UNIVERSIDADE FEDERAL DO MARANHÃO

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE ELETRICIDADE

PABLO LUÍS CASTRO DE MATOS

UM *FRAMEWORK* DE SEGURANÇA BASEADO EM ENGENHARIA DIRIGIDA POR MODELOS PARA PLATAFORMAS DE COMPUTAÇÃO EM NUVEM: Uma Abordagem para Modelos SaaS

São Luís – MA
2015

PABLO LUÍS CASTRO DE MATOS

UM *FRAMEWORK* DE SEGURANÇA BASEADO EM ENGENHARIA DIRIGIDA POR MODELOS PARA PLATAFORMAS DE COMPUTAÇÃO EM NUVEM: Uma Abordagem para Modelos SaaS

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia de Eletricidade da Universidade Federal do Maranhão, para a obtenção do título de mestre em Engenharia de Eletricidade – Área de Concentração: Ciência da Computação.

Orientador: Ph. D. Zair Abdelouahab

Coorientador: Dr. Denivaldo Lopes

São Luís – MA
2015

Matos, Pablo Luís Castro de.

Um framework de segurança baseado em engenharia dirigida por modelos para plataformas de computação em nuvem: uma abordagem para modelos SaaS / Pablo Luís Castro de Mattos. — São Luís, 2015.

108 f.

Orientador: Zair Abdelouahab.

Coorientador: Denivaldo Lopes.

Dissertação (Mestrado) – Universidade Federal do Maranhão, Programa de Pós-Graduação em Engenharia de Eletricidade, 2015.

1. Computação em nuvem. 2. Framework de segurança. 3. Engenharia dirigida por modelos. 4. Modelo SaaS. I. Título.

CDU 004.771

UM FRAMEWORK DE SEGURANÇA BASEADO EM ENGENHARIA DIRIGIDA POR MODELOS PARA PLATAFORMAS DE COMPUTAÇÃO EM NUVEM: UMA ABORDAGEM PARA MODELOS SAAS

Pablo Luís Castro de Matos

Dissertação aprovada em 31 de agosto de 2015.

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

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

Profa. Karla Donato Fook, Dra.
(Membro da Banca Examinadora)

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

Dedico este trabalho aos meus pais José Luís Dias de Matos e Aldina Ferreira Castro de Matos.

RESUMO

O desenvolvimento e a utilização de *softwares* baseados em computação em nuvem têm conquistado cada vez mais destaque na atualidade. A oferta de SaaS (*Software as a Service*) se mostra uma tendência não apenas para as grandes empresas, mas também para as pequenas e médias, adquirindo espaço também na computação pessoal de forma transparente. Esta relativa popularização do serviço traz consigo muitos desafios no que se refere à segurança da informação manipulada pelos seus fornecedores e a vulnerabilidade de suas respectivas aplicações. Neste trabalho, propomos um *framework* de desenvolvimento de SaaS, fazendo uso da Engenharia Dirigida por Modelos (MDE) aliada a técnicas de fusão de modelos do domínio de segurança a modelos do domínio da aplicação. Esta abordagem envolve a utilização de técnicas de MDE para se alcançar tal adaptação e auxiliar na condução do processo de desenvolvimento do *software*. Através da adoção da abordagem MDE é possível realizar a junção de elementos de modelos diferentes, a partir de modelos fonte alcançando-se um modelo alvo pela utilização de técnicas de *weaving*. Um protótipo implementa o *framework* proposto e reutiliza as ferramentas *Mapping Tool for Model Driven Engineering* (MT4MDE) e *Semi-Automatic Matching Tool for Model Driven Engineering* (SAMT4MDE) na demonstração da metodologia usada. Os resultados demonstram a viabilidade e os benefícios da combinação de vários aspectos de segurança no processo de desenvolvimento de um SaaS.

Palavras-chave: Computação em Nuvem, Segurança, Engenharia Dirigida por Modelos, Modelo *Weaving*, Transformações de Modelos

ABSTRACT

The development and use of software based on cloud computing have been highlighted more and more nowadays. Software as a Service (SaaS) has been considered as a trend for small, medium and large companies, subtly acquiring presence in personal computing too. This service popularizing brings with it many challenges concerning to information security handled by their suppliers and the vulnerability of their applications. In this work, we propose a SaaS development framework by combining the Model-Driven Engineering (MDE) with merging techniques of domain-security models and domain-application model. This approach involves the use of MDE techniques for achieving such adaptation and assist in the software development process. By adopting the MDE approach, it is possible to combine elements of different models, from source models reaching a target model by using weaving techniques. A prototype implements the proposed framework and reuses the Mapping Tool for Model Driven Engineering (MT4MDE) and Semi-Automatic Matching Tool for Model Driven Engineering (SAMT4MDE) in order to demonstrate the used methodology. The results demonstrate the feasibility and benefits of combining several security aspects in the development process of SaaS.

Keywords: Cloud Computing, Security, Model-Driven Engineering, Weaving Model, Model Transformations

Agradecimentos

A Deus, autor e consumador da vida, por ter me permitido concluir mais essa etapa da minha vida acadêmica.

Ao meu orientador, professor Ph.D. Zair Abdelouahab, pela oportunidade concedida, pela orientação, dedicação e pelo conhecimento repassado.

Ao meu coorientador, professor Dr. Denivaldo Lopes, pela oportunidade concedida, pela orientação, pela paciência, compreensão, pelo conhecimento repassado, pelo apoio e incentivo ao longo deste período.

Ao programa de Pós-Graduação em Engenharia de Eletricidade da Universidade Federal do Maranhão, pela oportunidade de realização do curso.

A CAPES, pelo apoio financeiro através da concessão de bolsa de pesquisa, e ao CNPq e FAPEMA, pelo apoio concedido ao longo do mestrado.

Aos integrantes e ex-integrantes do LESERC, em especial Wesley Lima, Amanda Serra, Michael Costa, Steve Ataky e Gustavo Dominices que contribuíram direta ou indiretamente para realização desse trabalho.

Aos ex-companheiros de mestrado Gerson Lobato, Débora Stefanello, Helaine Cristina, Jéssica Bassani e Marcus Vinícius pelas contribuições, apoio e incentivo para enfrentar essa caminhada.

Aos meu pais José Luís Dias de Matos e Aldina Ferreira Castro de Matos, meus maiores incentivadores e melhores conselheiros durante toda a vida.

A minha noiva Danielle Medeiros Vidal, pelo apoio, incentivo e paciência durante todo o desenvolvimento deste trabalho.

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

Lista de Figuras

Lista de Tabelas

Lista de Siglas

ATL Atlas Transformation Language.

ATM Asynchronous Transfer Mode.

DDM Design Decision Model.

EBNF Extended Backus-Naur Form.

EMF Eclipse Modeling Framework.

EML Epsilon Merging Language.

GMF Graphical Modeling Framework.

HuaaS Humans as a Service.

IaaS Infrastructure as a Service.

MDA Model-Driven Architecture.

MDE Model-Driven Engineering.

MMT Mapping Modeling Tool.

MOF Meta Object Facility.

MT4MDE Mapping Tool for MDE.

NIST National Institute of Standards and Technology.

OHRS Online Hotel Reservation System.

OMG Object Management Group.

PaaS Platform as a Service.

PDM Platform Description Model.

PIM Platform Independent Model.

PSM Platform Specific Model.

RBAC Role-based Access Control.

SaaS Software as a Service.

SAMT4MDE Semi-Automatic Matching Tool for MDE.

SLA Service Level Agreement.

UML Unified Modeling Language.

WM Weaving Model.

WMM Weaving Metamodel.

WSDL Web Services Description Language.

XMI XML Metadata Interchange.

XML eXtensible Markup Language.

XSLT Extensible Stylesheet Language Transformations.

Sumário

1 Introdução

Neste capítulo, o contexto em que a dissertação foi desenvolvida, a problemática a ser tratada, a solução proposta e seus objetivos serão apresentados. A metodologia empregada para a execução deste trabalho de pesquisa será mostrada, além da apresentação dos demais capítulos desta dissertação.

1.1 Contexto

Atualmente, a computação em nuvem tem sido uma área bastante difundida na aplicação das soluções de negócio de grandes, médias e pequenas empresas, baseada em uma nova forma de fornecer computação, bem como em uma nova forma de se cobrar pela utilização da mesma. Este é o mais recente modelo de computação no campo da tecnologia de comunicação e informação, onde os recursos computacionais – *hardware*, *software*, ambiente de desenvolvimento e outras infraestruturas – são disponibilizados como serviços, remotamente sobre uma rede (intranet ou *Internet*) [1].

A computação em nuvem através da utilização de técnicas já conhecidas pela computação como, por exemplo, a virtualização, aliada a tecnologias *Web* modernas e a abstração de infraestruturas de TI, trouxe grandes benefícios aos seus utilizadores como escalabilidade e elasticidade, tendo como características principais o amplo acesso e o fornecimento de serviços sob demanda [2].

O NIST – *National Institute of Standards and Technology*, estabeleceu uma definição frequentemente citada para computação em nuvem, especificando quatro diferentes modelos de serviço [3], dentre eles o modelo SaaS – *Software as a Service*. Uma nuvem SaaS é uma plataforma de vários locatários que utiliza recursos comuns e uma única instância de ambos – o código de objeto de uma aplicação, bem como o banco de dados subjacente – para suportar vários clientes simultaneamente [4]. Um dos grandes pontos positivos do modelo SaaS está na versatilidade do acesso via *Internet*. Onde houver conectividade à rede, os sistemas podem ser acessados [5].

Da mesma forma que ocorre com os demais modelos de serviços em nuvem, muitos são os problemas de segurança relacionados a SaaS. Dentre eles podemos citar [5]: a gestão de identidades quanto a falta de recursos sofisticados aos provedores de serviço na nuvem no momento de integrar sua plataforma aos serviços de verificação de identidades localizados atrás dos *firewalls* da rede corporativa; a ausência de padrões de segurança específicos para a nuvem e consequente utilização de padrões pouco robustos com a finalidade de comprovar o controle dos provedores sobre os dados; e ainda, a falta de transparência quanto aos procedimentos e estratégias de segurança

adotadas pelos provedores. Porém, um ponto ainda não tão abordado no quesito segurança em ambientes SaaS diz respeito a vulnerabilidade dos ativos das empresas quanto a manipulação não autorizada dos dados por terceiros após a implementação e disponibilização de serviços SaaS.

No contexto de desenvolvimento de *software*, diversas abordagens podem ser utilizadas, dentre elas temos a Engenharia Dirigida por Modelos (*Model-Driven Engineering* – MDE). A MDE visa auxiliar no desenvolvimento, manutenção e evolução de aplicações diminuindo a complexidade de desenvolvimento de *software*. A Engenharia Dirigida por Modelos é uma abordagem que guia o processo de desenvolvimento de *software* de qualidade na construção e transformação de modelos. A MDE fornece meios para adaptar novas tecnologias com sistemas legados e permitir a integração entre diferentes tecnologias [6].

Dessa forma, a Engenharia Dirigida por Modelos é capaz de fornecer suporte ao desenvolvimento de serviços SaaS, podendo se utilizar de transformações entre modelos para criar modelos específicos para esta plataforma em nuvem, com o intuito de prover melhor qualidade do *software* desenvolvido.

1.2 Problemática

Há apenas dois pontos de entrada em um ambiente SaaS: o *front-end*, o qual os usuários utilizam; e o *back-end*, usado pelo fornecedor SaaS para manutenção e gerenciamento. Limitar a entrada elimina todas as maneiras das quais os dados são perdidos ou roubados. A entrada pelo front-end é sempre através de uma conexão de identidade segura e criptografada VPN e de acesso baseado em funções [7].

É desejável que o nível de segurança adotado pelos fornecedores de serviço em nuvem ofereça garantias de que o acesso aos serviços/dados de seus clientes (*front-end*) está restrito a pessoas que realmente tenham direitos previamente estabelecidos e legais sobre tais informações. Porém mais interessante ainda, é que independentemente da existência de falhas ou brechas que permitam o acesso de pessoas não autorizadas a informações privilegiadas, também sejam adotadas medidas de controle de acesso internamente a aplicação SaaS, bem como de detecção e prevenção de ataques sobre o sistema (*back-end*), evitando assim danos maiores não apenas aos bens de um cliente, mas também ao sistema como um todo, que atende uma ampla rede de clientes.

Acredita-se que a utilização de mecanismos de desenvolvimento de *software* baseados na metodologia da arquitetura dirigida a modelos pode automatizar consideravelmente o processo de concepção de sistemas mais

robustos, aliando aspectos de segurança de sistemas de TI para aplicação em domínios de nuvem. Tal ação minimizaria os efeitos de possíveis ataques e adicionaria um nível a mais de segurança às aplicações SaaS, consequentemente convergindo em benefícios aos utilizadores destes serviços.

O problema a ser pesquisado neste trabalho é a falta de padrão de segurança e a ausência de modelagem de segurança dos sistemas de *software*.

1.3 Solução Proposta

A integração de modelos é o ponto chave da solução proposta. Compreende a tarefa de mesclar elementos de dois ou mais modelos através de correspondências entre os elementos de seus respectivos metamodelos ou análise de características funcionais dos modelos. Tal tarefa será suportada pelo *framework* proposto, construído baseado no trabalho de D. Lopes [8] no qual se utiliza da abordagem MDE e geração de regras de transformação em modelos. O *framework* reutiliza e estende o trabalho desenvolvido por D. Lopes com o intuito de realizar a fusão de modelos de domínios distintos através da técnica de tecelagem (*weaving*). É utilizado o *Eclipse Modeling Framework* (EMF) [9] para a implementação dos conceitos da abordagem MDE. O EMF provisiona recursos de criação em nível de modelos, metamodelos, definições de transformação de modelos e a interface gráfica para interações com o especialista de domínio [10].

Para o *framework* proposto, o especialista deve abstrair inicialmente os modelos da aplicação e de segurança; os modelos devem estar em conformidade com seus respectivos metamodelos baseados em cada domínio. É aplicada a técnica de *weaving* a esses modelos através da ferramenta de tecelagem, gerando um modelo intermediário que combina elementos de ambos os modelos de entrada. Após análise do especialista, o modelo intermediário é então transformado através de um motor de transformação em um modelo de saída específico de plataforma (PSM) de serviço *web*. Por fim, este modelo por sua vez, é novamente transformado em código-fonte.

A fusão dos modelos de entrada em um único modelo de saída ocorre de acordo com a intervenção do especialista. Após a execução do *weaving*, o sujeito responsável por analisar e avaliar quais correspondências são válidas é o especialista, minimizando o problema de integrações inconsistentes.

O uso da abordagem MDE pode auxiliar no desenvolvimento de aplicações SaaS mais seguras. O fato de permitir a integração de aplicações de domínios diferentes para este propósito se mostra uma proposta bastante promissora; ao elevar a fase de desenvolvimento em nível de modelos, torna-se

bem mais fácil a manipulação das correspondências e consequentes transformações para se chegar ao código final.

O foco da solução proposta se concentra na integração de modelos de aplicações SaaS com aplicações do domínio de segurança, focando em autenticação e autorização. A motivação para o trabalho de pesquisa foi a melhoria do aspecto segurança nas ferramentas disponíveis atualmente e aplicações futuras, desenvolvidas para ambientes de nuvem. Portanto, é tratado neste trabalho apenas a integração de tais domínios especificamente.

1.4 Objetivos

O objetivo geral do trabalho de pesquisa é fornecer um modelo de desenvolvimento para sistemas de computação em nuvem em ambientes SaaS baseado nas técnicas da Engenharia Dirigida por Modelos aliando a isto aspectos de segurança.

1.4.1 Objetivos Específicos

Os objetivos específicos são os seguintes:

- Construção de um *framework* para realizar integração de modelos baseados na abordagem MDE;
- Construção de uma metodologia para desenvolvimento de aplicações SaaS através da separação de preocupações, abordando aspectos relativos à segurança;
- Aplicação do *framework* para integração de aplicações SaaS e aplicações do domínio de segurança;
- Aplicação de técnicas de *weaving* e *merging* no processo de correspondência entre modelos representando as aplicações;
- Definição de transformações a partir da fusão dos modelos utilizados;
- Desenvolvimento de uma aplicação de serviço *web* para validar a proposta e os resultados alcançados em uma plataforma SaaS.

1.5 Metodologia de pesquisa

A metodologia proposta para o desenvolvimento deste trabalho de pesquisa pode ser listada a seguir:

1. Pesquisa bibliográfica para coleta de informações de livros, artigos, teses, dissertações, periódicos, anais de congressos, tutoriais e *websites* (*IEEE, ACM, Wiley, SpringerVerlag*) sobre o estado da arte dos tópicos a seguir:

- Engenharia Dirigida por Modelos, incluindo linguagens de transformação, definição de transformação, especificação de correspondências [6][11][12];
- Correspondência de modelos [13][14][15];
- Integração com *weaving* [15][16][17];
- Computação em nuvem [18][19][20][21].

2. Proposta de construção de um *framework* para realizar integração de modelos baseados na abordagem MDE;
3. Uso do *Eclipse Modeling Framework* (EMF) [22] para construção de modelos e metamodelos das aplicações;
4. Uso do *Eclipse Modeling Project* (EMF) [22] para construção do *framework* de integração de modelos com um *plug-in* Eclipse;
5. Uso do ambiente de programação Eclipse [23] durante o desenvolvimento do trabalho proposto;
6. Adaptação e extensão das ferramentas MT4MDE e SAMT4MDE para suportar a integração de modelos através de *weaving*;
7. Desenvolvimento de uma aplicação de serviço *web* em uma plataforma SaaS, utilizando o *framework*, a metodologia e o protótipo propostos.

1.6 Apresentação da Dissertação

Este trabalho de dissertação está estruturado em 7 (sete) capítulos, como segue.

O primeiro Capítulo apresenta o contexto tecnológico no qual o tema do trabalho se encontra. É apresentada a problemática que motivou a pesquisa, descrevendo o cenário para o desenvolvimento de uma proposta de integração de aplicações de domínios diferentes, voltando-se para o aspecto da segurança em plataformas SaaS. São também apresentados neste primeiro capítulo, o objetivo geral, os objetivos específicos e a metodologia de pesquisa.

O segundo Capítulo traz a fundamentação teórica para direcionar e ambientar o leitor a respeito do desenvolvimento da solução. Neste capítulo são apresentados conceitos como: *Model-Driven Engineering* – MDE, *Model-Driven Architecture* – MDA, e *Eclipse Modeling Framework* – EMF; são descritos conceitos de modelos *weaving* e *merging*, demonstrando suas utilizações no processo de integração das aplicações através da abordagem MDE. E encerrando, é apresentada a plataforma de computação em nuvem SaaS, a qual é o campo de atuação onde a solução proposta está inserida.

No terceiro Capítulo, o estado da arte da pesquisa e os trabalhos relacionados as várias partes integrantes deste trabalho de pesquisa são apresentados, tendo como foco as técnicas de *weaving*, a abordagem MDE e aspectos de segurança para ambientes SaaS.

O quarto Capítulo apresenta a abordagem utilizada, bem como os metamodelos para suportar a integração das aplicações utilizadas no *framework*. São apresentadas a forma como serão integradas e as definições de transformação para criação de modelos intermediários que servirão de apoio ao processo de obtenção de modelos mais refinados.

O quinto Capítulo demonstra a composição da arquitetura e a implementação do *framework* de fusão de modelos através das ferramentas para suportar a integração entre eles.

O sexto Capítulo apresenta um exemplo ilustrativo e uma avaliação através de modelagens. O *framework* executa a metodologia para integrar as aplicações a fim de alcançar um modelo de aplicação com informações do domínio da aplicação SaaS e do domínio de segurança.

Encerrando a dissertação, o sétimo Capítulo apresenta as conclusões da pesquisa, as contribuições, as limitações e dificuldades encontradas, os resultados alcançados e os trabalhos futuros da pesquisa desenvolvida.

2 Fundamentação Teórica

Este capítulo apresenta a fundamentação teórica na qual se baseia este trabalho. Tal fundamentação se torna essencial para a melhor compreensão da solução proposta. Os conceitos da abordagem de desenvolvimento de *software* *Model-Driven Engineering* (MDE), a especificação *Model-Driven Architecture* (MDA) – um projeto padronizado pela *Object Management Group* (OMG), e a ferramenta de modelagem *Eclipse Modeling Framework* (EMF) são apresentados. Conceitos e técnicas de integração de modelos através de *weaving* e técnicas de fusão de modelos através de *merging* também são apresentados. Ainda neste capítulo, os conceitos e definições de Computação em Nuvem são apresentados, bem como suas características e classificações quanto ao tipo de serviço oferecido.

2.1 *Model-Driven Engineering* (MDE)

Não é de hoje que a constante evolução tecnológica tem ocasionado um considerável aumento do interesse por *softwares* de qualidade. Porém, um dos principais aspectos aos quais a qualidade de *software* está diretamente ligada diz respeito ao seu processo de desenvolvimento. Muitos são os esforços para se identificar ou aperfeiçoar os processos de desenvolvimento tradicionais mais recomendáveis para atender cada demanda de mercado. Como consequência à rápida crescente demanda está o aumento da complexidade do desenvolvimento de *software* para atendê-la.

Model-Driven Engineering (MDE) é uma abordagem onde o desenvolvimento de *software* é baseado em modelos [24]. Modelos são geralmente usados para representar situações do mundo real, podendo definir diferentes representações de um mesmo sistema [25]. Essa abordagem visa auxiliar na questão da complexidade do desenvolvimento de *software* de qualidade e em um menor tempo possível.

Modelos são concebidos através de uma linguagem de modelagem bem definida a qual compreende uma sintaxe e uma semântica que norteia a representação dos seus elementos e suas relações [26]. Um modelo está em conformidade com um metamodelo [17][25]. Neste contexto, um metamodelo é um modelo que define os elementos de outro modelo e como esses elementos se relacionam [27]. A seguir, são apresentados dois exemplos de abordagens baseadas em MDE: a especificação Arquitetura Dirigida por Modelos (MDA) da *Object Management Group* (OMG), e a ferramenta de modelagem *Eclipse Modeling Framework* (EMF).

2.1.1 *Model-Driven Architecture* (MDA)

No processo de desenvolvimento de *software* guiado pela MDA utiliza-se artefatos que são modelos formais capazes de serem entendidos por computadores [33][25]. Segundo a OMG, a modelagem pode ser dividida em quatro camadas hierárquicas representadas pelos seguintes níveis:

- Metametamodelo: camada na qual são estabelecidos conceitos para a criação e definição de metamodelos, que devem ser conforme seu metametamodelo. Como por exemplo, o metametamodelo MOF (*Meta Object Facility*) [28];
- Metamodelo: camada que representa como a próxima camada (de modelos) deve ser criada. Esta camada especifica quais elementos poderão ser criados na camada de modelo, bem como as relações entre elementos. Exemplo: metamodelo UML (*Unified Modeling Language*) [29];
- Modelo: nesta camada são definidos conceitos conforme a camada de metamodelo. Neste nível de representação são especificadas abstrações do mundo real, como por exemplo, uma classe carro que é composta por nome e marca. Essa camada especifica as instâncias de objetos carros com essas propriedades que irão compor a próxima camada;
- Objeto: camada que especifica objetos e características do modelo definido representando um conceito do mundo real em uma determinada plataforma. Por exemplo, uma instância da classe carro, o qual tem o nome ”Palio” e a marca “Fiat”.

É importante salientar também alguns outros conceitos que formam a MDA. Tais conceitos estão relacionados a definição de modelos (*Platform Independent Model* – PIM e *Platform Specific Model* – PSM) e de transformações entre modelos (Figura 2.1), conforme segue:

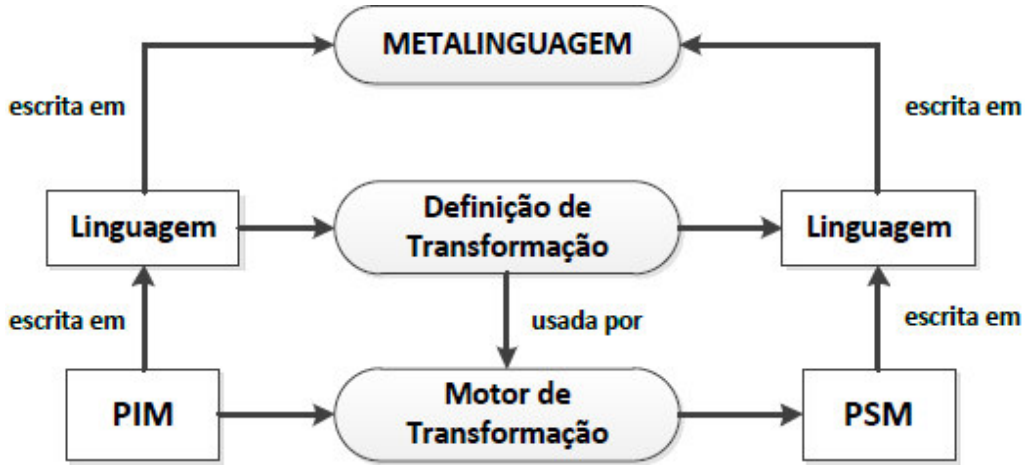


Figura 2.1: *Framework* básico MDA [12]

- ***Platform Independent Model* (PIM)**: modelo de alto nível de abstração, no qual são modelados aspectos e funcionalidades do sistema, porém sem contemplar nenhuma tecnologia de implementação específica para a solução;
- ***Platform Specific Model* (PSM)**: modelo concebido a partir de transformação do modelo PIM, com a definição de uma plataforma específica, levando em consideração detalhes da implementação de acordo com informações do PIM;
- **Definição de Transformação**: descrição de como elementos de um modelo fonte podem ser transformados em elementos de um modelo alvo, independentemente do nível de abstração dos modelos manipulados;
- **Motor de Transformação**: mecanismo responsável por executar a conversão de um modelo em outro através das definições de transformação que declaram como isso ocorre. O motor de transformação recebe um modelo de entrada, executa as definições de transformação e gera como saída um outro modelo [10].

O princípio base da MDA é o desenvolvimento de modelos independentes de plataforma (PIM), representando as funcionalidades de negócios de um sistema, e sua transformação para modelos específicos de plataforma (PSM) para uma determinada plataforma tecnológica [16]. Dessa maneira, o código-fonte é obtido após sucessivas transformações, resultante do processo de desenvolvimento.

A utilização da MDA se propõe a oferecer certas contribuições ao processo de desenvolvimento de *software* em aspectos de produtividade, portabilidade e interoperabilidade, como descrito a seguir [12]:

- **Produtividade**: utilizando-se a abordagem MDA, grande parte do código é gerado a partir das transformações do PIM para o PSM e do PSM para o código, dessa forma reduz-se consideravelmente o tempo de desenvolvimento;
- **Portabilidade**: através da MDA e das definições de transformações, um mesmo PIM pode ser transformado para diferentes PSMs, cada um deles representando diferentes plataformas;
- **Interoperabilidade**: a MDA possibilita que se estabeleçam relacionamentos entre diferentes PSMs gerados a partir de um mesmo PIM, permitindo dessa forma que conceitos de uma plataforma sejam convertidos em conceitos de outras plataformas.

2.1.2 *Eclipse Modeling Framework* (EMF)

O EMF [9] é um *framework* de modelagem para Eclipse utilizado para a implementação dos conceitos da abordagem MDE. Para tal, o EMF reúne em

um só conjunto tecnologias como Java [30], *eXtensible Markup Language* (XML) [31] e *Unified Modeling Language* (UML) [29], como ilustra a Figura 2.2 a seguir.

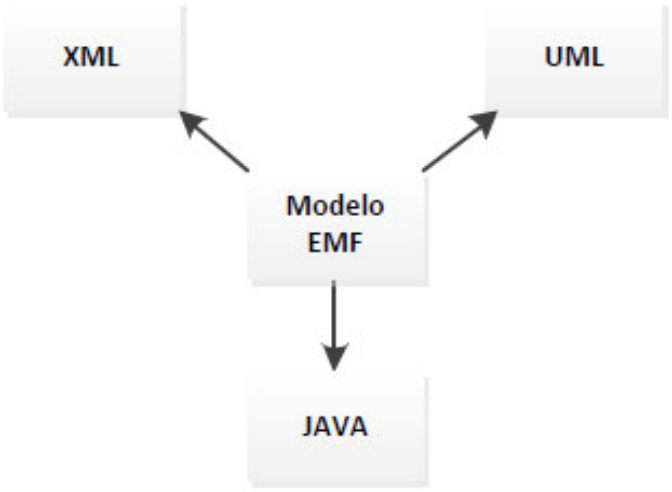


Figura 2.2: Tecnologias reunidas em EMF [9]

Neste trabalho o EMF foi utilizado na construção dos metamodelos, criação dos modelos fonte e modelos alvo, bem como no processo de transformação dos modelos. Os modelos em EMF podem ser representados de várias maneiras como por exemplo, através do formato *Ecore*, que é um subconjunto do metamodelo UML [9], em formato XML *Metadata Interchange* (XMI), padrão da OMG [32] para troca de informações baseado em XML, e através de Java, utilizando *@annotations*.

2.2 Conceitos de *Model Weaving*

Modelos *weaving* (tecelagem) podem ser utilizados como uma forma genérica de estabelecer correspondências entre elementos de modelos distintos; um modelo *weaving* pode também ser utilizado por uma linguagem de transformação de modelo para traduzir modelos fonte para modelos alvo [15]. O modelo *weaving* é um modelo que especifica diferentes tipos de mapeamentos entre elementos de metamodelos [17]. O processo de criação de modelos *weaving* é contido em uma operação de gerenciamento de modelos chamada *Match* [34].

Segundo Del Fabro et al. em [35] a metamodelagem é uma maneira conveniente para isolar preocupações de um sistema de tal forma que um metamodelo age como um filtro especificando o conjunto de preocupações que deve ser levado em consideração enquanto se cria um modelo. Dessa maneira,

com metamodelos separados, é possível lidar com domínios distintos ou diferentes níveis de abstração. Uma vez que essas preocupações foram expressas separadamente, elas podem ser novamente reagrupadas através de um mapeamento de correspondências, como no caso da criação de correspondências entre elementos de modelos que são utilizadas como base para executar diferentes operações.

Na técnica de tecelagem os modelos podem ser trançados automaticamente através de métodos de correspondências de elementos e estruturais, ou manualmente através de uma ferramenta apropriada que dê suporte para essa tarefa, como por exemplo a ferramenta *ATLAS Model Weaving* (AMW) [35].

O contexto operacional de uma operação de *weaving* de modelos é descrito pela Figura 2.3 a seguir. Esta operação produz um modelo de *weaving* (WM – *Weaving Model*) representando o mapeamento entre os metamodelos LeftMM e RightMM [35]; o modelo WM deve estar conforme um metamodelo específico (WMM – *Weaving Metamodel*).

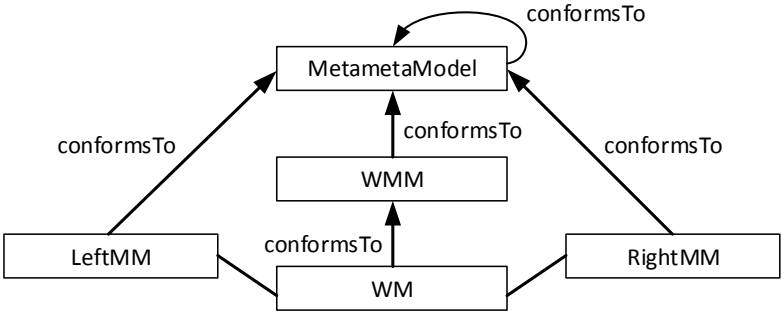


Figura 2.3: Operação de *Weaving* [35]

Dessa maneira, cada metamodelo pode representar uma linguagem específica de domínio lidando com uma visão particular de um sistema, enquanto as ligações de *weaving* possibilitam a descrição dos seus aspectos tanto separadamente quanto combinados [36]. Algumas questões devem ser levadas em consideração quanto à diferença entre transformações de modelos e operações de *weaving* de modelos [35]:

- *Arity*: normalmente uma transformação toma um modelo como entrada e produz um outro como saída, ainda que se trate de múltiplas entradas e saídas. Enquanto que o *weaving* toma dois modelos como entrada e um metamodelo *weaving*;
- Automaticidade: a transformação é uma operação automática enquanto que no *weaving* pode ser necessária a ajuda adicional de regras e métodos ou de orientação para auxiliar o usuário a realizar a operação;

- Variabilidade: a transformação está em conformidade com um metamodelo da linguagem de transformação, enquanto que no caso do *weaving* não há um metamodelo padrão, considerando que para cada aplicação diferente é necessário criar um novo metamodelo.

2.3 Conceitos de *Model Merging*

O *merging* de modelos é compreendido como a ação de fundir dois ou mais modelos mediante a especificação de correspondências previamente estabelecidas. Dessa forma, o *merging* vem a ser parte importante do processo de integração de modelos. Um modelo *weaving* em um cenário de *merging* indica como fundir elementos de dois modelos, elementos estes que serão fundidos como resultado de uma operação de *merge* [37]. A operação de *merge* retorna uma cópia de todos os objetos dos modelos de entrada, com exceção dos objetos dos modelos de entrada que são identificados como iguais, sendo tais objetos especificados como um único objeto no modelo de saída [34].

A OMG tem promovido na proposta de MDA a ideia da organização em "Y" como uma metodologia para desenvolvimento (conforme Figura 2.4), onde um modelo PIM deve ser mesclado com um modelo de descrição de plataforma (PDM) para produzir um modelo fundido PSM [35].

