uma relação binária da teoria dos conjuntos seja bidirecional ou unidirecional.

Na Figura 6.27, o predicado do esquema da Figura 6.26 é apresentado de forma mais organizada no formato LATEX.

$$
\begin{array}{|l}
T : DOM\_FONTE \leftrightarrow IMA\_ALVO \\
\hline
\mathrm{dom}\ T = DOM\_FONTE \\
\mathrm{ran}\ T = IMA\_ALVO \\
\land\ T \in DOM\_FONTE \leftrightarrow IMA\_ALVO \\
\land\ e\_fonte? \in DOM\_FONTE \\
\land\ e\_alvo! \in IMA\_ALVO \\
\land\ T\ applies\$\ to\ e\_fonte? \\
\forall\ T : DOM\_FONTE \leftrightarrow IMA\_ALVO;\ e\_fonte? : DOM\_FONTE;\ e\_alvo! : IMA\_ALVO \bullet \\
T\ applies\$\ to\ e\_fonte? \Rightarrow T(e\_fonte?) = e\_alvo! \lor (T\ applies\$\ to\ e\_fonte? \Rightarrow T(e\_fonte?) = e\_alvo! \\
\Leftrightarrow T^{\sim}\ applies\$\ to\ e\_alvo! \Rightarrow T^{\sim}(e\_alvo!) = e\_fonte?)
\end{array}
$$

Figura 6.27: Axioma Genérico da transformação de modelos (formato LATEX)

A explicação do predicado do axioma geral da Figura 6.27 é dada da seguinte forma: tem-se abaixo da linha divisória o *dom T* (domínio de *T*) é igual ao *DOM_FONTE* e o *ram* (imagem) de *T* é igual a *IMA_ALVO* e (∧) a transformação *T* pertencente (∈) a relação binária do *DOM_FONTE* para *IMA_ALVO*.

Neste predicado, o elemento fonte (*e_fonte*?) pertence (∈) ao conjunto *DOM_FONTE* e (∧) o elemento alvo (*e_alvo*!) pertence (∈) ao conjunto *IMA_ALVO* e (∧) a transformação *T* (que neste caso passa a ser uma relação binária) aplica-se (*applies$to*) ao elemento *e_fonte*?.

Generalizamos que: para todo (∀) e qualquer que seja uma relação binária T aplicada do conjunto *DOM_FONTE* para o conjunto *IMA_ALVO*, tem-se um elemento *e_fonte*? pertencente (∈) ao *DOM_FONTE* e um elemento *e_alvo*! pertencente (∈) a IMA_ALVO. Onde a relação binária *T* aplicada (*applies$to*) ao elemento fonte *e_fonte?* transforma um elemento *e_fonte?* em um elemento *e_alvo!* ou (∨) existe uma outra relação

binária T aplicada (*applies$to*) ao elemento fonte *e_fonte?* que transforma um elemento fonte *e_fonte?* em um elemento alvo *e_alvo!* se, e somente se (⇔) existir uma relação inversa $T^{\sim}$ aplicada (*applies$to*) ao elemento *e_alvo!* que transforma um elemento *e_alvo!* em um elemento *e_fonte?*.

A regra depois do símbolo (∨) na verdade trata a transformação como uma função bijetiva que é um tipo especial de relação binária.

Este axioma garante que as transformações em MDE são mais adequadas serem representas por uma relação binária do que uma função. Este resultado valida o uso da relação binária como o artefato matemático ideal para representar a transformação. Assim, valida-se o *framework* conceitual proposto no capítulo 5 ao tratar a transformação de modelos como uma relação binária.

6.5 TRANSFORMAÇÃO DO METAMODELO UML PARA O METAMODELO JAVA COMO UMA RELAÇÃO BINÁRIA

Para comprovar a validade do axioma da Figura 6.27, retomou-se o exemplo do estudo de caso da transformação bidirecional do fragmento de metamodelo UML para o de JAVA, e substituiu-se a função bijetiva anteriormente usada para representar esta transformação pela relação binária e em seguida validou-se no Z/EVES. O esquema resultante deste novo exemplo esta ilustrado na Figura 6.28.

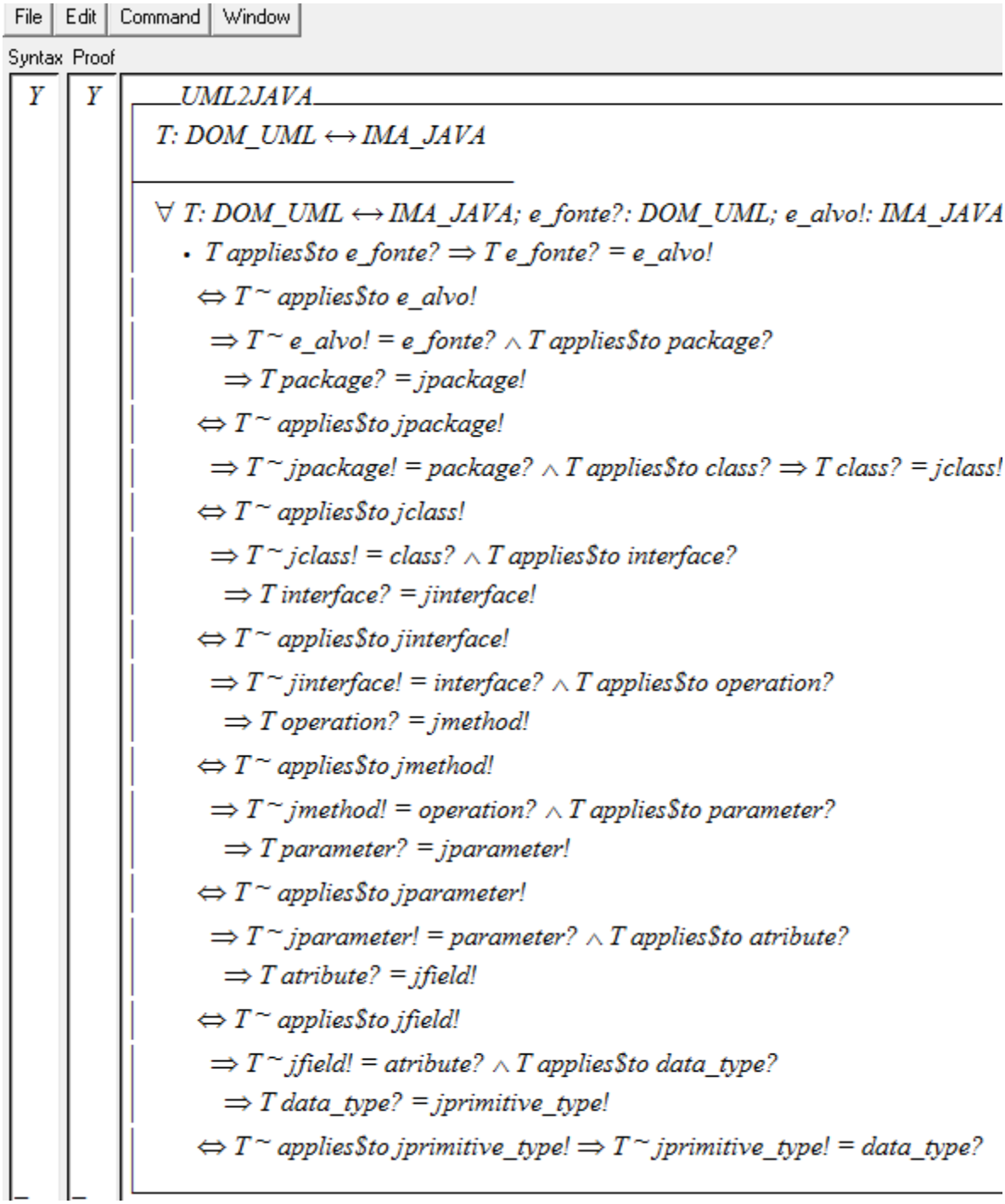

Figura 6.28: Transformação de UML para JAVA validada no Z/EVES

Nas colunas da esquerda da Figura 6.28, o Z/EVES exibe o resultado da checagem do esquema e verifica-se que não houve nenhum tipo de inconsistência na prova do predicado quando se mudou de **função bijetiva** para **relação binária**. Este resultado comprova que a função bijetiva que representou a transformação bidirecional de UML para JAVA pode ser substituída por uma relação binária. Concluindo-se que a relação binária pode ser utilizada para representar tanto uma transformação unidirecional quanto bidirecional.

O predicado abaixo da linha divisória do esquema garante que a transformação pode ser especificada como uma relação binária, pois o símbolo ↔ entre o *DOM_UML* e *IMA_JAVA* formaliza a transformação como uma relação binária.

