UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

UM FRAMEWORK BASEADO EM PADRÕES DE SEGURANÇA PARA TRANSFORMAÇÕES DE MODELOS

DISSERTAÇÃO DE MESTRADO

FÁBIO SARTURI PRASS

Santa Maria, RS, Brasil
2012

UM FRAMEWORK BASEADO EM PADRÕES DE SEGURANÇA PARA TRANSFORMAÇÕES DE MODELOS

por

FÁBIO SARTURI PRASS

Dissertação apresentada ao Curso de Mestrado em Computação do Programa de Pós-Graduação em Informática (PPGI), Área de Concentração em Computação, da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de
Mestre em Ciência da Computação

Orientadora: Prof.ª. Dr.ª Lisandra Manzoni Fontoura
Co-Orientador: Prof. Dr. Osmar Marchi dos Santos

Santa Maria, RS, Brasil.
2012

Universidade Federal de Santa Maria

Centro de Tecnologia

Programa de Pós-Graduação em Informática

A Comissão Examinadora, abaixo assinada,

aprova a Dissertação de Mestrado

UM FRAMEWORK BASEADO EM PADRÕES DE SEGURANÇA PARA TRANSFORMAÇÕES DE MODELOS

elaborada por

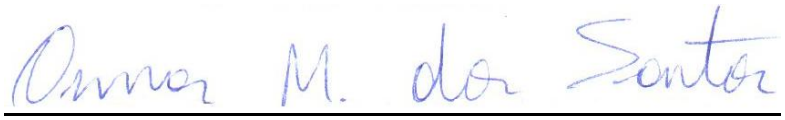
Fábio Sarturi Prass

como requisito parcial para obtenção do grau de

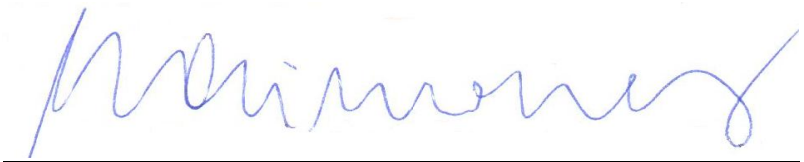
Mestre em Ciência da Computação

COMISSÃO EXAMINADORA:

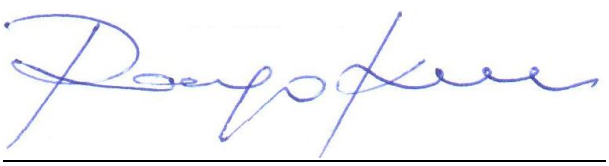
Lisandra Manzoni Fontoura, Dr.ª
(Presidente/Orientadora)



Osmar Marchi dos Santos, Dr. (UFSM)
(Co-Orientador)



Eugênio de Oliveira Simonetto, Dr. (UFSM)



Rodrigo dos Santos Keller, Dr. (ULBRA)

Santa Maria, 18 de abril de 2012.

AGRADECIMENTOS

Primeiramente, agradeço à minha esposa Emília, que sempre esteve ao meu lado, me ajudando, me compreendendo e principalmente me motivando nos momentos de desânimo e lamentação. Seu apoio foi fundamental para a realização deste trabalho.

À minha família por todo apoio que me foi dado não somente nesta jornada de estudos, mas também em toda minha vida. Ao meu pai e a minha mãe pela atenção, a dedicação, o carinho e a busca de me proporcionarem sempre a melhor educação possível. Ao meu irmão e irmã pela amizade, companheirismo e conselhos.

Aos meus orientadores Prof.ª Dr.ª Lisandra Manzoni Fontoura e Prof. Dr. Osmar Marchi dos Santos, por terem me orientado, além da paciência, confiança e compreensão dos momentos difíceis. Muito obrigado por compartilharem parte de seu conhecimento e suas ideias, mostrando-me o caminho da pesquisa.

Aos professores e colegas de mestrado, pela troca de experiências.

Por fim, gostaria de agradecer a todos que contribuíram de alguma forma para a realização deste trabalho ou para a minha formação.

RESUMO

Dissertação de Mestrado
Programa de Pós-Graduação em Informática
Universidade Federal de Santa Maria

UM FRAMEWORK BASEADO EM PADRÕES DE SEGURANÇA PARA TRANSFORMAÇÕES DE MODELOS

AUTOR: FÁBIO SARTURI PRASS
ORIENTADORA: PROF.ª DR.ª LISANDRA MANZONI FONTOURA
CO-ORIENTADOR: PROF. DR. OSMAR MARCHI DOS SANTOS

Santa Maria, 18 de abril de 2012.

O aumento da automatização nos processos de desenvolvimento de sistemas vem ganhando espaço no contexto atual devido ao aumento da complexidade. Com isto, cada vez mais surgem ferramentas baseadas na ideia de geração automática de código a partir de modelos. Por causa desta complexidade, os sistemas de informação estão sujeitos a erros e vulnerabilidades. Com isso, surgem as necessidades do aumento da automatização e do nível de abstração no desenvolvimento de software, permitindo assim maior segurança na aplicação. Essas necessidades são respondidas pela utilização da abordagem da Engenharia Dirigida por Modelos que permite a modelagem e aplicação de transformações sobre os modelos, visando à obtenção do software de forma automatizada. Este trabalho propõe um *framework* baseado em padrões de segurança orientado a modelo, fornecendo diretrizes para implementação do modelo e a validação correta do uso dos padrões. A segurança é inserida implicitamente no sistema por meio de transformação entre modelos e codificação automática, garantindo que a segurança não será violada em nenhum nível e não será suscetível a erros ou alterações do código. Estas transformações são definidas através de uma sintaxe XMI e um conjunto de regras implementadas em linguagem Java e ATL, e podem ser posteriormente executadas de maneira unidirecional, através da aplicação de transformações implementada para apoiar o uso da abordagem proposta. É apresentado também, um exemplo de transformação de modelos para a plataforma Java.

Palavras-chaves: Segurança, padrão de segurança, transformação, modelos, MDE.

ABSTRACT

Master's Dissertation
Post-Graduate Program in Informatics
Federal University of Santa Maria

A FRAMEWORK BASED ON SECURITY PATTERNS FOR TRANSFORMATIONS OF MODELS

AUTHOR: FÁBIO SARTURI PRASS
ADVISOR: PROF.ª DR.ª LISANDRA MANZONI FONTOURA
CO-ADVISOR: PROF. DR. OSMAR MARCHI DOS SANTOS

Santa Maria, abril 18, 2012.