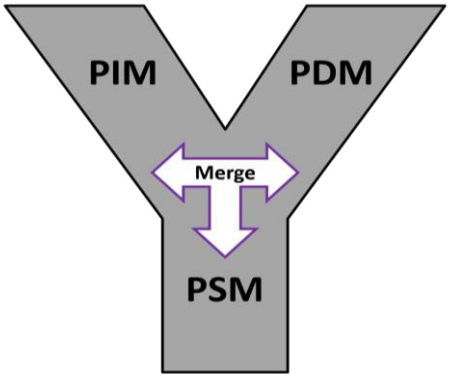


Figura 2.4: MDA e o processo de desenvolvimento em Y [35]

O problema de fundir modelos está relacionado ao fato de envolver muitas aplicações de metadados, tais como a integração de visões, a criação de esquemas para integração de dados, e a fusão de ontologias [13]. Para se alcançar o modelo resultante da fusão é necessário resolver conflitos que podem ocorrer no processo [13]:

- Conflitos de representação: são causados por representações conflitantes do mundo real, a nível de modelos. Por exemplo quando há duas representações de uma classe *Actor*, onde em um modelo A é representado *Name* por apenas um elemento "*ActorName*", e em um modelo B é representado por dois elementos, "*FirstName*" e "*LastName*". Se faz necessária a intervenção manual do usuário para solucionar tais conflitos;
- Conflitos de metamodelos: são conflitos causados pelas restrições a nível de metamodelos. Por exemplo quando determinados conceitos dos metamodelos de entrada não podem ser importados para o resultado do *merge*. Tais conflitos podem ser resolvidos utilizando-se um outro operador após o *merge*;
- Conflitos fundamentais: são causados pela violação de restrições a nível de metametamodelos. Ocorre por exemplo quando um elemento resultante de uma fusão tem mais de um único tipo. Conflitos fundamentais devem ser solucionados por *merge* visto que os operadores subsequentes esperam que o resultado da fusão seja um modelo bem-definido para ser possível manipulá-los.

2.4 Computação em Nuvem

O termo nuvem foi usado pela primeira vez no início dos anos 90 para se referir a grandes redes ATM (*Asynchronous Transfer Mode*). A computação em nuvem começou de fato no início deste século, com o advento dos *Web Services* da Amazon [38]. Atualmente, a computação em nuvem tem sido uma área bastante difundida na aplicação das soluções de negócio de grandes, médias e pequenas empresas. Baseada em uma nova forma de se fornecer computação, bem como em uma nova forma de se cobrar pela utilização da mesma.

Este é o modelo de computação no campo da tecnologia de comunicação e informação, onde os recursos computacionais – *hardware*, *software*, ambiente de desenvolvimento e outras infraestruturas – são disponibilizadas como serviços, remotamente sobre uma rede (intranet ou *Internet*) [39]. Através do agrupamento de recursos (armazenamento, processamento, memória, rede, etc.) é possível fornecer desde uma infraestrutura básica de *hardware*, para desempenhar tarefas comuns, até a entrega de um sistema de *software* completo que se adequa dinamicamente a demanda que lhe for exigida em determinado espaço de tempo.

A computação em nuvem através da utilização de técnicas já conhecidas pela computação como, por exemplo, a virtualização, aliada a tecnologias *Web* modernas e a abstração de infraestruturas de TI, trouxe grandes benefícios aos seus utilizadores como escalabilidade e elasticidade, tendo como características

principais o amplo acesso e o fornecimento de serviços sob demanda [40]. Vale ressaltar também a transferência da responsabilidade e preocupação com o melhor gerenciamento e administração de recursos computacionais, que deixa de ser até então do cliente e passa a ser do fornecedor de tais serviços.

O NIST – Instituto Nacional de Padrões e Tecnologia estabeleceu uma definição frequentemente citada para computação em nuvem, especificando cinco características principais, quatro diferentes modelos de serviço, e três diferentes modelos de desenvolvimento [41]. Segundo o NIST, os principais aspectos que caracterizam a computação em nuvem são: fornecimento de serviços sob demanda, amplo acesso à rede, agrupamento de recursos, elasticidade, e a capacidade de mensurar a qualidade do serviço oferecido. Quanto aos modelos de desenvolvimento cita-se o modelo público, privado e híbrido, dos quais dizem respeito à forma como se disponibiliza o serviço além da forma como este serviço é operado.

Ainda de acordo com a definição do NIST, dentre os modelos de serviços fornecidos pela computação em nuvem, apresentam-se a IaaS – Infraestrutura como serviço, PaaS – Plataforma como serviço, SaaS – Software como serviço, e ainda o HuaaS – Humano como serviço, conforme ilustra a Figura 2.5 a seguir.

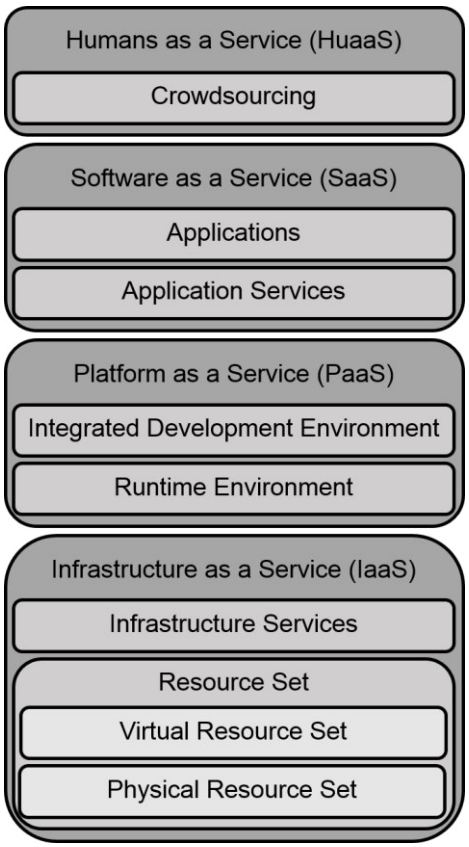


Figura 2.5: Diagrama da arquitetura *Cloud* [41]

Os modelos de serviços estão relacionados ao nível de abstração e entrega dos serviços, onde IaaS fornece recursos de *hardware* físico e recursos virtuais como instâncias virtuais através de tecnologias de virtualização, PaaS fornece recursos de ambientes de programação e de execução, onde *softwares* proprietários podem ser desenvolvidos, SaaS fornece aplicações de *software* em nuvem que tratam diretamente com o usuário final, isentando os clientes da necessidade de instalar o *software* localmente bem como de fornecer os recursos necessário, e HuaaS onde um grupo de recursos humanos usam a *Internet* para realizar tarefas de diferentes âmbitos e níveis de complexidade para um cliente [41].

Uma nuvem SaaS é uma plataforma de vários locatários que utiliza recursos comuns e uma única instância de ambos – o código de objeto de uma aplicação, bem como o banco de dados subjacente – para suportar vários clientes simultaneamente [5]. Um dos grandes pontos positivos do modelo SaaS está na versatilidade do acesso via *Internet*. Onde houver conectividade à rede, os sistemas podem ser acessados [5].

Da mesma forma que ocorre com os demais modelos de serviços em nuvem, são muitos os problemas de segurança relacionados a SaaS. Dentre eles, podemos citar [5]: a gestão de identidades no que diz respeito a falta de recursos sofisticados aos provedores de serviço na nuvem no momento de integrar sua plataforma aos serviços de verificação de identidades localizados atrás dos *firewalls* da rede corporativa; a ausência de padrões de segurança específicos para a nuvem e consequente utilização de padrões pouco robustos com a finalidade de comprovar o controle dos provedores sobre os dados; a falta de transparência quanto aos procedimentos e estratégias de segurança adotadas pelos provedores; a questão da mobilidade no acesso que traz a vantagem da conveniência em detrimento da segurança por não haver a garantia de que o ponto de conexão é seguro; e ainda o problema de localização dos dados no que se refere a regulamentações que dão garantia de que dados confidenciais estão de fato localizados em determinados territórios. Pode-se observar também o risco de violação de dados devido a possíveis erros de configuração. Para garantir que seus dados estão bem cuidados, uma empresa deve garantir que seu fornecedor de serviços escreva exatamente como seus dados de clientes serão protegidos, em seu contrato de Acordo de Nível de Serviço (SLA) [42].

2.5 Síntese

Neste capítulo, uma visão holística dos principais conceitos e tecnologias utilizadas no desenvolvimento deste trabalho de pesquisa foi apresentada. A finalidade deste capítulo foi de situar o leitor sobre os temas relacionados

visando uma melhor compreensão do *framework* proposto para suportar a integração de aplicações SaaS com aplicações do contexto de segurança.

Foram apresentados os conceitos da abordagem de desenvolvimento de *software* MDE bem como suas características, sua importância e suas aplicações. Dois exemplos de utilização de MDE, a arquitetura MDA e a ferramenta EMF também foram apresentados. Ambas abordagens foram similarmente conceituadas e caracterizadas; tais tecnologias foram utilizadas na construção do *framework* proposto.

Neste capítulo, os conceitos e características de *Model Weaving* e *Model Merging*, processos fundamentais para a integração das aplicações conforme proposto também foram abordados. Por fim, foram apresentados os conceitos de Computação em Nuvem, caracterizando as infraestruturas de acordo com o tipo de serviço oferecido pela nuvem, ressaltando dentre eles a infraestrutura SaaS, sobre a qual a solução proposta do trabalho de pesquisa foi desenvolvida.

3 Estado da Arte

Este capítulo tem como objetivo apresentar as principais abordagens existentes na literatura incluindo pesquisas para realizar transformações de modelos no contexto de MDE. Algumas informações complementares sobre especificação de correspondência, definição de transformação, e linguagens de transformação de modelos são apresentadas.

Foram selecionados trabalhos que abordam o processo de desenvolvimento de *software* utilizando MDA, além de trabalhos que tratam da utilização dos modelos de *merging* e modelos de *weaving*, e também soluções que abordam MDA para desenvolvimento de aplicações de *software* em nuvem.

Os trabalhos relacionados são analisados e comparados segundo alguns critérios estabelecidos, promovendo um maior embasamento para o desenvolvimento da pesquisa aqui proposta.

3.1 Especificação de correspondência e definição de transformação

O processo de transformação de modelos envolve duas etapas que, segundo D. Lopes em [43], devem ser claramente separadas: a especificação de correspondência e a definição de transformação; sendo que a primeira precedendo a segunda. A especificação de correspondência, ou especificação de mapeamento, envolve a definição de relacionamentos estabelecidos entre dois metamodelos; por exemplo, as correspondências entre um metamodelo para a construção de um PIM e outro metamodelo para a construção de um PSM. Enquanto que a definição de transformação determina regras para transformar elementos de um metamodelo fonte em elementos de um metamodelo alvo, utilizando-se uma linguagem de transformação.

Um modelo de correspondência gera um modelo de transformação, que por sua vez é a base de um programa de transformação [44]. No trabalho de D. Lopes [43] é apresentada uma arquitetura para transformação de modelos, como descrito na Figura 3.1 a seguir, onde pode se perceber a separação entre os modelos de correspondência e de transformação.

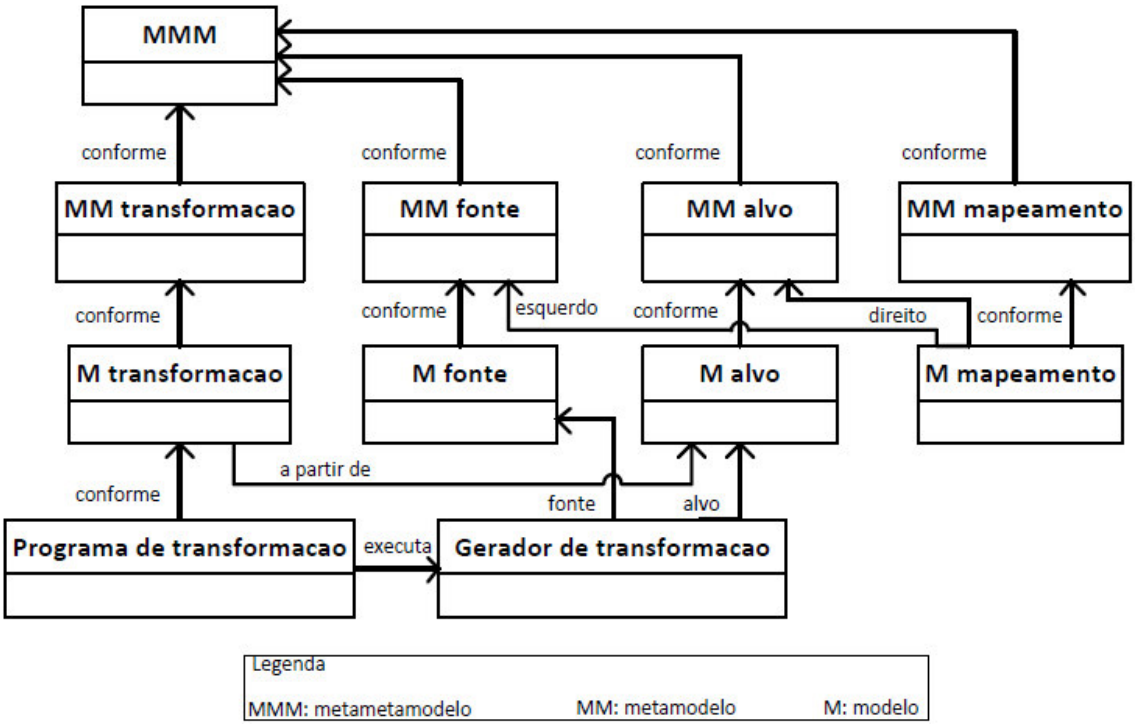


Figura 3.1: Uma proposta de arquitetura para transformação de modelos [43]

A arquitetura é constituída das seguintes entidades:

- MMM (metametamodelo): por exemplo, MOF ou Ecore;
- MM fonte e MM alvo (metamodelo fonte e metamodelo alvo): por exemplo, o metamodelo UML;
- M fonte e M alvo (modelo fonte e modelo alvo): por exemplo, um modelo de um sistema de reservas de hotel modelado em UML que posteriormente deve ser transformado em um modelo de WSDL [46];
- MM mapeamento (metamodelo de especificação de correspondência): utiliza uma linguagem para modelagem de correspondências entre os elementos de um metamodelo fonte e os elementos de um metamodelo alvo;
- M mapeamento (modelo de especificação de correspondência): modelo onde as correspondências entre dois metamodelos estão armazenadas;
- MM transformação (metamodelo de transformação): baseado em uma linguagem de transformação que permite a criação exata de transformações de um modelo fonte em um modelo alvo;
- M transformação (modelo de transformação): modelo que descreve como elementos de um metamodelo fonte são transformados em elementos de um metamodelo alvo;
- Programa de transformação: um programa executável para realizar a transformação de um modelo fonte em um modelo alvo.

Existem muitas linguagens para transformação de modelos, como por exemplo a MOF QVT da OMG [28], o *MOFScript* do *Eclipse Project* [45], e a ATL da Universidade de Nantes [11], sendo esta última a linguagem utilizada para o desenvolvimento do trabalho de dissertação. A linguagem ATL é utilizada em um ambiente na forma de *plug-in* para o Eclipse. Através da linguagem ATL é possível realizar transformações tanto modelo-a-modelo quanto modelo-a-texto, manipulando regras que descrevem correspondências entre modelos e produzindo elementos de um modelo alvo a partir de modelos fonte.

No trabalho de D. Lopes [43] um metamodelo de correspondência também é proposto para dar suporte à especificação de correspondência definida na arquitetura proposta para transformação de modelos. A Figura 3.2 a seguir ilustra este metamodelo de correspondência.

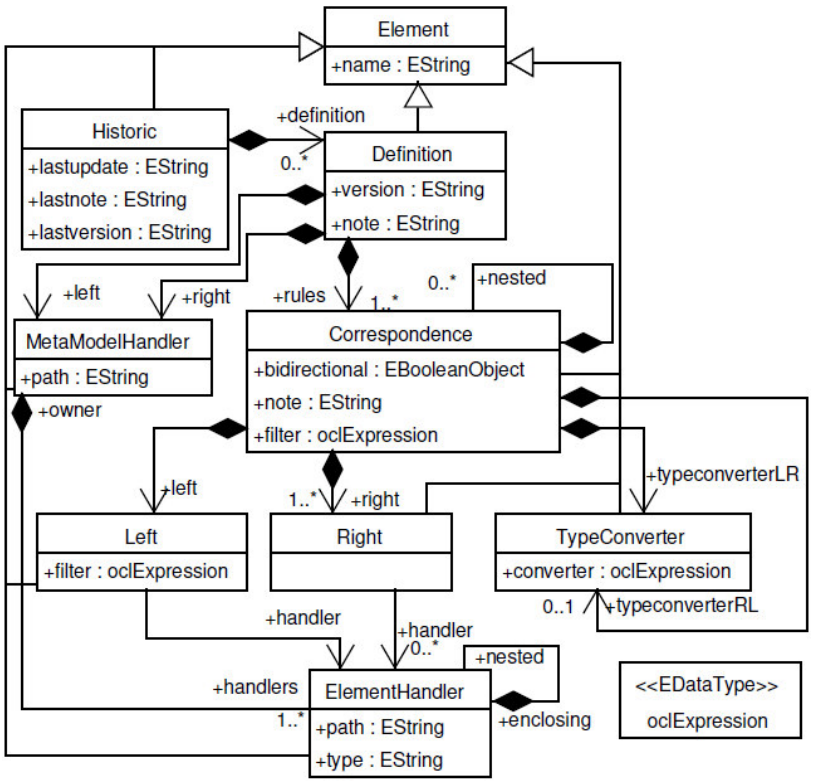


Figura 3.2: Metamodelo de especificação de correspondência [43]

O metamodelo é constituído dos seguintes elementos:

- Element: é uma generalização para outros elementos;
- Historic: mantém um histórico de especificação de correspondências e um conjunto de definições que representam as diferentes opções de correspondência;
- Definition: é o elemento principal que contém toda a correspondência entra as definições da esquerda e da direita;

- Correspondence: elemento que permite a especificação de correspondência entre dois ou mais elementos dos metamodelos da esquerda e da direita;
- MetaModelHandler: elemento que permite a navegação nos metamodelos participantes da especificação de correspondência;
- ElementHandler: elemento que permite o acesso aos elementos mapeados sem alterá-los;
- Left: identifica um elemento à esquerda no metamodelo de correspondência;
- Right: identifica um ou mais elementos à direita no metamodelo de correspondência.

Neste trabalho de D. Lopes [43] é implementada a ferramenta *Mapping Modeling Tool* (MMT) para validar o metamodelo de especificação de correspondência. A ferramenta foi concebida com a utilização do projeto *Eclipse Modeling Framework* (EMF) e é aplicada na geração de definição de transformações em ATL de um sistema de *software* definido em UML para WSDL [46].

3.2 Trabalhos relacionados

Nesta seção, trabalhos relacionados ao tema abordado são apresentados, nos quais foram detectadas várias iniciativas relacionadas a pontos diversos que envolveram e conduziram o processo de desenvolvimento como um todo da abordagem de segurança proposta.

3.2.1 Towards a new software development process for MDA [16]

Belangour et al. propõem em [16] um novo processo de desenvolvimento chamado M2T (MDA™ 2 Tracks). Neste processo de desenvolvimento é fornecida uma implementação da abordagem MDA baseada em um ciclo de desenvolvimento em forma de Y, no qual o ramo (ou faixa) da esquerda do ciclo Y corresponde ao modelo independente de plataforma (PIM) enquanto o ramo direito corresponde ao modelo de descrição de plataforma (PDM) representando a plataforma de destino. A evolução dos dois ramos corresponde a um conjunto de transformações. Durante a combinação dos dois modelos são capturados os recursos de mapeamento entre os ramos superiores para se produzir então um modelo específico de plataforma (PSM). A Figura 3.3 a seguir descreve o processo de desenvolvimento em Y do método M2T.

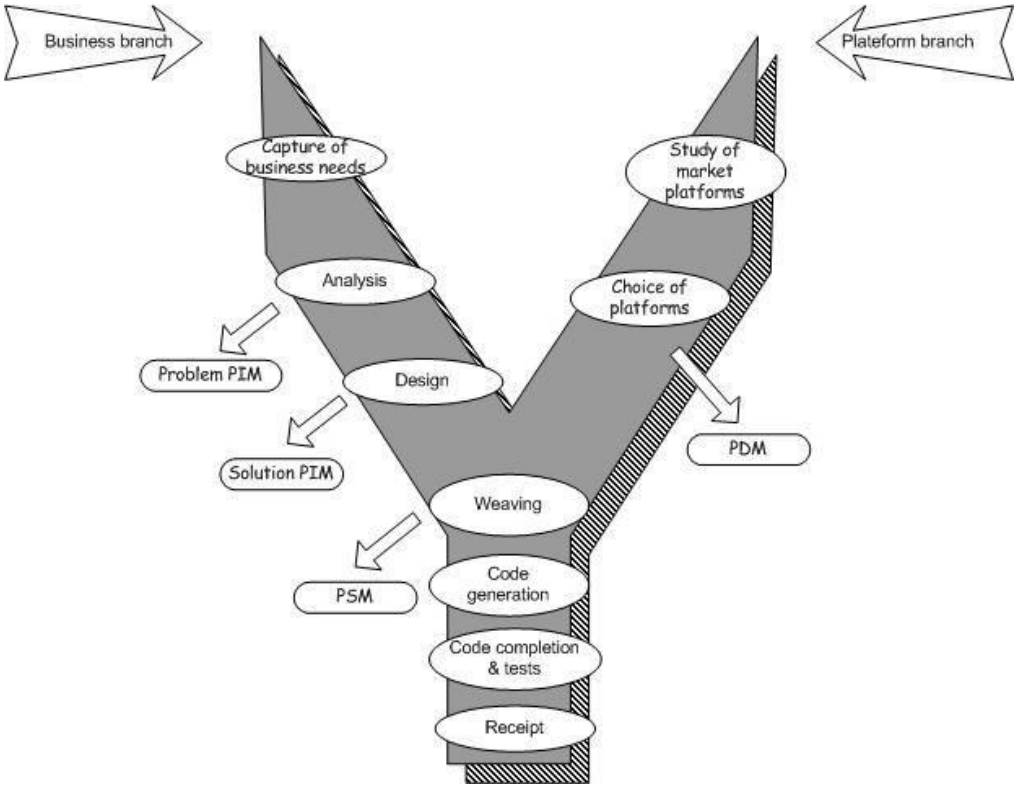


Figura 3.3: Processo de desenvolvimento em Y do método M2T [16]

Como pode-se observar na Figura 3.3 o processo M2T parte da fase de análise como ponto de partida e segue uma abordagem baseada na integração de MDA™ como sua metodologia principal. Nessa abordagem reúnem-se as fases habituais de desenvolvimento de *software* com as transformações de PIM para PSM providas pela MDA.

A combinação do PIM e do PDM é posteriormente guiada por um Modelo de Decisão de *Design* (DDM) que captura os recursos de mapeamento entre os dois ramos superiores do "Y" para produzir um modelo específico de plataforma (PSM) [16]. A atividade do ramo da direita é limitada a escolha da plataforma da aplicação que o PIM irá se ligar após um estudo das plataformas disponíveis no mercado. Essa plataforma é chamada de modelo PDM e é necessária para que a transformação ocorra.

A fase de *weaving* ocorre em duas etapas [16]:

- Definição do Modelo de Decisão de Design (DDM): referente à construção de um modelo de decisões de *design* conforme o metamodelo DDM. Tal modelo permite vincular os elementos do PIM com os recursos do PDM. Conforme o metamodelo DDM esses vínculos podem ser de substituição, utilização ou extensão de ligações;

- Execução da transformação: o objetivo dessa fase é gerar o PSM da aplicação realizando o *weaving* automático entre PIM e PDM.

O *weaving* entre o PIM e o PDM representando uma dada plataforma supõe que o usuário tem um conhecimento razoável da plataforma alvo para ser possível tomar boas decisões de *weaving*, limitando-se à escolha da plataforma da aplicação que o PIM irá se ligar.

3.2.2 UMLX: A graphical transformation language for MDA [47]

No trabalho [47] Willink também aborda o modelo de transformações em forma de Y. Seu conteúdo cita a utilização de modelos como uma linguagem de programação no âmbito da MDA, através da exploração do XMI (*XML Metadata Interchange*) para o intercâmbio de modelo e o XSLT (*Extensible Stylesheet Language Transformations*) para a transformação de modelo. Neste artigo é descrita uma linguagem de transformação gráfica, que envolve apenas pequenas extensões para UML, mas constitui uma linguagem de alto nível para as transformações.

Willink [47] primeiramente apresenta as extensões baseadas nas notações padrões UML para *schema* para suportar as transformações. São apresentados conceitos para definições de *schema* e instâncias dessas definições, bem como exemplos de transformações *schema* para *schema*. Em seguida são apresentados conceitos da abordagem MDA, modelos PIM, PSM e PDM, bem como transformações baseadas na forma Y onde é possível fundir ao PIM informação extra de plataforma através do PDM conforme ilustra a Figura 3.4 a seguir.

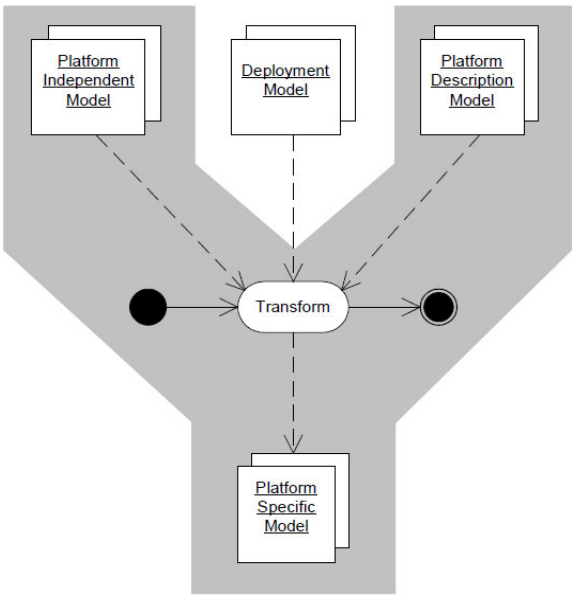


Figura 3.4: O formato Y em um diagrama de atividade UML [16]

Para se preservar a reusabilidade do PIM e manter o relacionamento entre PIM e PSM não se deve alterar o modelo PIM, entretanto o modelo PDM também pode ser reutilizado incluindo descrições de [16]:

- Sistemas do tipo *assembler*/linguagem;
- Interfaces de *driver*/*hardware*;
- Conjunto de recursos componente/sistema operacional/instruções;
- Protocolos de comunicação;
- Conectividade de rede;
- Recursos de bibliotecas convencionais;
- Recursos de bibliotecas *novel*;
- Ferramentas.

Após essas etapas o autor apresenta uma transformação a nível de compilador baseada em MDA. É apresentada a notação UMLX e algumas de suas transformações a nível de compilador a compilador. UMLX é uma linguagem de transformação gráfica que se integra com UML como um mapeamento entre *schema*, e por ser uma linguagem declarativa oferece a possibilidade de suportar otimizações poderosas.

3.2.3 Merging Models with the Epsilon Merging Language (EML) [48]

Kolovos et al. em [48] desenvolvem um trabalho focado em operações de validação e transformação de modelo. Neste trabalho são discutidos os requisitos especiais do modelo de fusão (*merging*) e é introduzida a EML (*Epsilon Merging Language*), uma linguagem baseada em regras, com suporte a ferramentas para a mesclagem de diversos metamodelos e tecnologias. Pesquisas já existentes [13][49] demonstram que modelos de *merging* podem ser decompostos em quatro fases distintas: comparação, verificação de conformidade, fusão e reconciliação (ou reestruturação).

Na fase de comparação são identificadas as correspondências entre elementos equivalentes dos modelos fonte, dessa maneira elementos duplicados não são reproduzidos no modelo resultante da fusão. Na fase de verificação de conformidade os elementos que são identificados como correspondentes na fase anterior são analisados para checar a conformidade entre eles com o objetivo de identificar potenciais conflitos que tornariam a fusão inviável. Na fase de fusão são observados dois pontos fracos de duas entre várias abordagens propostas; nos métodos avaliados só foi abordada a questão da fusão de modelos do mesmo metamodelo, outro ponto é a utilização de um algoritmo de fusão inflexível que não fornece meios para estender ou personalizar a sua lógica. Em seguida, na fase de reconciliação são removidas as inconsistências que precisam ser corrigidas para refinar o modelo a fim de alcançar a sua forma final.

EML é construída sobre a *Extensible Platform for Specification of Integrated Languages for Model mOdel management* (Epsilon), plataforma que fornece infraestrutura essencial para implementar linguagens de gerenciamento de modelos de tarefas específicas, como por exemplo, transformação de modelo, fusão de modelo e linguagens de validação de modelos. Os autores em [48] apresentam dois cenários de uso da EML, onde em um é realizada a fusão de um PIM (em UML) com diferentes PDMs para obter diferentes PSMs, e em outro é realizada a fusão entre dois modelos complementares em UML.

No primeiro cenário o PDM contém apenas uma metaclasse chamada *PrimitiveTypeMapping* com dois atributos (*independent* e *specific*) que é utilizada para definir mapeamentos entre os tipos de plataformas independentes e específicas [48]. A fusão é obtida com a especificação EML exibida na Listagem 3.1 a seguir.

Listagem 3.1: Especificação de fusão de PIM com PDM [48]

```
1  rule PimTypeMapping
2        match pimType : PIM!Class
3        with mapping : PDM!PrimitiveTypeMapping {
4
5  guard {
6        return pimType.stereotype.exists(s|s.name = 'primitive');
7  }
8
9  compare {
10       return pimType.name = mapping.independent;
11 }
12 }
13
14 auto rule PimTypeToPsmType
15       merge pimType : PIM!Class
16       with mapping : PDM!PrimitiveTypeMapping
17       into psmType : PSM!Class {
18
19       psmType.name := mapping.specific;
20 }
```

Na regra de correspondência *PimTypeMapping*, um tipo primitivo do PIM é declarado para corresponder com um mapeamento (*PrimitiveTypeMapping*) se o valor *name* do tipo primitivo for igual ao valor do atributo *independent* do mapeamento. Na regra de fusão *PimTypeToPsmType*, pares correspondentes de tipos primitivos do PIM e mapeamentos do PDM são mesclados para criar os tipos específicos de plataforma no PSM.

No segundo cenário, dois modelos UML que foram desenvolvidos independentes um do outro foram mesclados, através da especificação apresentada na Listagem 3.2. Como os modelos fonte tem o mesmo metamodelo com o modelo alvo, foi usada a estratégia na qual elementos correspondentes da mesma metaclasse poderiam ser fundidas automaticamente, e os elementos dos modelos fonte que não fossem correspondentes no modelo oposto poderiam ser inteiramente copiados no modelo alvo.

Listagem 3.2: Fusão parcial de UML [48] (fragmento)

```
1  abstract rule ModelElements
2        match left : Left!ModelElement
3        with right : Right!ModelElement {
4
5  compare {
6        return (left.name = right.name and left.namespace.matches
7  (right.namespace));
8  }
9  }
10
11 auto rule ModelWithModel
12       merge left : Left!Model
13       with right : Right!Model
14       into merged : Merged!Model {
15 merged.name := left.name + ' and ' + right.name;
16 }
```

O corpo da regra *ModelElements* define que para dois elementos de modelo se corresponderem seus *names* e seus *namespaces* devem ser iguais. A invocação da operação *matches()* irá resultar na chamada de outra regra, *Package* ou *Models* dependendo de onde o elemento de modelo é contido. Na regra *ModelWithModel* é definido que ao invés de receber o nome do modelo da esquerda, o modelo fundido deve ter um nome igual a concatenação dos nomes dos dois modelos separados por uma *string* "*and*".

3.2.4 Applying Generic Model Management to Data Mapping [15]

O trabalho apresentado por Del Fabro et al. [15] aborda uma solução capaz de controlar o equilíbrio entre genericidade, expressividade e eficiência dos mapeamentos, devido às limitações causadas pelo uso de determinada linguagem de mapeamento e que torna o gerenciamento deste difícil. É ilustrado o problema da troca de dados entre sistemas de biblioteca para padronização de formatos de catálogos. São apresentados dois cenários: no primeiro são estabelecidos mapeamentos de correspondência como igualdade, chave estrangeira e aninhamento, em um cenário de mapeamento entre esquema relacional e esquema XML; no segundo são estabelecidos mapeamentos como equivalência, igualdade, aninhamento e ordenamento para integrar um esquema XML a uma ontologia.

Na solução apresentada é definido o modelo de tecelagem (*weaving*) como uma forma genérica para estabelecer correspondências entre elementos; dessa forma, modelos *weaving* podem ser utilizados por uma linguagem de transformação de modelo para traduzir os modelos fonte para modelos alvo. A operação de *weaving* de modelo é implementada pelo operador *Weave*, conforme descrito na Listagem 3.3 a seguir.