Na Figura 6.29, o predicado do esquema da Figura 6.28 é apresentado de forma mais organizada no formato LATEX.

$$
\begin{array}{|l}
\textit{UML2JAVA} \\
T : DOM\_UML \leftrightarrow IMA\_JAVA \\
\hline
\forall\, T : DOM\_UML \leftrightarrow IMA\_JAVA;\ e\_fonte? : DOM\_UML;\ e\_alvo! : IMA\_JAVA \bullet \\
Tapplies\$\ to\ e\_fonte? \Rightarrow T(e\_fonte?) = e\_alvo! \\
\Leftrightarrow T^{\sim} applies\$\ to\ e\_alvo! \Rightarrow T^{\sim} e\_alvo! = e\_fonte? \\
\\
\wedge\ Tapplies\$\ to\ package? \Rightarrow T(package?) = jpackage! \\
\Leftrightarrow T^{\sim} applies\$\ to\ jpackage! \Rightarrow T^{\sim}(jpackage!) = package? \\
\\
\wedge\ Tapplies\$\ to\ class? \Rightarrow T(class?) = jclass! \\
\Leftrightarrow T^{\sim} applies\$\ to\ jclass! \Rightarrow T^{\sim}(jclass!) = class? \\
\\
\wedge\ Tapplies\$\ to\ interface? \Rightarrow T(interface?) = jinterface! \\
\Leftrightarrow T^{\sim} applies\$\ to\ jinterface! \Rightarrow T^{\sim}(jinterface!) = interface? \\
\\
\wedge\ Tapplies\$\ to\ operation? \Rightarrow T(operation?) = jmethod! \\
\Leftrightarrow T^{\sim} applies\$\ to\ jmethod! \Rightarrow T^{\sim}(jmethod!) = operation? \\
\\
\wedge\ Tapplies\$\ to\ parameter? \Rightarrow T(parameter?) = jparameter! \\
\Leftrightarrow T^{\sim} applies\$\ to\ jparameter! \Rightarrow T^{\sim}(jparameter!) = parameter? \\
\\
\wedge\ Tapplies\$\ to\ atribute? \Rightarrow T(atribute?) = jfield! \\
\Leftrightarrow T^{\sim} applies\$\ to\ jfield! \Rightarrow T^{\sim}(jfield!) = atribute? \\
\\
\wedge\ Tapplies\$\ to\ data\_type? \Rightarrow T(data\_type?) = jprimitive\_type! \\
\Leftrightarrow T^{\sim} applies\$\ to\ jprimitive\_type! \Rightarrow T^{\sim}(jprimitive\_type!) = data\_type?
\end{array}
$$

Figura 6.29: Transformação de UML para JAVA (formato LATEX)

No esquema da Figura 6.29, observa-se que foi considerada do axioma geral da Figura 6.27, apenas a parte do predicado que está depois do símbolo ∨ (ou), pois é a parte que trata da transformação de modelos como uma função bijetiva, que é um tipo especial de relação binária, isto quando esta transformação for bidirecional.

A explicação do predicado da Figura 6.29 é dada da seguinte forma: na primeira linha do predicado tem-se a regra geral do axioma da Figura 6.27, onde para todo (∀) e qualquer que seja uma transformação *T* partindo do *DOM_UML* para o *IMA_JAVA* especificada como uma relação binária em Z. E um elemento fonte *e_fonte?* tipo do conjunto

DOM_UML (*e_fonte?: DOM_UML)* e um elemento alvo *e_alvo!* tipo do conjunto *IMA_JAVA* (*e_alvo!: IMA_JAVA*).

A relação binária *T* aplicada (*applies$to*) ao elemento fonte *e_fonte?* implica ($\Rightarrow$) na transformação do elemento *e_fonte?* no elemento alvo *e_alvo!*, se e semente se ($\Leftrightarrow$) a relação binária $T^{\sim}$ aplicada ao elemento alvo *e_alvo!* transformar através da transformação reversa o elemento alvo *e_alvo!* no elemento fonte *e_fonte?*.

E em seguida têm-se as transformações propriamente ditas entre os elementos selecionados do metamodelo UML que estão mapeados aos elementos selecionados do metamodelo JAVA.

Por exemplo, de acordo com a regra geral anterior, se *T* é aplicado ao elemento *package?* que está contido no conjunto *DOM_UML* isso implica que a relação binária *T* transforma o elemento fonte *package?* no elemento alvo *jpackage*!, se e somente se ($\Leftrightarrow$) a relação inversa transformar o elemento alvo *jpackage*! no elemento fonte *package?* e assim por diante ocorre para as demais correspondências estabelecidas, ver na Figura 6.29.

Observa-se na Figura 6.29, que apenas a parte entre parênteses do axioma geral da Figura 6.27:

$$(T\ applies\$\ to\ e\_fonte? \Rightarrow T(e\_fonte?) = e\_alvo \Leftrightarrow T^{\sim}\ applies\$\ to\ e\_alvo! \Rightarrow T^{\sim}(e\_alvo!) = e\_fonte?)$$

depois do símbolo (V) foi considerada, isto ocorre devido a transformação ser bidirecional e exigir que exista a transformação reversa no retorno do metamodelo alvo de JAVA para o metamodelo fonte de UML.

Este exemplo foi útil para comprovar que uma transformação bidirecional representada como uma função bijetiva poder ser também representada por uma relação binária em linguagem Z, em razão da função bijetiva ser um tipo especial de relação binária.

6.6 TRANSFORMAÇÃO DO METAMODELO UML PARA METAMODELO WSDL

A fim de testar a validade do axioma da Figura 6.27, realizou-se uma transformação de um fragmento de metamodelo UML para um fragmento de metamodelo WSDL, envolvendo um estudo de caso de um sistema de atendimento ao cliente de uma agência de viagens extraída do artigo de *Bézivin et at* [31]. Em [31], os seus autores não utilizam a linguagem Z, eles utilizam a linguagem ATL [35] para realizar as transformações entre os modelos.

Para compreender com mais clareza este exemplo, um diagrama de caso de uso da aplicação é ilustrado na Figura 6.30.

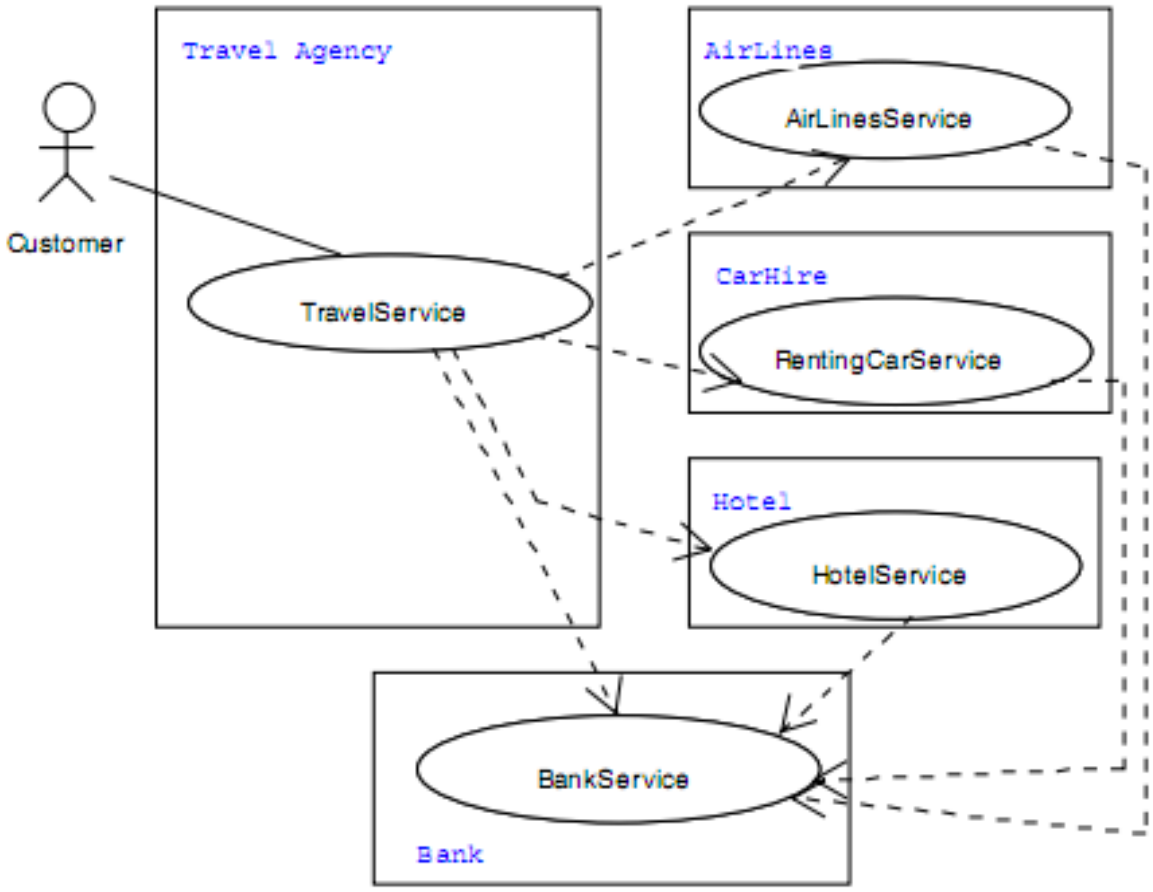

Figura 6.30: Diagrama de Caso de Uso da Agência de Viagens [31]

Os requisitos desta aplicação são: O cliente (`Customer`) acessa os serviços da agência de viagens. A agência de viagens (`TravelAg`) fornece os serviços para encontrar: reserva, pagar e cancelar um bilhete de avião, realizar um aluguel de carro ou de um quarto. As linhas aéreas (`AirLines`), a locadora de carros (`CarHire`) e o hotel (`Hotel`) fornecem os serviços para encontrar uma reserva, realizar pagamento e cancelar os seus produtos. O Banco realiza as operações financeiras entre as empresas [31], coforme é ilustrado na Figura 6.30.

De acordo com o estudo de caso da Figura 6.30, a agência de viagens vende bilhetes de avião, reserva quartos do hotel e oferece aluguel de automóveis. A fim de prestar estes serviços para seus clientes, uma agência de viagens precisa estabelecer laços comerciais com outras empresas, ou seja, companhias aéreas, empresas de aluguel de automóvel e hotéis. Neste contexto, uma instituição financeira, ou seja, um banco é necessário para facilitar as

transações financeiras entre um cliente e uma empresa, ou entre uma empresa e outra empresa [31].

A Figura 6.30 mostra um cliente que pretende fazer uma viagem. A fim de fazer esta viagem, ele acessa o site de uma agência de viagens que vende bilhetes de avião, oferece aluguel de carros e reservas de quartos [31].