The increased automation in the process of systems development is becoming more popular in the current context because of the increased complexity. With this increasingly arise tools based on the idea of automatic code generation from models. As result of this complexity, information systems may have errors and vulnerabilities. So, there is a need for increasing the level of abstraction and automation in the software development processes, thereby allowing greater security in the application. These needs are answered by using the approaches of Model-Driven Engineering that allows the modeling and the application of transformations onto models aiming at obtaining software in an automatized way. This study proposes a framework based on security patterns oriented to model is proposed in this paper. This provides guidelines for implementing the model application abd the correct validation of using patterns. The security is implicitly included into the system through a transformation between models, which automatically encodes the security pattern, ensuring that the generated code is not susceptible to errors or changes in code. These transformations are defined through XML syntax and a set of rule implemented in Java and ATL language. They can be further executed in a unidirectional way, through the transformation application implemented to support the use of the proposed approach. It is also presented an example of model transformation for the Java platform.

Keywords: Security, Security Pattern, transformation, models, MDE.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CIM	Computation Independent Model
DSL	Domain-Specific Language
EJB	Enterprise Java Beans
JEE	Java Enterprise Edition
MDA	Model Driven Architecture
MDD	Model Driven Development
MDE	Model Driven Engineering
MOF	Meta-Object Facility
OMG	Object Management Group
PIM	Platform Independent Model
PSM	Platform Specific Model
QVT	Query-View-Transformation
UML	Unified Modeling Language
XMI	XML Metadata Interchange

SUMÁRIO

1 INTRODUÇÃO

Este capítulo contextualiza o tema proposto nesse trabalho. Descreve a discussão do problema existente na área onde se desenvolve a pesquisa e os objetivos vislumbrados a partir do problema, o método escolhido para alcançá-los, as justificativas para o estudo, os resultados esperados, as limitações e a estrutura do trabalho.

1.1 Contextualização

Com a intenção de obter maior agilidade no desenvolvimento de sistemas e alcançar os benefícios inerentes, têm surgido ferramentas de geração de código, que se utilizam de *templates*, assistentes, desenvolvimento declarativo, entre outros recursos. O uso destas ferramentas no desenvolvimento de aplicações exige do desenvolvedor um valor extremamente alto, que é a dependência das soluções geradas, tornando mais complexo realizar alterações nos códigos-fonte automatizados por diversas razões, entre as quais, os algoritmos gerados podem ter alto nível de complexidade (BROWN, 2004).

Diversos *frameworks* e outras ferramentas que se utilizam de complementação de código têm surgido com o mesmo intuito, o de acelerar os processos de codificação dos aplicativos (ROSADO *et al.*, 2006). Embora a complementação de código seja extremamente satisfatória, a essência e a responsabilidade da codificação correta continuam sendo atribuídas ao desenvolvedor. Mesmo que estas ferramentas façam constantes verificações sintáticas, elas não podem garantir, em tempo de desenvolvimento, a utilização correta dos objetos e nem mesmo da completude e coerência operacional dos algoritmos (FINK, KOCH e PAULS, 2006).

A *Object Management Group* (OMG), em busca de uma alternativa para a automatização do desenvolvimento, criou a abordagem *Model Driven Architeture* (MDA), como alternativa para o aproveitamento da documentação e geração de

código de forma automatizada, com o intuito de oferecer um passo evolutivo no processo de desenvolvimento de software. O MDA utiliza as noções de modelos gerados na etapa de modelagem para o desenvolvimento de software. Esta arquitetura busca ainda permitir a especificação de um sistema de forma independente da plataforma e prover meios de transformação desta especificação para plataformas específicas (OMG, 2003b).

Como auxílio à elaboração de sistemas seguros, tem-se os padrões de segurança, que descrevem boas soluções aos problemas de segurança e podem ser reutilizáveis em diferentes contextos. Contudo, para obter os benefícios da utilização destes padrões é necessário saber quando, onde e como utilizá-los, a partir de suas definições, pois, falhas de segurança podem ser introduzidas pela incorreta implementação dos padrões de segurança (SCHUMACHER, 2003).

Padrões de segurança, em combinação com MDE, permitem que a segurança esteja integrada na modelagem de sistemas à medida que os modelos de projeto são integrados aos modelos de segurança e, desta forma, são utilizados para automatizar a construção de sistemas a partir desses modelos, auxiliando a criação de estruturas flexíveis e garantindo que os requisitos de segurança sejam considerados durante todas as fases do desenvolvimento do sistema. Com isso, as falhas de segurança podem ser evitadas desde a modelagem, sendo a implementação mantida consistente com a política de segurança modelada além de permitir a migração para novas plataformas (CUNHA, 2007).

Aplicações como softwares, comércio eletrônico e outras similares no âmbito da tecnologia da informação, possuem a segurança como requisito fundamental e apesar da grande variedade de estruturas disponíveis, constantemente há notícias sobre vulnerabilidades e falhas de segurança em sistemas de software (FINK, KOCH e PAULS, 2006). À medida que as estruturas de segurança aparecem, elas se tornam rapidamente superadas por diferentes motivos. Na maioria das vezes não oferecem um modelo arquitetural suficientemente flexível para acompanhar as necessidades do negócio, o qual está em desenvolvimento ou, ainda, porque limitam o escopo da arquitetura que o negócio necessita. Devido à diversidade de requisitos de negócio de cada aplicação, arquitetos de software e analistas necessitam cada vez mais criar suas próprias estruturas e manter suas integridades e coerência em relação às necessidades do negócio as quais atendem (CUNHA, 2007).

O conhecimento desses desenvolvedores pode ser incorporado em padrões, que descrevem soluções comprovadas para problemas recorrentes. Padrões, assim, facilitam a geração e documentação de soluções conhecidas e estabelecidas. Estas soluções encapsuladas são muito valiosas na elaboração de novos sistemas, na avaliação de sistemas existentes, e podem, ainda, melhorar o processo de comunicação e aprendizagem (WEISS e MOURATIDIS, 2008).