Listagem 3.3: Algoritmo para mesclar dois modelos [15]

```
1  Weave (M1, M2, MMw, Mw)
2    If Mw is null
3     Mw = createWeavingModel(MMw);
4   for all ei in M1 do
5   begin
6    for all mej in MMw do
7    begin
8     corresp = SearchCorresp (mej, ei, M1, M2);
9     for all ek in Mw
10     if not exists ek with corresp
11     begin
12      mnew = Create(mej, corresp);
13      Mw = add ( mnew );
14     end
15   end
16  end
17 return Mw;
```

O algoritmo *Weave* primeiramente cria um modelo *weaving* conforme ao *MMw*. Depois, para cada elemento e_i de M1, e todo me_j de *MMw*, é realizada uma busca por elementos correspondentes em M1 ou M2, retornando as correspondências encontradas. A busca é realizada na função *SearchCorresp*. Essa função não é genérica, logo, necessita ser modificada para manipular qualquer estrutura diferente definida por cada me_j. As correspondências retornadas são utilizadas para criar o elemento *mnew*, o qual os associa de acordo com a estrutura de me_j. Esse elemento é chamado de *weaving link*.

Na publicação [17], os princípios de MDE são utilizados para automatizar o processo de integração de dados através da geração de transformações entre as várias visões em que os complexos metamodelos podem ser decompostos. Para este propósito são gerados modelos *weaving* entre as diferentes visões usando heurísticas de correspondência e produzidos transformações de modelos a partir do modelo *weaving*.

3.2.5 Automated Verification of Role-based Access Control Security Models Recovered from Dynamic Web Applications [50]

Alalfi et al. em [50] apresentaram uma abordagem e uma ferramenta para analisar modelos de segurança RBAC (*Role-based Access Control*) automaticamente recuperados de aplicações *web* dinâmicas existentes. A abordagem é baseada na Engenharia Dirigida por Modelos (MDE) para suportar a verificação e testes de propriedades de segurança. A abordagem inicia transformando-se o modelo em um modelo formal baseado em Prolog e posteriormente checando para verificar a conformidade da aplicação com as propriedades específicas de segurança de controle de acesso.

O trabalho [50] apresenta o metamodelo SecureUML (conforme Figura 3.5), que é uma implementação da abordagem Segurança Dirigida a Modelos (uma especialização de MDA). SecureUML integra claramente os aspectos de segurança aos modelos de aplicação e fornece suporte para a transformação de modelo.

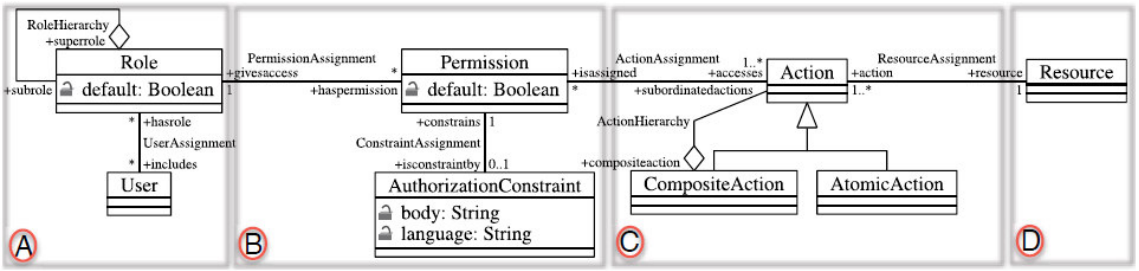


Figura 3.5: Metamodelo SecureUML [50]

A sintaxe abstrata do SecureUML é baseada no metamodelo RBAC, que define o controle de acesso baseado em funções. De acordo com o metamodelo SecureUML é possível definir funções para determinados usuários, permissões de ações para determinadas funções de usuários, e recursos os quais serão manipulados conforme as permissões estabelecidas para cada função. A abordagem voltada para o controle de acesso baseado em funções com foco em autenticação e autorização foi uma das contribuições utilizadas na concepção da ideia deste trabalho de dissertação.

3.2.6 Cloud SaaS: Models and Transformation [4]

No trabalho [4], Sharma e Sood ilustram uma abordagem baseada em MDA para o desenvolvimento de aplicações de *software* em nuvem, tornando-as mais robustas, flexíveis e ágeis, na sequência de evolução das tecnologias, através da adoção de uma metodologia de desenvolvimento de *software* onde os efeitos indesejáveis da mudança de tecnologia em aplicações de *software* são mínimos.

O desenvolvimento de uma aplicação SaaS baseado em MDA permite definir serviços de s*oftware* em nuvem independente da tecnologia, exercendo um papel significativo na melhoria da qualidade desses serviços. É utilizado como exemplo ilustrativo para desenvolver uma aplicação SaaS, o *Online Hotel Reservation System* (OHRS), conforme Figura 3.6. O OHRS é uma aplicação de *software* executando como um serviço na nuvem podendo ser utilizada por empresas de pequeno ou médio porte em uma base de pagamento. O OHRS também é utilizado como exemplo ilustrativo para o desenvolvimento deste trabalho de dissertação que envolve a construção de um *framework* de

segurança baseado em Engenharia Dirigida por Modelos para a plataforma SaaS.

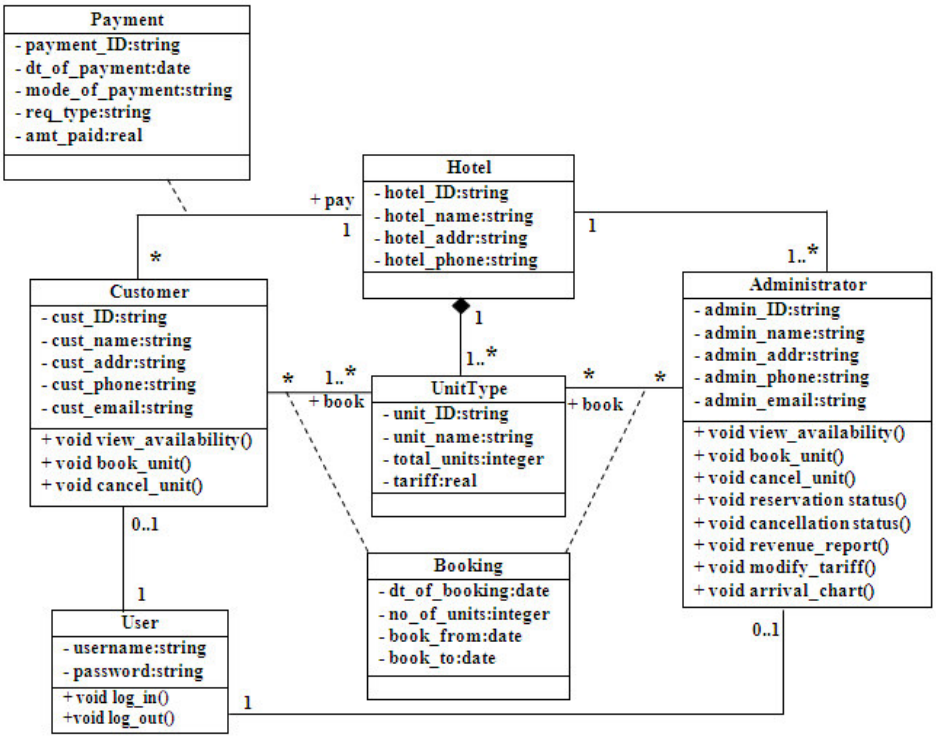


Figura 3.6: Modelo Independente de Plataforma para OHRS [4]

Os autores apresentam ainda alguns dos principais benefícios em unir computação em nuvem e MDA. Cita-se a capacidade de a partir de um mesmo PIM se gerar diferentes PSMs conforme as tecnologias vão evoluindo, e ainda descartando a necessidade de redefinir a estrutura, comportamento e funcionalidade do aplicativo para diferentes plataformas tecnológicas específicas graças as ferramentas de transformação [4]. Sharma e Sood também citam o trabalho [1] onde a adoção da abordagem MDA conduz a automação de transformações modelo a modelo e modelo a código, contemplando ambientes SaaS.

3.3 Análise dos trabalhos relacionados

Nesta seção, os trabalhos relacionados foram analisados destacando os pontos relevantes para se fazer uma comparação com o trabalho de pesquisa proposto nesta dissertação. Os seguintes aspectos foram utilizados na avaliação:

1. Utilização da abordagem MDE para lidar com o desenvolvimento de *software*;
2. Baseia-se no processo de desenvolvimento em Y;
3. Utilização do mecanismo de *weaving* e/ou *merging* para estabelecer mapeamentos;
4. Permissão de uso de ferramentas para definição de correspondências;
5. Geração automática de códigos-fonte.

Conforme apresentado na Tabela 3.1, todos os trabalhos pesquisados apresentaram aspectos da utilização da abordagem MDE no desenvolvimento de suas soluções.

Belangour et al. em [16] utilizaram a abordagem MDA baseando-se no processo de desenvolvimento em Y para representação dos modelos independente e específico de plataforma. O processo de mapeamento utiliza o mecanismo de *weaving* entre modelos dividindo-se em duas etapas, que é conduzido por um Modelo de Decisão de Plataforma (DDM) e pela execução das transformações. O artigo não descreve a geração de códigos fontes a partir do modelo PSM gerado.

Willink em [47] também utiliza o processo de desenvolvimento em Y baseado em MDA, sendo que o trabalho não utiliza os mecanismos de entrelaçamento, mas sim de fusão para acrescentar informação extra de plataforma através dos modelos PDM. O autor apresenta a notação UMLX como uma linguagem de transformação gráfica a nível de compilador baseada em pequenas extensões para UML, porém não contempla a geração de códigos fonte.

Kolovos et al. em [48] não utilizam a abordagem de desenvolvimento em Y, porém introduz uma linguagem baseada em regras chamada EML (*Epsilon Merging Language*) com suporte a ferramentas que realizam a fusão de diferentes metamodelos e tecnologias. No trabalho foi demonstrada a fusão de um modelo PIM a diferentes PDMs definindo-se mapeamentos entre plataformas independentes e específicas. Apresenta-se também a especificação de correspondências para fusão de modelos UML independentes, sem demonstrar a geração de códigos a partir dessas transformações.

Em [15], Del Fabro et al. apresentam uma definição do modelo *weaving* e o operador *Weave* para se estabelecer mapeamentos de correspondência entre

esquema relacional e esquema XML, e para realizar a integração de um esquema XML a uma ontologia. O modelo *weaving* descrito no trabalho pode ser utilizado por uma linguagem de transformação de modelos para traduzir modelos fonte em modelos alvo. Os autores não abordam a etapa de geração de códigos.

Alalfi et al. no trabalho [50] apresentam uma abordagem baseada na MDE para analisar modelos de segurança RBAC (*Role-based Access Control*), oferecendo suporte à verificação e testes das propriedades de segurança. Os autores apresentam o metamodelo SecureUML que é baseado no metamodelo RBAC e a ferramenta SecureUML2Prolog para recuperar modelos de segurança de aplicações *web* gerando consultas em Prolog para checar os aspectos importantes de segurança na aplicação alvo.

Sharma e Sood em [4] abordam o desenvolvimento de aplicações de *software* em nuvem baseado em MDA, citando a automação de transformações modelo a modelo e modelo a código em ambientes SaaS. No trabalho é apresentado o mapeamento entre um modelo independente de SaaS em UML e um modelo específico relacional. O artigo não descreve detalhes da geração de código a partir das transformações apresentadas, contudo ressalta os benefícios da utilização da MDA em computação em nuvem.

Cada um dos trabalhos relacionados apresenta abordagens distintas baseadas em MDE, enquanto que a ausência da abordagem de aspectos de segurança para desenvolvimento de aplicações SaaS demonstra a necessidade de desenvolvimento de mecanismos e ferramentas para dar suporte a esta proposta. Portanto, a proposta de um *framework* para desenvolvimento de SaaS com a preocupação voltada para aspectos de segurança pode trazer diversos benefícios neste âmbito de pesquisa.

Tabela 3.1: Análise dos trabalhos relacionados

Itens Avaliados	**Towards a new software development process for MDA [16]**	**UMLX: A graphical transformation language for MDA [47]**	**Merging Models with the Epsilon Merging Language (EML) [48]**	**Applying Generic Model Management to Data Mapping [15]**	**Automated Verification of Role-based Access Control Security Models Recovered from Dynamic Web Applications [50]**	**Cloud SaaS: Models and Transformation [4]**
Utiliza a abordagem MDE no desenvolvimento de *software*	Sim	Sim	Sim	Sim	Sim	Sim
Baseado no processo de desenvolvimento em Y	Sim	Sim	Não	Não	Não	Não
Utiliza o mecanismo de weaving e/ou merging para mapeamento	Sim	Sim	Sim	Sim	Não	Não
Permite o uso de ferramentas de correspondência	Sim	Sim	Sim	Sim	Sim	Sim
Geração automática de códigos fonte	Não	Não	Não	Não	Sim	Não

3.4 Síntese

Este capítulo apresentou o Estado da Arte, relatando pesquisas acadêmicas desenvolvidas relacionadas a abordagem MDA, modelagem, transformações de modelos, linguagens de transformação, modelos *weaving*, *merging*, controle de acesso baseado em funções, e computação em nuvem. Nota-se com o material pesquisado a grande vertente que envolve a computação em nuvem com os benefícios proporcionados pela modelagem e transformações de modelos. E mais ainda, percebe-se uma lacuna no que diz respeito ao que se pode proporcionar no quesito melhoria de qualidade de *software* observando o aspecto de segurança de aplicações SaaS.

Seis trabalhos dos temas anteriormente citados foram descritos. Cada tema pode ser encaixado em um processo único de desenvolvimento para atender a demanda de concepção de aplicações SaaS mais robustas no quesito segurança. Uma análise dos trabalhos relacionados foi apresentada abordando aspectos relevantes para fazer uma comparação com o trabalho de pesquisa proposto nesta dissertação.

Com base nos referidos trabalhos mencionados, o presente trabalho de dissertação será direcionado à abordagem de aspectos relativos aos princípios da MDA com aplicação para segurança em computação em nuvem em ambientes SaaS, utilizando-se o ciclo de desenvolvimento em forma de Y e os modelos de *weaving* e *merging* para estabelecer a combinação e correspondências entre elementos de modelo fonte PIM do SaaS correspondente, e modelo PDM referente a plataforma de destino com nível de segurança aplicado, bem como suas transformações para o modelo alvo, PSM.

4 Uma abordagem baseada em MDE para suportar o desenvolvimento de SaaS seguros

Neste capítulo, é descrita uma abordagem baseada em MDE para suportar o desenvolvimento de SaaS com a separação de preocupações (*concerns*), e adoção de aspectos de segurança. A abordagem proposta visa satisfazer alguns objetivos de segurança para aplicações SaaS conforme mostra a Tabela 4.1 adiante. Modelos de segurança são propostos com o objetivo de definir aspectos como autenticação, controle de acesso, criptografia e persistência. No trabalho de dissertação, abordaremos a questão da segurança de aplicações SaaS voltada para o aspecto do controle de acesso focando em autenticação e autorização, ilustrando dessa forma a viabilidade de integrar sistemas de contextos tecnológicos diferentes.

Tabela 4.1: Objetivos de segurança para aplicações SaaS (baseado em [51])

Objetivos de Segurança	Requisitos de Segurança	Tipo de Política
Confidencialidade e integridade da informação armazenada	O acesso aos recursos, suas propriedades e operações deve ser controlado	Políticas de controle de acesso baseadas em funções de usuários
	Para utilizar o sistema o usuário deve ser autenticado	Políticas de autenticação
Confidencialidade e integridade das informações transmitidas	O conteúdo das mensagens trocadas deve ser mantido em confidencialidade	Políticas de confidencialidade a nível de mensagem
	O conteúdo das mensagens deve ser verificado se não foi modificado durante sua transmissão	
Confidencialidade da informação interceptada	O conteúdo das mensagens trocadas deve ser transformado da sua forma original para outra forma ilegível	Políticas de criptografia de dados
Persistência da informação armazenada	O conteúdo deve ser armazenado persistentemente	Políticas de serialização de dados

Neste capítulo, a metodologia que direciona o desenvolvimento do *framework* para suportar a solução proposta e os metamodelos utilizados na abordagem também são apresentados; são apresentados os metamodelos

necessários para a integração dos modelos distintos, além das técnicas utilizadas de *merging* e *weaving* para obter o modelo mesclado do sistema de negócios e de segurança.

4.1 MDE e *Weaving*

Nesta seção, apresentamos o cenário que aplicamos como base para propor a abordagem. Normalmente, o processo de *merging* de modelos ocorre em cenários de aplicações similares, com o objetivo de fundir dados de diferentes representações destes sistemas, ou integrar visões de um mesmo sistema ou partes dele. Nós propomos a utilização de MDE e *weaving* para unir modelo de negócios (PIM – *Platform Independent Model*) e modelos de segurança (PDM – *Platform Description Model*) conforme ilustra a Figura 4.1. Os modelos PIM e PDM são entrelaçados através do modelo *Weaving*, e uma definição de transformação toma um PIM, um PDM e um modelo *Weaving* (WM) como entrada e gera como saída um modelo específico de plataforma (PSM).

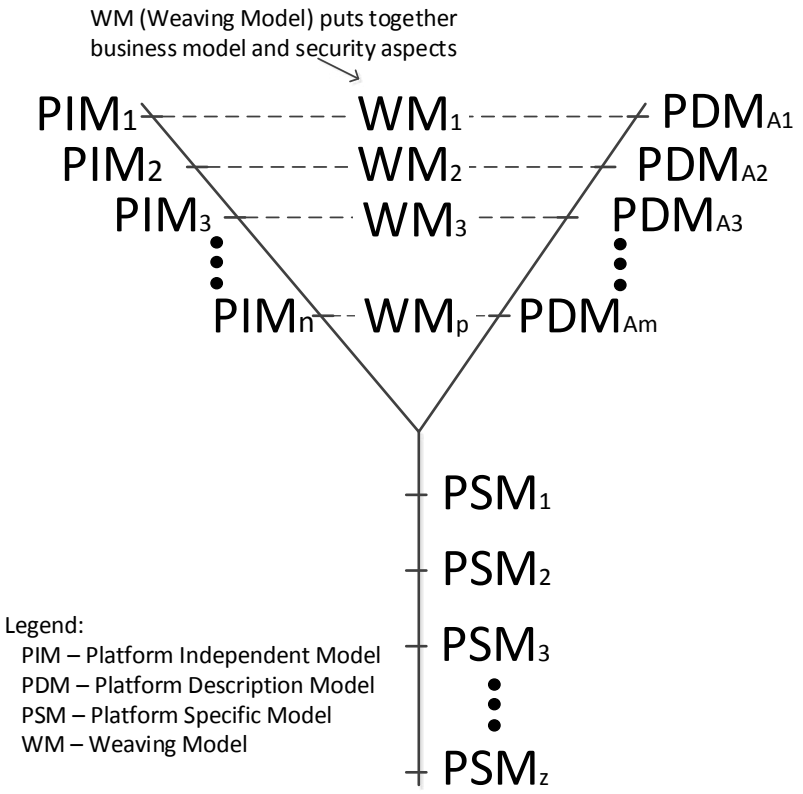


Figura 4.1: MDE, modelos *weaving*, e o processo de desenvolvimento em Y (baseado em [52])

Semelhante a proposição que Bézivin et al. apresentam em [52], podemos tomar como exemplo um cenário onde um modelo PIM está conforme um

metamodelo UML, um modelo PDM está conforme um metamodelo que inclui aspectos de segurança, um modelo WM está conforme um metamodelo *Weaving*, e um PSM está conforme um metamodelo de *Web Service*.

4.2 O *framework* FD3S para desenvolvimento de SaaS Seguro

Nesta seção, é apresentado o *framework* FD3S (*Framework* para Desenvolvimento de SaaS Seguro) para suportar o desenvolvimento de aplicações da plataforma SaaS baseado em MDE e *Weaving*, com a separação de preocupações. A figura 4.2 descreve o *framework* em maiores detalhes.

No *framework*, tem-se um PIM conforme um metamodelo UML, e um PDM conforme o metamodelo de segurança proposto. No centro, uma ferramenta de *weaving* auxilia na criação e edição de um modelo *weaving* que é conforme um metamodelo *Weaving*. Uma vez que o modelo *weaving* está criado, um Gerador de Modelo Intermediário toma como entradas, um modelo PIM, um modelo *Weaving* e um modelo PDM, e fornece como saída um modelo intermediário. O último modelo mescla as informações do PIM, do modelo *Weaving* e do PDM. Um modelo intermediário é necessário antes da geração de um PSM pois contém as informações de fusão do PIM, modelo *Weaving* e PDM. Depois disso, um mecanismo de transformação toma como entrada uma definição de transformação modelo-a-modelo, um modelo intermediário e fornece como saída um PSM. Essa definição de transformação modelo-a-modelo se relaciona acima com um metamodelo intermediário e abaixo com outro metamodelo como por exemplo um metamodelo *Web Service*. Em seguida, um mecanismo de transformação toma como entrada outra definição de transformação modelo-a-modelo e um PSM abstrato e fornece como saída um PSM concreto. Essa definição de transformação modelo-a-modelo se relaciona acima com um metamodelo *Web Service* e abaixo com um metamodelo Java e o padrão para aplicação da API Java de *Web Service*. Após isso, um outro mecanismo de transformação toma como entrada uma definição de transformação modelo-a-texto e um PSM concreto e fornece como saída um código fonte. Essa definição de transformação modelo-a-texto relaciona um *Web Service* Java e API Java para *Web Service* e uma gramática EBNF da linguagem de programação Java.

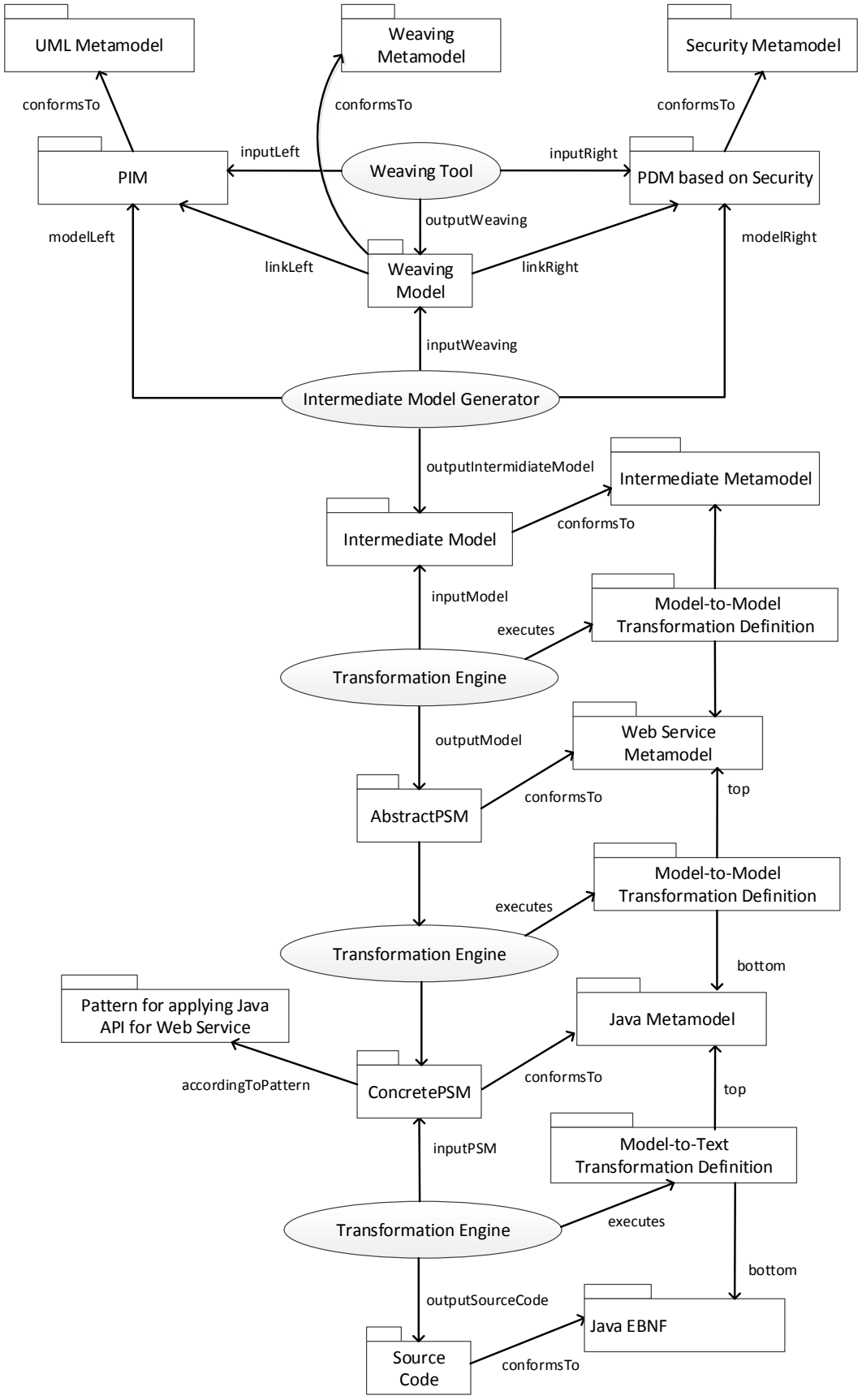


Figura 4.2: Um *framework* para desenvolvimento de SaaS seguro baseado em MDE e *Weaving* (FD3S)

4.3 Metodologia para desenvolvimento de SaaS seguros

A metodologia proposta que direciona o desenvolvimento do *framework* para suportar o desenvolvimento de SaaS abordando aspectos de segurança é ilustrada pela Figura 4.3. A metodologia envolve as fases de criação dos modelos PIM, PDM e *Weaving*, partindo para a geração do modelo intermediário, execução das definições de transformações, edição do modelo PSM e do código fonte. Cada fase é descrita a seguir:

- Create PIM: nesta fase é criado um modelo de negócios (PIM);
- Create PDM (baseado em segurança): nesta fase é criado um modelo contendo aspectos de segurança voltados para autenticação e autorização;
- Create Weaving Model: nesta etapa é criado um modelo *Weaving* com o propósito de relacionar um PIM e um PDM;
- Generate Intermediate Model: nesta fase é gerado o modelo que contém a informação da fusão do PIM, do modelo *Weaving* e do PDM;
- Execute Transformation Definition M2M: neste ponto é realizada uma transformação modelo-a-modelo com o objetivo de gerar um modelo PSM a partir de um modelo intermediário;
- Edit PSM: esta fase é realizada com o objetivo de incluir informações específicas de uma plataforma que não estejam presentes no modelo intermediário, sejam informações do PIM ou do PDM;
- Execute Transformation Definition M2T: nesta fase é realizada uma nova transformação para criação de um código fonte conforme uma gramática EBNF de uma linguagem de programação;
- Edit Source Code: esta etapa é realizada com o propósito de completar o código fonte para estar pronto para ser compilado.

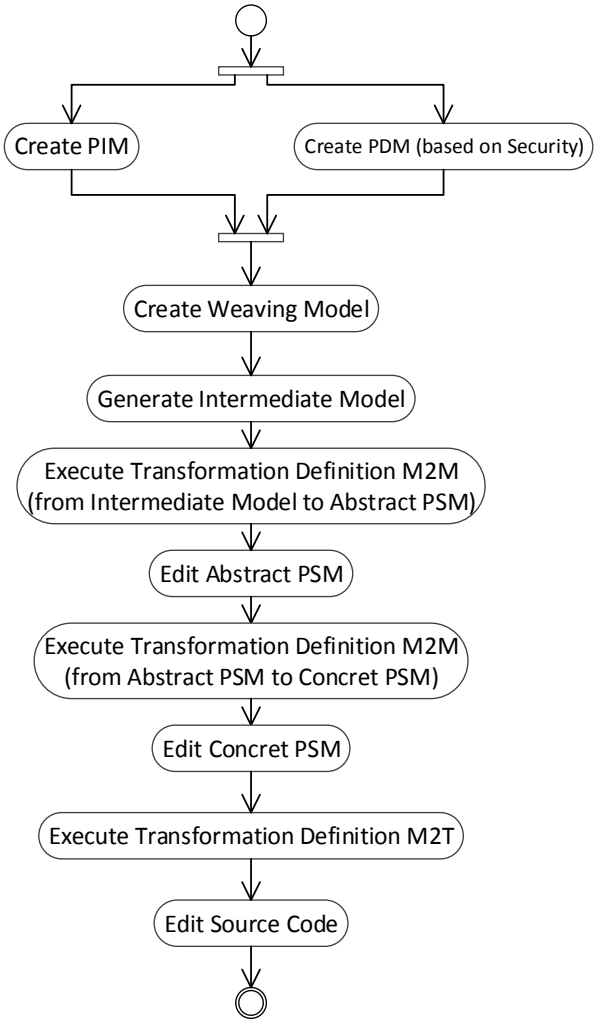


Figura 4.3: Uma metodologia para desenvolvimento de SaaS seguros

4.4 Metamodelos propostos para a fusão dos modelos

Os metamodelos apresentados nesta seção visam auxiliar no processo metodológico descrito na seção anterior. São detalhados os metamodelos para criação do modelo *Weaving*, do modelo PDM baseado em aspectos de segurança com foco em autenticação e autorização, do modelo intermediário, e dos modelos PSM abstrato e concreto. O modelo PIM é baseado no metamodelo UML.

4.4.1 Metamodelo para *Weaving*

O metamodelo para *Weaving* é usado para auxiliar na criação e edição de um modelo *Weaving*, utilizado como uma forma de estabelecer um entrelaçamento de informações de dois ou mais modelos distintos, definindo como estes se relacionam. Este metamodelo estende e adapta o metamodelo para especificação de mapeamento apresentado em [53]. O metamodelo *Weaving* é descrito na Figura 4.4 e é composto pelos seguintes elementos:

- Historic: este elemento contém as definições de *Weaving* (*WeavingDefinition*) e armazena um histórico através dos atributos *update*, *note*, *version* e *timestamp* para indicar o *WeavingDefinition* atual;
- WeavingDefinition: é um contêiner para os elementos que definem o *weaving*. Por exemplo, o *WeavingDefinition* contém o elemento *ModelHandlers* que aponta para modelos a serem relacionados, contém o elemento *WeavingNode* que relaciona dois ou mais elementos apontados por *LinkLeft* e *LinkRight*, ou seja, um *ElementModelHandler*;
- ModelHandler: este elemento é um ponteiro para modelos que são relacionados por um modelo *Weaving*. Um *ModelHandler* contém *ElementModelHandlers* que apontam para os elementos de um metamodelo que se relacionam com outros elementos de outro metamodelo.

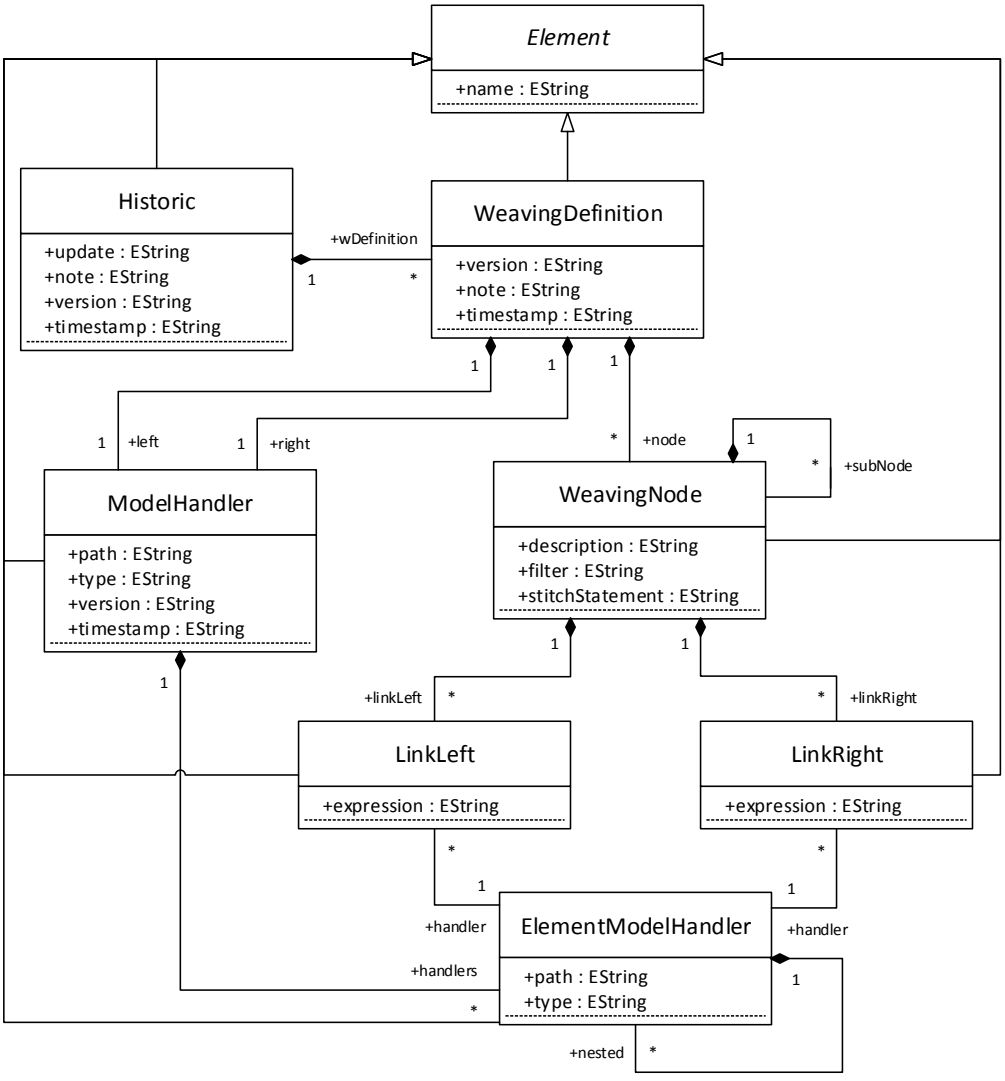


Figura 4.4: Um metamodelo para *Weaving* (baseado em [53])

4.4.2 Metamodelo para PDM com aspectos de segurança

Para a criação do modelo PDM baseado em aspectos de segurança com o foco em autorização e autenticação foi utilizado um metamodelo que aborda elementos relacionados a mecanismos de controle de acesso baseado em funções. Este metamodelo foi baseado em [50][54]. A Figura 4.5 a seguir descreve o metamodelo para PDM e é composto pelos seguintes principais elementos:

- RoleSpace: este elemento é um contêiner para funções (*Roles*) que determinam as ações (*Actions*) executadas pelos usuários (*Users*) de acordo com as permissões (*Permissions*) definidas;
- User: o elemento *User* contém os elementos *Property* e *Operation*. O elemento *Property* define características estáticas, e o elemento *Operation* define características de comportamento do usuário;
- Permission: este elemento está relacionado a restrições sobre as ações (*Actions*) que podem ser executadas sobre os recursos (*Resources*) dos quais o usuário tem acesso. Uma ação pode ser de leitura, atualização, exclusão e execução.