Nesta aplicação, o cliente entra com suas necessidades. A agência de viagens recebe os pedidos do cliente e os envia para a companhia aérea, de aluguel de automóveis e para o hotel. A agência de viagens recebe as propostas dos três parceiros e escolhe as melhores soluções para os vôos, carros e hotéis, e envia para o cliente que escolhe, reserva e paga por um bilhete de avião, um carro e um quarto de hotel. O pagamento é feito com o apoio de um banco [31].

Em [31], para se alcança a implementação do sistema da agência de viagens através da abordagem da MDE, são feitas quatro tipos de transformações de modelos: de um metamodelo UML para um metamodelo JAVA, de um metamodelo UML para um metamodelo WSDL, de um metamodelo EDOC para um metamodelo JAVA e de um metamodelo EDOC para um metamodelo WSDL.

Destas quatro escolheu-se apenas a transformação de um metamodelo UML para um metamodelo WSDL que é suficiente para testar a veracidade do axioma da Figura 6.27, quando a transformação de modelos apresentar um caso de mapeamento de um-para-muitos.

Os possíveis mapeamentos existentes entre os elementos do metamodelo UML com os do metamodelo WSDL são ilustradas na Figura 6.31.

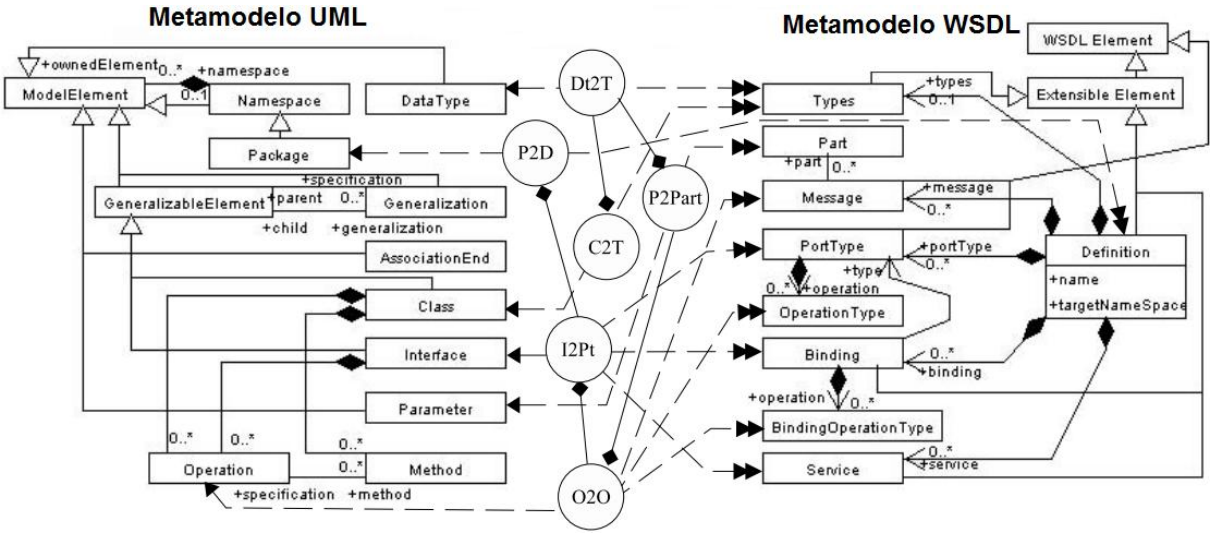

Figura 6.31: Transformação do metamodelo UML para o de WSDL [31]

A Figura 6.31 mostra a existência de um mapeamento do elemento *Interface* para os elementos *PortType*, *Binding* e *Service* que é do tipo um-para-muitos, este tipo de

mapeamento contraria a definição de função utilizada no megamodelo de Favre [3], mas é compatível com a definição de relação binária utilizada no nosso *framework* conceitual.

Também na transformação de UML para WSDL ilustrada na Figura 6.31, existe um mapeamento de muitos-para-um, onde os elementos *Class* e *DataType* do metamodelo UML são mapeados para o elemento *Type* no metamodelo WSDL. Esta transformação é adequada para testar a validade do axioma da Figura 6.27.

A Tabela 6.3 apresenta os casos de mapeamento: de um-para-muitos (*Interface?* para *PortType!*, *Binding!* e *Service!*), de muitos-para-um *(Class?* e *DataType?* para *Type!)* e de um-para-um (*Parameter?* para *Part!*), que foram extraídos dos possíveis casos de mapeamento de elementos entre o metamodelo UML com o metamodelo WSDL ilustrados na Figura 6.31 para serem especificados em linguagem Z e provados no Z/EVES.

Tabela 6.3: Mapeamentos entre os elementos do metamodelo UML com os de WSDL

UML	WSDL
Interface?	*PortType!*
Interface?	*Binding!*
Interface?	*Service!*
Class?	*Type!*
DataType?	*Type!*
Parameter?	*Part!*

Estes casos de mapeamento podem ser analisados nos diagramas de Venn, conforme a Figura 6.32.

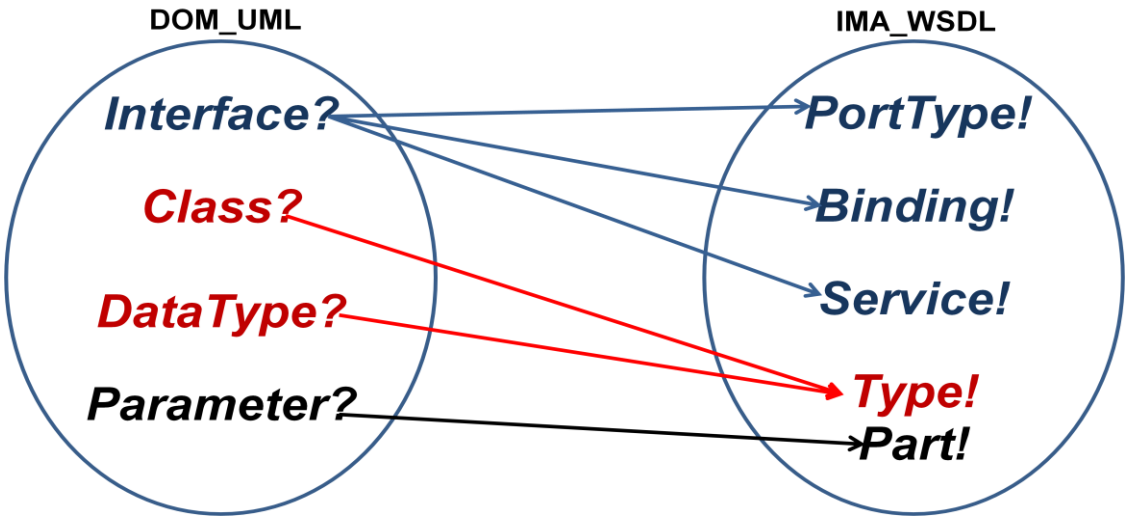

Figura 6.32: Mapeamento de UML para WSDL em Diagramas de Venn

Através dos diagramas de Venn, ilustrados na Figura 6.32, verifica-se que a transformação de UML para WSDL não pode ser associada a nenhum tipo de função em Z, devido ao mapeamento de um-para-muitos (o elemento *Interface?* é mapeado para os elementos *PortType!*, *Binding!* e *Service!*) contrariar a definição de função estabelecida no megamodelo de Favre [3].

Com isso, a relação binária é o artefato matemático ideal para representar esta transformação, pois tanto o mapeamento de um-para-muitos (*Interface?* mapeado para os elementos *PortType!*, *Binding!* e *Service!*) quanto o mapeamento de muitos-para-um (*Class?* e *DataType?* mapeado para o elemento *Type!*) são suportados pela relação binária.

Na Figura 6.33, a tela principal de especificação do Z/EVES é ilustrada, e exibe nas duas colunas do lado esquerdo o resultado da checagem do esquema, a letra Y para a sintaxe (*Syntax*) e para a prova (*Proof*) indica que o predicado não apresenta erros na sintaxe e nem na prova.

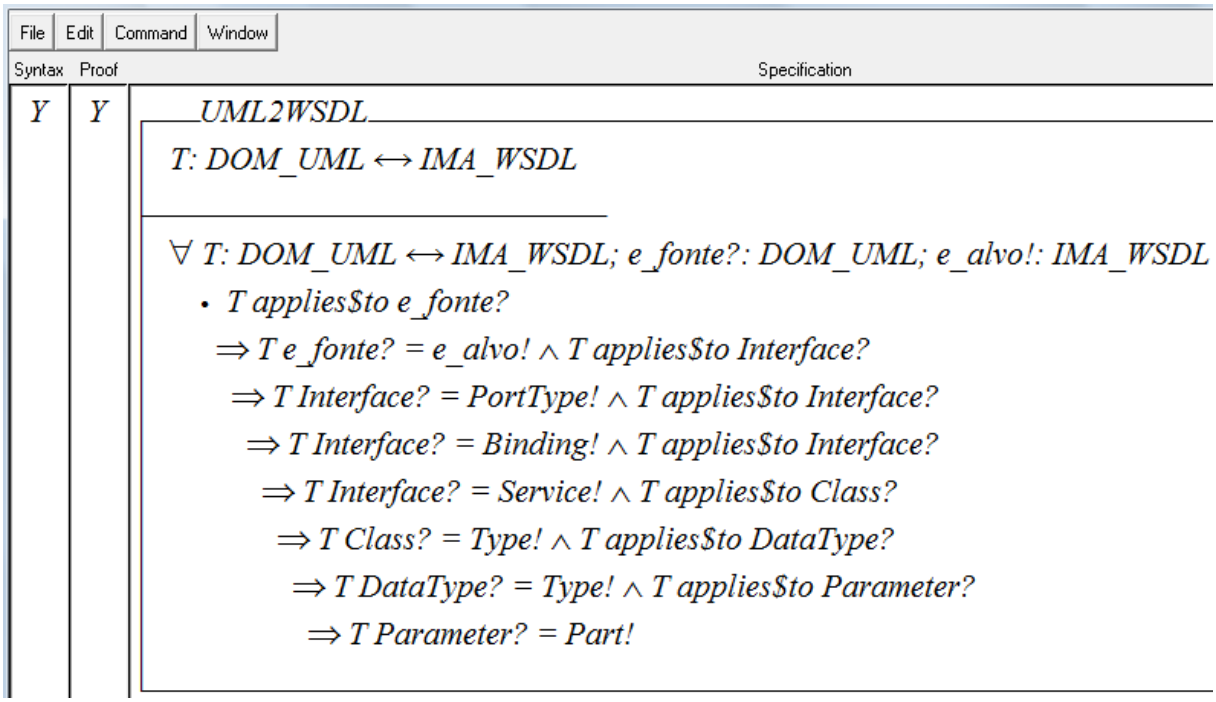

Figura 6.33: Esquema da Transformação de UML para WSDL no Z/EVES

O esquema ilustrado na Figura 6.33 utiliza uma parte da regra do predicado do axioma da Figura 6.27, que é a seguinte:

$$\forall T : DOM\_UML \leftrightarrow IMA\_WSDL;\ e\_fonte? : DOM\_UML;\ e\_alvo! : IMA\_WSDL \bullet T\ applies\$\ to\ e\_fonte? \Rightarrow T(e\_fonte?) = e\_alvo!$$

Esta regra é apenas a parte do predicado do axioma da Figura 6.27 que está antes do símbolo (V), e que trata a transformação entre modelos como uma relação binária, quando a transformação não for do tipo bidirecional.

O predicado da Figura 6.33 foi criado para garantir que a transformação de UML para WSDL seja validada no Z/EVES unicamente como uma relação binária.

Na Figura 6.34, o esquema da Figura 6.33 é apresentado de forma mais organizada no formato LATEX, onde o esquema pode ser analisa linha por linha.

$$
\begin{array}{|l}
\textit{UML2WSDL} \\
T : DOM\_UML \leftrightarrow IMA\_WSDL \\
\hline
\forall\, T : DOM\_UML \leftrightarrow IMA\_WSDL;\ e\_fonte? : DOM\_UML;\ e\_alvo! : IMA\_WSDL \bullet \\
T\ applies\$\ to\ e\_fonte? \Rightarrow T(e\_fonte?) = e\_alvo! \\
\land\ T\ applies\$\ to\ Interface? \Rightarrow T(Interface?) = PortType! \\
\land\ T\ applies\$\ to\ Interface? \Rightarrow T(Interface?) = Binding! \\
\land\ T\ applies\$\ to\ Interface? \Rightarrow T(Interface?) = Service! \\
\land\ T\ applies\$\ to\ Class? \Rightarrow T(Class?) = Type! \\
\land\ T\ applies\$\ to\ DataType? \Rightarrow T(DataType?) = Type! \\
\land\ T\ applies\$\ to\ Parameter? \Rightarrow T(Parameter?) = Part! \\
\hline
\end{array}
$$

Figura 6.34: Esquema da Transformação de UML para WSDL (formato LATEX)

Na primeira linha do predicado do esquema da Figura 6.34, a transformação *T* é especificada como uma relação binária do conjunto DOM_UML (fragmento de metamodelo UML) para o conjunto IMA_WSDL (fragmento de metamodelo WSDL), em seguida tem-se o elemento fonte (*e_fonte?*) pertencente ao conjunto DOM_UML e o elemento alvo (*e_alvo!*) pertencente ao conjunto WSDL.

Na segunda linha do esquema da Figura 6.34, tem-se a regra que será seguida para especificar o mapeamento dos elementos definidos no conjunto DOM_UML como os do conjunto IMA_SQL, onde nesta linha tem-se: a transformação *T* (que é uma relação binária) é aplicada (*applies$ to*) ao elemento fonte (*e_fonte?)* implicando que *T* transforma o elemento *e_fonte?* no elemento alvo *e_alvo!*,ou seja, *T (e_fonte?) = e_alvo!*.

Da terceira a quinta linha do esquema da Figura 6.34, especifica-se os outros mapeamentos entre os elementos, onde tem-se a transformação *T* aplicada ao elemento *Interface?* que resulta nas seguintes transformações: *T(Interface?) = PortType!*, *T(Interface?) = Binding!* e *T(Interface?) = Service!* este é um caso de mapeamento de um-para-muitos, este caso não gera ambiguidade no nosso *framework* conceitual, pois nele a transformação é tratada como uma relação binária, assim suportando o caso de mapeamento de um-para-muitos.

Na quinta e na sexta linha, tem-se as transformações *T(Class?) = Type!* e *T(DataType?) = Type!*. Este é um caso de mapeamento de muitos-para-um e por fim *T(Parameter?) = Part!* é um caso de mapeamento de um-para-um. Estes casos de mapeamento também são suportados pela relação binária, assim comprova-se a validade do axioma da Figura 6.27, que especifica as transformações em MDE como uma relação binária.

6.7 ANÁLISES DOS RESULTADOS OBTIDOS

Os exemplos de mapeamento dos elementos dos metamodelos: de UML para JAVA, de UML para SQL e de UML para WSDL foram feitas seguindo os passos da metodologia proposta no Capítulo 4.

Ao se especificar os mapeamentos entre os elementos dos metamodelos fonte e alvo em linguagem Z, verificou-se que a transformação bidirecional de UML para JAVA pode ser representada como uma função bijetiva. Isto é compatível com o *framework* utilizado para representar o processo de transformação, mesmo que o termo utilizado no *framework* para representar a transformação seja relação binária, pois como já visto, uma função bijetiva é um tipo especial de relação binária.

Neste caso de transformação de modelos, do metamodelo UML para o metamodelo JAVA, nem todos os elementos dos metamodelos foram considerados para realizar a transformação, caso fossem considerados todos os elementos dos metamodelos então a transformação não seria mais uma função bijetiva.

Para a transformação de UML para SQL, ao se aplicar a metodologia, verificou-se que a transformação não poderia ser representada como uma função bijetiva, o que é comprovado quando se valida o esquema no Z/EVES, pois esta ferramenta retorna um erro na prova ao exibir a letra N na coluna da prova. No entanto, quando se substitui a função bijetiva pela relação binária, o Z/EVES exibe a letra Y, provando que a transformação de UML para SQL só pode ser representa por uma relação binária.

Este resultado é compatível com a representação da transformação no *framework*, que utiliza o termo relação binária para representar a transformação.

Com os resultados obtidos, comprovamos que a transformação bidirecional do fragmento do metamodelo UML para o de JAVA pode ser generalizada para uma relação binária, uma vez que, quando se substitui no Z/EVES a função bijetiva pela relação binária, a ferramenta valida o predicado com sucesso, exibindo a letra Y na coluna da sintaxe (*Syntax*) e da prova (*Proof*). Provando assim que a transformação bidirecional também é uma relação binária.

O resultado da transformação de UML para WSDL comprova que a relação binária é muito mais abrangente do que a função, uma vez que, esta transformação contém mapeamentos dos tipos: um-para-muitos, de muitos-para-um e de um-para-um. Com este

exemplo, o Z/EVES valida com sucesso o predicado do axioma proposto para generalizar a transformação bidirecional e unidirecional como uma relação binária.

Estes resultados comprovam que os modelos ilustrativos utilizados para representar o processo de transformação de modelos, apresentados no Capítulo 3, não são totalmente completos para formalizar a transformação entre modelos, pois eles não foram construídos em uma linguagem padrão.

Favre [3] na tentativa de formalizar a transformação de modelos, utiliza o termo função em seu megamodelo [3] para representar a transformação, porém a função é compatível apenas com os casos de mapeamento de um-para-um e de muitos-para-um como pode ser ilustrado na Figura 6.35. Os demais casos de mapeamento: de um-para-muitos e de muitos-para-muitos contrariam a definição de função, tornando assim o megamodelo de Favre inconsistente para representar as transformações que apresentam estes casos de mapeamento.