Diferentes tipos de padrões têm sido considerados na literatura: os padrões são desenvolvidos em diferentes níveis de abstração, variando de paradigmas fundamentais para a estruturação de sistemas de software ou padrões de gestão para implementações concretas de decisões de projeto particular (WEISS, MOURATIDIS, 2008). Desta forma, uma abordagem orientada a modelo pode auxiliar o desenvolvimento de sistemas seguros, promovendo uma implementação facilitada dos padrões de segurança e, desta forma, garantindo aderência às diretrizes arquiteturais da organização e às questões de segurança.

1.2 Definição do problema

Um dos principais problemas encontrados no desenvolvimento de aplicações na atualidade é tornar a implementação de sistemas de software produtivo, e, ao mesmo tempo, manter uma documentação consistente com o que está codificado. É comum encontrar aplicações já desenvolvidas que não possuem qualquer tipo de documentação. Com isso, futuras alterações na mesma tornam-se mais onerosas, demoradas e complexas, pois sem um documento apropriado sobre como está estruturado o sistema, os programadores muitas vezes necessitam desvendar o que determinados trechos de códigos-fonte fazem ou então se gasta um longo tempo analisando o código até que se possa entender o funcionamento do mesmo.

Outro problema que pode ser destacado é a perda da rastreabilidade do sistema, onde apesar do empenho utilizado na geração e validação das informações acerca dos relacionamentos e interdependências entre o mundo real, os requisitos e o sistema de software, estas informações de rastreamento normalmente não são confiáveis em razão da inconsistência gerada pela evolução distinta do sistema e dos seus requisitos. A principal ocorrência desta inconsistência é a perda efetiva do

seu valor para o desenvolvedor, que passa a não utilizar as informações armazenadas. Neste contexto é complexo para o desenvolvedor ter condições de saber qual informação é correta e qual se encontra desatualizada.

Além da desatualização das informações de rastreamento, uma série de outros fatores dificulta a obtenção da rastreabilidade: os padrões de desenvolvimento existentes não fornecem uma boa orientação de como obter a rastreabilidade; a não adoção de uma forma sistemática de rastreabilidade pelas organizações de desenvolvimento; os mecanismos e modelos de rastreamento existentes modificam-se e muitas vezes não são compreendidos; a dificuldade de realizar o mapeamento entre os domínios do problema e da solução; e o alto custo para obter, atualizar e validar as informações de rastreamento.

Outra questão a ser considerada diz respeito às vulnerabilidades encontradas nos aplicativos acerca da segurança empregada. Uma solução dita segura é aquela que identifica todas as suas possíveis vulnerabilidades, implementando mecanismos que tenham o objetivo de reduzir qualquer tipo de dano, caso ela seja vítima de um ataque malicioso. As vulnerabilidades estão espalhadas em todas as camadas da solução, seja em rede, servidores e aplicação. Qualquer fraqueza em qualquer ponto da solução pode ser explorada por um usuário malicioso.

1.3 Objetivos

1.3.1 Objetivo Geral

Elaborar de *framework* para desenvolvimento de software seguro usando padrões de segurança em uma abordagem baseada em MDE aplicado a transformações de modelos. O *framework* é formado pelo modelo baseado em padrões de segurança e pelas regras de transformação entre modelos.

1.3.2 Objetivos Específicos

- Elaborar uma estrutura de segurança orientada a modelo onde a segurança é introduzida no sistema através de transformação entre modelos e codificação automática.
- Gerar um *framework* de padrões de segurança, baseados em Shumacher et al. (2006), para ser inserido em regras de transformações de modelos, uma vez que, os padrões são menos compreendidos em comparação com os registrados no nível de execução e a prevenção inicial de vulnerabilidades de segurança é ponderada para ser menos onerosa em relação às ações corretivas tomadas tardiamente.

1.4 Questão problema

Padrões de segurança podem agregados ao MDE visando prover uma maior segurança em uma aplicação

1.5 Justificativa

A proposta deste trabalho é a pesquisa e criação de um *framework* que permita tornar mais ágil e eficaz o desenvolvimento de uma aplicação, fazendo o uso de regras de transformações baseadas em padrões de segurança. Espera-se que a aplicação do *framework* apresente ao desenvolvedor um ganho significativo em termos de velocidade de desenvolvimento. A espera de ganho é baseada na seleção de novos conceitos que anunciam tornar o desenvolvimento de sistemas mais ágil e eficaz.

Para auxiliar na melhoria deste contexto e, visando a elaboração de sistemas seguros, têm-se os padrões de segurança, que são soluções reutilizáveis aos problemas de segurança. Contudo, para obter os benefícios da utilização destes

padrões é necessário saber quando, onde e como utilizá-los, a partir de suas definições, pois, pode-se acabar introduzindo notórias falhas de segurança (FERNANDEZ e PAM, 2001). É necessário que haja na equipe de desenvolvimento um alto grau de conhecimento de padrões de segurança e de estrutura de segurança. Todavia, esse conhecimento faz com o que o custo do desenvolvimento do software aumente consideravelmente (YOSHIOKA, HONIDEN e FINKELSTEIN, 2004).

Padrões de segurança, em combinação com MDE, permitem que a segurança esteja integrada na modelagem de sistemas à medida que os modelos de projeto sejam combinados com os modelos de segurança e, técnicas de geração automática de código sejam utilizadas para automatizar a construção de sistemas a partir desses modelos, auxiliando a criação de estruturas flexíveis e garantindo que os requisitos de segurança sejam levados em consideração durante todas as fases do processo de desenvolvimento do sistema. Com isso, as falhas de segurança podem ser identificadas de forma mais rápida no processo de desenvolvimento e a implementação é mantida consistente com a política modelada, além de permitir a migração para novas plataformas (KIENZLE *et al.*, 2011).

Neste contexto, uma abordagem orientada a modelo pode auxiliar o processo de desenvolvimento de sistemas seguros, promovendo uma implementação controlada dos padrões de segurança e, desta forma, garantindo aderência às diretrizes arquiteturais da organização e às questões de segurança. Esta pesquisa se concentra, principalmente, nas linguagens de modelagem e métodos para gerar um *framework* de padrões de segurança para ser inserido em regras de transformações de modelos.

1.6 Resultados esperados

Os resultados esperados ao término deste trabalho são averiguar a viabilidade da aplicação da arquitetura MDE para explorar a transformação de modelos aplicados à segurança.

Espera-se que esses resultados contribuam para que o desenvolvimento de sistema seja focado em padrões de segurança, evitando assim falhas nas fases do ciclo de vida do software.