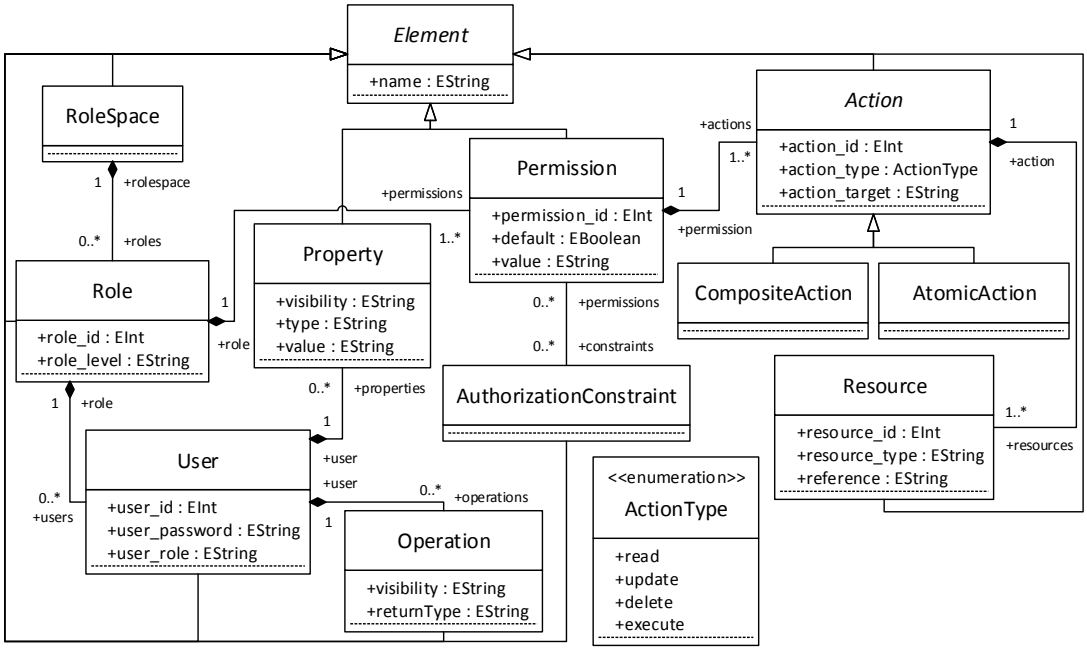


Figura 4.5: Um metamodelo para definir um PDM baseado em aspectos de segurança (baseado em [50][54])

A definição de quais elementos do PIM estão relacionados a cada elemento do PDM de segurança ocorre mediante a interação do especialista que determina as correspondências de acordo com o conhecimento do domínio.

4.4.3 Metamodelo para modelo intermediário

Foi proposto o metamodelo intermediário para criação de modelos intermediários. O metamodelo intermediário oferece suporte para a criação de classes, propriedades e métodos semelhante a um metamodelo UML. Sendo que, o metamodelo intermediário foi aperfeiçoado para possibilitar a definição de uma extensão para armazenar os elementos mesclados (*MergedElements*) que apontam para elementos de outros metamodelos. Dessa maneira, é possível criar um modelo intermediário que contém as informações de fusão dos elementos de um PIM, de um modelo *Weaving* e de um PDM, mas que ainda mantem a ligação com os elementos originais de onde esses elementos do modelo intermediário foram derivados. A seguir, a Figura 4.6 ilustra o metamodelo intermediário e é composto pelos seguintes elementos:

- Merge: este elemento traz a definição dos modelos utilizados no processo de *weaving*; o modelo da esquerda, o modelo da direita e o modelo *weaving* são descritos. O elemento Merge também contém a definição das classes geradas a partir da fusão dos modelos de entrada;
- MergedElements: este elemento é uma extensão que armazena informações dos elementos dos modelos originários da fusão. MergedElements registra os valores de *name*, *path*, *timestamp* e *version* dos modelos e seus elementos;
- Class: o elemento Class reflete as classes provenientes da fusão dos modelos da esquerda e da direita, contém também em sua definição os elementos Method e Property, que descrevem as características e comportamentos das suas respectivas classes.

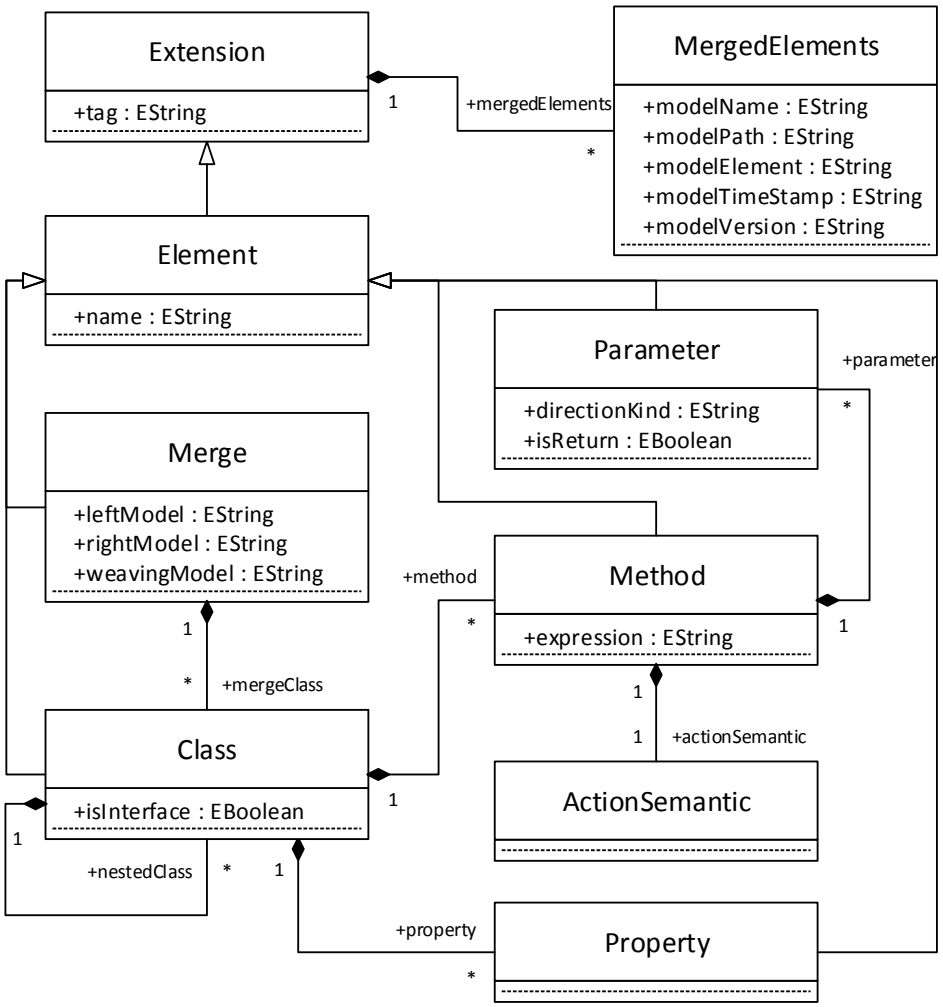


Figura 4.6: Metamodelo intermediário

4.4.4 Metamodelos para PSMs

Por definição do *framework* proposto são gerados modelos PSM abstratos e concretos. O modelo PSM abstrato foi baseado em um metamodelo de *Web Service* e o modelo PSM concreto é conforme a plataforma Java aplicando padrões de sua API para *Web Services* na qual a solução será desenvolvida de fato. A Figura 4.7 apresenta o metamodelo para criação do PSM abstrato baseado na linguagem WSDL utilizada para descrever *Web Services*. O metamodelo é representado através dos seguintes elementos:

- Definition: este é o elemento raiz de todo documento WSDL e contém atributos para definir os namespaces utilizados no documento WSDL;
- Service: é o elemento que juntamente com o elemento Port identifica e define a localização real dos serviços;
- Binding: este elemento especifica a interface que mapeia os elementos de operação em um elemento PortType para um protocolo específico;

- Message: o elemento Message corresponde as operações do *Web Service* definindo os dados a serem transmitidos, e se relaciona com os elementos Part que formam as partes reais da operação.

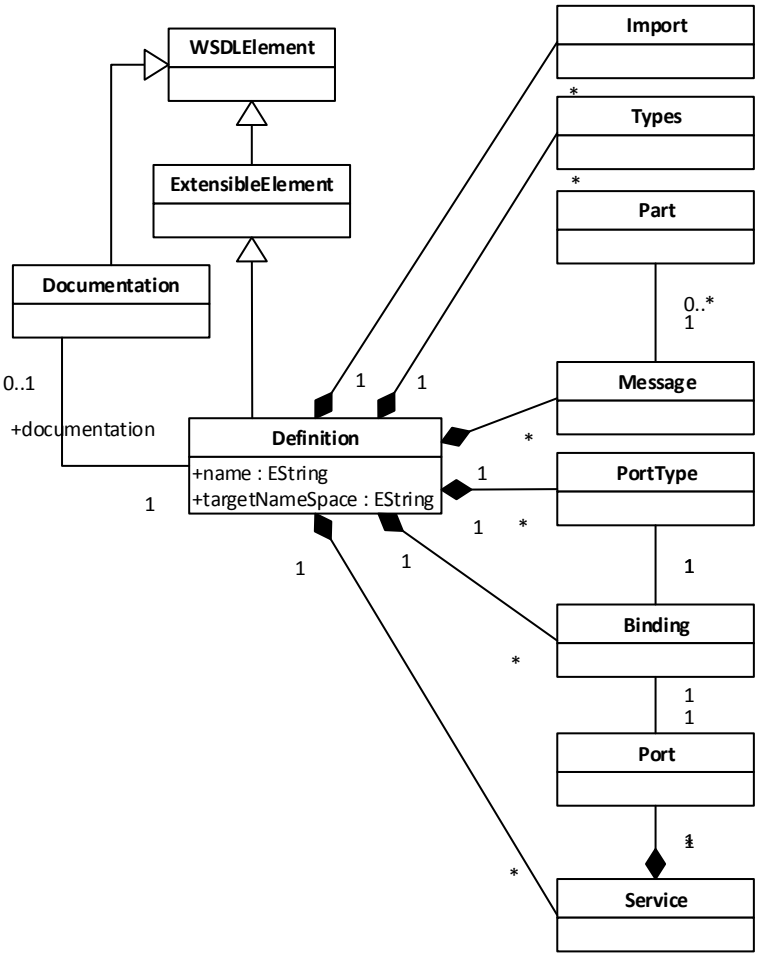


Figura 4.7: Um metamodelo para *Web Services* (extraído de [14])

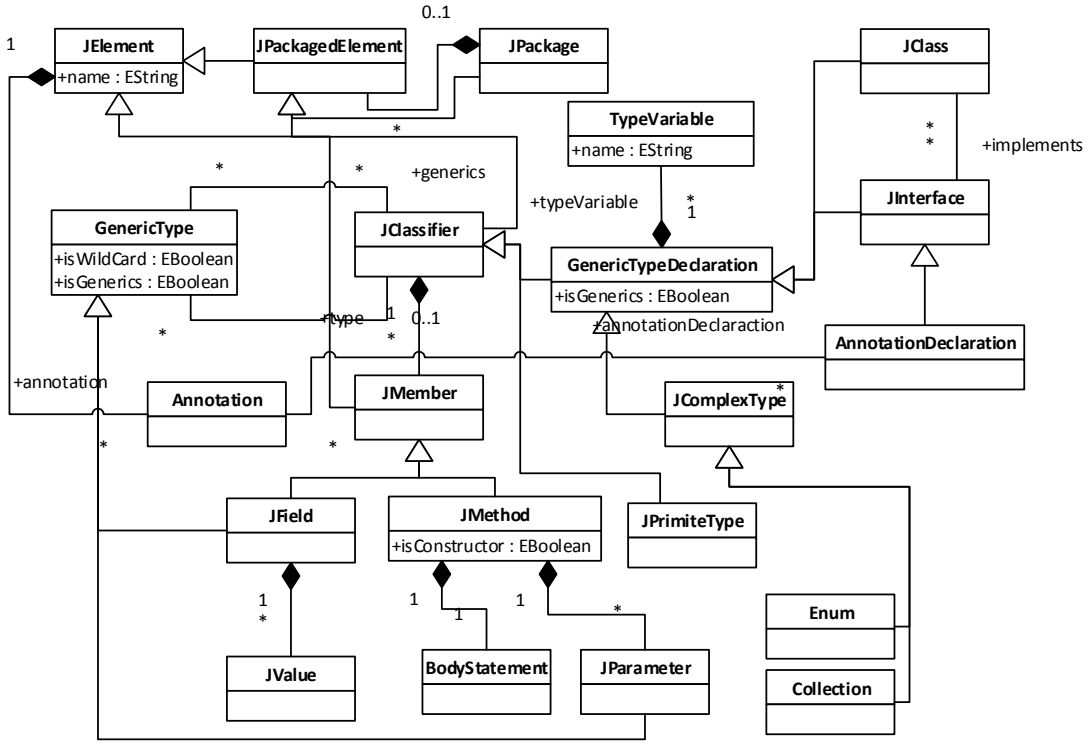
A Figura 4.8 a seguir demonstra os metamodelos para criação do PSM concreto baseado em Java e na API Java para *Web Services*. No metamodelo Java os principais elementos são:

- JPackage: este elemento é um contêiner para todas as classes e interfaces de Java;
- JClassifier: este elemento é a base para os elementos JPrimitiveType, JClass e JInterface; o JClassifier define as associações de herança e contém o elemento JMember, que é a base para os elementos JField e JMethod;
- JClass: é uma especialização de um JClassifier, que contém os elementos JField e JMethod como membros;

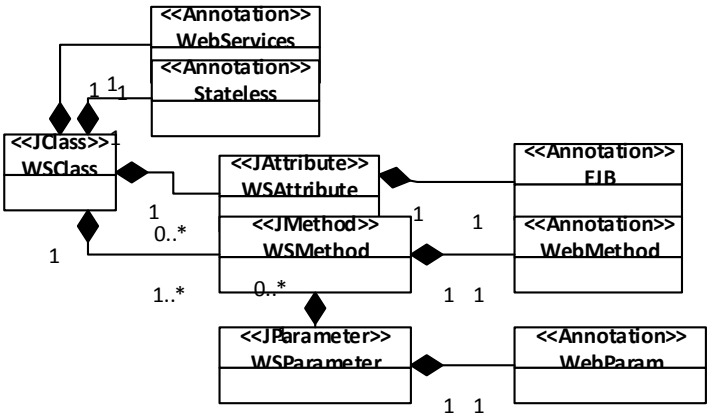
- JField: este elemento é uma composição de apenas um valor, JValue, e tem um tipo JPrimitiveType, JClass ou JInterface, através do JClassifier;
- JMethod: é o elemento que contém a assinatura da operação, o elemento JParameter e o corpo do método;
- JParameter: elemento que corresponde ao argumento de um JMethod, podendo ser de entrada ou de saída.

Nas definições de um padrão para aplicação da API Java para *Web Services* temos os seguintes elementos:

- WSClass: este elemento representa um contêiner das definições de classes em Java *Web Services*; contém os elementos WSAttribute e WSMethod, e também o elemento WebServices através de annotations;
- WSAttribute: é o elemento que define os atributos de uma classe em Java *Web Services*; contém a definição do elemento EJB utilizando annotations;
- WSMethod: este elemento define os métodos de uma classe em Java *Web Services* e contém o elemento WSParameter que descreve os parâmetros do método. WSMethod também contém a definição do elemento WebMethod através de annotations.



(a)



(b)

Figura 4.8: (a) Metamodelo Java (adaptado de [14]) e **(b)** Padrão para aplicação da API Java para *Web Services*

4.5 Síntese

Neste capítulo, uma abordagem baseada em MDE utilizada para suportar o desenvolvimento de SaaS foi apresentada, envolvendo neste processo aspectos de segurança com o propósito de atender objetivos específicos de segurança relativos a autenticação e controle de acesso. Para aplicação neste processo foi apresentada a técnica de *weaving*, na qual são estabelecidos relacionamentos entre um modelo PIM e modelos PDM, para mesclar seus elementos em um modelo único.

No *framework* apresentado, os níveis do processo de desenvolvimento da solução proposta até se alcançar o nível final foram descritos em detalhes. Também foi apresentada a metodologia que irá guiar o desenvolvimento do *framework*, descrevendo-se as fases de criação dos modelos e de transformações de modelos.

Diversos metamodelos utilizados na abordagem foram apresentados, desde os metamodelos utilizados para a criação e integração dos modelos de entrada, até os metamodelos para os modelos intermediário e de saída.

5 Implementação do protótipo do *framework* FD3S

Neste capítulo, um protótipo de implementação do *framework* para suportar o desenvolvimento de SaaS adotando aspectos de segurança é apresentado. O *framework* foi implementado com o auxílio da ferramenta EMF (*Eclipse Modeling Framework*), ferramenta que oferece suporte à geração de código utilizando como base a definição de modelos. Foi utilizada a linguagem de transformação ATL, ferramenta que se integra ao Eclipse em forma de *plug-in* e oferece suporte a transformações de modelos através de regras de correspondência definidas. Foram definidos metamodelos em Ecore para suportar os processos de *merging* e *weaving*, bem como utilizada a ferramenta *GenModel* do EMF responsável por gerar *plug-ins* em Eclipse para criação e edição dos modelos utilizados no trabalho.

5.1 Prototipagem do *framework* FD3S

A arquitetura do protótipo que implementa o *framework* proposto, a metodologia, os metamodelos e as ferramentas são apresentadas a seguir conforme ilustra a Figura 5.1.

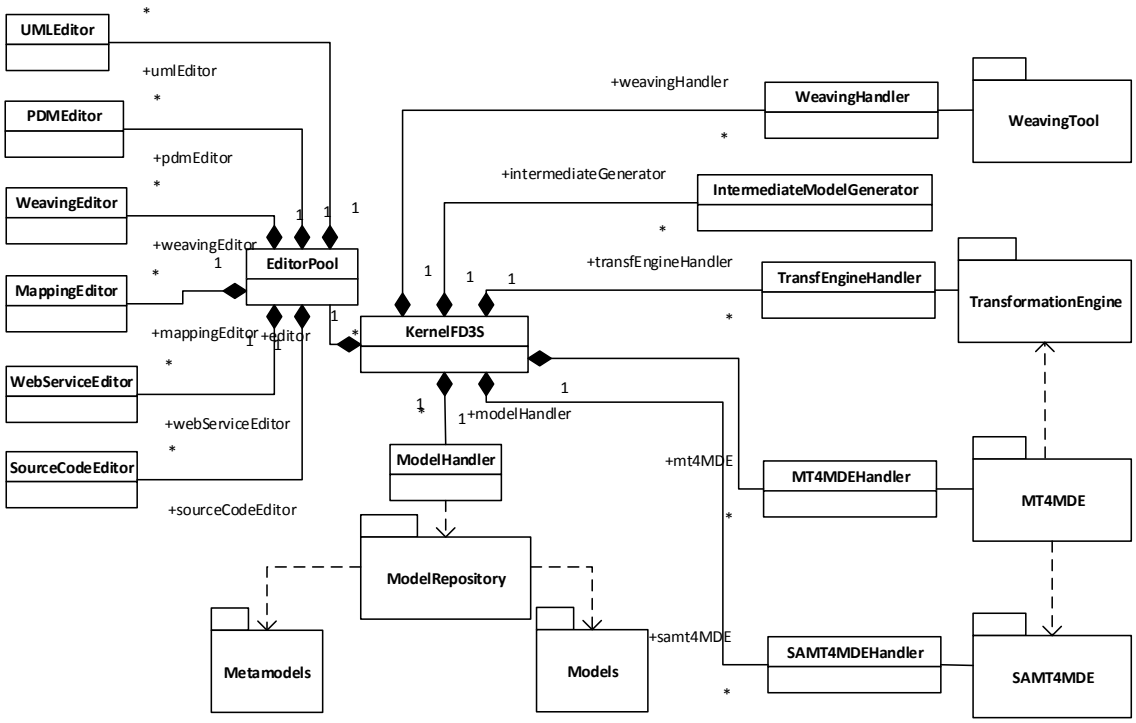


Figura 5.1: Protótipo para FD3S

No protótipo para FD3S, o KernelFD3S contém as funcionalidades básicas para iniciar o processo. O KernelFD3S é composto por manipuladores para integrar outras ferramentas tais como: uma ferramenta de *weaving* (através do *WeavingHandler*), uma ferramenta de transformação (através do *TransfEngineHandler*), uma ferramenta de mapeamento (como por exemplo através de um *MappingHandler*), uma ferramenta de correspondência (como por exemplo através de um *MatchingHandler*).

O elemento KernelFD3S também integra um grupo de editores para criação e edição de modelos UML, Modelo de Descrição de Plataforma (PDM), modelos *Weaving*, modelos de mapeamento, *Web Services* e códigos fonte em Java. O KernelFD3S gerencia ainda o *ModelRepository* que inclui modelos e metamodelos.

Este protótipo de FD3S foi desenvolvido usando as seguintes ferramentas:

- Java SE Development Kit (J2SDK) versão 7 [30].
- IDE Eclipse versão Luna Service Release 1 [56].
- Eclipse Modeling Framework versão 2.6.0 [22].
- ATL SDK para Eclipse versão 3.5.0 [57].
- Editor UML Papyrus versão 0.9.2 [58].
- IDE NetBeans versão 6.5 [59] (incluindo Glassfish Server Open Source Edition 4.1).
- MT4MDE versão 0.5.1 [55].
- SAMT4MDE versão 0.8.2 [55].

O protótipo utilizou as ferramentas MT4MDE (*Mapping Tool for Model Driven Engineering*) e SAMT4MDE (*Semi-Automatic Matching Tool for MDE*) desenvolvidas por D. Lopes [55]. Tais ferramentas foram concebidas com o propósito de estabelecer correspondências entre metamodelos Ecore e também gerar regras de transformação em ATL. O protótipo foi desenvolvido baseado em uma série de adaptações dessas ferramentas para que pudéssemos incluir o suporte ao *weaving* de modelos. Tomando como base as ferramentas MT4MDE e SAMT4MDE, foram adaptados o metamodelo de *weaving*, o metamodelo intermediário, as classes de manipulação e as regras de transformações de modelos.

A ferramenta MT4MDE é composta por uma interface para permitir a interação gráfica do especialista com a ferramenta e interfaces dos algoritmos a serem implementadas tanto para as definições de correspondências quanto para as definições de transformações entre metamodelos; ela contém também módulos de importação e exportação para tradução dos metamodelos, além dos metamodelos de mapeamento. A MT4MDE visa especificamente a criação e

edição de correspondências entre elementos de metamodelos gerando definições de transformações a partir destas utilizando a linguagem ATL.

A ferramenta SAMT4MDE é uma extensão da MT4MDE, pois em complemento a esta última, ela acrescenta a busca de correspondências entre elementos de metamodelos de modo semiautomático, visto que ainda é necessário a interação humana para confirmar se as correspondências localizadas são de fato consistentes. A SAMT4MDE é composta por mecanismos de busca por sinônimos para detectar elementos similares utilizando uma base de dados; a ferramenta também apresenta uma interface para interação do especialista validando as correspondências localizadas. Para permitir tal funcionalidade a ferramenta também conta com manipuladores que garantem a navegabilidade entre elementos dos metamodelos, além de um construtor de modelo de correspondências com base na validação das correspondências encontradas.

Para implementar o *framework* FD3S foram desenvolvidas ainda as seguintes partes do protótipo:

- Editor Weaving versão 0.1.0 e Weaving Tool versão 0.1.0.
- Editor PDM baseado no metamodelo apresentado na Figura 4.5.
- Gerador de modelo intermediário baseado no metamodelo apresentado na Figura 4.6.
- Editor Web Service baseado no metamodelo apresentado na Figura 4.7.
- Definições de transformação apresentadas na Listagem 5.5, Listagem 5.6, Listagem 5.7 e Listagem 5.8.

A imagem do protótipo para o *framework* FD3S é apresentado na Figura 5.2 a seguir. São apresentados o modelo PIM de OHRS à esquerda, o modelo *weaving* ao centro, e o modelo PDM de segurança à direita. Como ilustrado, os *handlers* do modelo *weaving* são responsáveis por manipular os elementos dos modelos da esquerda e da direita.

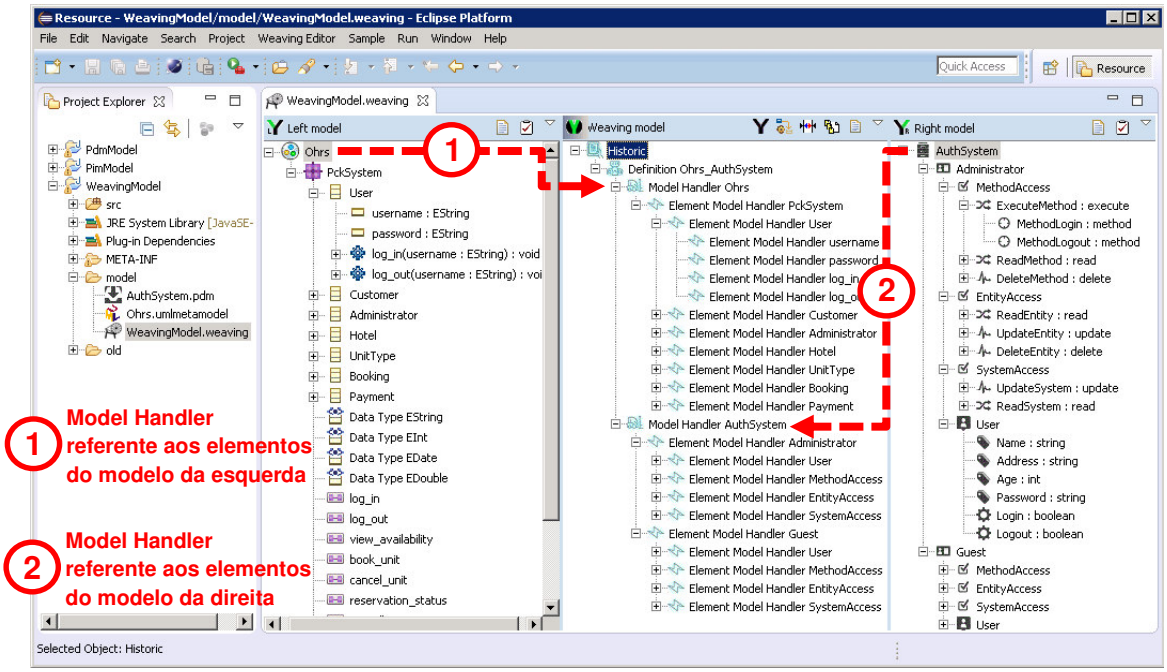


Figura 5.2: *Screenshot* do protótipo para FD3S (versão 0.1.0)

Além da adaptação das ferramentas MT4MDE e SAMT4MDE para a criação da estrutura de edição de modelos com suporte ao *weaving*, também foi necessário gerarmos as definições de transformações que darão suporte as transformações de modelo-a-modelo e modelo a código.

5.1.1 Implementação do protótipo

Seguindo a abordagem MDE através da ferramenta *Eclipse Modeling Framework*, foram criados os metamodelos propostos no capítulo, conforme o metamodelo Ecore fornecido pelo EMF. Também foram utilizadas classes Java fornecidas pelo EMF para criar instâncias e manipular modelos Ecore. Para cumprir parte das etapas de transformações de modelos foi necessário utilizar a linguagem ATL.

A Figura 5.3 mostra o modelo gerador **pdmmetamodel.genmodel** criado a partir do modelo **pdmmetamodel.ecore**. Este modelo corresponde ao metamodelo do PDM de segurança que será instanciado mais adiante, e que será integrado ao PIM conforme descrito no *framework*. O modelo gerador é responsável por criar os códigos de modelo, códigos de teste e códigos de edição em forma de *plug-in* para Eclipse. O modelo instanciado através da ferramenta GenModel permite a edição das suas informações pelo especialista de domínio. É possível criar novos elementos, conforme ilustra a Figura 5.4, definindo características conforme o respectivo metamodelo, e os modelos

criados ficam armazenados podendo ser utilizados para novos processos de integração.

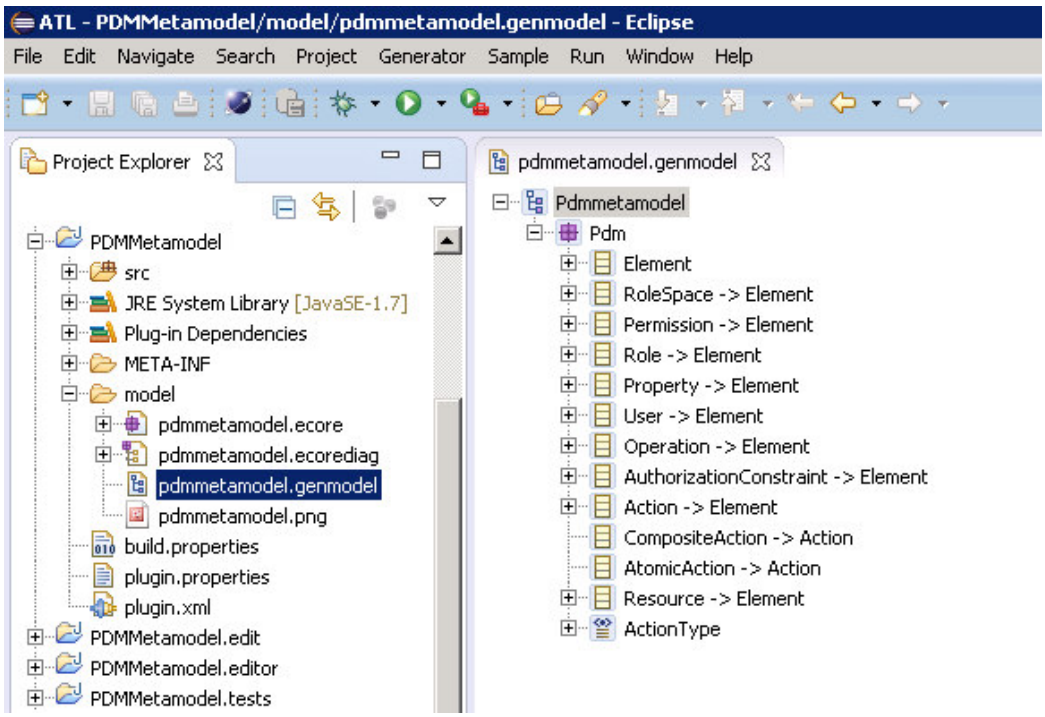


Figura 5.3: Modelo pdmmetamodel.genmodel construído a partir de pdmmetamodel.ecore

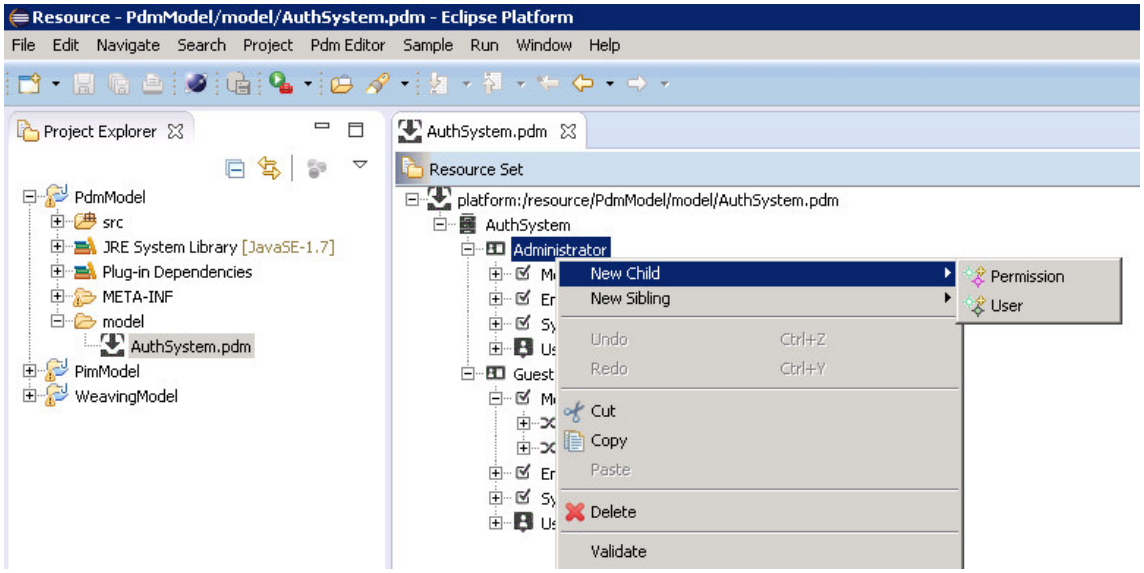


Figura 5.4: Interface de criação de uma nova permissão para a função de administrador do modelo PDM instanciado AuthSystem.pdm

O PIM a ser integrado ao modelo PDM pode ser importado ou modelado da mesma forma utilizando os recursos que o EMF disponibiliza. Neste caso também foi utilizado o GenModel para gerar um modelo conforme um metamodelo UML, sendo que o respectivo modelo instanciado obedeceu a todas as definições apresentadas na Figura 3.3 no capítulo 3, que dizem respeito ao PIM do Sistema de Reserva de Hotel Online utilizado no exemplo ilustrativo na seção seguinte. A Figura 5.5 ilustra um fragmento do modelo gerador **umlmetamodel.genmodel** criado a partir do modelo **umlmetamodel.ecore**.