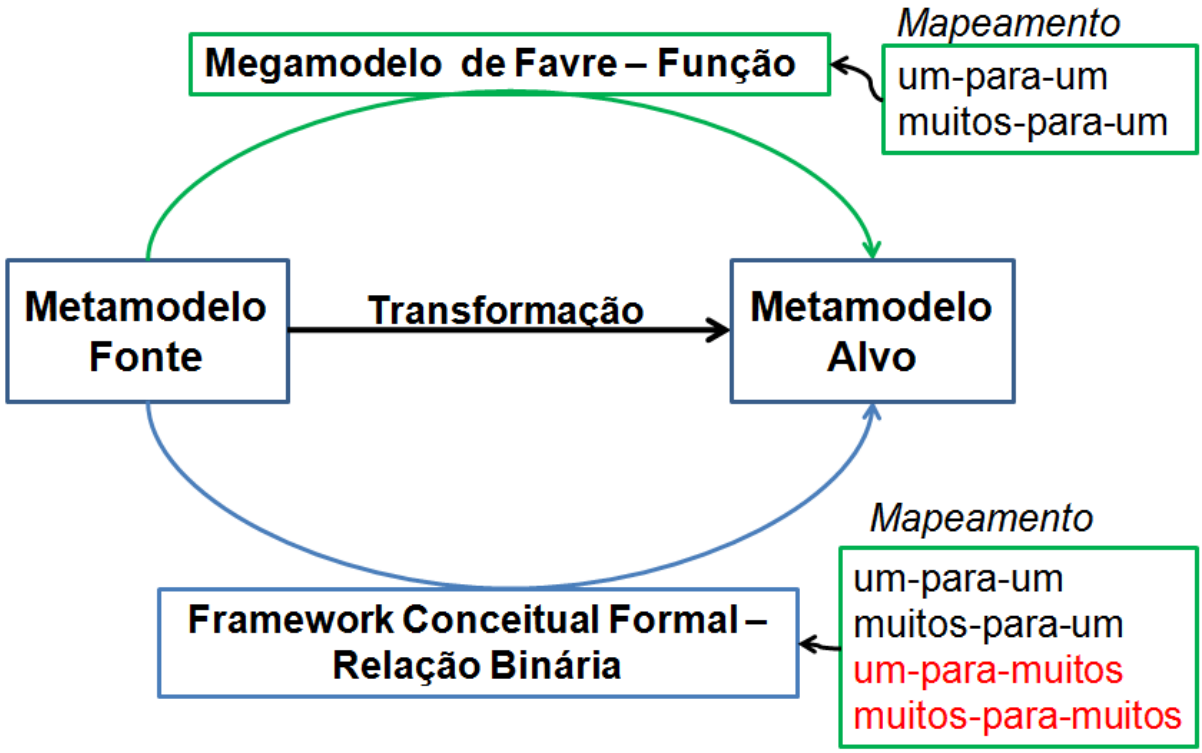

Figura 6.35: Resumo dos resultados obtidos

Na Figura 6.35, resumem-se os resultados obtidos no estudo de caso e verifica-se que o *framework* conceitual proposto nesta dissertação, ao utilizar o artefato matemático relação binária complementa o megamodelo de Favre [3], pois a relação binária é compatível tanto com os casos de mapeamento de um-para-um e de muitos-para-um quanto os casos de mapeamento de um-para-muitos e de muitos-para-muitos.

Assim, conclui-se que, de acordo com os exemplos feitos neste capítulo, a transformação é mais complexa do que um simples mapeamento de elementos de um-para-um, como por exemplo, a transformação do metamodelo UML para o de WSDL é mais complexa, pois contêm os casos de mapeamento de um-para-muitos, de muitos-para-um e de um-para-um.

Com isso, o modelo que expresse o processo de transformação em MDE deve representar a transformação como uma relação binária de acordo com o proposto no *framework* conceitual discutido no Capítulo 5 e ilustrado na Figura 5.2.

Desta forma, ambiguidades na interpretação do processo de transformação de modelos são eliminadas com o uso do *framework* conceitual proposto neste trabalho.

6.8 RESUMO

Neste capítulo, apresentou-se um estudo de caso para provar a veracidade do *framework* conceitual desenvolvido. Através dos estudos de caso, comprovou-se que é mais adequado representar uma transformação em MDE por uma relação binária do que uma função.

7 CONCLUSÃO E TRABALHOS FUTUROS

Neste capítulo, discute-se sobre as conclusões da pesquisa, os resultados obtidos, as contribuições deste trabalho, as limitações encontradas e os trabalhos futuros.

7.1 CONCLUSÕES DA PESQUISA

Esta dissertação apresentou a proposta de um *framework* conceitual formal desenvolvido com base na **Teoria dos Conjuntos e na Linguagem Formal** *Z*. Este *framework* tem como objetivo formalizar o processo de transformação de modelos sem apresentar ambiguidades em sua interpretação.

Para construir este *framework,* realizou-se um estudo formal aprofundado do processo de transformação em MDE. Este estudo formal teve como objetivo compreender com mais clareza quais os elementos da teoria dos conjuntos e da linguagem Z seriam utilizados para representar os elementos envolvidos no processo de transformação de modelos e reuni-los no *framework* conceitual.

Com o estudo formal, construiu-se uma **Hierarquia de Artefatos Matemáticos** de modo que os elementos da teoria dos conjuntos foram organizados do mais abrangente até o mais específico, onde o mais abrangente é o **Sistema** e o mais específico é a **Função Bijetiva**. Estes resultados foram organizados em uma tabela relacionando os elementos matemáticos da linguagem Z com os elementos presentes no processo de transformação de modelos.

A prova da relevância do *framework* foi feita ao testar a transformação do metamodelo UML para JAVA, do metamodelo UML para SQL e do metamodelo UML para WSDL na ferramenta de prova matemática Z/EVES. Por não apresentar inconsistências no predicado, comprovou-se que a relação binária é mais adequada para representar os diferentes tipos de transformação que apresentam os casos de mapeamentos de elementos de um-para-um, um-para-muitos, muitos-para-um e muitos-para-muitos.

Os resultados obtidos no Z/EVES validando a transformação de modelos como uma relação binária, prova a consistência do *framework* conceitual, pois este representa a transformação como uma relação binária, eliminando assim ambiguidades que venham a surgir na interpretação gráfica da transformação.

7.2 RESULTADOS OBTIDOS

Os primeiros resultados foram obtidos com o estudo formal da transformação de modelos, este estudo forneceu informações que ajudaram a construir a Hierarquia de Artefatos Matemáticos adicionada da classe *Função Bijetiva* associada às transformações bidirecionais.

Obteve-se também uma tabela que descreve os elementos da linguagem Z que são adequados para representar os elementos de MDE presentes no processo de transformação.

Obteve-se também outra tabela, onde reuniu-se os elementos de MDE que estão presentes no *framework* e não estão presentes no megamodelo de Favre [3]. Estes elementos constituem os incrementos que complementam a representação do processo de transformação de modelos apresentado no megamodelo de Favre [3].

Desenvolveu-se uma metodologia que auxilia na classificação de uma transformação em uma função bijetiva ou uma relação binária.

Alcançou-se uma representação mais consistente para a transformação de modelos através do *Framework* Conceitual Formal, ao representar a transformação como uma relação binária e ao ilustrar os relacionamentos entre os elementos envolvidos no processo de transformação.

No ano de 2010 na conferência internacional "*International Conferences on Computer, Information, and Systems Sciences, and Engineering"* foi publicado um artigo intítulado como: *"An Approach based on Z Language for Formalization of Model Transformation Definition"* [34].

7.3 CONTRIBUIÇÕES DESTA PESQUISA

Dentre as contribuições deste trabalho, citam-se as seguintes:

- O *Framework* Conceitual Formal é a principal contribuição desta pesquisa, pois, este elimina ambiguidades na interpretação da transformação em MDE;
- Uma representação mais clara e simples dos relacionamentos entre os elementos de MDE através dos símbolos χ**(*ConformsTo*)** e μ ***(are)*** presentes no processo de transformação;
- A representação da transformação como uma relação binária no *framework*, garantido assim contemplar os casos de mapeamento de elementos de um-para-muitos, de muitos-para-um, de muitos-para-muitos e um-para-um;

- A relação clara entre os elementos matemáticos da linguagem Z que melhor representam os elementos envolvidos no processo de transformação;
- Um axioma genérico que trata em um único predicado a transformação bidirecional e unidirecional de formas distintas, que pode ser estendido para outros exemplos diferentes dos apresentados nesta dissertação.
- A adição da classe *Função Bijetiva* na hierarquia de artefatos matemáticos, referente às transformações bidirecionais.

7.4 LIMITAÇÕES

Devido à linguagem Z apenas estabelecer o que deve ser feito para alcançar um objetivo, não como se fazer para alcançar tal objetivo, a principal limitação foi a ausência de uma ferramenta que implemente os esquemas feitos na linguagem Z, pois o Z/EVES é uma ferramenta que valida a parte formal da especificação dos predicados, ele não implementa as regras estabelecidas.

7.5 TRABALHOS FUTUROS

Como proposta para trabalho futuro, propõe-se criar uma ferramenta que disponibilize um *plug-in* que possibilite criar um projeto em uma linguagem de programação para implementar os esquemas feitos na linguagem Z e validados no Z/EVES, com o objetivo de transformar de forma automática os elementos fontes em elementos alvos baseado na relação semântica entre eles.

REFERÊNCIAS

[1] LOPES, Denivaldo; HAMMOUDI, Slimane; ABDELOUAHAB, Zair., **Schema Matching in the context of Model Driven Engineering: From Theory to Practice**, In: Advances in Systems, Computing Sciences and Software Engineering, Spinger, p. 253–264, 2006.

[2] KURTEV, I.; BÉZIVIN, J.; AKSIT, M., **Technological Spaces: an Initial Appraisal,** CoopIS, DOA'2002, Industrial track. p. 1–6, 2002.

[3] FAVRE , Jean-Marie, **Towards a Basic Theory to Model Driven Engineering**, In Workshop on Software Model Engineering (WISME 2004). p 1–13, 2004.

[4] OMG. **MDA Guide** Version 1.0.1 Document Number: omg/2003-06-01, 2003.

[5] KLEPPE, A.; WARMER, J.; BAST, W., **MDA Explained: The Model Driven Architecture: Practice and Promise**, Editora Addison-Wesley, 1° Edição, 2003.

[6] SPIVEY, J.M., **The Z notation: A Reference Manual,** Editora Prentice Hall International (UK) Ltd, 2°Edição, 1992.

[7] KENT, S., **Model Driven Engineering, Integrated Formal Methods. Integrated Formal Methods** - IFM, p. 286–298, May 2002.

[8] LOPES, D., **Engenharia Dirigida por Modelos: Abordagem e Aplicações,** Universidade Federal do Maranhão – UFMA. Pós Graduação em Engenharia da Eletricidade.

[9] LOPES, D., **Introdução a Engenharia Dirigida por Modelos**, I Escola Regional de Computação Ceará Maranhão Piauí (ERCEMAPI), 2007.