Almeja-se, ainda, a elaboração de um conjunto de padrões de segurança aplicáveis a regras de transformações para serem inserido em modelos UML.

1.7 Limitações do trabalho

Este trabalho se limita a um diagrama de classes para a implementação de uma aplicação integrada ao *framework* de padrões de segurança utilizando MDE para adaptação às regras de transformações entre modelos. Não faz parte do escopo deste trabalho detalhar características de plataforma, sistema operacional, *hardware*, outros *frameworks*, *containers* ou qualquer outro conceito ou tecnologia não relacionados ao MDE.

As transformações de modelos ocorrem somente no modelo PIM e só podem ser definidas de forma unidirecional a partir de uma única especificação.

1.8 Estrutura do trabalho

Os demais capítulos dessa dissertação estão estruturados a seguir.

No Capítulo 2 é apresentado uma explicação mais detalhada dos procedimentos científicos aplicados na verificação da hipótese de pesquisa e do método científico de pesquisa escolhidos para alcançar o objetivo do trabalho.

No Capítulo 3 é descrito uma síntese do que é a MDE, incluindo motivos, conceitos, definições e seu estado atual. O capítulo enfoca os conceitos mais relacionados a este trabalho, tais como modelos, transformações e mapeamentos entre modelos. Detalham-se as transformações e os mapeamentos, apresentando-se: como os mapeamentos são usados nas transformações, as abordagens de

transformação propostas, os tipos de mapeamentos e exemplos dos mapeamentos, ilustrando como são realizados.

No Capítulo 4 são apresentados conceitos de segurança além da definição de padrões de segurança e a função destes no desenvolvimento de sistemas, visando possibilitar o entendimento da proposta.

No Capítulo 5 é discutida a elaboração do modelo a partir de um conjunto de padrões de segurança e a implementação de uma ferramenta para integração do modelo ao modelo do cliente, gerando assim, transformações genéricas para unificação dos modelos baseadas na arquitetura MDE.

No Capítulo 6 é descrito um estudo de caso visando analisar os mapeamentos e as transformações nas ferramentas, detalhando como o estudo de caso foi feito, o que foi analisado e os resultados obtidos. Aqui também é apresentado um guia geral de soluções possíveis para os problemas de modelagem encontrados na criação dos modelos do sistema para cada uma das ferramentas.

No Capítulo 7 são discutidos alguns trabalhos relacionados que propõem aplicação de padrões de segurança. Além disso, faz-se a comparação desses trabalhos com o proposto nessa dissertação.

No Capítulo 8 são apresentadas as conclusões e contribuições do presente trabalho e uma proposta de trabalhos futuros.

2 CLASSIFICAÇÃO DA PESQUISA E MÉTODO CIENTÍFICO

Neste capítulo são apresentados os procedimentos científicos aplicados na verificação da hipótese de pesquisa exposta no Capítulo anterior.

Os procedimentos são divididos em: características metodológicas, procedimentos técnicos, estratégia e planejamento. Esses são passos utilizados na repetição e validação de qualquer pesquisa para confirmação do objetivo proposto, obtenção dos resultados e verificação da legitimidade científica (WAZLAWICK, 2010), (SÓRIA, 2006). Unidos, eles caracterizam e classificam a pesquisa.

Em cada seção, após a explanação conceitual de cada procedimento científico, apresenta-se a sua relação com essa dissertação. Por fim, é apresentado o método científico utilizado na análise dos dados extraídos com os procedimentos científicos.

2.1 Características metodológicas

Com base nas classificações apresentadas por Silva e Menezes (2001), essa dissertação é classificada conforme apresentado na Tabela 1:

Tabela 1 - Classificação da pesquisa

Classificação Geral de Pesquisas	
Ponto de Vista	**Classificação da Pesquisa**
Natureza da pesquisa	Aplicada
Abordagem do problema de pesquisa	Qualitativa
Objetivos de pesquisa	Exploratória
	Descritiva
Procedimentos técnicos de pesquisa	Pesquisa Bibliográfica
	Levantamento

2.2 Natureza da Pesquisa

Neste trabalho será usada a pesquisa aplicada, que de acordo com Barros e Lehfeld (2000), tem como motivação a necessidade de produzir conhecimento para aplicação de seus resultados, com o objetivo de “contribuir para fins práticos, visando à solução mais ou menos imediata do problema encontrado na realidade”. Appolinário (2004) salienta que pesquisas aplicadas têm o objetivo de “resolver problemas ou necessidades concretas e imediatas”.

No presente caso, objetiva-se gerar conhecimento para aplicação prática na área de implantação de regras de transformações baseadas em padrões de segurança para o desenvolvimento de software. Com isso, pretende-se aumentar consideravelmente o nível de segurança e reduzir o tempo gasto no desenvolvimento de aplicações.

2.3 Abordagem do problema de pesquisa

Foca-se pela abordagem qualitativa, onde a fonte direta para coletas de dados é o ambiente natural e o pesquisador é a ferramenta chave. A pesquisa é descritiva e os pesquisadores tendem a analisar seus dados indutivamente, e os focos principais da abordagem são os processos e seus significados.

2.4 Objetivos de pesquisa

Quanto ao objetivo, essa pesquisa se classifica como exploratória e descritiva. Appolinário (2004) entende pesquisas exploratórias como sendo aquelas cujos objetivos se concentram em conhecer melhor o objeto a ser investigado. Segundo o autor, pode-se dizer que estas pesquisas têm como objetivo principal o aprimoramento de ideias ou a descoberta de intuições. Pode ser considerada ainda como exploratória, pois de acordo com o entendimento de Gil (2000), toda pesquisa

exploratória busca constatar algo num organismo ou num fenômeno, identificando fatores determinantes para a ocorrência dos fenômenos.

2.5 Procedimentos técnicos de pesquisa

Essa pesquisa caracteriza-se como pesquisa bibliográfica e de levantamento baseada na descrição de Appolinário (2004). A pesquisa bibliográfica é a atividade de localização e consulta de fontes diversas de informações escritas, para coletar dados gerais ou específicos a respeito de um tema. O material recolhido deve ser submetido a uma triagem, a partir da qual é possível elaborar um plano de leitura. Trata-se de uma leitura atenta e sistemática que se faz acompanhada de anotações que, eventualmente, poderão servir à fundamentação teórica do estudo.