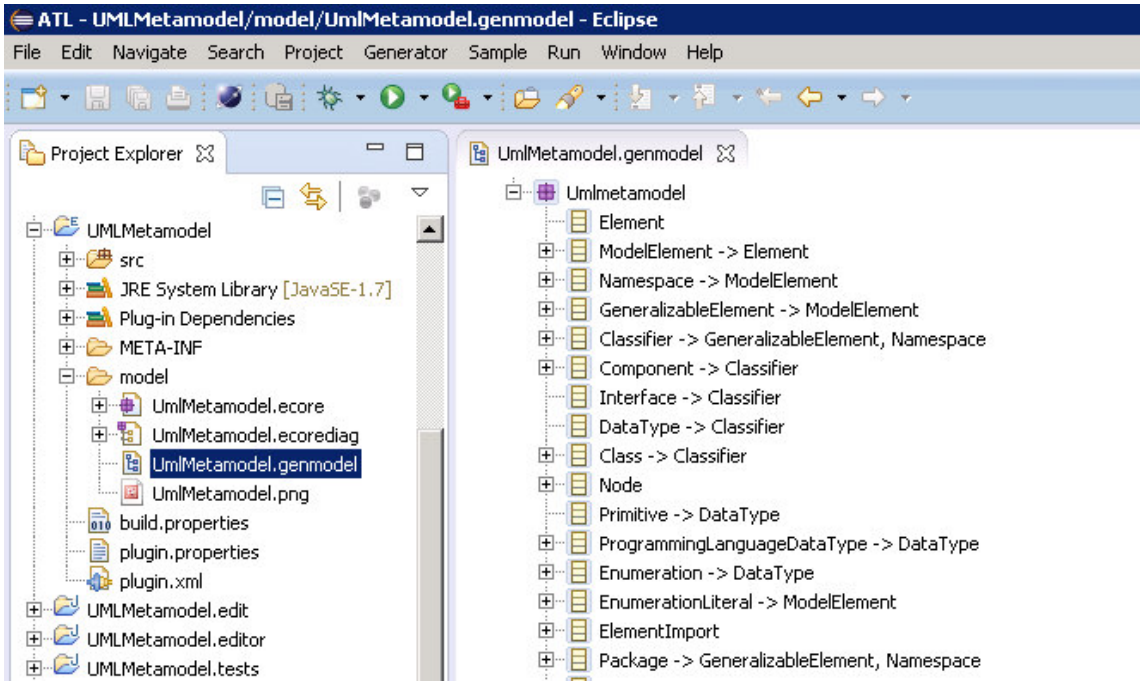


Figura 5.5: Modelo umlmetamodel.genmodel construído a partir de umlmetamodel.ecore

O modelo instanciado mostrado na Figura 5.6 ilustra a interface de edição do modelo PIM conforme o metamodelo UML, através da qual é possível criar ou apagar elementos conforme a necessidade do especialista de domínio. Da mesma forma foram modelados também os metamodelos das plataformas de destino para as transformações, mas com uma finalidade diferente. Na definição de transformações em ATL os metamodelos de entrada e saída precisam ser especificados, logo, se fez necessário também a modelagem dos respectivos metamodelos descritos no capítulo anterior referentes aos PSMs abstrato e concreto. Assim foram gerados os metamodelos **webservices.ecore** e **javawebservices.ecore**.

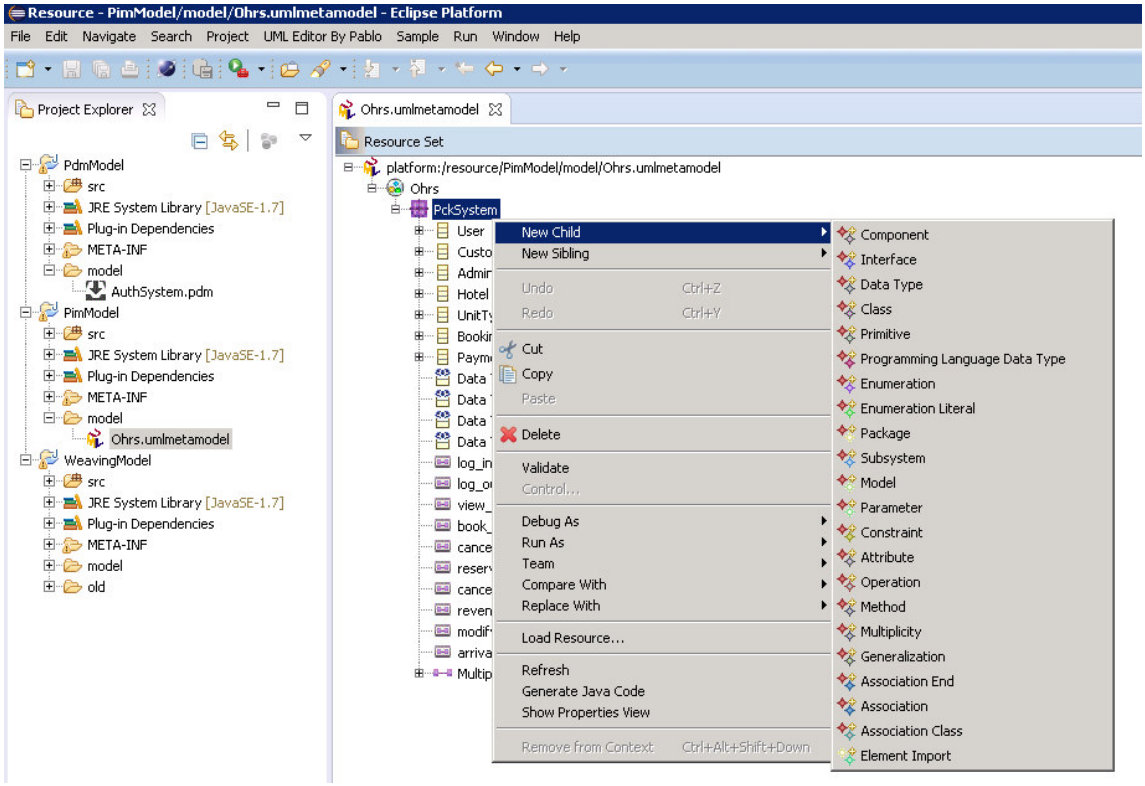


Figura 5.6: Interface de criação de novas classes, atributos, etc., para o modelo PIM instanciado Ohrs.umlmetamodel

Além dos modelos e metamodelos já citados até agora, ainda há o modelo gerador do *weaving*, conforme mostra a Figura 5.7, a partir do qual tiveram seus códigos gerados, e especialmente adaptados para oferecer suporte à integração de modelos PIM e PDM.

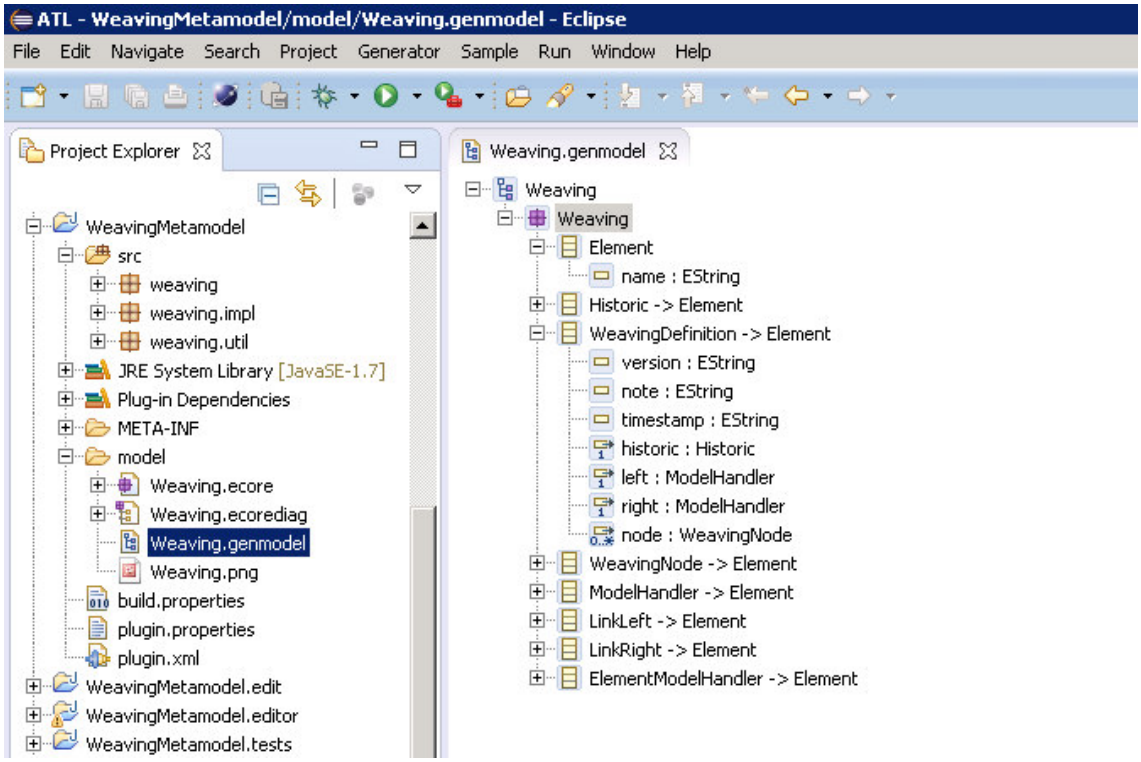


Figura 5.7: Modelo weaving.genmodel construído a partir de weaving.ecore

Dos projetos gerados a partir do GenModel do *weaving*, dois receberam ajustes especiais para que o resultado da instanciação do metamodelo *weaving* fosse correspondente à interface apresentada pela Figura 5.2. Os projetos *weavingmetamodel.edit* e *weavingmetamodel.editor* foram os que sofreram maiores adaptações visando uma melhor usabilidade e funcionalidade da ferramenta, bem como o suporte ao *weaving* dos modelos de entrada.

A partir do modelo *weaving* é possível gerar um modelo intermediário que combina as informações analisadas dos modelos de entrada. Para tanto, o metamodelo intermediário também precisou ser modelado no início do processo, por ser peça fundamental no processo de transformação e de armazenamento dos dados mesclados. Na Figura 5.8 observa-se o modelo gerador **intermediate.genmodel** gerado a partir do metamodelo intermediário **intermediate.ecore**. Tais modelos refletem aspectos de classes, métodos e propriedades herdadas dos modelos fundidos, e traz em seu modelo instanciado informações da origem de seus elementos.

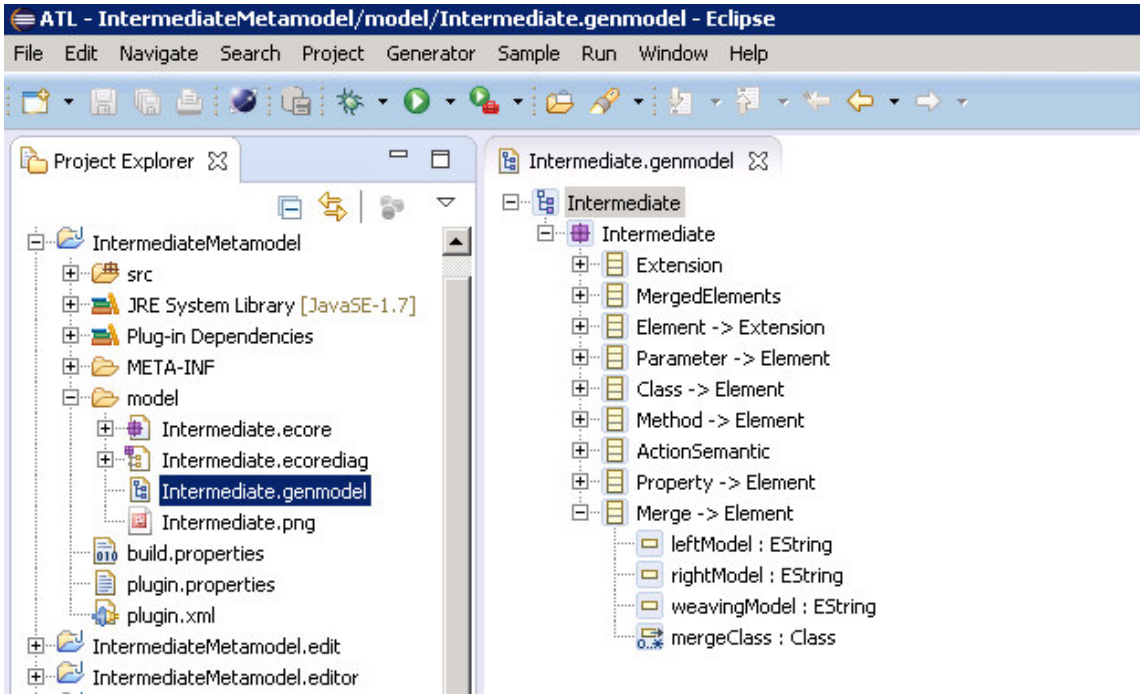


Figura 5.8: Modelo intermediate.genmodel construído a partir de intermediate.ecore

O modelo instanciado do metamodelo intermediário será a representação da análise feita pelo especialista de domínio. No modelo intermediário estarão presentes as informações de quais elementos sofreram fusão mediante conhecimento do domínio de segurança e do domínio do modelo de negócios. A Figura 5.9 a seguir representa um fragmento do que pode ser editado conforme o metamodelo intermediário.

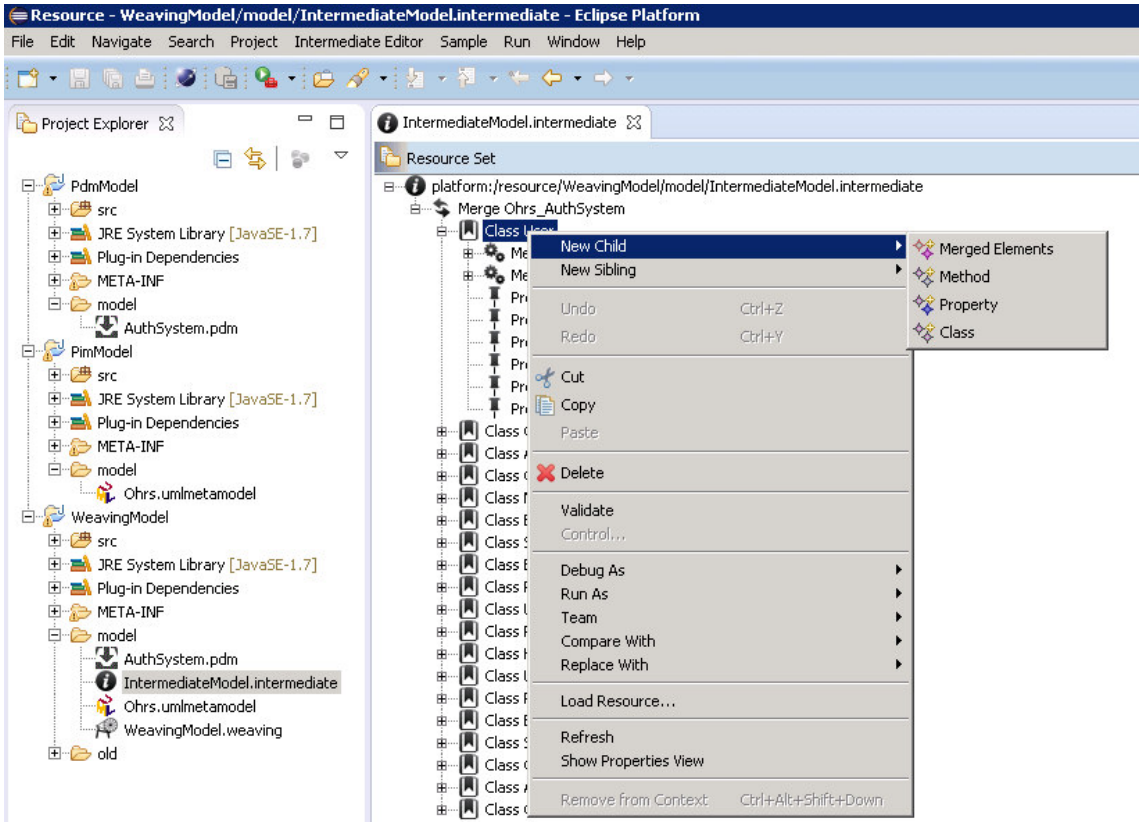


Figura 5.9: Modelo intermediário instanciado intermediatemodel.intermediate

Conforme descrito no *framework* um modelo *weaving* é criado a partir da combinação de elementos de dois modelos de entrada, um modelo PIM à esquerda e um modelo PDM à direita. A princípio, o modelo *weaving* tem seus manipuladores de modelos (*Model Handlers*) populados automaticamente com todos os elementos de cada modelo de entrada, em seguida, através do processo de tecelagem os elementos de ambos modelos de entrada são entrelaçados, populando dessa forma os nós do modelo *weaving*. O modelo intermediário é criado pelo especialista de domínio a partir de um gerador, e armazena as informações dos elementos que foram fundidos na etapa do *weaving* para análise e melhor refinamento do que é de fato válido na fusão.

Além dos metamodelos definidos em Ecore e seus respectivos modelos geradores de *plug-ins*, a linguagem de transformação ATL também foi utilizada para realizar as transformações de modelo-a-modelo e modelo-a-código. Foi criado um projeto ATL para armazenar as regras definidas para cada etapa de transformação descrita no *framework*. Foram criadas definições de regras para transformar o modelo intermediário em modelo PSM abstrato estabelecendo correspondências entre elementos de ambos os modelos. Sequencialmente foram definidas regras para transformação do modelo PSM abstrato para PSM concreto, e finalmente, definidas regras de transformação de PSM concreto para o código. Os modelos criados são conforme seus respectivos metamodelos.

5.2 Síntese

Neste capítulo, o protótipo de implementação do *framework* FD3S foi apresentado inicialmente. Os elementos que compõem o protótipo foram descritos, bem como as ferramentas utilizadas para o desenvolvimento do mesmo e suas respectivas funcionalidades.

Em seguida, a implementação do protótipo foi apresentada etapa por etapa. Os metamodelos para criação de modelos PIM e PDM foram apresentados e as interfaces de criação dos modelos foram ilustradas. O metamodelo para criação de um modelo *Weaving* foi apresentado, salientando os ajustes necessários para que fosse possível se alcançar os resultados desejados no modelo *Weaving*. O metamodelo intermediário também foi apresentado, de modo que um modelo intermediário pudesse representar a análise realizada pelo especialista de domínio.

Por fim, o capítulo destaca a utilização da linguagem ATL para a realização das transformações, definidas através de regras especificadas para cada etapa de transformação descrita no *framework*.

6 Exemplo Ilustrativo

Neste capítulo, um exemplo ilustrativo é apresentado com o propósito de auxiliar na melhor compreensão do *framework*, demonstrando suas respectivas funcionalidades. Este exemplo consiste na utilização do modelo PIM de um Sistema de Reserva de Hotel Online (OHRS) [4] para a aplicação de técnicas de *weaving* para fundir aspectos de segurança relacionados a controle de acesso baseado em funções. Para tanto, são descritos o modelo de segurança utilizado, o modelo *weaving* e intermediário gerados, bem como as definições de transformações presentes durante o processo do *framework*.

6.1 Modelo de negócio

O modelo PIM utilizado na implementação do protótipo foi do OHRS conforme já ilustrado no capítulo 3 deste trabalho. Porém, para efeito de compatibilidade com os demais modelos produzidos em Ecore e utilizados na aplicação da metodologia, o modelo PIM foi redesenhado utilizando os recursos do EMF. Primeiramente foi definido um metamodelo UML utilizando Ecore conforme ilustrado na Figura 6.1 a seguir.

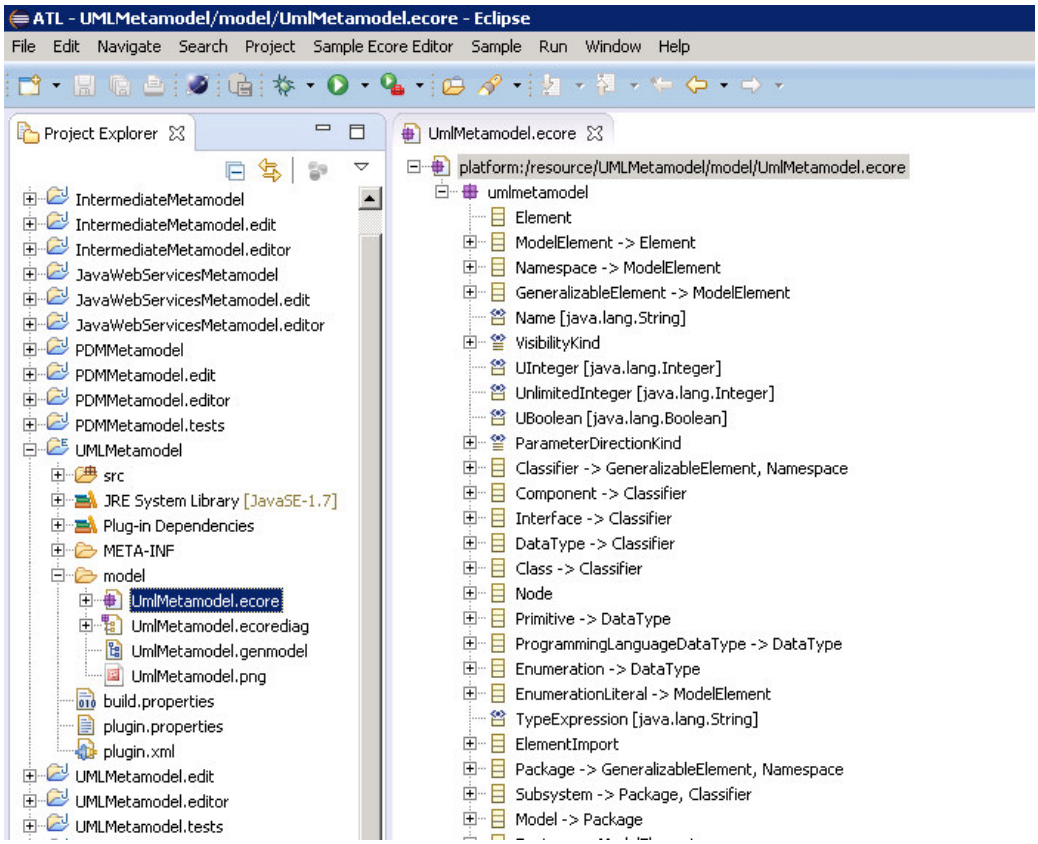


Figura 6.1: Metamodelo UML umlmetamodel.ecore

O metamodelo **umlmetamodel.ecore** expressa de forma bem ampla os elementos essenciais que um modelo UML deve conter, como por exemplo pacotes, classes, métodos e propriedades. A partir desse metamodelo é criado um modelo gerador de UML **umlmetamodel.genmodel**, que auxilia na criação e edição de novos modelos. Este modelo permite criar uma estrutura de *plug-ins* capazes de automatizar a criação de modelos UML e é através dessa estrutura que o modelo PIM do Sistema de Reserva de Hotel Online é instanciado.

Nesta etapa são geradas primeiramente as entidades, pacotes e *factories* para criar instâncias do metamodelo criado em Ecore através da opção *Generate Model Code* conforme ilustra a Figura 6.2. O EMF permite ainda criar um projeto de *plug-in* em Eclipse através da opção *Generate Edit Code*, contendo provedores para exibir um modelo em uma interface de usuário, e o *plug-in* de editor para criar e modificar instâncias de um modelo através da opção *Generate Editor Code*. Através da edição dos códigos gerados a partir do *genmodel* é possível customizar a interface do assistente de criação do modelo PIM, bem como a interface do usuário após o modelo instanciado.

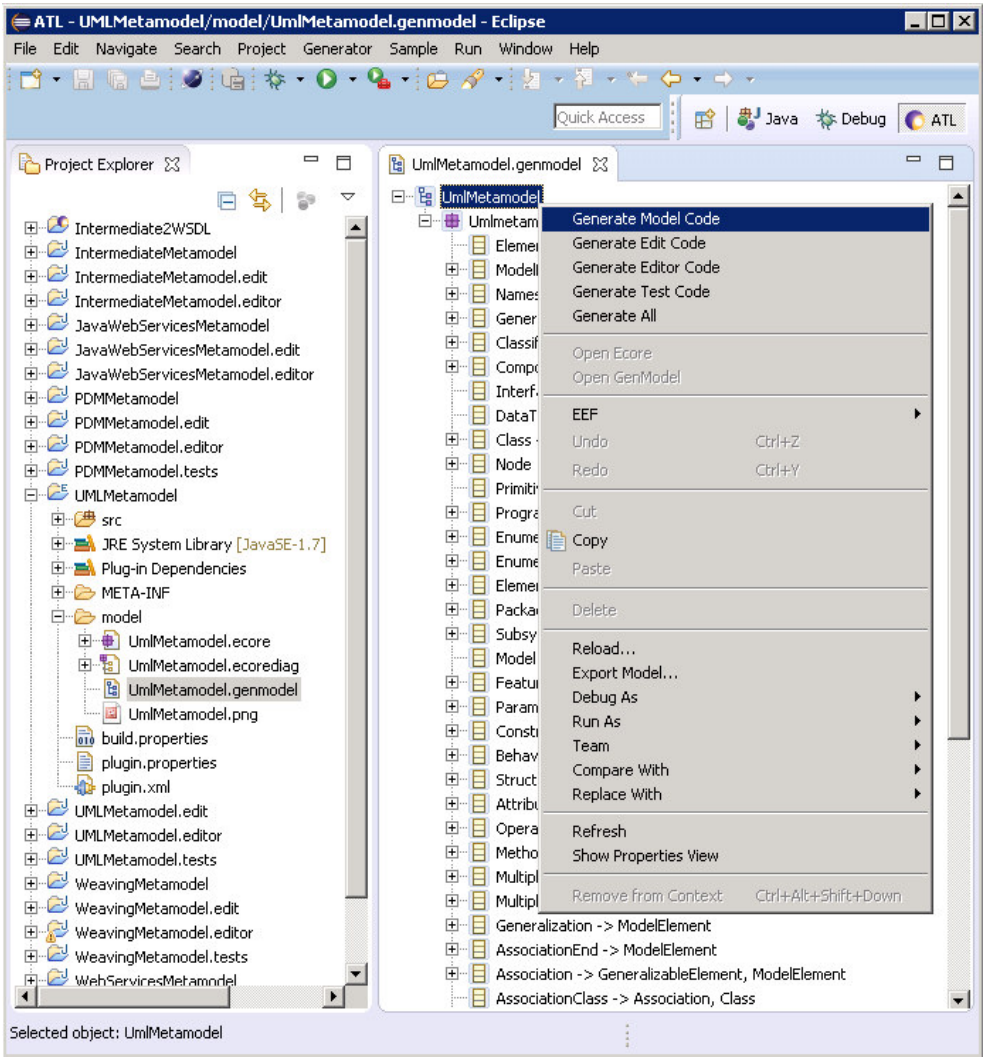


Figura 6.2: Geração de *plug-ins* a partir do umlmetamodel.genmodel

Dessa forma é possível alterar a maneira como os elementos do modelo são apresentados no momento de sua instanciação, seus ícones e quais características desses elementos são visíveis em seus rótulos; a Figura 6.3 demonstra o resultado de tal recurso e ilustra o modelo OHRS baseado em UML, construído a partir das respectivas definições de seu modelo PIM original [4]. O modelo exibe as classes *User*, *Customer*, *Administrator*, *Hotel*, *UnitType*, *Booking* e *Payment*, e expande a classe *Customer* demonstrando seus atributos e métodos.

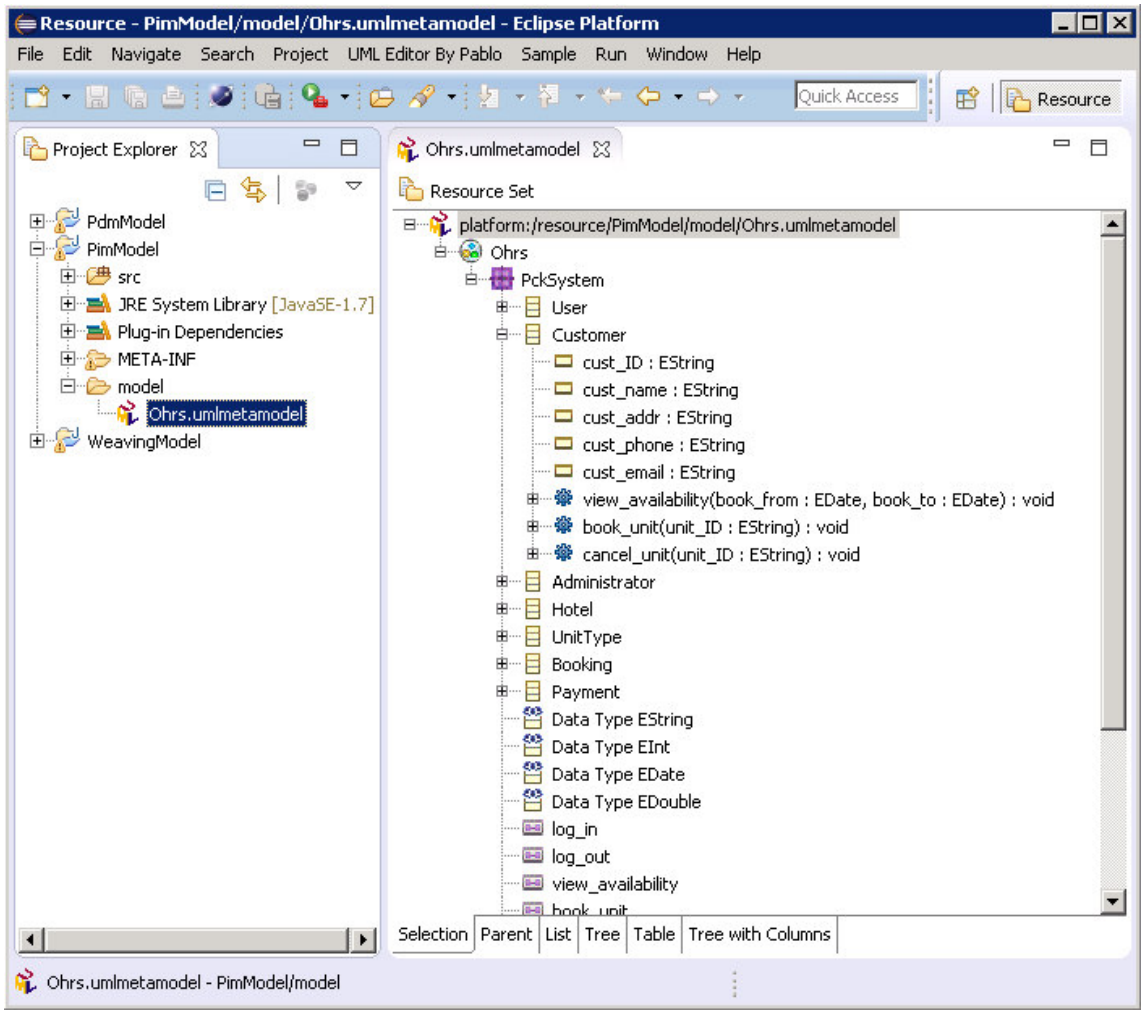


Figura 6.3: Modelo PIM de OHRS ohrs.umlmetamodel

6.2 Modelo de segurança

De semelhante modo, para obtermos o modelo PDM de segurança, utilizamos EMF para construir um metamodelo Ecore representando aspectos de segurança relacionados a controle de acesso e voltados para conceitos de autenticação e autorização. A Figura 6.4 a seguir ilustra os conceitos de

RoleSpace, *Permission*, *Role*, *Action* e *Resource*, conforme o metamodelo anteriormente descrito na Figura 4.5. Com base nesses conceitos é possível construir diferentes modelos de segurança para serem integrados ao modelo PIM, podendo inclusive reutilizar os mesmos modelos de segurança para novos processos de integração com diferentes aplicações de negócios.