[10] OMG. **Meta Object Facility (MOF) 2.0 Query/View/Transformation**. Version 1.0 Document Number: formal/2008-04-03, 2008.

[11] MELLOR, Stephen; SCOOTT, Kendall; UHL, Axel and WEISE, Dirk., **MDA Destilada: Princípios de Arquitetura Orientada por Modelos**, Editora Ciência Moderna, 2005.

[12] IRWIN, M. **Software Manual for Windows Z/EVES Version 2.4**. TR-97-5505-04h. Release date: June 2005.

[13] KEMP, K., **Formal Methods Specification and Verification Guidebook for Software and Computer Systems,** Volume I: Planning and Technology Insertion**.** NASA Office of Safety and Mission Assurance, Washington D.C., December 1998.

[14] SPIVEY, J. M., **An introduction to Z and formal specifications**. **Softw. Eng.**, J., Michael Faraday House, Herts, UK, UK, v. 4, n. 1, p. 40–50, 1989. ISSN 0268-6961.

[15] WING, J. M. **A specifier's introduction to formal methods**. Computer, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 23, n. 9, p. 8–23, 1990. ISSN 0018-9162.

[16] ABRIAL, J.R.; SCHUMAN, S. A.; and MEYER, B., **A Specification Language**, in On the Construction of Programs, Cambridge University Press, eds. A. M. Macnaghten and R. M. McKeag, p. 343–410, 1979.

[17] Developed by members of the Z Standards Panel, **Formal Specification - Z Notation - Syntax**, **Type and Semantics,** August 24, 2002

[18] SMITH, G., **The Object-Z Specification Language**, American Kluwer Publishers, 2000.

[19] WOODCOCK, J.; DAVIES, J., **Using Z: Specification, Refinement, and Proof**. [S.l.]: Prentice Hall International, 1996.

[20] KONZEN, A. A., **Especificação de Requisitos de Agentes de Usuário em Z,** 2002, 99 f. Dissertação de mestrado (Programa de Pós-Graduação em Ciência da Computação) Pontifícia Universidade Católica do Rio Grande do Sul - Faculdade de Informática, Rio Grande do Sul, 2002.

[21] CRAIGEN, D. et al., **Eves: An overview,** In: VDM '91: Proceedings of the 4th International Symposium of VDM Europe on Formal Software Development-Volume I. London, UK: Springer - Verlag, 1991. p. 389–405. ISBN 3-540-54834-3.

[22] KROMODIMOELJO, S. et al., **The Eves system. In: Functional Programming, Concurrency, Simulation and Automated Reasoning**, International Lecture Series 1991-1992, McMaster University, Hamilton, Ontario, Canada. London, UK: Springer -Verlag, 1993. p. 349–373. ISBN 3-540-56883-2.

[23] OMG. **Model Driven Architecture (MDA).** ormsc/2001-07-01, July 2001, Disponível em http://www.omg.org/cgi-bin/doc?ormsc/2001-07-01, Acessado em 29/07/2011.

[24] BÉZIVIN, J. & PLOQUIN, N., **Tooling the MDA Framework: a new Software Maintenance and Evolution Scheme Proposal,** Journal of Object-Oriented Programming (JOOP), p. 1–10, December 2001.

[25] KURTEV, I.; BÉZIVIN, J.; AKSIT, M., **Technological Spaces: an Initial Appraisal**, CoopIS, DOA'2002, Industrial track, p. 1– 6, 2002.

[26] BÉZIVIN, J., **In Search of a Basic Principle for Model-Driven Engineering**, Novatica Journal, Special Issue, p. 21–24, March 2004.

[27] GAŠEVIĆ, D.; KAVIANI, N.; and HATALA, M., **On Metamodeling in Megamodels**, G. Engels et al. (Eds.): MoDELS 2007, LNCS 4735, p. 91–105, 2007.

[28] FAVRE , Jean-Marie, **Foundations of Model (driven) (Reverse) Engineering - Episode I: Story of the Fidus Papyrus and the Solarus**, Dagsthul Seminar on Model Driven Approaches for Language Engineering, p. 1–31, May 2004.

[29] FAVRE , Jean-Marie, **Foundations of Meta-Pyramids: Languages and Metamodels - Episode II: Story of Thotis the Baboon**, Dagsthul Seminar on Model Driven Approaches for Language Engineering, p. 1–28, May 2004.

[30] Série **From Ancient Egypt to Model Driven Engineering**, disponível em: http://www-adele.imag.fr/mda, Acessado em 20/07/2011.

[31] BÉZIVIN, J.; HAMMOUDI, S.; LOPES, D.; JOUAULT, F., **Applying MDA Approach for Web Service Platform**, Proceedings of the 8th IEEE Intl Enterprise Distributed Object Computing Conf (EDOC 2004), p. 1 – 13, 2004.

[32] BERNSTEIN, Philip A., **Applying Model Management to Classical Meta Data Problems**, In: First Biennial Conference on Innovative Data Systems Research (CIDR), p. 1 – 12, 2003.

[33] IEZZI, G. & MURAKAMI, C., **Fundamentos de Matemática Elementar – Conjuntos e Funções**, Editora Atual, 3° ed., 1997, São Paulo – SP.

[34] MENDES, Carlos C.G.; ABDELOUAHAB, Zair; LOPES, D., **An Approach based on Z Language for Formalization of Model Transformation Definition,** In: CISSE 2010, p. 1 – 5, 2010.

[35] ATL – ATLAS group LINA & INRIA Nantes. ATL - **ATLAS Transformation Language.** ATL User Manual 0.7, 2006.

[36] JOHNSON, R. E., **Components, Frameworks, Patterns,** [S.1.: s.n.], 1997.

[37] SOMMERVILLE, I., **Engenharia de Software**, 7ª edição, Addison Wesley, 2007.

[38] Wang, H. H.; Gibbins, N.; Payne, T.; Saleh, A.; Sun, J., **A Formal Model of Semantic Web Service Ontology (WSMO) Execution,** In: The Thirteenth IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 2008), p. 1 – 10.

[39] SOUSA, H. C. S., **Construção Automatizada de Casos de Teste Usando Engenharia Dirigida por Modelos,** 2009, 142 f. Dissertação de mestrado (Programa de Pós-Graduação em Engenharia de Eletricidade, área de concentração Ciência da Computação) UFMA – Maranhão, 2009.

[40] LOPES, D., **Étude et applications de l'approche MDA pour des plates-formes de Services Web** 2005, 264 p. Tese de doutorado (*Sciences et Technologies de L'Information des Matériaux) Université de Nantes*, 2005.

[41] EMF, **Eclipse Modeling Framework Project,** Disponível em: http://www.eclipse.org/modeling/emf/?project=emf. Acessado em: 01/03/2010.

[42] PRESSMAN, R. S., **Engenharia de Software**, 6° edição, McGraw-Hill, 2006.

[43] OLIVEIRA, D. P. R., **Sistemas, organizações e métodos: uma abordagem gerencial,** 13. ed. São Paulo, 2002.

ANEXOS

ANEXO A – A FERRAMENTA DE VALIDAÇÃO FORMAL Z/EVES

A.1 A JANELA DE EDIÇÃO (MINI EDITOR)

A janela de edição permite criar novos parágrafos ou modificar parágrafos já existentes. O editor dar suporte para edição padrão, tais como: a inserção e exclusão de caracteres, recortar e colar. Além disso, oferece uma paleta de símbolos abaixo da área de edição baseada no *toolkit* da linguagem Z.

A Figura A.1 apresenta a janela de edição de parágrafos e o *toolkit* da linguagem Z com os símbolos que representam as principais funcionalidades de Z.