2.6 Aplicação da pesquisa

O método de validação de uma pesquisa por meio de um estudo de caso enquadra-se como uma abordagem qualitativa e é frequentemente utilizado para coleta de dados na área de estudos organizacionais.

Yin (2001) discute que a adoção do Método do Estudo de Caso é adequada quando são propostas questões de pesquisa do tipo “como” e “por que”, e nas quais o pesquisador tenha baixo controle de uma situação que, por sua natureza, esteja inserida em contextos sociais. Embora o pesquisador utilize um quadro teórico referencial como ponto de partida para utilização do método, alguns estudos organizacionais enquadram-se em situações em que o pesquisador se vê frente a frente com problemas a serem compreendidos e para os quais estudos experimentais não podem ser aplicados; ou em situações nas quais estudos de natureza predominantemente quantitativa não dão conta dos fenômenos sociais complexos que estejam envolvidos nas mesmas.

3 MODEL-DRIVEN ENGINEERING

O presente capítulo trata da Engenharia Dirigida por Modelos (*Model-Driven Engineering* - MDE). Apresentam-se alguns dos conceitos encontrados na especificação do MDE e o processo de desenvolvimento descrito por esta. Em particular, detalham-se os conceitos de mapeamentos e transformações, apresentam-se os tipos de transformações e os tipos de mapeamentos definidos e mostram-se alguns exemplos de como estes mapeamentos podem ser realizados.

3.1 Abordagem MDE

Tanto a empresa como os requisitos e as especificações de um sistema podem ser representados de diversas maneiras. Neste trabalho, essas representações serão tratadas na perspectiva da Engenharia Dirigida por Modelos (MDE). A MDE é um paradigma de desenvolvimento e manutenção de sistemas intensivos de software que, segundo Den Haan (2009), é baseado no princípio de que tudo é modelo.

MDE visa elevar o nível de abstração na especificação do programa e aumentar a automação no desenvolvimento do programa. A ideia, promovida pelo MDE, é a utilização de modelos em diferentes níveis de abstração para desenvolvimento de sistemas, aumentando assim o nível de abstração na especificação do programa. Um aumento da automação no desenvolvimento de programas é alcançado através de transformações executáveis do modelo. Os modelos de um nível superior são transformados em modelos de um nível mais baixo até que o modelo possa ser executável usando a geração de código ou a interpretação modelo (DEN HAAN, 2009).

Um modelo é especificado em alguma notação ou linguagem de modelo. Desde que as linguagens do modelo sejam concebidas para um determinado domínio, tal linguagem é muitas vezes chamada de Linguagem de Domínio Específico (*Domain-Specific Language* - DSL), a qual pode ser visual ou textual. A

linguagem de descrição contém uma sintaxe abstrata, uma ou mais descrições da sintaxe concreta, os mapeamentos entre a sintaxe abstrata e concreta, e uma descrição semântica. A sintaxe abstrata de uma linguagem muitas vezes é definida usando um metamodelo. A semântica também pode ser definida usando um metamodelo, mas na maioria dos casos, na prática, a semântica não é explicitamente definida, ela têm que ser derivada a partir do comportamento de tempo de execução (DEN HAAN, 2009).

MDE é muitas vezes confundido com a Arquitetura Dirigida a Modelos (*Model Driven Architecture*- MDA), que pode ser vista como a visão da OMG (*Object Management Group*) em MDE. O MDA se concentra sobre a variabilidade técnica em software, ou seja, como especificar software de uma forma independente de plataforma.

MDA é uma abordagem de desenvolvimento de sistemas de software orientada por modelos que se caracteriza por separar a lógica do negócio da plataforma que lhe dá suporte. Ser orientada por modelos significa utilizar modelos como artefatos principais do desenvolvimento: são eles que dirigem o entendimento, o projeto, a construção, a implantação, a operação, a manutenção e a modificação do sistema. A separação da especificação da operação de um sistema de seus detalhes, e de como o sistema usa uma plataforma, é a principal diferença entre o MDA e outros enfoques de desenvolvimento dirigido por modelos (OMG, 2003b).

3.2 A utilização da UML na descrição dos modelos

Seguindo o princípio do MDE de que tudo é modelo, a transformação também pode ser vista como um modelo (BÉZIVIN, 2006). Assim como qualquer outro modelo, a transformação deve estar em conformidade com um metamodelo e esse modelo em conformidade com um metametamodelo. Essa ideia é apresentada esquematicamente na Figura 1. Nela é representada uma transformação em que o modelo de origem tem um metamodelo diferente do modelo de destino, mas ambos os metamodelos estão em conformidade com o mesmo metametamodelo. Com isso a transformação opera sobre esses modelos e/ou metamodelos (dependendo da abordagem seguida), dentro de um mesmo Espaço Técnico. A transformação em si

está em conformidade com outro metamodelo, condizente com o mesmo metametamodelo que os metamodelos de origem e de destino (SIQUEIRA, 2011).

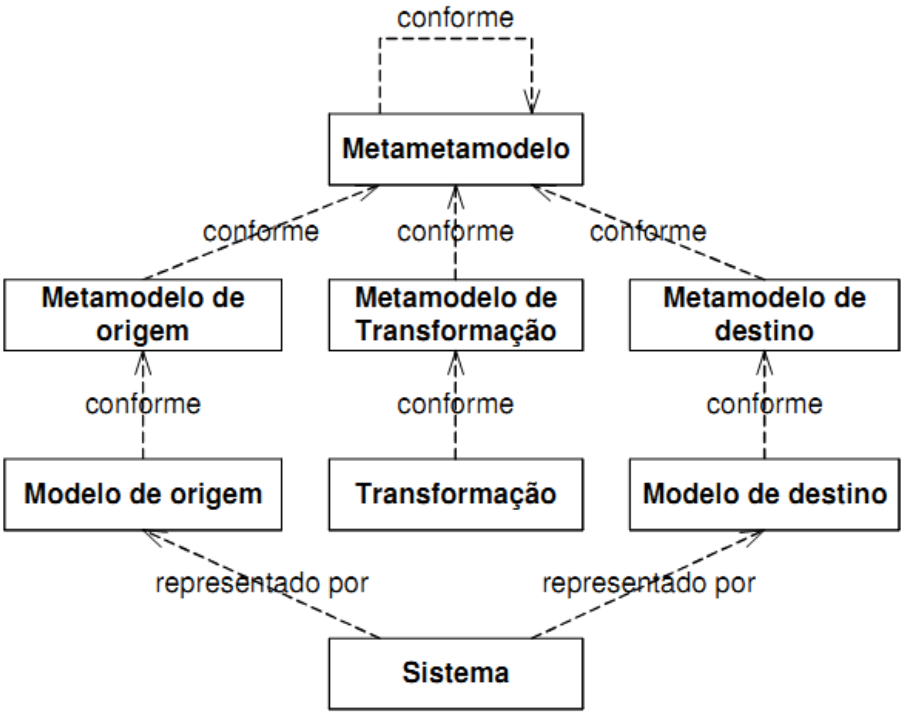


Figura 1 - Os modelos envolvidos em uma transformação de modelos e suas relações. Disponível em Siqueira (2011).