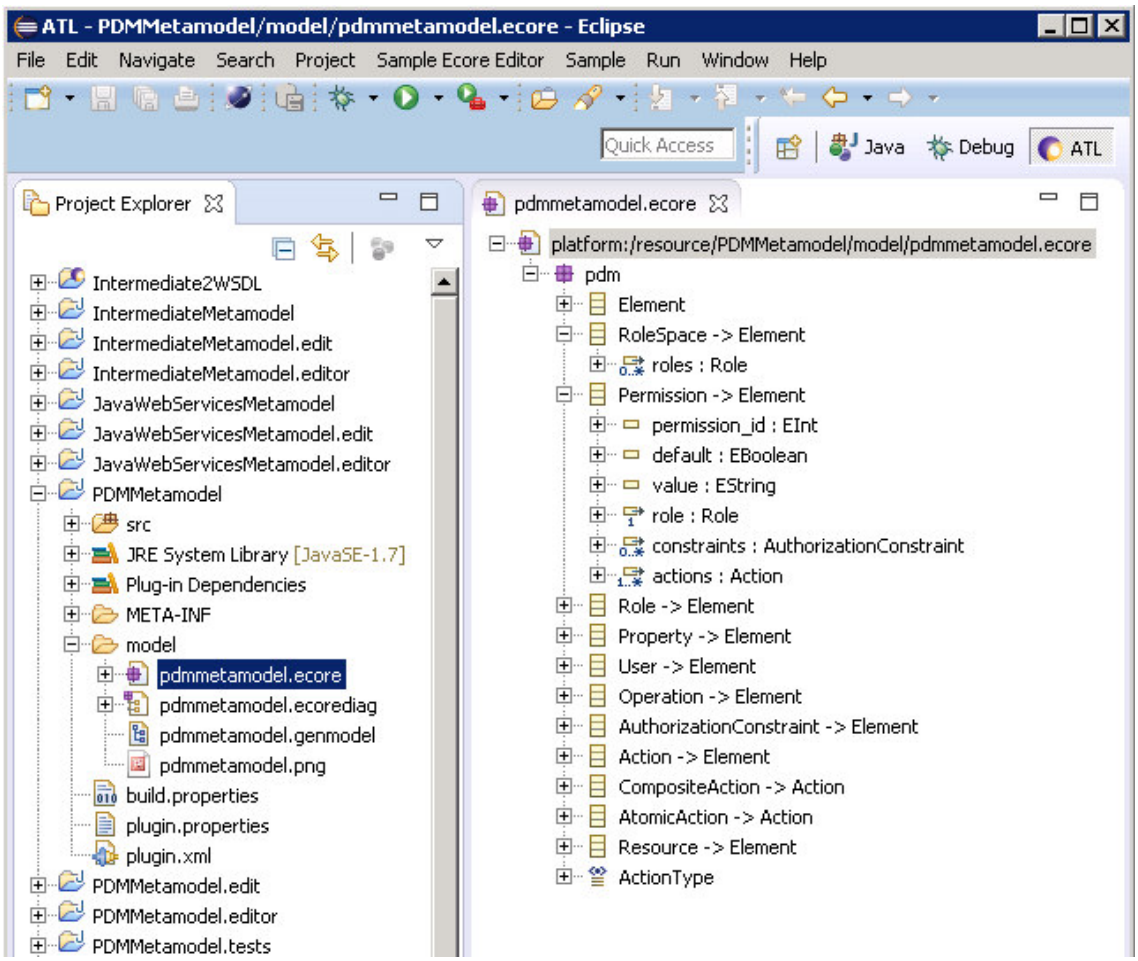


Figura 6.4: Metamodelo pdmmetamodel.ecore para definição de PDM de segurança

A partir do metamodelo **pdmmetamodel.ecore** é criado o modelo gerador de PDM **pdmmetamodel.genmodel** que deu origem ao conjunto de *plug-ins* para edição e criação do modelo PDM. A partir do modelo gerador, os projetos PDMMetamodel.edit e PDMMetamodel.editor são criados em forma de *plug-in* em Eclipse. A edição dos códigos desses *plug-ins* possibilitou a customização da interface do assistente de criação além dos rótulos e ícones dos elementos criados em tempo de execução de uma nova instância do modelo PDM, como mostra a Listagem 6.1 e 6.2.

Listagem 6.1: Trecho de código que define o rótulo do elemento *Action*

```
73  /**
74  * This returns the label text for the adapted class.
75  * <!-- begin-user-doc -->
76  * <!-- end-user-doc -->
77  * @generated NOT
78  */
79  @Override
80
81  //ESPECIFICA O RÓTULO DO ELEMENTO ATOMIC ACTION
82
83  public String getText(Object object) {
84        String label = ((AtomicAction)object).getName();
85        return label == null || label.length() == 0 ?
86              getString("_UI_AtomicAction_type") :
87              label + " : " +
88              ((AtomicAction)object).getAction_type().getName();
89  }
```

Listagem 6.2: Trecho de código que define o ícone do elemento *Role*

```
144  /**
145  * This returns Role.gif.
146  * <!-- begin-user-doc -->
147  * <!-- end-user-doc -->
148  * @generated NOT
149  */
150  @Override
151
152  //ESPECIFICA O ÍCONE DO ELEMENTO ROLE
153
154  public Object getImage(Object object) {
155        return overlayImage(object,
156        getResourceLocator().getImage("full/obj16/Role"));
157  }
```

A execução dos *plug-ins* como uma nova aplicação do Eclipse gerou o modelo AuthSystem.pdm descrito na Figura 6.5, o qual denota aspectos de controle de acesso baseado em funções. A instância do modelo descreve o espaço de funções (*RoleSpace*) AuthSystem como elemento principal do modelo, contendo as funções *Administrator* e *Guest*. Cada uma dessas funções tem permissões especiais dentro deste sistema representado. Os usuários (*User*) contidos sob a função *Administrator* por exemplo, tem permissões definidas como: acesso a métodos (*MethodAccess*), acesso a entidades (*EntityAccess*) e acesso ao sistema como um todo (*SystemAccess*). Por outro lado, os usuários atribuídos a função *Guest* têm permissões semelhantes, porém limitadas, como no caso da permissão SystemAccess, na qual só é permitida a ação de leitura (*ReadSystem*) sob os recursos entidades (*AuthSystemEntity*), métodos (*AuthSystemMethod*) e atributos (*AuthSystemAttribute*). Nota-se que cada elemento *User* tem seus atributos (ex.: *name* e *password*) e métodos (ex.: *login* e *logout*) próprios.

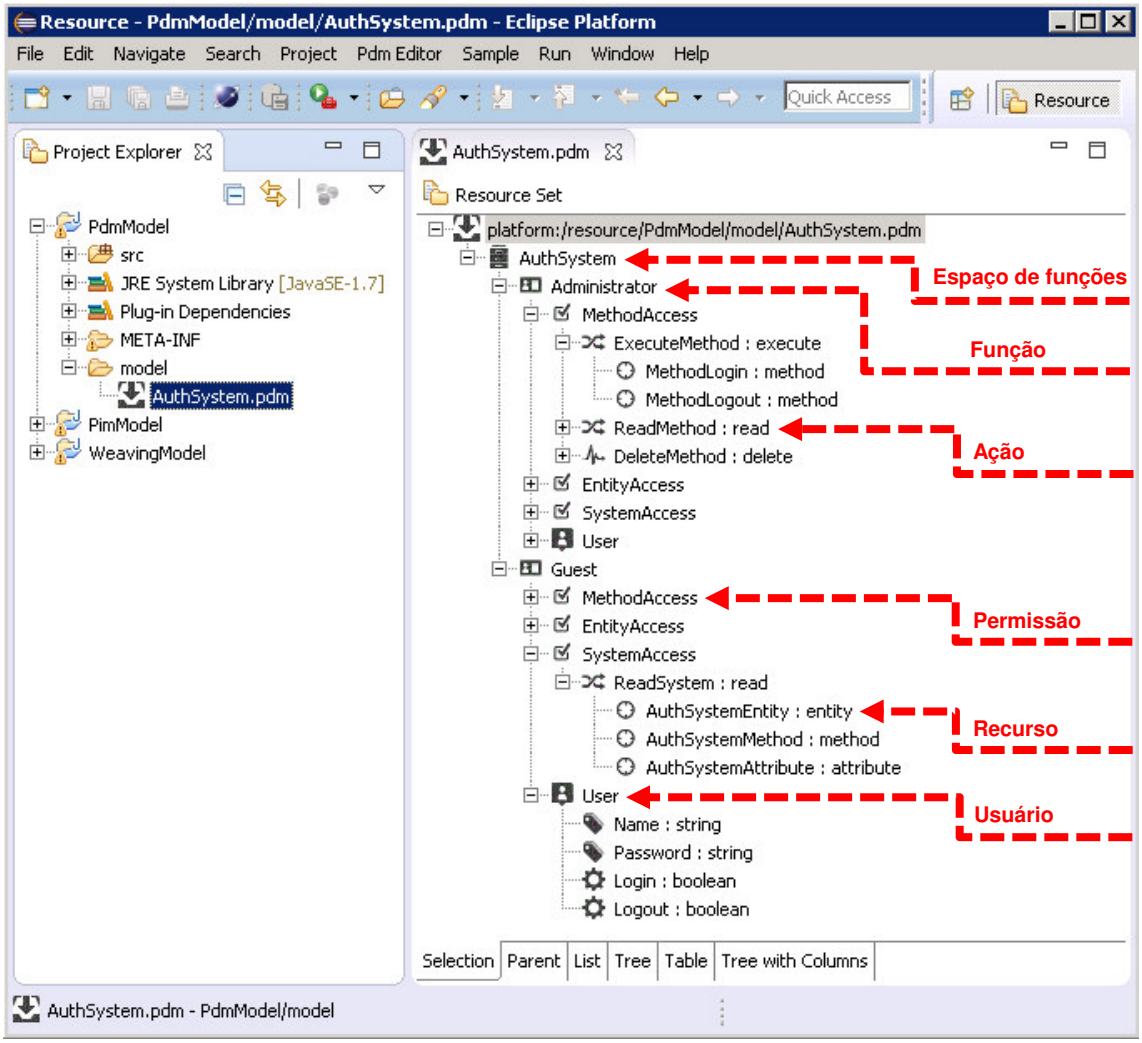


Figura 6.5: Modelo PDM de segurança AuthSystem.pdm

6.3 Modelo *Weaving*

Após a definição dos modelos PIM e PDM, é apresentada a definição do modelo *weaving*. O modelo *weaving* foi construído conforme o metamodelo para *weaving* apresentado na Figura 4.4 do capítulo 4. Assim como os outros metamodelos propostos, o metamodelo *weaving* também foi modelado através da ferramenta EMF do Eclipse para permitir a criação e adaptação dos *plug-ins* com o objetivo de viabilizar o recurso de fusão de modelos PIM e PDM.

A Figura 6.6 denota o metamodelo weaving.ecore com seus respectivos elementos para criação de modelos *weaving*. O metamodelo apresenta uma estrutura suficiente para armazenar informações do histórico de definições de *Weaving*, que por sua vez contêm em seus manipuladores ponteiros para os elementos dos modelos a serem relacionados, armazenando dessa forma a referência para os metamodelos de origem.

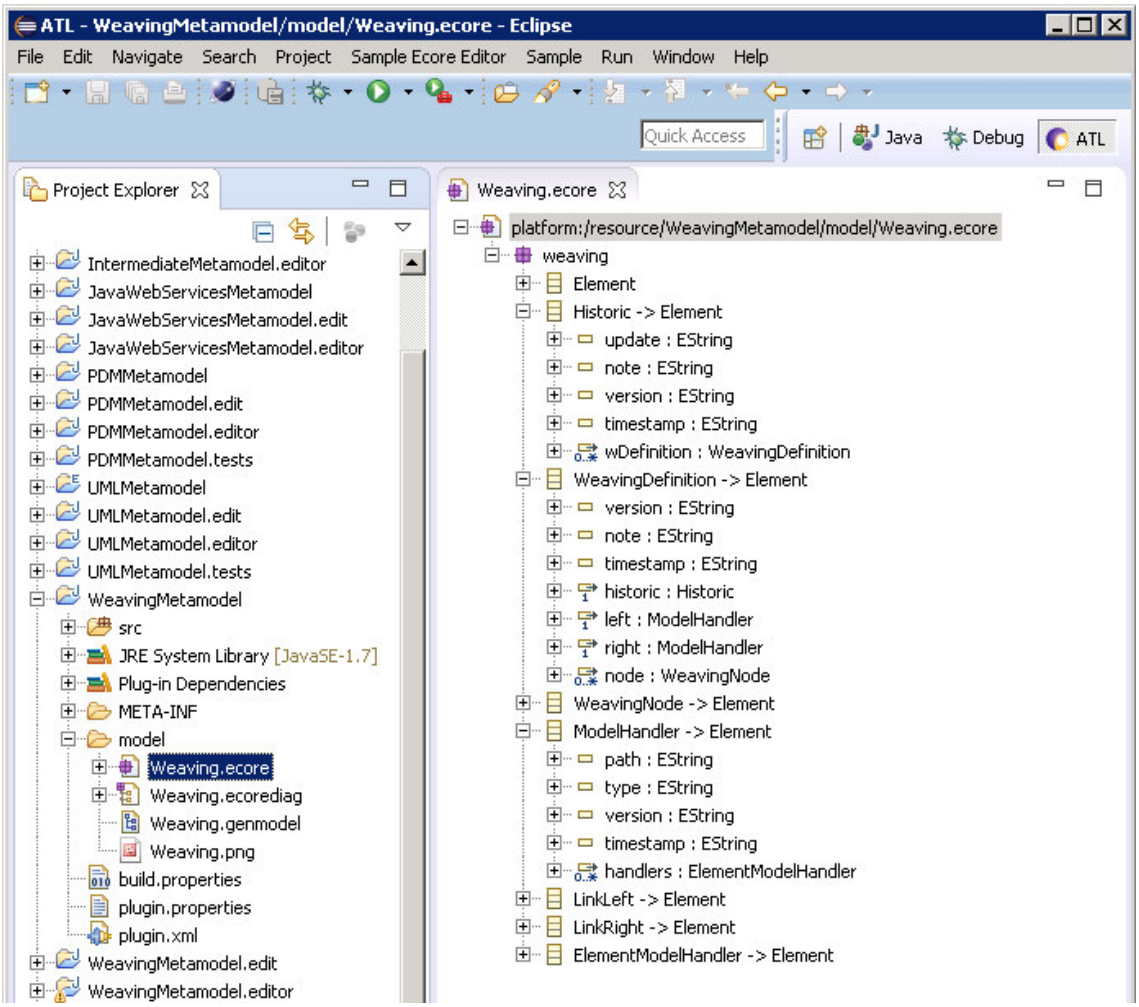


Figura 6.6: Metamodelo *Weaving* weaving.ecore

Com base no metamodelo **weaving.ecore** é criado o modelo gerador de *Weaving* **weaving.genmodel**. Tal modelo contém as informações adicionais para a geração de código, incluindo os parâmetros de controle sobre como o código deve ser gerado. O **weaving.genmodel** gerou os projetos de *plug-ins* WeavingMetamodel.edit e WeavingMetamodel.editor. Mediante adaptações nos códigos dos provedores do *plug-in* Edit, foi possível customizar os rótulos de descrição dos elementos e seus respectivos ícones do modelo.

Através da modificação dos códigos do *plug-in* Editor, foi necessário adaptar a interface do assistente para que fosse incluído o suporte à integração dos dois modelos de entrada bem como o suporte a visualização dos dois modelos de entrada no painel principal, juntamente com os *handlers* do modelo *weaving* populados com os elementos de cada modelo de entrada. A Figura 6.7 adiante demonstra a tela de criação do modelo *weaving* no momento em que o assistente solicita a escolha dos modelos de entrada, da esquerda e da direita. Neste caso foram selecionados o modelo OHRS e o modelo AuthSystem.

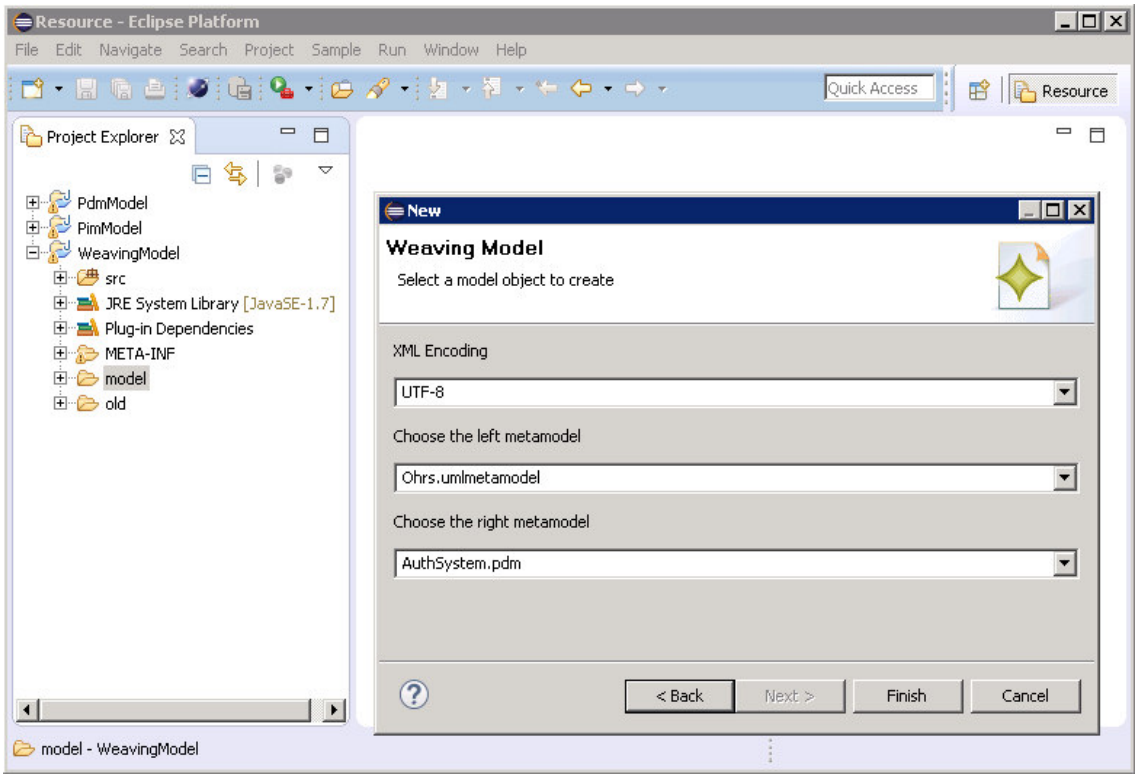


Figura 6.7: Interface de criação do modelo *Weaving*

Destaca-se a necessidade de alterar o arquivo WeavingModelWizard.java no *plug-in* Editor, incluindo principalmente a programação dos trechos de código responsáveis pelo carregamento dos modelos da direita e da esquerda na interface do assistente. A Listagem 6.3 detalha este trecho de código. Outro trecho importantíssimo que também sofreu modificações trata da parte responsável por navegar pelos modelos e popular os *handlers* tanto da direita quanto da esquerda. A Listagem 6.4 demonstra um trecho desse código.

Listagem 6.3: Trecho de código que carrega os modelos no assistente

```
312  //Obter os pacotes.
313  Namespace leftModel = null;
314  RoleSpace rightModel = null;
315
316  try {
317        //Carregar o modelo da esquerda
318        editingDomain.loadResource(fileLeftURI.toString());
319
320        //Obter o pacote do modelo da esquerda
321        leftModel = (Namespace) ((Resource)
322  editingDomain.getResourceSet().getResources().get(0)).getContents().get(0);
323
324        Package eLeftModel = (Package) leftModel.getOwnedElement().get(0);
325
326        JOptionPane.showMessageDialog(null, "Modelo '" +
327  leftModel.getName() + "' carregado com sucesso!");
328
329        } catch (Exception e) {
330
```

```
331                 MessageDialog.openError(null,
332 WeavingEditorPlugin.INSTANCE.getString("_UI_Wizard_label"), e.toString());
333                 return null;
334           }
335
336           try {
337                 //Carregar o modelo da direita
338                 editingDomain.loadResource(fileRightURI.toString());
339
340                 //Obter o pacote do modelo da direita
341                 rightModel = (RoleSpace) ((Resource)
342 editingDomain.getResourceSet().getResources().get(1)).getContents().get(0);
343
344                 Role eRightModel = (Role) rightModel.getRoles().get(0);
345
346                 JOptionPane.showMessageDialog(null, "Modelo '" +
347 rightModel.getName() + "' carregado com sucesso!");
348
349           } catch (Exception e) {
350                 MessageDialog.openError(null,
351 WeavingEditorPlugin.INSTANCE.getString("_UI_Wizard_label"), e.toString());
352                 return null;
           }
```

Listagem 6.4: Trecho de código que navega pelos modelos

```
349 //Popular cada ModelHandler com Handlers
350 EList leftHandlers = getHandlersLeft(leftModel, left);
351
352 if (leftHandlers.size() == 0) {
353       MessageDialog.openError(null,
354 WeavingEditorPlugin.INSTANCE.getString("_UI_Wizard_label"), "The handler
355 for " + left.getName() + " cannot be created.");
356       return null;
357 }
358
359 EList rightHandlers = getHandlersRight(rightModel, right);
360
361 if (rightHandlers.size() == 0) {
362                   MessageDialog.openError(null,
363 WeavingEditorPlugin.INSTANCE.getString("_UI_Wizard_label"), "The handler
364 for " + right.getName() + " cannot be created.");
365       return null;
366 }
367
368 left.getHandlers().addAll(leftHandlers);
369 right.getHandlers().addAll(rightHandlers);
370
371 left.setName(leftModel.getName());
372 right.setName(rightModel.getName());
373
374 def.setLeft(left);
375 def.setRight(right);
376
377 String leftName = left.getName();
378 String rightName = right.getName();
379
380 def.setName(leftName + "_" + rightName);
381
382 if (rootObject instanceof Historic) {
383       Historic rootHistoric = (Historic) rootObject;
384       rootHistoric.getWDefinition().add(def);
385 }
386 else{
387
388
```

```
389         MessageDialog.openError(null,
390 WeavingEditorPlugin.INSTANCE.getString("_UI_Wizard_label"), "The root
390 element is not Historic!");
391         return null;
392 }
393
            return rootObject;
    }
```

Dessa forma também foi ajustado o modo de visualização dos modelos através da edição de código do arquivo WeavingTreeViewer.java do *plug-in* Editor, de maneira que o assistente pudesse ter uma melhor experiência na manipulação dos elementos a serem relacionados. Como resultado, o assistente retorna a interface ilustrada na Figura 6.8 onde do lado esquerdo é exibido o modelo PIM do Sistema de Reserva de Hotel Online no formato de árvore, e à direita é exibido modelo PDM AuthSystem relativo aos aspectos de controle de acesso. Ao centro é exibido o modelo Weaving com seus *handlers* já populados com cada elemento dos modelos de entrada da esquerda e da direita. A princípio o modelo *weaving* apenas lista cada elemento, um por um, e os classifica como seus *Element Model Handlers*. Posteriormente na ação do *weaving*, uma outra parte do *weaving* é populada, os nós (*Nodes*) da árvore do *Weaving Definition* Ohrs_AuthSystem. Na primeira seleção dos elementos correspondentes determinados através da intervenção manual do especialista, a análise dos principais elementos expostos no modelo é feita. Os nós carregam informações sobre o nome, a descrição e o ponto de ligação entre o elemento da esquerda e o elemento da direita.

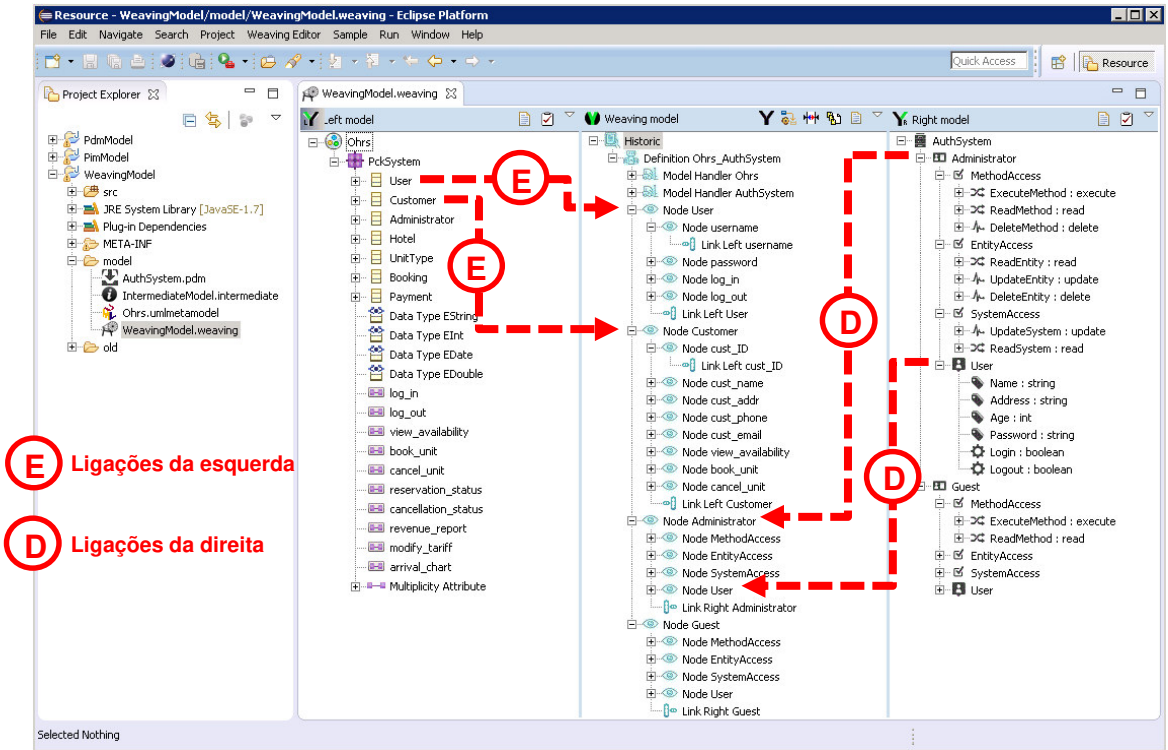


Figura 6.8: Interface de criação dos nós do modelo *Weaving*

A partir deste ponto, o especialista já dispõe de informações suficientes para abstrair como e quais elementos são os melhores candidatos a sofrerem fusão. Essa escolha ocorre de maneira manual visto que a ideia de unificar dois modelos com funcionalidades e semânticas distintas não se trata de uma tarefa trivial. Porém com o conhecimento prévio de ambos os domínios, é possível chegar ao próximo modelo do *framework* que será tratado na seção adiante.

6.4 Modelo intermediário

O modelo intermediário foi definido conforme o metamodelo intermediário proposto no capítulo 4, ilustrado pela Figura 4.6. No modelo intermediário estão presentes as informações de classes, propriedades e métodos, correspondentes aos elementos mesclados durante a fase de *weaving*. Este modelo apresenta um conteúdo mais refinado, do que realmente será manipulado nas transformações para modelos mais concretos. A partir do modelo intermediário aplicamos as primeiras definições de transformações para chegarmos as especificações de plataformas alvo. Apresentamos primeiramente através da Figura 6.9 o metamodelo definido em Ecore para a criação de modelos intermediários.

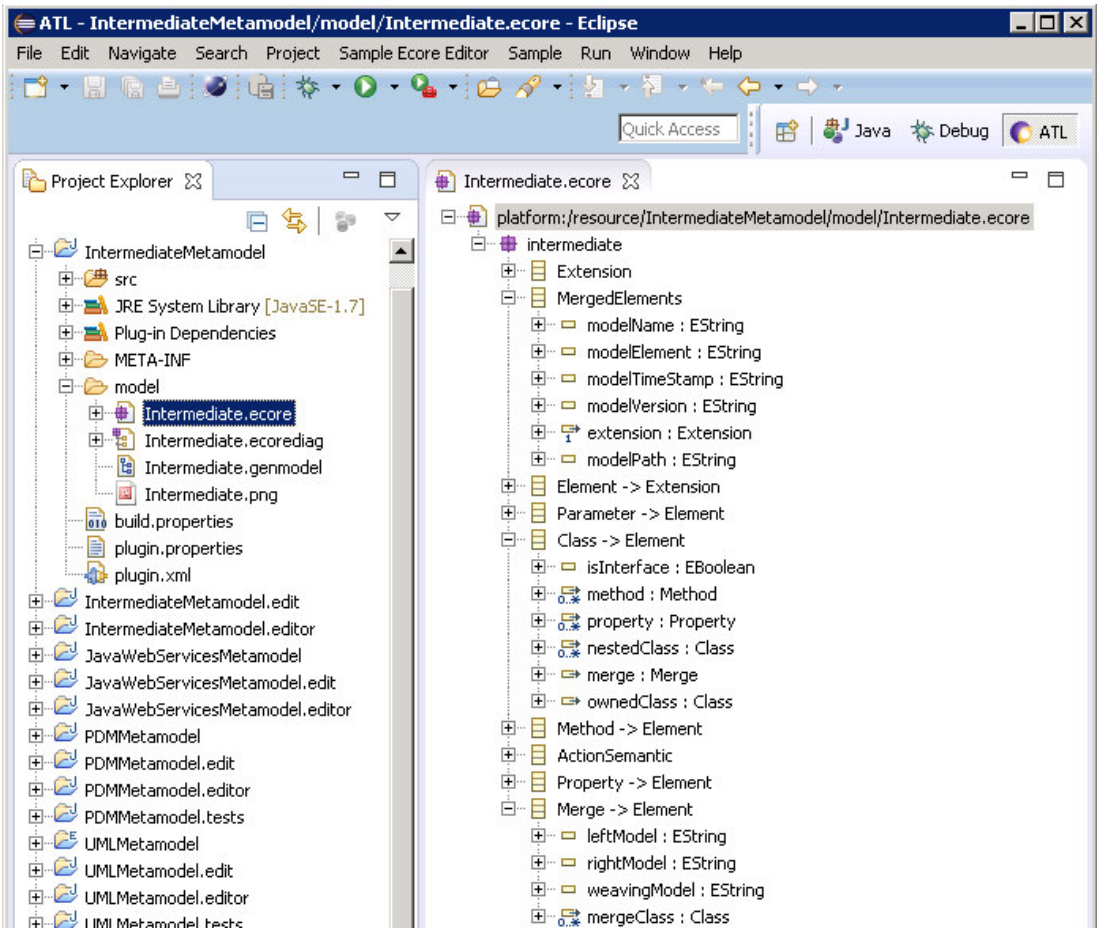


Figura 6.9: Metamodelo intermediário Intermediate.ecore

O elemento *MergedElements* contém a referência para o modelo de origem de cada elemento de modelo mesclado, com sua versão e caminho especificados. O elemento *Merge* faz referência aos modelos da esquerda, da direita, e ao modelo *weaving* utilizados no processo de fusão.

Para a criação do modelo intermediário também foram utilizados os recursos de EMF através da geração dos *plug-ins* que permitiu a edição dos elementos do modelo em tempo de execução. A Figura 6.10 descreve o modelo intermediário instanciado após realizada a fusão entre os elementos do modelo PIM do Sistema de Reserva de Hotel Online e os elementos do modelo PDM do sistema de controle de acesso baseado em funções.

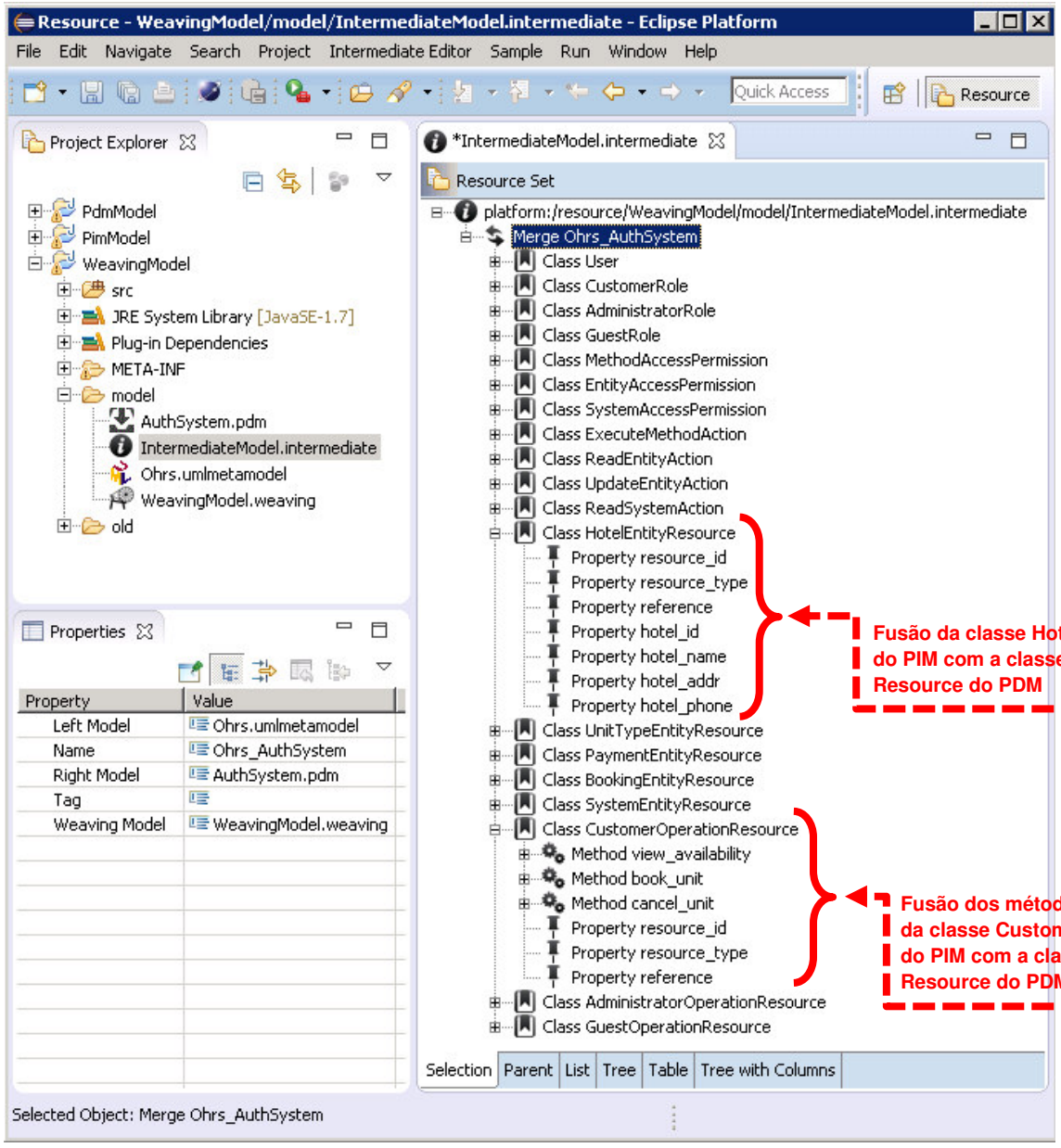


Figura 6.10: Modelo intermediário IntermediateModel.intermediate

A lógica utilizada para a escolha dos elementos a serem mesclados se baseou na assimilação da funcionalidade de cada sistema. No exemplo ilustrativo foram analisadas primeiramente as funcionalidades do Sistema de Reserva de Hotel Online. Neste sistema o usuário pode ser um cliente (*Customer*) ou um administrador (*Administrator*). Dependendo da sua função no sistema o usuário pode se limitar a fazer reservas, consultar a disponibilidade de quartos e realizar pagamentos, ou desempenhar além destas, outras tarefas de gerenciamento como realizar cancelamentos de reservas, modificar tarifas e gerar relatórios. Logo, observou-se que as entidades *Customer* e *Administrator* são características de funções, as quais habilitam o usuário a desempenhar ações específicas dentro do sistema. Tais ações têm efeito sobre entidades como *Hotel*, *Payment*, *UnitType* e *Booking*, denotando características de recursos. Então, numa breve análise em paralelo das funcionalidades do sistema de controle de acesso baseado em funções, identificou-se que várias características encontradas no OHRS tinham relacionamento direto com muitos dos elementos descritos no sistema de segurança em questão. Tal análise denotou a viabilidade de se mesclar elementos de ambos modelos baseada nas correspondências encontradas.