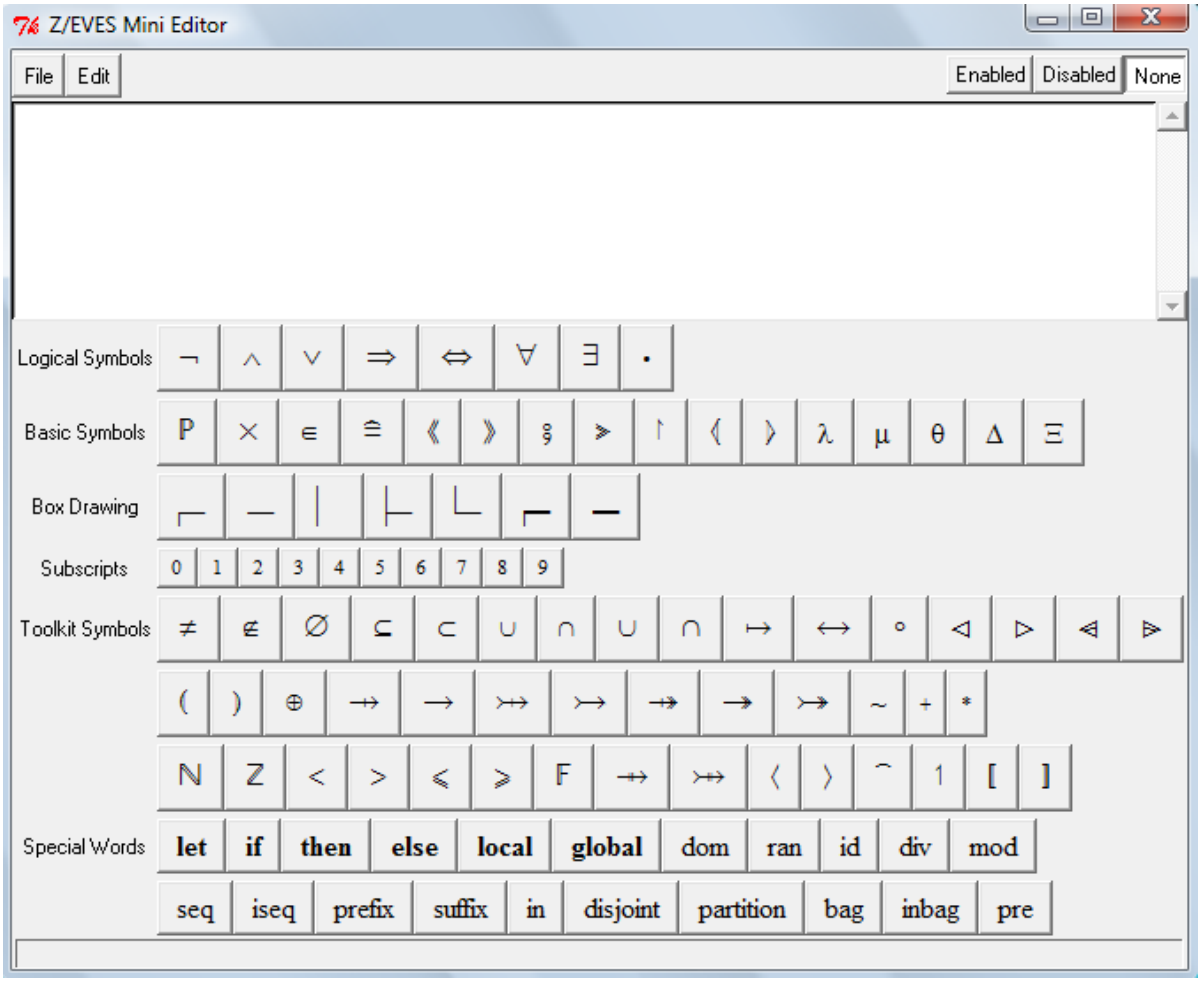

Figura A.1: A Janela de Edição (Mini Editor) [12]

Antes de um parágrafo ser criado na janela de edição é necessário definir o tipo de "*Box*" apropriado para a especificação a partir do menu "*Edit*" como está ilustrado na Figura

A.2. Os tipos de "*Box's"* que podem ser utilizados são: um "*Schema Box" (esquema comum)*, "*Axiom Box" (axioma matemático),* ou *"Generic Box" (esquema genérico)* de acordo com a necessidade da especificação a ser validada. Neste momento o conhecimento bem detalhado da aplicação é que vai definir o tipo de "*Box*" a ser escolhido no menu "*Edit*".

Na Figura A.2, o menu *"Edit"* também apresenta as funções "*Clear*" que apaga uma certa especificação feita, "*Copy*" que copia um determinado dado de uma parágrafo ou o parágrafo inteiro, "*Cut*" que cola um determinado dado de uma parágrafo ou o parágrafo inteiro e "*Paste*" que cola um determinado dado de uma parágrafo ou o parágrafo inteiro.

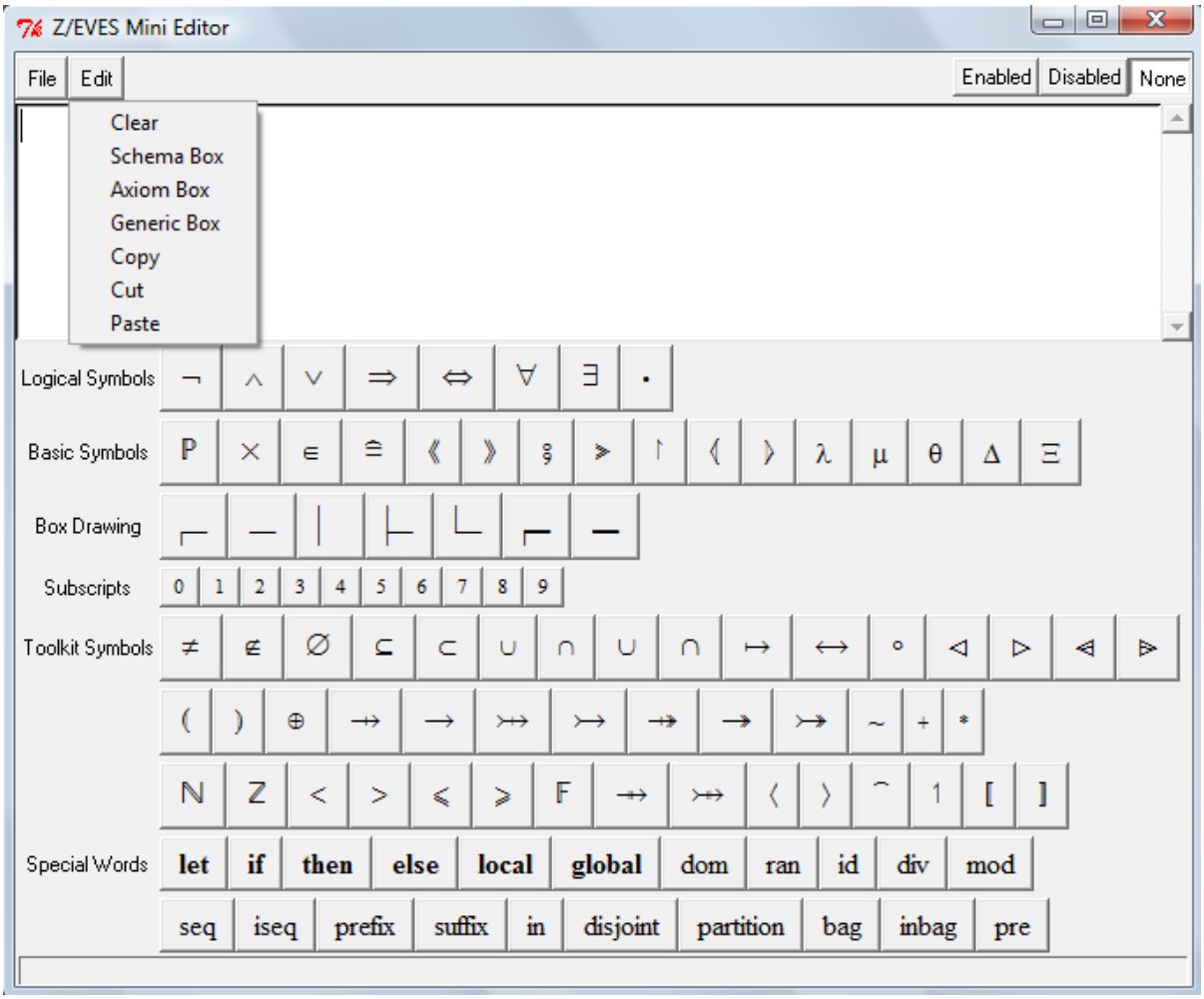

Figura A.2: O menu "*Edit*" [12]

A Figura A.3 descreve um parágrafo especificado como *"Schema Box"*, ou seja, um esquema comum.

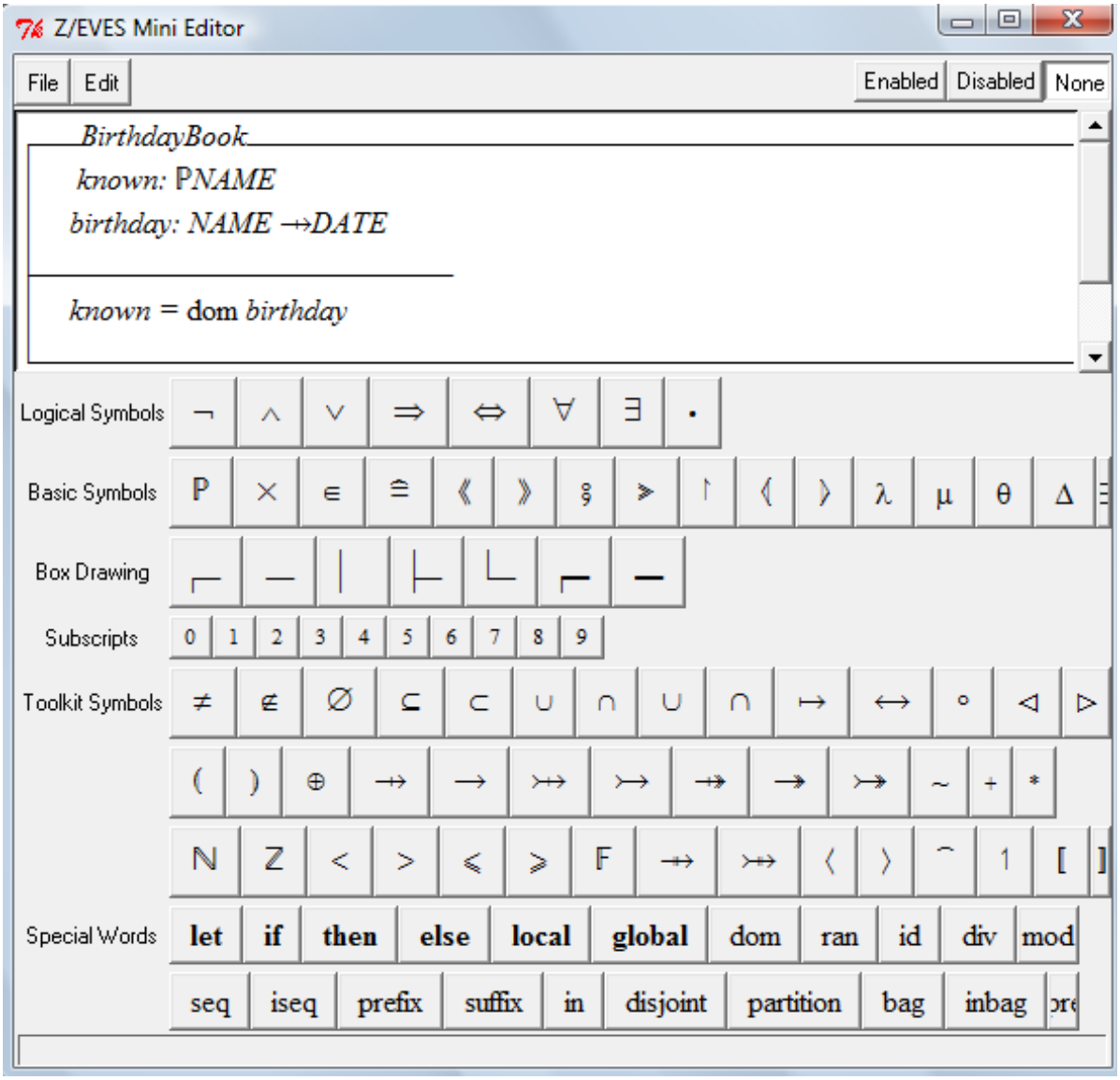

Figura A.3: Exemplo de um "*Schema Box*" [12]

A Figura A.4 descreve um parágrafo especificado como um *"Axiom Box"*, ou seja, um axioma matemático.