O objetivo do metamodelo é descrever a semântica de um modelo, pode-se dizer que um metamodelo é um modelo de um conjunto de modelos. A ligação entre o modelo e o metamodelo é estabelecida através da conformidade do primeiro com relação ao segundo. Além disso, deve-se notar que pode-se distinguir várias noções de conformidade, como por exemplo, a conformidade sintática e a conformidade semântica (BÉZIVIN, 2006).

Assim, estereótipos definem novos tipos de elementos de modelo ampliando a semântica dos tipos existentes ou de classes em um modelo UML. A notação consiste do nome do estereótipo escrito entre os sinais de maior e menor, por exemplo, <<*nomeDoEstereotipo*>>. A notação é anexada a um elemento do modelo, que recebe a definição descrita pelo estereótipo. Anexar um valor rotulado a um elemento do modelo é uma forma de definir uma propriedade para esse elemento. Um valor etiquetado é um par "*nome-valor*". A notação correspondente é

“*nome=valor”* com o valor constituindo atribuído ao nome. Essas estruturas são utilizadas para assinalar os elementos do PSM e tornam o modelo específico para plataforma.

3.3 MOF - Meta-Object Facility

O MOF é uma extensa especificação. O foco capital é o gerenciamento de metadados, provendo um *framework* que suporta qualquer tipo de metadados e comporte que novos tipos sejam agregados se necessário. O que comporta isso é sua arquitetura fundamentada na arquitetura tradicional de metamodelos, com quatro camadas que são padronizados por órgãos e normas. Estas camadas são assim descritas (GARDNER, 2003):

- Informação: É a camada utilizada pelos Objetos, e abrangem as informações que se almeja descrever. Estas informações são caracteristicamente referenciadas como “dados”;
- Modelo: Esta camada conglomera as informações descritas pelos metadados que são informalmente adicionados como modelos.
- Metamodelo: Abrange as descrições que determinam a estrutura e semântica dos metadados.
- Metametamodelo: Abrange as descrições que determinam a estrutura e semântica dos meta-metadados, ou seja, é a linguagem para definição de diferentes tipos de metadados.

A Figura 2 ilustra a arquitetura *Meta Object-Facility* (MOF) englobando modelos e metamodelos, onde se tem os padrões MOF, *Unified Modeling Language* (UML) e *XML Metadata Interchange* (XMI) e as camadas: metametamodelo (M3), metamodelo (M2), o modelo (M1) e, o mundo real (M0) (DJURIC, GAŠEVIC e DEVEDŽIC, 2006). A camada superior nesta arquitetura (M3) determina uma linguagem abstrata e a estrutura para especificar, construir e gerenciar metamodelos. É o embasamento para a determinação de qualquer linguagem de modelagem. MOF, também define um *framework* para a implementação de repositórios que contêm metadados (modelos, por exemplo) descrito por metamodelos (OMG, 2003a). O objetivo das quatro camadas com um

metametamodelo comum é amparar múltiplos metamodelos e modelos para permitir a sua extensibilidade, integração e modelo genérico e gestão metamodelo.

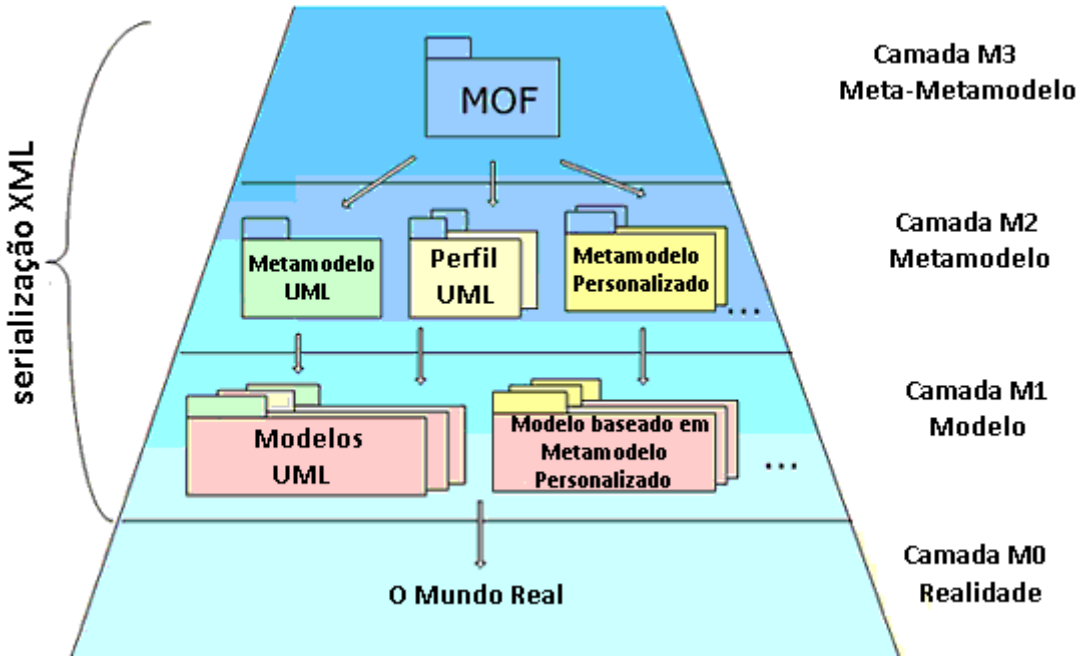


Figura 2 - Arquitetura MDA em quatro camadas baseada em MOF. Adaptado de Djuric, Gaševic e Devedžic (2006).