A Figura 6.11 ilustra algumas dessas correspondências encontradas na análise dos dois modelos de entrada, onde a classe *Administrator* do PIM caracteriza uma função da classe *Role* do PDM, os métodos da classe *Administrator* caracterizam a classe *Action* do PDM, e a classe *UnitType* caracteriza a classe *Resource* do PDM.

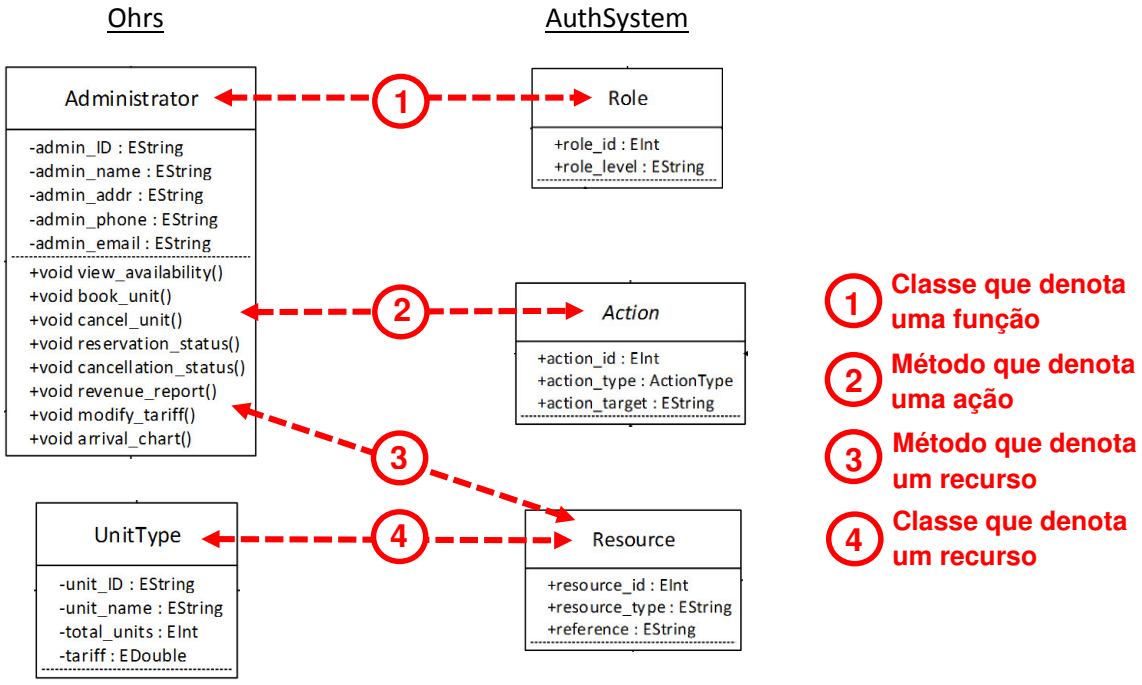


Figura 6.11: Algumas características correspondentes entre modelos

No novo cenário do modelo intermediário, algumas classes foram unificadas e outras tiveram apenas alguns de seus elementos importados para uma nova classe, como é o caso das propriedades das classes *Customer* e *Administrator*. Considerando que estas classes assumiram a partir de então a nova funcionalidade de *Roles*, suas propriedades foram migradas para a entidade *User* a qual define as características pessoais de cada usuário, como nome, endereço, ID, telefone, nome de usuário, senha, etc.

As classes *Hotel*, *UnitType*, *Payment*, e *Booking* assumiram o valor de *Resources* tendo suas propriedades mescladas em uma única classe contendo as informações de identificação de cada recurso e seu tipo. Os métodos que até então eram vinculados ao tipo de usuário, foram separados e agora integram características de recursos, acessados através da *Action* de execução de métodos. A Figura 6.12 ilustra de maneira mais clara através de um diagrama o que seria o novo modelo integrado com base nessa série de modificações.

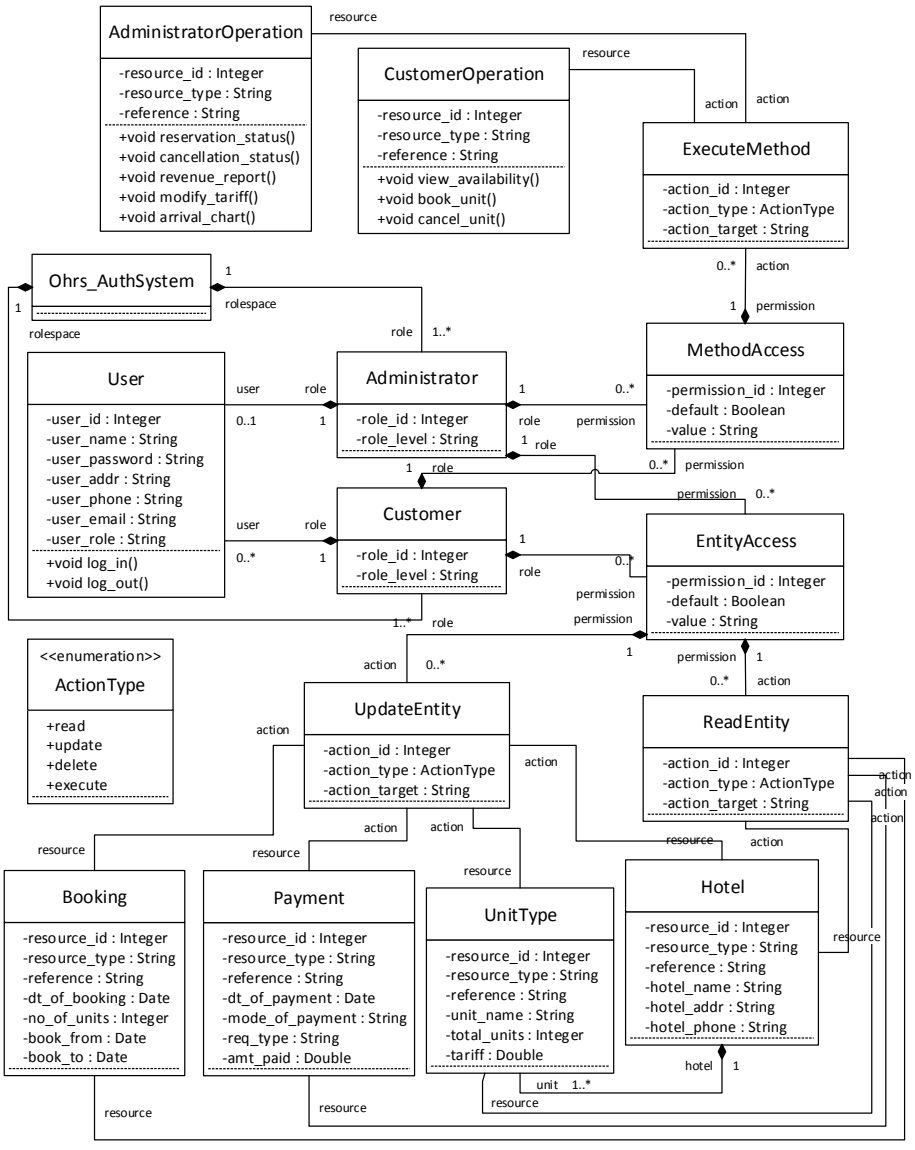


Figura 6.12: Modelo Ohrs_AuthSystem

A etapa seguinte a ser realizada trata da criação das definições de transformação, que em um primeiro momento tomam o modelo intermediário criado como entrada, e geram um modelo PSM abstrato de saída. Em um segundo momento o modelo PSM abstrato passa por uma nova etapa de transformação conforme descrito no *framework*, onde é então alcançado um modelo PSM concreto. Por fim, este modelo PSM concreto passa por uma nova iteração de transformação gerando então o código fonte a ser implementado da aplicação.

6.5 Definições de transformação

Para a etapa de transformação, um projeto em ATL foi desenvolvido, armazenando todas as definições de transformação de modelos utilizadas no processo do *framework* proposto. Três definições de transformação referente a cada etapa de evolução do processo foram criadas. A primeira definição descreve a transformação do modelo intermediário, adquirido na iteração anterior, para o modelo PSM abstrato descrito em WSDL. Para essa definição utilizou-se o metamodelo intermediário e o metamodelo para *Web Services*, definidos nas seções 4.4.3 e 4.4.4 respectivamente deste trabalho. Nesta primeira definição de transformação chamada intermediate2wsdl toma-se como entrada o modelo IntermediateModel.intermediate, tendo como saída o modelo psm_wsdl.xmi, conforme ilustrado na Figura 6.13 a seguir.

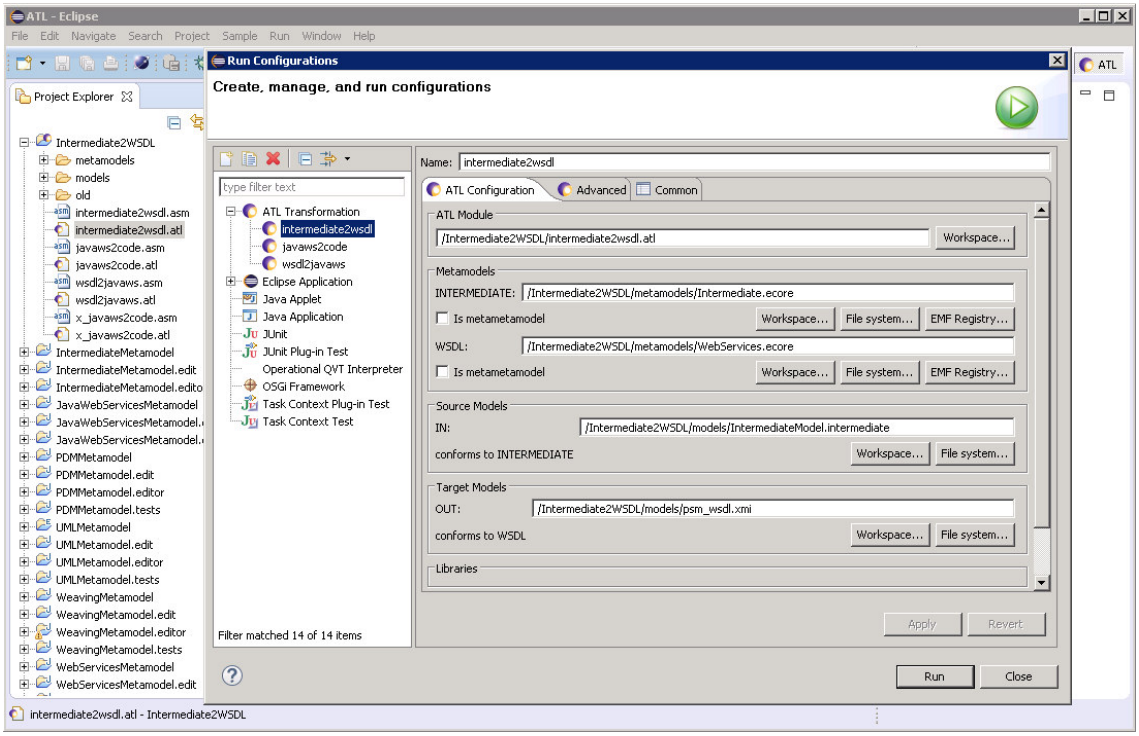


Figura 6.13: Definição de transformação intermediate2wsdl

A Listagem 6.5 adiante apresenta as regras definidas em ATL para transformação do modelo intermediário para o modelo WSDL.

Listagem 6.5: Fragmento de código em ATL que transforma modelo intermediário em modelo WSDL

```
1  -- @path INTERMEDIATE=/Intermediate2WSDL/metamodels/Intermediate.ecore
2  -- @path WSDL=/Intermediate2WSDL/metamodels/WebServices.ecore
3
4  module intermediate2wsdl;
5  create OUT : WSDL from IN : INTERMEDIATE;
6
7  rule Merge2Definition {
8        from merge : INTERMEDIATE!Merge
9        to definition : WSDL!Definition (
10             name <- 'Service_' + merge.name,
11             targetNameSpace <- 'urn://' + merge.name + '.wsdl'
12             )
13 }
14
15 rule Class2Service {
16       from class : INTERMEDIATE!Class
17       to service : WSDL!Service (
18             name <- class.name,
19             ownerServ <- class.merge,
20             port <- port
21             ),
22             port : WSDL!Port (
23                    name <- class.name + 'Port',
24                    binding <- binding
25             ),
26             binding : WSDL!Binding (
27                    name <- class.name + 'Binding',
28                    ownerBind <- class.merge,
29                    type <- portType
30             ),
31             portType : WSDL!PortType (
32                    name <- class.name,
33                    ownerPTyp <- class.merge,
34                    binding <- binding
35             )
36 }
37
38 rule Property2Types {
39       from property : INTERMEDIATE!Property
40       to types : WSDL!Types (
41             name <- property.name,
42             ownerTyp <- property.class.merge,
43             documentation <- property.tag
44             )
45 }
46
47 rule Method2Message {
48       from method : INTERMEDIATE!Method
49       to message : WSDL!Message (
50             name <- method.name,
51             ownerMess <- method.class.merge,
52             documentation <- method.tag
53             )
54 }
```

Conforme a definição de transformação apresentada na Listagem 6.5, a regra Merge2Definition transforma os elementos *Merge* do modelo intermediário em elementos *Definition* do modelo *Web Service*. A regra Class2Service captura as informações dos elementos *Class* do modelo intermediário para produzir os elementos *Service*, *Port*, *Binding* e *PortType* do modelo *Web Services*. Do mesmo modo, as regras *Property2Types* e *Method2Message* transformam os elementos *Property* e *Method* do modelo intermediário em elementos *Type* e *Message* do modelo *Web Services*.

O modelo PSM abstrato baseado em *Web Services* psm_wsdl.xmi é apresentado a seguir na Figura 6.14 no formato em árvore, como resultado da definição de transformação anterior descrita na Listagem 6.5. O modelo PSM abstrato apresenta a definição para *Web Services* Ohrs_AuthSytem. Essa definição contém elementos representando os componentes *Type*, *Message*, *Port Type*, *Binding* e *Services*. Tais elementos estão diretamente relacionados ao modelo intermediário, no qual *Properties* foram transformados em *Types*, *Methods* foram transformados em *Messages*, *Classes* foram transformados em *Services*, e algumas características de *Classes* foram transformadas em *Port Types* e *Bindings*.

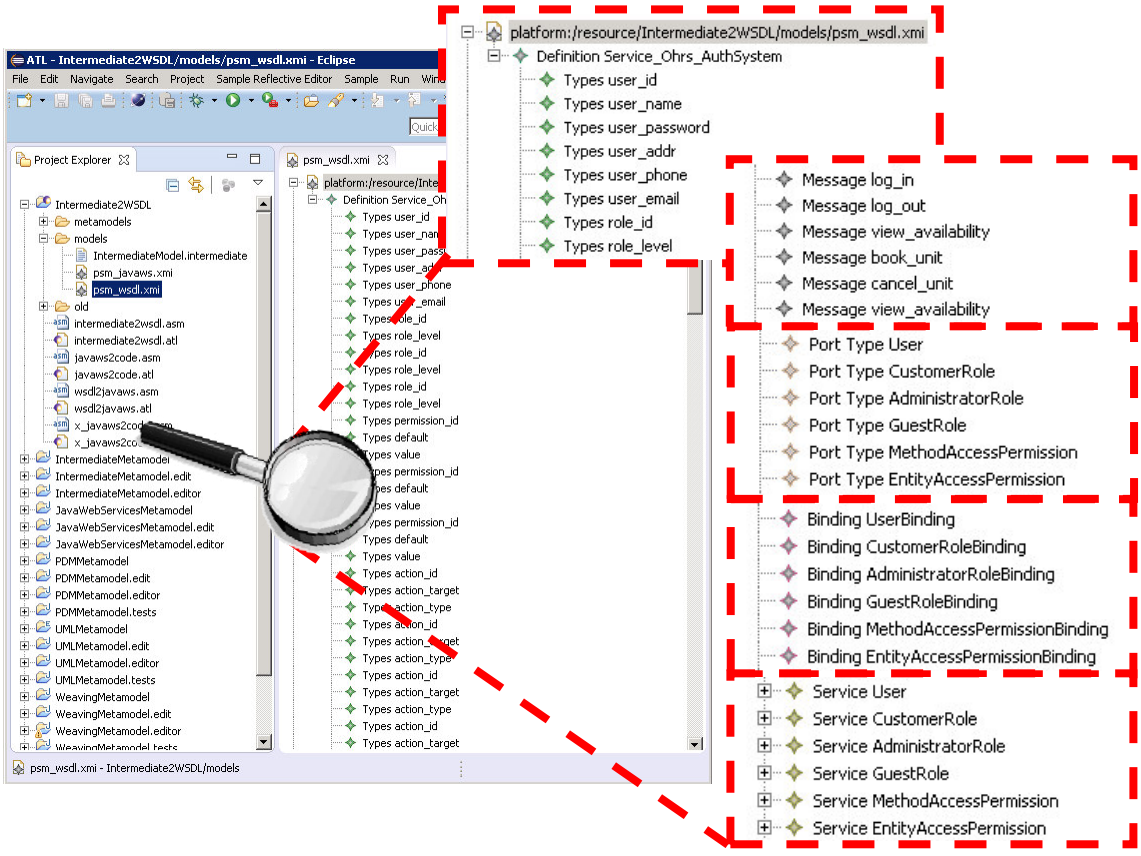


Figura 6.14: Modelo PSM abstrato baseado em *Web Services*

De acordo com o *framework*, a transformação que se segue trata da transformação do modelo PSM abstrato em um modelo PSM concreto. Na configuração da definição desta transformação utilizou-se o metamodelo para *Web Services* e o metamodelo para *Java Web Services*, conforme definidos respectivamente nas Figuras 4.7 e 4.8 do capítulo 4. Esta segunda definição de transformação foi chamada de wsdl2javaws e tomou como entrada o modelo psm_wsdl.xmi resultante da transformação anterior, tendo como saída o modelo psm_javaws.xmi conforme ilustra a Figura 6.15 adiante.

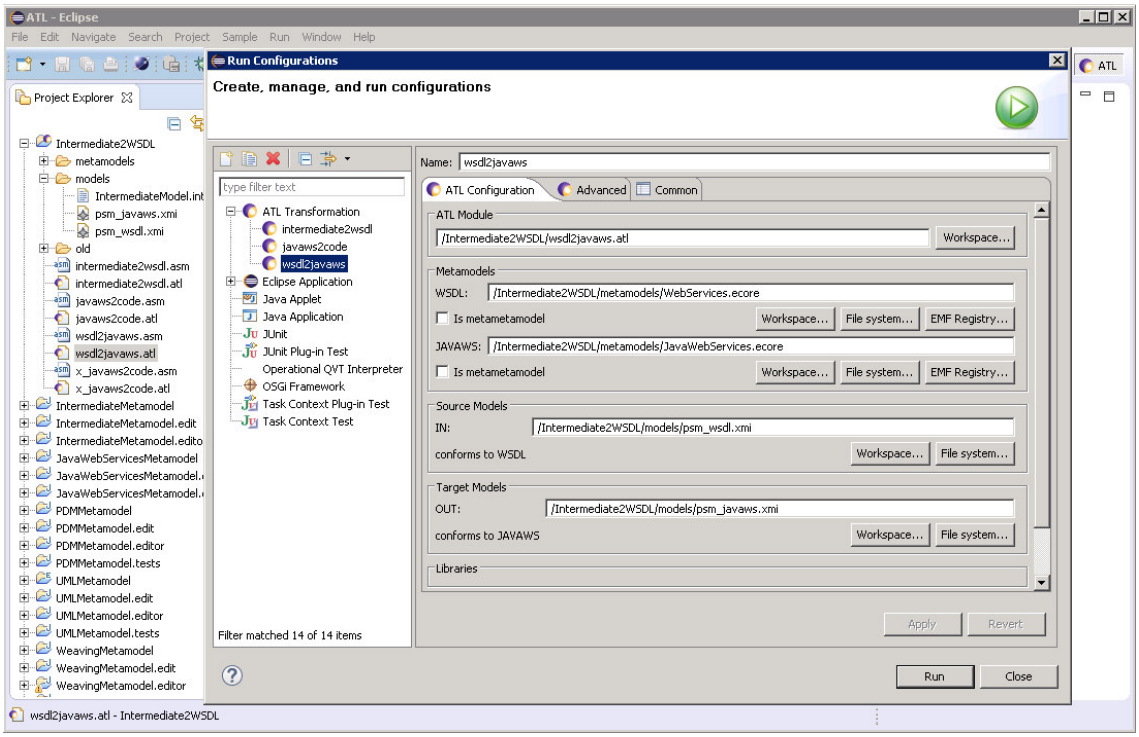


Figura 6.15: Definição de transformação wsdl2javaws

A Listagem 6.6 a seguir apresenta as regras definidas em ATL para transformação do modelo PSM abstrato WSDL em modelo PSM concreto *Java Web Services*.

Listagem 6.6: Fragmento de código em ATL que transforma modelo WSDL em modelo Java *Web Services*

```
1  -- @path WSDL=/Intermediate2WSDL/metamodels/WebServices.ecore
2  -- @path JAVAWS=/Intermediate2WSDL/metamodels/JavaWebServices.ecore
3
4  module wsdl2javaws;
5  create OUT : JAVAWS from IN : WSDL;
6
7  rule Definition2WebServices {
```

```
8         from definition : WSDL!Definition
9         to wservice : JAVAWS!WebServices (
10              name <- definition.name,
11              visibility <- 'public'
12              )
13  }
14
15  rule Service2WSClass {
16        from service : WSDL!Service
17        to wsclass : JAVAWS!WSClass (
18              name <- service.name,
19              visibility <- 'public',
20              isActive <- true,
21              owner <- service.ownerServ
22        )
23  }
24
25  rule Types2WSAttribute {
26        from types : WSDL!Types
27        to wsattribute : JAVAWS!WSAttribute (
28              name <- types.name,
29              visibility <- 'public',
30              owner <- types.ownerTyp.service -> select (e|e.name =
31  types.documentation)
32        )
33  }
34
35  rule Message2WSMethod {
36        from message : WSDL!Message
37        to wsmethod : JAVAWS!WSMethod (
38              name <- message.name,
39              visibility <- 'public',
40              owner <- message.ownerMess.service -> select (e|e.name =
41  message.documentation)
42        )
43  }
```

Segundo as regras de definição de transformação da Listagem 5.6, a regra Definition2WebServices transforma os elementos de *Definition* do modelo *Web Services* em elementos *Web Services* do modelo *Java Web Services*. A regra Service2WSClass transforma os elementos *Service* do modelo *Web Services* em elementos *Web Service Class* do modelo *Java Web Services*. A regra Types2WSAttributes transforma os elementos *Type* do modelo *Web Services* em elementos *Web Service Attribute* do modelo *Java Web Services*. A regra Message2WSMethod transforma os elementos *Message* do modelo *Web Services* em elementos *Web Service Method* do modelo *Java Web Services*.

A Figura 6.16 a seguir descreve no formato de árvore o modelo PSM concreto baseado em *Java Web Services* gerado pela definição de transformação anterior. Neste modelo PSM concreto é apresentado o *Java Web Services* chamado Service_Ohrs_AuthSystem. Este serviço contém os elementos *WS Class*, *WS Attribute* e *WS Method*, os quais representam os elementos transformados do modelo *Web Services*. Como resultado, *Services* foram transformados em *Java Web Service Classes*, *Types* foram transformados em *Java Web Service Attributes*, e *Messages* foram transformados em *Java Web Service Methods*.

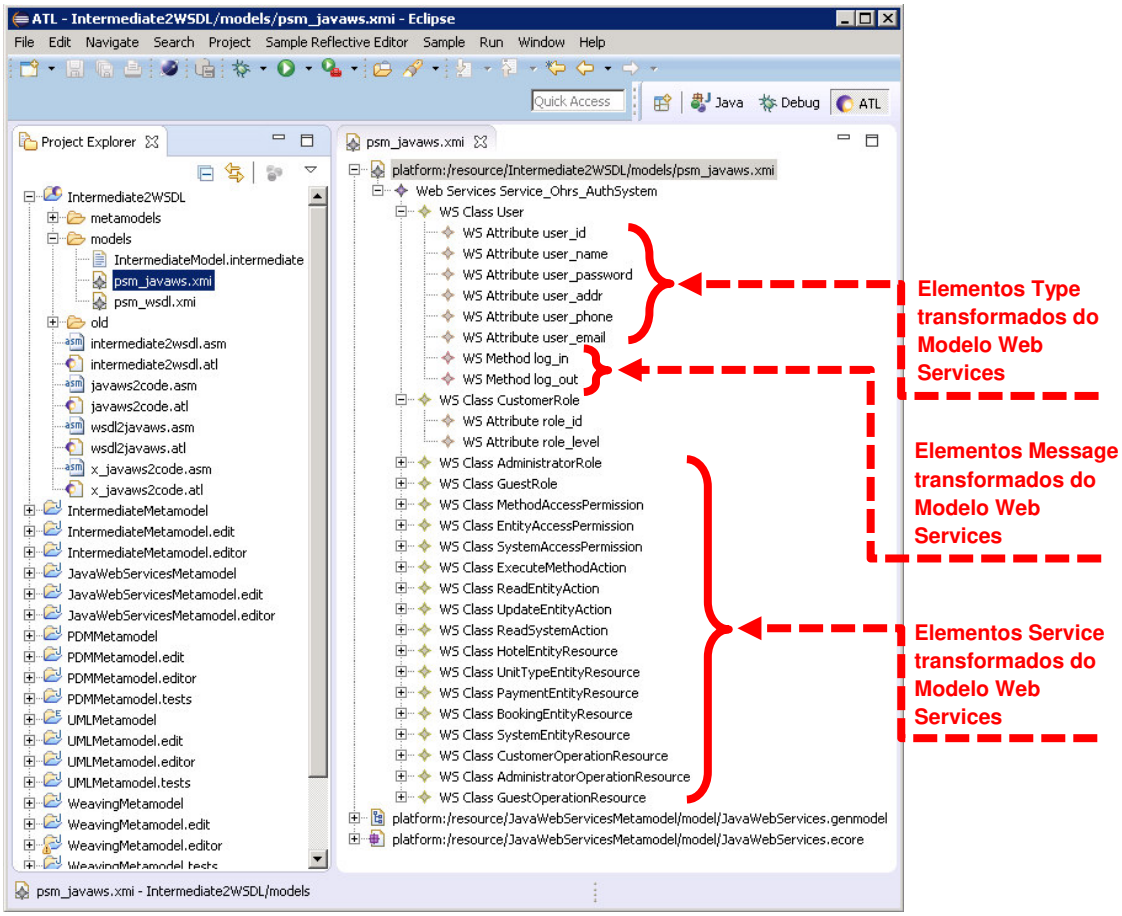


Figura 6.16: Modelo PSM concreto baseado em Java *Web Services*

Até então, as definições de transformação trabalhadas se baseavam em nível de transformações modelo-a-modelo. A terceira etapa de transformações descrita no *framework* se refere as definições de transformação a nível de modelo-a-texto, ou seja, do modelo PSM concreto em código fonte. Neste cenário alguns outros recursos são requeridos, como por exemplo a estrutura de *query* e *library*. Em ATL a função da *query* é navegar pela estrutura do modelo coletando suas informações com a ajuda de uma biblioteca definida para o modelo. O conceito de *library* denota a função de auxiliadores que capturam as características de elementos de um determinado modelo de entrada e os compõe de forma textual.

Para cumprir esta etapa, a definição de transformação javaws2code foi criada, a qual utilizou o metamodelo para *Java Web Services* apresentado na seção 4.4.4 do capítulo 4 e teve como entrada o modelo criado na transformação anterior psm_javaws.xmi. Nesta definição de transformação não há a definição de um modelo alvo, porém é necessário especificar a *library* utilizada x_javaws2code, que será apresentada mais adiante. A Figura 6.17 descreve a configuração para a definição de transformação javaws2code.

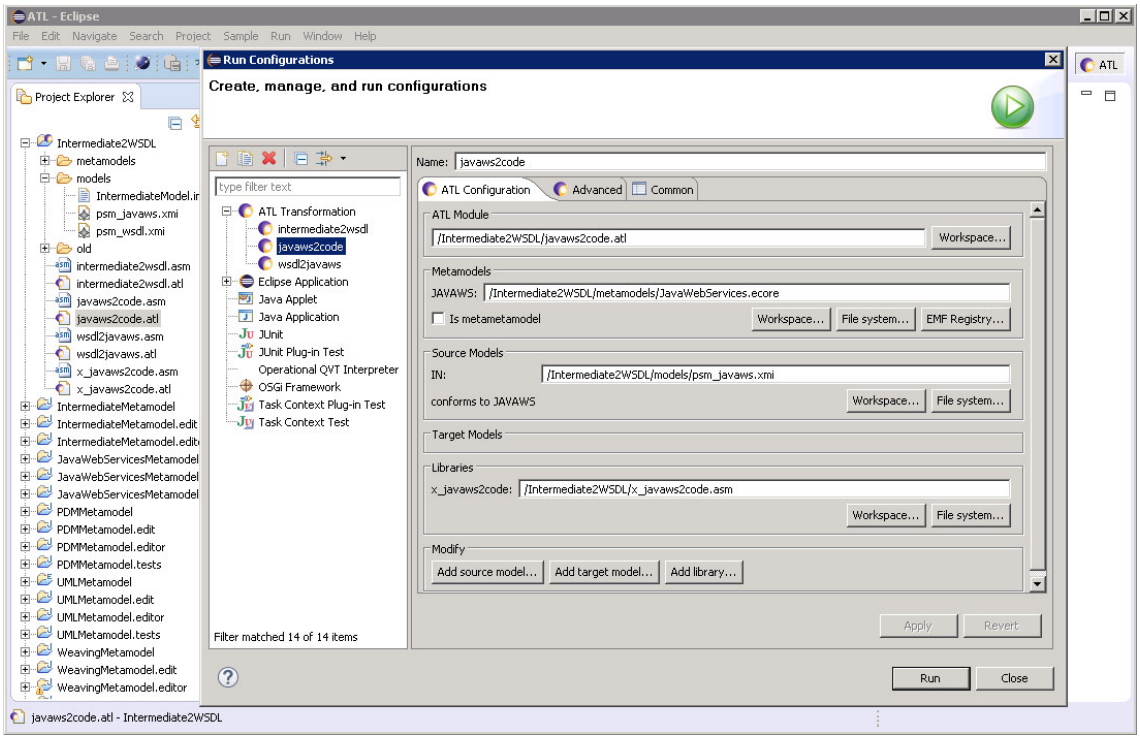


Figura 6.17: Definição de transformação javaws2code

As Listagens 6.7 e 6.8 apresentam as definições de transformação do modelo PSM concreto para o código fonte. A Listagem 6.7 descreve a definição de transformação javaws2code, a qual contem a *query* javaws2code_query que aciona a transformação de modelo *Java Web Services* a código. A operação *allInstances()* retorna uma coleção de todas as instâncias das classes do modelo *Java Web Services*. Enquanto que a operação *collect()* recupera cada elemento da coleção e chama a operação *toString()* para cada um deles. A *query* usa os auxiliadores definidos na *library* x_javaws2code conforme descrita na Listagem 6.8, que apresentam os *helpers* para pesquisar e extrair informações do modelo *Java Web Services* escrevendo-os em um arquivo Java no destino local "C:/SourceCode/JavaWS/".