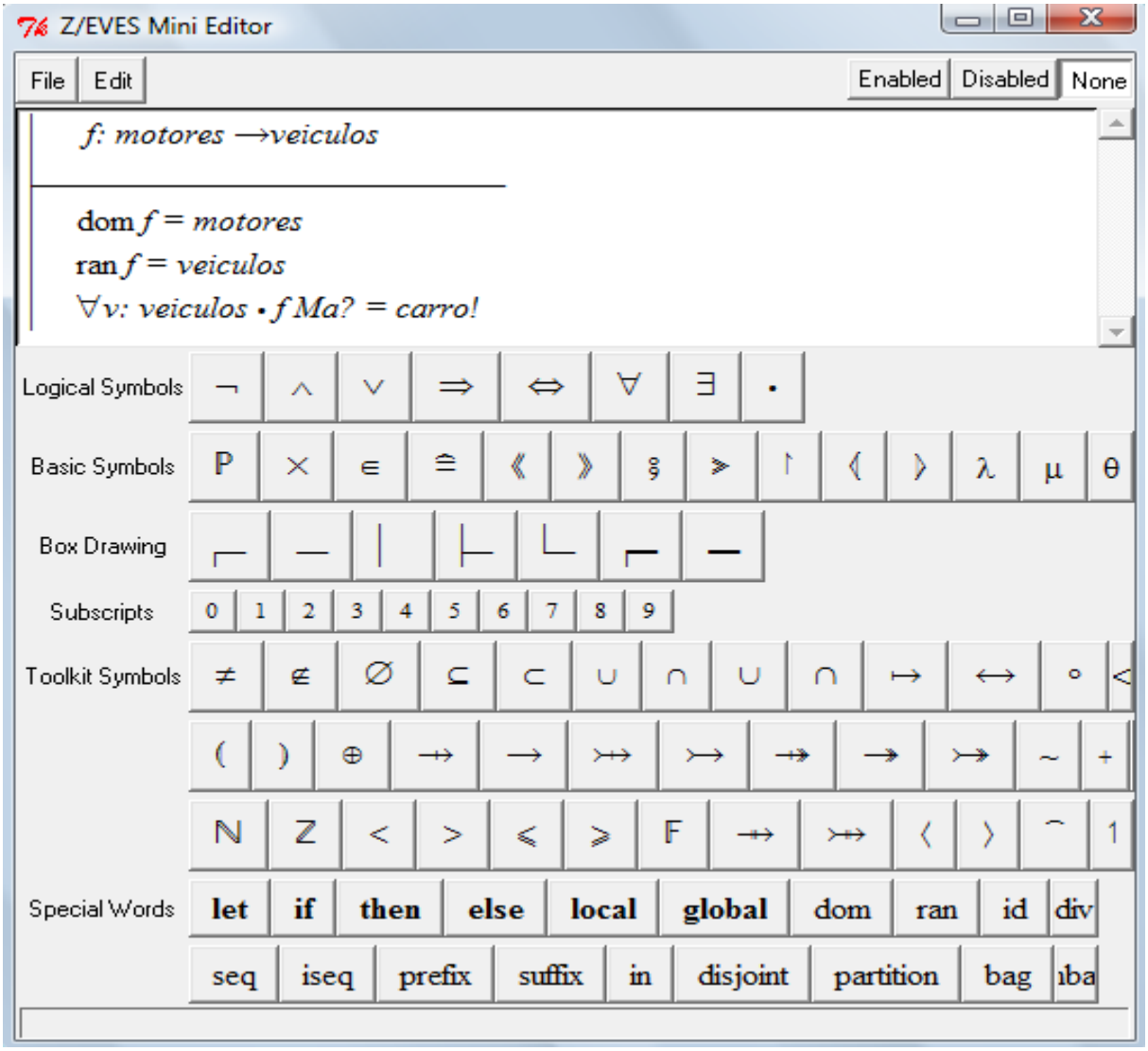

Figura A.4: Exemplo de um "*Axioma Box*" [12]

A Figura A.5 descreve um parágrafo especificado como "*Generic Box*", ou seja, como um esquema genérico.

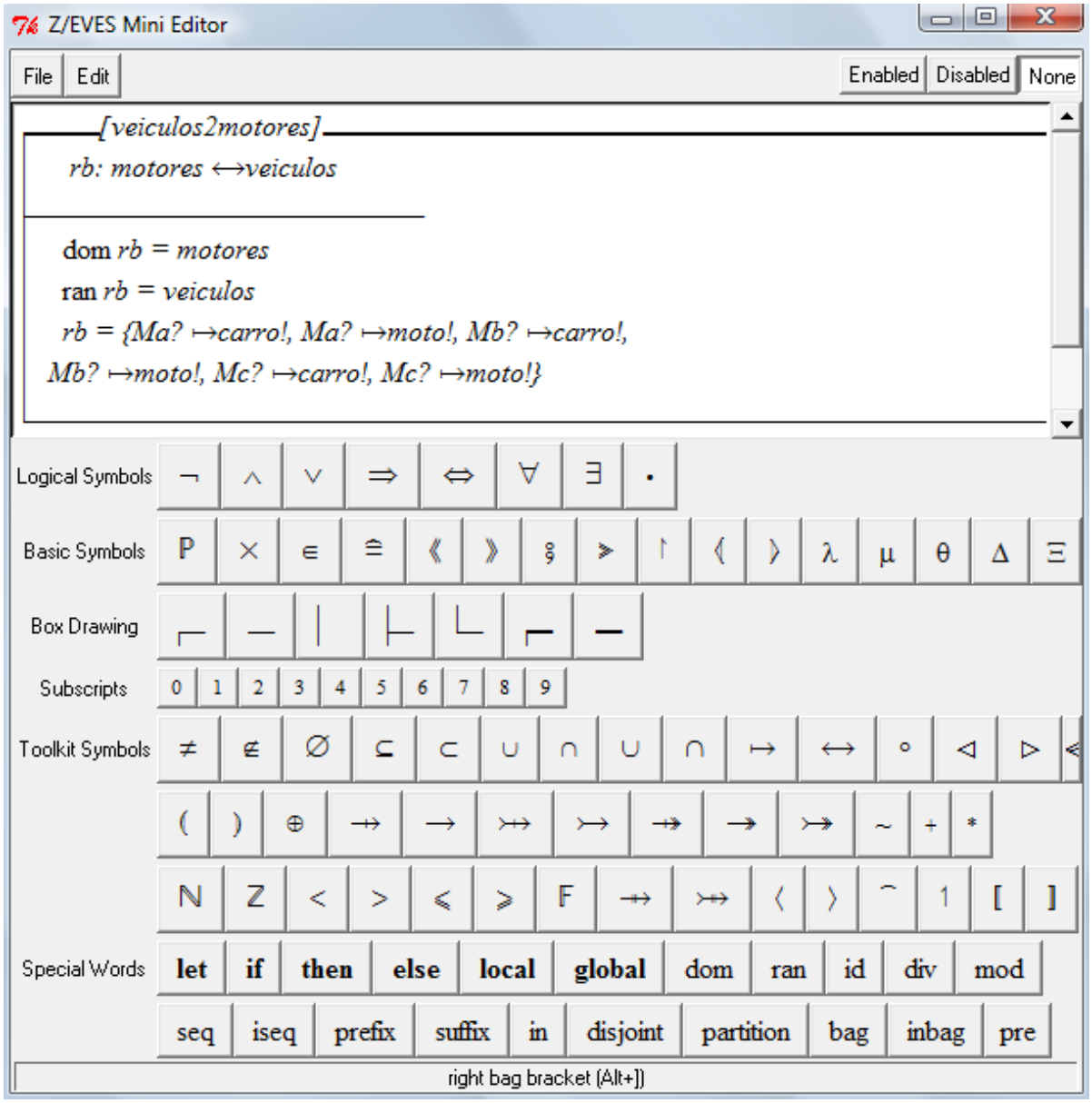

Figura A.5: Exemplo de um "*Generic Box*" [12]

Cada parágrafo da especificação pode ser criado e checado no mesmo momento, além da possibilidade de poder ser re-checado após uma alteração.

No *toolkit* existem regras de reescrita, lemas e teoremas usados durante a construção de uma prova. Existem teoremas no *toolkit* que por padrão são utilizados automaticamente pelo Z/EVES, mas na maioria das vezes é necessário guiar o provador a utilizar um teorema para que seja possível dar mais um passo na prova de alguma propriedade da especificação.

É necessário que se prove as obrigações de provas geradas pela ferramenta quando a especificação é inserida e também teoremas que representam propriedades do

sistema. Para auxiliar na automação dessas provas a ferramenta dá a possibilidade se criar "*rewriting rules"*, "*forward rules"* e "*assumption rules"*.