Todos os metamodelos determinados pelo MOF são dispostos na camada M2. Uma delas é a UML, uma linguagem de modelagem gráfica para especificar, visualizar e documentar sistemas de software. Com perfis UML, podem-se estender os conceitos básicos de UML (classes, associações, entre outras), com novos conceitos (uso de estereótipos) e adaptá-las às necessidades específicas de modelagem. Os modelos do mundo real, representado por conceitos definidos no metamodelo correspondente na camada M2 (metamodelo UML, por exemplo) estão na camada M1. Finalmente, na camada M0 são eventos do mundo real. Embora a abordagem no MDA foque em considerar os eventos do mundo real como exemplos de elementos de modelo, este foi alterado em novas abordagens para uma abordagem mais natural, onde o modelo é o mais próximo da realidade (DJURIC, GAŠEVIC e DEVEDŽIC, 2006).

Outro padrão desta arquitetura é o XMI, que define o mapeamento de MOF metamodelos a documentos XML e esquemas. O XML permite o metametamodelo do metamodelo, e partilha modelo através de XMI.

3.4 Atlas Transformation Language - ATL

A ATL é uma linguagem de modelagem exclusiva de domínio para representar transformações de um conjunto de modelos fonte em modelos destino, descritos por um ou vários metamodelos. Isto é feito com um estilo de programação híbrido que mistura os estilos imperativo e declarativo (JOUAULT e KURTEV, 2006).

Transformações ATL são definidas utilizando módulos. Um módulo possui uma seção *header* (cabeçalho) indispensável, uma seção *import* (importadora), um número de *helpers* (ajudantes) e regras de transformação que determinam a funcionalidade da transformação. A seção *header* dá o nome do módulo de transformação e declara os modelos fonte e destino. Os modelos fonte e destino são tipados por seus metamodelos (JOUAULT e KURTEV, 2006).

Em uma regra, a seção *from* define qual classe do modelo fonte será transformada em qual classe do modelo destino (definido na seção *to*). Duas seções adicionais necessitam ainda surgir na regra: a seção *using* que define variáveis locais a serem empregadas, e a seção que permite o uso de código imperativo. Além do mais, regras podem tanto ser acopladas como chamadas. Uma regra acoplada segue uma abordagem declarativa, ou seja, ela será executada sempre que existam classes no modelo fonte que combine com sua seção *from*, e uma regra chamada poderá ser executada apenas se for invocada por outra regra.

3.5 Transformações

Transformação é a atividade de gerar um modelo alvo a partir de um modelo fonte, de acordo com um mapeamento, formado por um conjunto de regras de

mapeamentos, que juntas descrevem como um modelo na linguagem fonte pode ser transformado em um modelo na linguagem alvo (KLEPPE *et al.*, 2003).

Por mais que um dos pontos fundamentais do MDE seja a automação, não necessariamente a transformação deve ser realizada de forma automática. No caso da abordagem via marcação é obrigatória a intervenção humana, já que é necessário que alguém marque o modelo de origem. Nas outras formas de transformação – e mesmo adicionalmente à marcação – podem também ser acrescentadas algumas informações, adicionando conhecimento ao modelo de origem e guiando a transformação (OMG, 2003a). Em um caso extremo, a transformação em si pode ser realizada manualmente (OMG, 2003b). O importante para que essa transformação siga a ideia da MDE é que a origem e o destino da transformação sejam modelos e que haja um registro da transformação realizada.

3.6 Mapeamentos

Em MDA, uma das características chave é a noção de mapeamento. Mapeamento é um conjunto de regras e técnicas usadas para modificar um modelo a fim de gerar outro. Os mapeamentos são usados para transformar (OMG, 2003b):

- PIM para PIM – Esta transformação é usada quando os modelos são refinados, filtrados ou especializados durante o ciclo de vida do desenvolvimento sem necessitar nenhuma informação que dependa da plataforma. O mapeamento mais óbvio é o da análise para o projeto. Mapeamentos PIM para PIM são usados geralmente para o refinamento do modelo.
- PIM para PSM – Esta transformação é usada quando o PIM está refinado o suficiente para ser projetado para a infraestrutura específica. A projeção é baseada nas características da plataforma. A descrição destas características deve ser feita usando uma descrição UML e eventualmente um profile para descrever conceitos comuns da plataforma. Ir de um modelo de componente lógico a um modelo de componente comercial existente, como EJB para a plataforma J2EE ou CCM para a plataforma CORBA, é um tipo de mapeamento PIM para PSM.

- PSM para PSM – Esta transformação é necessária para a concretização e a extensão dos componentes. Por exemplo, empacotamento dos componentes é realizado através da seleção dos serviços e sua configuração. Uma vez empacotados, a entrega dos componentes pode ser feita pela especificando inicial dos dados, máquina salvo, geração e configuração do *container*, etc. Mapeamentos PSM para PSM são usados geralmente para refinar o modelo dependente da plataforma.
- PSM para PIM – Esta transformação é requerida para a abstração de modelos implementados em uma plataforma específica em um modelo de plataforma independente. Este procedimento assemelha-se frequentemente a um processo de "mineração" que é difícil ser automatizado inteiramente. Porém pode ser suportado por ferramentas. Idealmente, o resultado deste mapeamento resultará no PIM correspondente ao mapeamento PSM.

A Figura 3 é destinada a ser sugestiva e genérica uma vez que o PIM e outras informações são combinados com a transformação para produzir um modelo específico de plataforma. É permitida ainda a geração de transformações bidirecionais (com recursos de engenharia reversa e foco em regras de negócio), inclusive entre modelos do mesmo nível, como, por exemplo, a geração de um ou mais PSMs a partir de outro PSM.