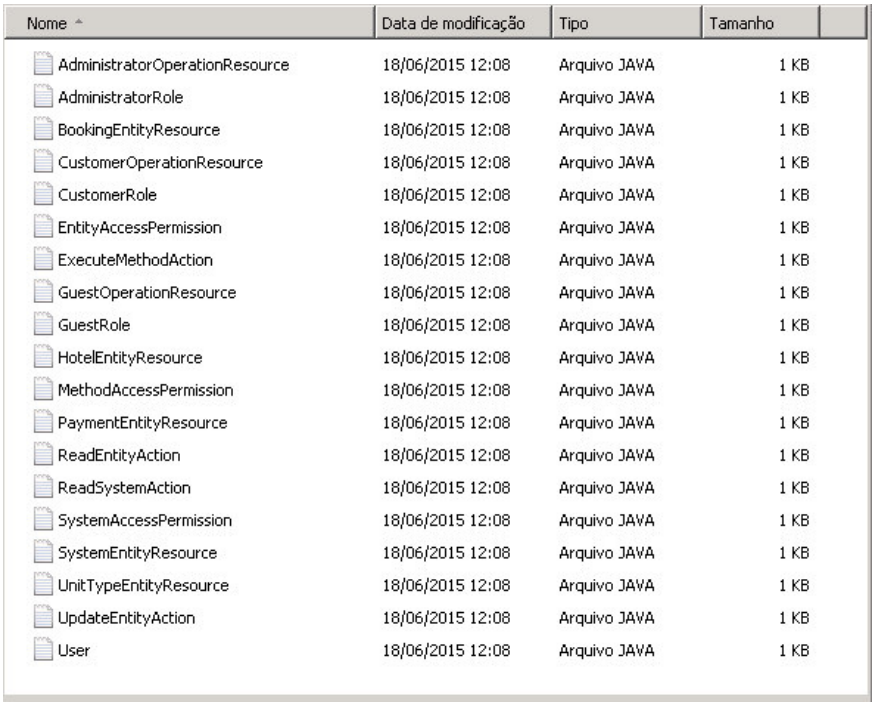
Listagem 6.7: *Query* para definição de transformação de modelo PSM concreto em código fonte

```
1  query javaws2code_query = JAVAWS!WSClass.allInstances() ->
2        collect (x | x.toString().writeTo
3        ('C:/SourceCode/JavaWS/' + x.owner.name + '/' + x.name +
4  '.java'));
5
   uses x_javaws2code;
```

Listagem 6.8: *Library* para definição de transformação de modelo PSM concreto em código fonte

```
1  library x_javaws2code;
2
3  helper context JAVAWS!WSClass def : toString() : String =
4        'package ' + self.owner.name + ';' + '\n\n' +
5        'import javax.jws.WebMethod;' + '\n' +
6        'import javax.jws.WebParam;' +
7        '\n' + 'import javax.jws.WebService;' +
8
9        '\n\n' + '@WebService()' + '\n' +
10       if (self.visibility = 'public') then 'public' else 'private' endif
11 +     ' class ' + self.name + ' { \n \n' +
12             self.wsAttribute -> iterate (i; acc : String = '' | acc +
13             i.toString()) +
14             self.wsMethod -> iterate (i; acc : String = '' | acc +
15             i.toString()) +
16       '} \n\n';
17
18 helper context JAVAWS!WSAttribute def : toString() : String =
19       '\t' + '@EJB \n \t' + if (self.visibility = 'public') then
20 'public'     else 'private ' endif + ' /*type*/ ' + self.name + '; \n';
21
22 helper context JAVAWS!WSMethod def : toString() : String =
23       '\n \t' + '@WebMethod \n \t' + if (self.visibility = 'public')
24 then   'public' else 'private ' endif +
25       ' /*returnType*/ ' + self.name + '() { \n' + '\t } \n';
```

A Figura 6.18 apresenta os arquivos Java gerados após a transformação do modelo PSM concreto para código fonte. Os arquivos Java gerados contêm o código para cada elemento *Class* baseado no modelo *Java Web Services*, incluindo seus respectivos *Methods* e *Attributes*.



Nome	Data de modificação	Tipo	Tamanho
AdministratorOperationResource	18/06/2015 12:08	Arquivo JAVA	1 KB
AdministratorRole	18/06/2015 12:08	Arquivo JAVA	1 KB
BookingEntityResource	18/06/2015 12:08	Arquivo JAVA	1 KB
CustomerOperationResource	18/06/2015 12:08	Arquivo JAVA	1 KB
CustomerRole	18/06/2015 12:08	Arquivo JAVA	1 KB
EntityAccessPermission	18/06/2015 12:08	Arquivo JAVA	1 KB
ExecuteMethodAction	18/06/2015 12:08	Arquivo JAVA	1 KB
GuestOperationResource	18/06/2015 12:08	Arquivo JAVA	1 KB
GuestRole	18/06/2015 12:08	Arquivo JAVA	1 KB
HotelEntityResource	18/06/2015 12:08	Arquivo JAVA	1 KB
MethodAccessPermission	18/06/2015 12:08	Arquivo JAVA	1 KB
PaymentEntityResource	18/06/2015 12:08	Arquivo JAVA	1 KB
ReadEntityAction	18/06/2015 12:08	Arquivo JAVA	1 KB
ReadSystemAction	18/06/2015 12:08	Arquivo JAVA	1 KB
SystemAccessPermission	18/06/2015 12:08	Arquivo JAVA	1 KB
SystemEntityResource	18/06/2015 12:08	Arquivo JAVA	1 KB
UnitTypeEntityResource	18/06/2015 12:08	Arquivo JAVA	1 KB
UpdateEntityAction	18/06/2015 12:08	Arquivo JAVA	1 KB
User	18/06/2015 12:08	Arquivo JAVA	1 KB

Figura 6.18: Arquivos de código fonte do modelo mesclado entre OHRS e AuthSystem (baseado em Java e API Java para *Web Service*)

A título de exemplificação, a Listagem 6.9 descrimina o código fonte do arquivo AdministratorOperationResource.java. Este arquivo descreve a classe AdministratorOperationResource com seus atributos e métodos. Neste ponto, é possível observar que a ferramenta *weaving* fundiu os dois modelos de entrada.

Conforme descrito na Listagem 6.9, as referências do elemento *Resource* do modelo PDM de segurança, representadas pelos elementos *resource_id*, *resource_type* e *reference* foram mescladas com as operações do elemento *Administrator* do modelo PIM de negócios, representadas pelos elementos *view_availability()*, *book_unit()*, *cancel_unit()*, *reservation_status()*, *cancellation_status()*, *revenue_report()* e *modify_tariff()*. Logo, esta classe foi criada assumindo uma nova estrutura na qual operações específicas abstraídas do modelo de negócio (PIM) foram definidas como recursos para uma função específica (*Administrator*) do modelo de segurança (PDM).

O mesmo ocorreu com os demais elementos do modelo de negócio que foram abstraídos como usuários, funções e recursos, tendo suas respectivas ações controladas pelas respectivas permissões baseadas em funções de usuário. Por exemplo, o elemento *Administrator* do modelo PIM que antes descrevia as características e comportamentos específicos de um usuário de nível administrativo no sistema OHRS, após a fusão com o modelo PDM, assumiu nova funcionalidade no sistema, passando a representar uma função, através das propriedades *role_id* e *role_level*, a qual determina quais permissões são concedidas aos usuários atrelados a esta função. Tais características, desse e dos demais elementos, são refletidas no código de cada arquivo gerado após as transformações de modelo a texto.

Listagem 6.9: Código fonte do arquivo AdministratorOperationResource.java

```
1  package Service_Ohrs_AuthSystem;
2
3  import javax.jws.WebMethod;
4  import javax.jws.WebParam;
5  import javax.jws.WebService;
6
7  @WebService()
8  public class AdministratorOperationResource {
9        @EJB
10       public /*type*/ resource_id;
11       @EJB
12       public /*type*/ resource_type;
13       @EJB
14       public /*type*/ reference;
15
16       @WebMethod
17       public /*returnType*/ view_availability() {
18        }
19
20       @WebMethod
21       public /*returnType*/ book_unit() {
22        }
23
```

```
24          @WebMethod
25          public /*returnType*/ cancel_unit() {
26           }
27
28          @WebMethod
29          public /*returnType*/ reservation_status() {
30           }
31
32          @WebMethod
33          public /*returnType*/ cancellation_status() {
34           }
35
36          @WebMethod
37          public /*returnType*/ revenue_report() {
38           }
39
40          @WebMethod
41          public /*returnType*/ modify_tariff() {
42           }
43
44          @WebMethod
45          public /*returnType*/ arrival_chart() {
46           }
47  }
```

Uma vez que todas as etapas do *framework* proposto tenham sido realizadas, os arquivos fonte podem ser utilizados como ponto de partida para implementação e desenvolvimento de um *Web Service* com suas demais funcionalidades. O sistema a ser idealizado pode seguir a mesma lógica de negócio, baseado no sistema de reserva de hotel online ou ainda ser alterado para um sistema de outro contexto, onde os recursos de controle de acesso sejam aplicáveis.

6.6 Avaliação dos resultados

A obtenção de códigos fonte Java para *Web Services* a partir da fusão de dois sistemas funcionalmente distintos utilizou o protótipo do *framework* para desenvolvimento de SaaS adicionando aspectos de segurança através da MDE e *Weaving*, e foi motivada pela demonstração da viabilidade de utilização dos metamodelos e a metodologia proposta nesta pesquisa.

O processo de *weaving* dos modelos através do protótipo proposto demonstrou ser um recurso prático quando se trata de unir aplicações logicamente diferentes. Tal afirmativa é comprovada pelos seguintes fatos:

- Possibilidade de obter correspondências independentes de análises léxicas e semânticas, evitando que, baseado nesses termos, o especialista de domínio se preocupe com relacionamentos inválidos;
- Fornece flexibilidade para refinar e adaptar novos elementos durante as etapas iniciais da modelagem, se permitindo obter um melhor resultado final;

- Mantém um registro da origem dos elementos na fase de modelagem intermediária, o que auxilia no processo de reutilização em transformações futuras;
- Recurso de transformação para código fonte conforme a plataforma desejada eliminando a necessidade de escrever as partes essenciais da codificação.

O protótipo utilizou como exemplo ilustrativo a aplicação apresentada na abordagem de Ritu Sharma e Manu Sood [4] para desenvolvimento de aplicações de *software* em nuvem mais robustas, flexíveis e ágeis, seguindo as tecnologias em constante evolução. A Tabela 6.1 a seguir apresenta um comparativo entre a aplicação baseada na solução proposta deste trabalho e a aplicação apresentada pela abordagem dos autores Ritu Sharma e Manu Sood para desenvolvimento de aplicações de *software* em nuvem.

Tabela 6.1: Comparativo entre a abordagem apresentada em [4] e a abordagem proposta

	OHRS [4]	**OHRS_AuthSystem (abordagem proposta)**
Segurança	Limitada ao mecanismo de validação de usuários	Aprimorada pelas políticas de controle de acesso baseadas em funções de usuários
Flexibilidade	Concedida pelo processo de *design* de *software*	Concedida pelo processo de *design* de *software* com adição de aspectos de segurança desde o início do processo
Autenticação	Fornecida pelo mecanismo de validação de usuários	Fornecida pelo mecanismo de validação de usuário e seus respectivos níveis de função
Controle de acesso	Baseado no mecanismo de validação de usuários	Baseado nas funções e permissões de usuário
Confidencialidade	Implementada com base nas definições das regras de negócio	Garantia de controle de acesso a recursos, suas propriedades e operações

Ambas aplicações utilizam MDE e cumprem com o papel inicialmente designado, de fornecer suporte a reserva de hotéis. O diferencial está no fato da

metodologia utilizada neste trabalho se preocupar com as questões relacionadas à segurança da aplicação desde o início do processo de concepção da mesma, agregando a ela aspectos para aprimorar seu nível de segurança.

Os pontos avaliados entre os trabalhos relacionados (conforme Seção 3.3) são retomados para uma avaliação da abordagem proposta nesta dissertação. Conforme ilustra a Tabela 6.2, uma nova coluna referente ao trabalho desenvolvido foi inserida com a finalidade de comparar com os trabalhos relacionados. A Tabela 6.2 descreve os seguintes pontos avaliados:

1. Utilização da abordagem MDE para lidar com o desenvolvimento de *software*. A abordagem proposta contempla a utilização da MDE com a construção de modelos e transformações para o desenvolvimento de aplicações SaaS da plataforma de computação em nuvem;
2. Baseia-se no processo de desenvolvimento em Y. A abordagem utiliza o processo de desenvolvimento em Y, no qual são apresentados os modelos da esquerda e da direita para serem fundidos;
3. Utilização do mecanismo de *weaving* e/ou *merging* para estabelecer mapeamentos. São utilizadas as técnicas de *weaving* e *merging* para realizar a fusão dos elementos dos modelos de entrada, observando que tais modelos de entrada pertencem a contextos totalmente diferentes;
4. Permissão de uso de ferramentas para definição de correspondências. A solução proposta permite o uso de ferramentas que auxiliam no momento de estabelecer correspondências entre modelos;
5. Geração automática de códigos-fonte. Na abordagem proposta são gerados vários arquivos correspondentes aos códigos-fonte resultantes da fusão realizada.

Tabela 6.2: Comparação e análise dos trabalhos relacionados

Itens Avaliados	**Towards a new software development process for MDA [16]**	**UMLX: A graphical transformation language for MDA [47]**	**Merging Models with the Epsilon Merging Language (EML) [48]**	**Applying Generic Model Management to Data Mapping [15]**	**Automated Verification of Role-based Access Control Security Models Recovered from Dynamic Web Applications [50]**	**Cloud SaaS: Models and Transformation [4]**	**Um Framework de Segurança Baseado em Engenharia Dirigida por Modelos para Plataformas de Computação em Nuvem: Uma Abordagem para Modelos SaaS**
Utiliza a abordagem MDE no desenvolvimento de *software*	Sim	Sim	Sim	Sim	Sim	Sim	Sim
Baseado no processo de desenvolvimento em Y	Sim	Sim	Não	Não	Não	Não	Sim
Utiliza o mecanismo de weaving e/ou merging para mapeamento	Sim	Sim	Sim	Sim	Não	Não	Sim
Permite o uso de ferramentas de correspondência	Sim	Sim	Sim	Sim	Sim	Sim	Sim
Geração automática de códigos fonte	Não	Não	Não	Não	Sim	Não	Sim

6.7 Síntese

Neste capítulo, um exemplo ilustrativo foi apresentado no qual foram demonstradas as fases de modelagem descritas no *framework* proposto até as etapas de transformações de modelo-a-modelo e modelo-a-código. A aplicação do protótipo a um exemplo real afirmou a funcionalidade dos metamodelos e da metodologia desenvolvida.

O protótipo foi avaliado utilizando o exemplo ilustrativo do Sistema de Reserva de Hotel Online, que tem a funcionalidade de cadastrar e consultar reservas de hotel, bem como realizar o gerenciamento do sistema como um todo.

Os modelos gerados em cada etapa do *framework* e as definições de transformação dos modelos foram apresentados, realizando dessa forma a fusão e o refinamento das informações até se chegar ao código fonte a ser usado como ponto de partida de desenvolvimento de um sistema SaaS com segurança aplicada.

Com a utilização do *framework* proposto obtivemos um mecanismo funcional utilizando princípios da MDE, que contribuiu com a geração de modelos a partir do entrelaçamento e fusão de modelos de contextos distintos. Tal fusão proporcionou o desenvolvimento de uma aplicação SaaS adicionando conceitos de segurança focando mais especificamente em aspectos de autorização e autenticação.

7 Conclusões e Trabalhos Futuros

Neste capítulo, os objetivos alcançados neste trabalho de pesquisa e suas limitações são apresentados. Também, as contribuições científicas proporcionadas pelo trabalho, bem como os trabalhos futuros concernentes a esta área de pesquisa são apresentados.

7.1 Conclusões do Trabalho

Este trabalho apresentou a proposta de um *framework* para suportar o desenvolvimento de aplicações SaaS baseado na Engenharia Dirigida por Modelos trabalhando com a separação de preocupações e abordando aspectos relacionados à segurança, com o diferencial da utilização de mecanismos de *weaving* para mesclar dois ou mais modelos com lógicas de negócio diferentes. Para tanto, foi apresentada uma metodologia para auxiliar no conhecimento de cada etapa relacionada ao *framework* proposto. Através dos recursos fornecidos pela EMF foram desenvolvidos metamodelos com o objetivo de trazer para o ambiente da MDE os sistemas utilizados, as plataformas alvo, e as ferramentas intermediárias responsáveis por estabelecer os relacionamentos identificados.

Foi desenvolvido um metamodelo de UML para instanciar o modelo correspondente ao SaaS de entrada, e um metamodelo PDM para instanciar o modelo correspondente a segurança. Além destes, também foram desenvolvidos um metamodelo *weaving* para instanciar o modelo contendo as entidades correspondentes dos modelos de entrada, e um metamodelo intermediário para instanciar o modelo que armazena todas as informações da fusão necessárias para ser transformado em PSMs. Os modelos específicos de plataforma foram criados a partir de metamodelos específicos para plataformas de computação em nuvem. Além dos metamodelos desenvolvidos foram criadas definições de transformações modelo-a-modelo e modelo-a-código utilizando ATL para se alcançar o estágio final do *framework*.

Foi construído um protótipo com a finalidade de avaliar o *framework* proposto, no qual foram utilizados os metamodelos e as definições de transformações desenvolvidas. O protótipo utilizou como exemplo ilustrativo a mesma aplicação SaaS de entrada, na qual foram aplicados conceitos e aspectos de segurança relativos a um sistema de controle de acesso baseado em funções.

7.2 Objetivos Alcançados

De acordo com o objetivo principal do trabalho que é fornecer um modelo de desenvolvimento para sistemas em ambientes SaaS com a aplicação de aspectos de segurança utilizando as técnicas de modelagem, os seguintes objetivos foram alcançados:

- Proposta de um *framework* para suportar a fusão de sistemas de domínios diferentes, sendo que um desses sistemas é relacionado a aspectos específicos de segurança;
- Demonstração da aplicação das técnicas de *weaving* e *merging* no processo de correspondência entre modelos através da construção de um protótipo para avaliar o *framework*;
- Considerando que as questões de segurança são geralmente gerenciadas como um recurso isolado do processo de *design* de *software*, o trabalho proposto abordou este aspecto desde o início do processo com a utilização da modelagem e das definições de transformação;
- Agregação de valores de confidencialidade e integridade de dados a aplicações SaaS através da aplicação da abordagem baseada na Engenharia Dirigida por Modelos para mesclar modelo de negócio e modelo de segurança definindo políticas de controle de acesso a recursos, propriedades e operações de um modelo Web Services Java.

7.3 Limitações

A despeito de haver se alcançado o seu objetivo principal, neste trabalho também foram identificadas algumas limitações conforme são apresentadas a seguir:

- A abordagem não abrange aspectos relacionados a criptografia e serialização de dados. Tais aspectos podem ser incluídos futuramente introduzindo novos processos de fusão e definições de transformação de modelos;
- O processo de fusão entre o modelo de negócios e o modelo de segurança não pode ocorrer completamente automatizado. É sempre necessário a intervenção do usuário para identificar os aspectos que se relacionam entre modelos e analisar se as correspondências entre os elementos dos modelos são válidas;
- Os modelos instanciados manualmente através do EMF utilizam por padrão o editor em forma de árvore. A utilização de um ambiente como o GMF (*Graphical Modeling Framework*) [60] por exemplo, poderia tornar a experiência mais intuitiva por utilizar elementos gráficos em seu formato de edição;

- O *framework* realiza as transformações de modelos usando a linguagem de transformação ATL; poderiam ser utilizadas outras linguagens como MOFScript, MOF QVT, etc.;
- A aplicação do *framework* foi analisada utilizando apenas modelos simples, sendo necessário um estudo de caso mais aprofundado que permitisse a integração de modelos de segurança mais complexos.

7.4 Contribuições

Esta seção apresenta as contribuições do trabalho de pesquisa no campo científico e tecnológico como descrito a seguir.

7.4.1 Científicas

A pesquisa obteve as seguintes contribuições científicas:

- Para a computação em nuvem, a melhoria da qualidade do sistema de *software*, melhoria da segurança, robustez e flexibilidade com a introdução de políticas de controle de acesso baseado em funções de usuário para aplicações do modelo de serviço SaaS;
- Adaptação do processo de *design* de *software* permitindo a adição de aspectos de segurança desde o início através da utilização das técnicas de *weaving*, onde é possível realizar a fusão de características de modelos diferentes;
- Modelagem e propagação dos aspectos de segurança em vários níveis de modelagem, compreendendo várias fases de transformação, nas quais aspectos, tais como controle de acesso, são propagados para as próximas fases de transformação em cada nível de evolução;
- Reutilização dos metamodelos adotados, permitindo também a reutilização das definições de transformação para manipular novos modelos de negócios, conforme especificado no *framework* proposto onde é possível instanciar além de novos modelos conforme UML, também novos modelos de segurança, *weaving* e intermediários.

7.4.2 Tecnológicas

Pode se citar dentre as contribuições tecnológicas:

- Desenvolvimento de um *framework* proposto como *plug-in* para Eclipse, incluindo EMF e ATL, para construção das definições de transformação

entre os modelos de entrada (PIM e PDM) e modelos de saída (*Intermediate* e PSMs) manipulados na proposta do *framework*;
- Criação dos metamodelos *Weaving* e *Intermediate*, além dos demais metamodelos referentes ao PIM, PDM e PSMs, e geração dos seus respectivos modelos.

7.5 Trabalhos Futuros

Nesta seção, os trabalhos futuros que podem ser sugeridos a partir da pesquisa atual são os seguintes:

- Aperfeiçoar o protótipo desenvolvido para Eclipse com a inclusão do ambiente de edição de modelos no formato gráfico através do GMF com o objetivo de tornar o processo mais intuitivo;
- Melhorar o mecanismo de correspondência com algoritmos para inclusão de automação na escolha dos elementos a serem entrelaçados, tendo em vista que a tomada de decisão de correspondência entre sistemas logicamente diferentes se demonstra um paradigma por não se tratar de uma tarefa trivial;
- Utilização dos códigos gerados pelo *framework* para desenvolvimento de uma aplicação SaaS real a fim de validar a proposta e os resultados alcançados em comparação com sistemas SaaS desenvolvidos utilizando a abordagem MDE convencional.

Referências Bibliográficas

[1] R. Sharma and M. Sood. Cloud SaaS and Model Driven Architecture. *Proc. of the International Conference on Advanced Computing and Communication Technologies* (ACCT 2011).

[2] X. Li, J. Chen, and M. Luo. A Simple Security Model based on Cloud Reference Model. *10th International Symposium on Distributed Computing and Applications to Business, Engineering and Science* – IEEE, 2011.

[3] C. Baun, M. Kunze, J. Nimis, and S. Tai. *Cloud Computing: Web-Based Dynamic IT Services*. Springer, 2011.

[4] R. Sharma and M. Sood. Cloud SaaS: Models and Transformation. *Proc. of International Conference on Computer Science and Information Technology* (CCSEIT 2011).

[5] J. Brodkin. Segurança em SaaS: cinco questões a considerar. Disponível em: http://computerworld.com.br/tecnologia/2010/12/13/seguranca-em-saas-cinco-questoes-a-considerar. Acesso em: 12/01/2015.

[6] D. C. Schmidt. Model-Driven Engineering. *IEEE Computer*, February 2006.

[7] M. Young. Security Concerns in the SaaS Environment. Disponível em: http://www.itworld.com/article/2782367/software-as-a-service/security-concerns-in-the-saas-environment.html. Acesso em: 12/01/2015.

[8] D. Lopes, S. Hammoudi, J. Bézivin, and F. Jouault. Mapping Specification in MDA: From Theory to Practice. *First International Conference INTEROP-ESA'2005 Interoperability of Enterprise Software and Applications* (February 2005).

[9] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks. EMF: Eclipse Modeling Framework, 2nd ed. Addison-Wesley Professional, 2008.

[10] M. Carvalho. Uma Abordagem Baseada na Engenharia Dirigida por Modelos para Suportar Merging de Base de Dados Heterogêneas. Master's thesis, Universidade Federal do Maranhão, 2014.

[11] A. Group, LINA, and INRIA. Atlas Transformation Language – ATL User Manual version 0.7, February 2006.

[12] A. Kleppe, J. Warmer, and W. Bast. MDA Explained: The Model Driven Architecture: Practice and Promise, 1st ed. Addison-Wesley, August 2003.

[13] R. A. Pottinger and P. A. Bernstein. Merging Models Based on Given Correspondences. Technical Report UW-CSE-03-02-03, University of Washington, 2003.

[14] J. Bézivin, S. Hammoudi, D. Lopes, and F. Jouault. Applying MDA Approach for Web Service Platform. *Proc. of the 8th Enterprise Distributed Object Computing Conference* (EDOC 2004) – IEEE, 2004.

[15] M. D. D. Fabro, J. Bézivin, F. Jouault, and P. Valduriez. Applying Generic Model Management to Data Mapping. *Proc. of Base de Données Avancées* (BDA 2005), October 17-20, 2005, Saint-Malo, France.

[16] A. Belangour, J. Bézivin, and M. Fredj. Towards a new software development process for MDA. *Proc. of the European Workshop on Milestones, Models and Mappings for Model-Driven Architecture* (3M4MDA). Bilbao, Spain, July 2006.

[17] A. Jossic, M. D. D. Fabro, J. Lerat, J. Bézivin, and F. Jouault. Model Integration with Model Weaving: a Case Study in System Architecture. *Proc. of the 2007 International Conference on Systems Engineering and Modeling* – IEEE, 2007.

[18] B. Sosinsky. *Cloud Computing Bible*. Wiley Publish, 2011.

[19] A. T. Velte, T. J. Velte, and R. Eslenpeter. *Computação em Nuvem: Uma abordagem prática*. Alta Books, 2010.

[20] Auditing Cloud Computing – A Security and Privacy Guide. Wiley Corporate, 2011.

[21] B. Halpert. *Auditing Cloud Computing – A Security and Privacy Guide*. Wiley Corporate, 2011.

[22] Eclipse Modeling Framework (EMF). Eclipse Modeling Project. Disponível em: http://www.eclipse.org/modeling/emf/. Acesso em: 14/01/2015.

[23] Eclipse, 2015. Eclipse IDE. Disponível em: http://www.eclipse.org/. Acesso em: 14/01/2015.

[24] J. M. Favre. Foundations of Model (Driven) (Reverse) Engineering: Models – Episode I, Stories of the Fidus Papyrus and of the Solarus. *Proc. of Dagstuhl Seminar 04101 on Language Engineering for Model-Driven Software Development*, Dagstuhl, Germany.

[25] J. Bézivin. Model driven engineering: An emerging technical space. *In Generative and Transformational Techniques in Software Engineering*, R. Lammel, J. Saraiva, and J. Visser, Eds., vol. 4143 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2006, pp. 36-64.

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

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

[28] OMG. OMG Meta Object Facility (MOF) Core Specification Version 2.4.1, Document Number: formal/2013-06-01. OMG, June 2013.

[29] OMG. OMG Unified Modeling Language™ (OMG UML), Infrastructure. Version 2.3, Document Number: formal/2010-05-03. OMG, May 2010.

[30] ORACLE. Plataforma Java. Disponível em: http://www.oracle.com/technetwork/java/index.html. Acesso em: 14/01/2015.

[31] W3C. XML Technology. Disponível em: http://www.w3c.org/standards/xml. Acesso em: 14/01/2015.

[32] OMG. Object Management Group. Disponível em: http://www.omg.org/. Acesso em: 16/01/2015.

[33] J. B. Oliveira. Uma abordagem baseada em Engenharia Dirigida por Modelos para Suportar o Teste de Sistemas de Software na Plataforma de Computação em Nuvem. Master's thesis, Universidade Federal do Maranhão, 2012.

[34] P. A. Bernstein. Applying Model Management to Classical Meta Data Problems. *Proc. of Conference on Innovative Data Systems Research* (CIDR 2003), pp 209-220.

[35] M. D. D. Fabro, J. Bézivin, F. Jouault, E. Breton, and G. Gueltas. AMW: A Generic Model Weaver. *Proc. of the 1ère Journée sur l'Ingénierie Dirigée par les Modèles* (IDM05). Paris: [s.n.], 2005.

[36] D. D. Ruscio. Specification of Model Transformation and Weaving in Model Driven Engineering. Ph.D. Thesis, Università di L'Aquila, Italy, 2007.

[37] M. D. D. Fabro, J. Bézivin, and P. Valduriez. Weaving Models with the Eclipse AMW plugin. *In Eclipse Modeling Symposium, Eclipse Summit Europe* (2006).

[38] R. Maggiani. Cloud computing is changing how we communicate. *In: IEEE International Professional Communication Conference*, pp.1-4 (2009).

[39] R. Sharma and M. Sood. Cloud SaaS and Model Driven Architecture. *Proc. of the International Conference on Advanced Computing and Communication Technologies* (ACCT 2011).

[40] X. Li, J. Chen, and M. Luo. A Simple Security Model based on Cloud Reference Model. *10th International Symposium on Distributed Computing and Applications to Business, Engineering and Science* – IEEE, 2011.

[41] C. Baun, M. Kunze, J. Nimis, and S. Tai. *Cloud Computing: Web-Based Dynamic IT Services*. Springer, 2011.

[42] SaaS - The Complete Cornerstone Guide to Software as a Service Best Practices Concepts, Terms, and Techniques for Successfully Planning, Implementing and Managing SaaS Solutions. Common SaaS problems that occur after implementation. Chapter Excerpt. Disponível em: http://searchitchannel.techtarget.com/feature/Common-SaaS-problems-that-occur-after-implementation. Acesso em: 08/11/2013.

[43] D. Lopes. Étude et applications de l'approche MDA pour des plates-formes de Services Web. Ph.D. Thesis, Université de Nantes, 2005.

[44] J. S. Jr, D. Lopes, D. B. Claro, and Z. Abdelouahab. A Step Forward in Semi-automatic Metamodel Matching: Algorithms and Tool. *11th International Conference on Enterprise Information Systems LNBIP* (2009).

[45] J. Oldevik. MOFScript User Guide. Version 0.9 (MOFScript v 1.4.0). February 2011.

[46] W3C. Web Services Description Language (WSDL). Version 2.0 – Part 0: Primer. W3C Recommendation, 26 June 2007. Disponível em: http://www.w3.org/TR/2007/REC-wsdl20-primer-20070626. Acesso em: 12/03/2015.

[47] E. D. Willink. UMLX: A graphical transformation language for MDA. England, May 2003.

[48] D. S. Kolovos, R. F. Paige, and F. A. C. Polack. Merging Models with the Epsilon Merging Language (EML). *Proc. of the 9th International Conference on Model Driven Engineering Languages and Systems*. Genova, Italy, October, 2006, pp. 215-229.

[49] C. Batini, M. Lenzerini, and S. B. Navathe. A Comparative Analysis of Methodologies for Database Schema Integration. ACM Computing Surveys, 18(4):323-364, December 1986.

[50] M. H. Alalfi, J. R. Cordy, and T. R. Dean. Automated Verification of Role-based Access Control Security Models Recovered from Dynamic Web Applications. *14th IEEE International Symposium on Web Systems Evolution* (WSE), Trento, p. 1-10, September 2012.

[51] N. A. Delessy and E. B. Fernandez. A Pattern-Driven Security Process for SOA Applications. 2008, IEEE.

[52] J. Bézivin, S. Hammoudi, D. Lopes, and F. Jouault. B2B Applications, BPEL4WS, Web Services and .NET in the Context of MDA, *Knowledge*

Sharing in the Integrated Enterprise, Springer, vol. 183, pages 225-236, 2005.

[53] D. Lopes, S. Hammoudi, J. Bézivin, and F. Jouault. Generating Transformation Definition from Mapping Specification: Application to Web Service Platform. *Lecture Notes in Computer Science*, v. 3520, p. 183-192, 2005.

[54] D. Basin, J. Doser, and T. Lodderstedt. Model Driven Security: from UML Models to Access Control Infrastructures. *ACM Transactions on Software Engineering and Methodology*, Vol. 15, No. 1, New York, p. 39-91, January 2006.

[55] D. Lopes, S. Hammoudi, and Z. Abdelouahab. Schema Matching in the Context of Model Driven Engineering: From Theory to Practice. *Proc. of the International Conference on Systems, Computing Sciences and Software Engineering* (SCSS 2005) (December 2005).

[56] Eclipse, 2015. IDE Eclipse Luna. Disponível em: http://www.eclipse.org/luna/. Acesso em: 15/01/2015.

[57] Eclipse, 2015. ATL – a model transformation technology. Disponível em: http://www.eclipse.org/atl/. Acesso em: 15/01/2015.

[58] Eclipse, 2015. Eclipse Projects – Papyrus. Disponível em: http://www.eclipse.org/papyrus/. Acesso em: 15/01/2015.

[59] Oracle, 2015. NetBeans IDE. Disponível em: http://www.netbeans.org/. Acesso em: 16/01/2015.

[60] Graphical Modeling Framework (GMF). Eclipse Modeling Project. Disponível em: http://www.eclipse.org/gmf-tooling/. Acesso em: 16/01/2015.