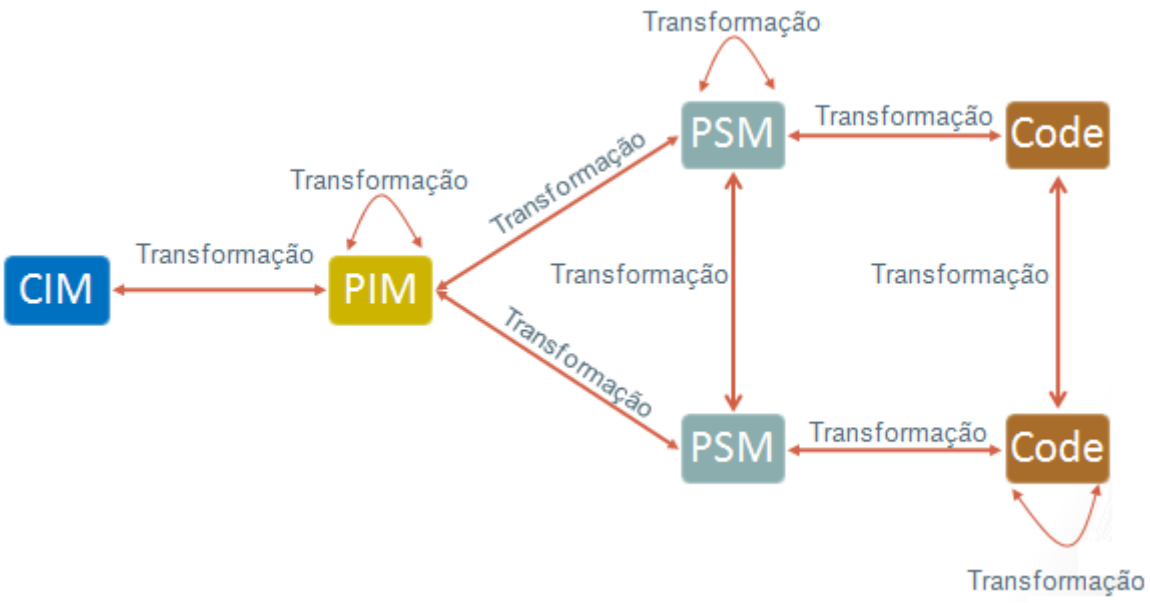


Figura 3 - Abordagem MDA com as possíveis transformações bidirecionais.

3.7 Transformações e Mapeamentos de Modelos

Juntamente com os conceitos de modelo e de metamodelo, o conceito de transformação de modelo faz parte do núcleo da MDE. Uma transformação de modelos estabelece uma relação entre dois modelos que representam diferentes níveis de abstração do sistema real modelado. Cada modelo sendo conforme a um dado metamodelo, uma transformação de modelos recebe em entrada um modelo conforme ao metamodelo origem e produz como saída o modelo equivalente para o metamodelo destino (Figura 4). Como exemplo de transformação de modelo, pode-se citar a transformação que recebe como entrada um modelo UML e produz na saída o documento XML correspondente.

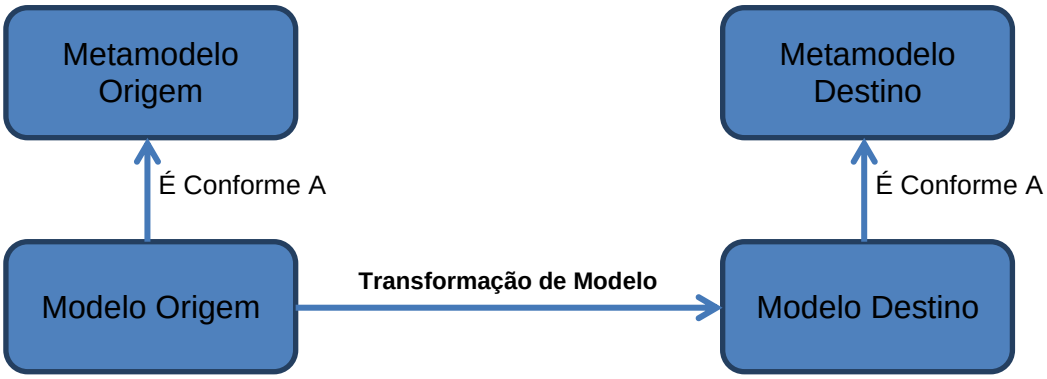


Figura 4 - Transformações de modelos. Adaptado de Medeiros (2009).

As transformações de modelos não necessitam serem fundamentalmente realizadas em uma etapa, elas podem ser feitas em partes para ajudar o desenvolvimento das transformações e atenuar sua complexidade. Em (ATKISON e KÜHNE, 2008) é introduzida à noção de uma plataforma abstrata empregada para transformações parciais de PIM para PSM. As plataformas abstratas descritas representam o suporte provido por plataformas que é ostentada pelo projetista na modelagem independente de plataforma. São tipos genéricos de plataformas que podem ser concretizados em mais de uma plataforma concreta.

3.8 Transformações de metamodelos

Metamodelos são modelos em uma linguagem de modelagem. Eles definem a estrutura, a semântica e as restrições para uma família de modelos, isto é, grupos de modelos que compartilham uma sintaxe e uma semântica comuns (MELLOR, 2004). Um metamodelo é uma definição precisa das construções e regras imprescindíveis para criar modelos semânticos. Modelos podem ser instanciados a partir de um metamodelo.

O metamodelo da UML é uma instância da Infraestrutura de Meta-Objetos (MOF), como visto em (OMG, 2003a). O MOF é um arcabouço de gerenciamento de metadados e um conjunto de serviços de metadados que possibilitam o desenvolvimento e interoperabilidade de sistemas dirigidos por modelos e metadados. Resumidamente, a MOF é designada para a criação de metamodelos, sendo chamado de metametamodelo.

Perfil UML é um mecanismo de extensão da UML que permite a personalização de metamodelos definidos pelo MOF, como a UML, para diferentes plataformas tecnológicas ou domínios de aplicação (OMG, 2003a). Os perfis UML permitem a personalização de qualquer metamodelo definido pela MOF, adicionando novos tipos de elementos da linguagem ou restringindo a linguagem para adaptar os metamodelos para diferentes propósitos. Um perfil UML pode também especificar outro perfil UML. A noção de perfil UML é utilizada pelo MDA.

O PIM é construído usando uma linguagem independente de plataforma especificada por um metamodelo. Escolhe-se uma plataforma e constrói-se, se já não existir, uma especificação de uma transformação mapeando os metamodelos do PIM no PSM. A transformação segue esse mapeamento. Esse processo é mostrado na Figura 5, extraída de (OMG, 2003b).

Os mapeamentos de tipos do modelo apenas são capazes de expressar transformações em termos de regras que geram elementos de tipos do PSM, a partir de elementos que são de um determinado tipo do PIM (por exemplo: todos os elementos do tipo classe da UML para elementos do tipo tabela em um metamodelo relacional). Com isso, a transformação é sempre executada de forma determinística e guiada por informações independentes de plataforma (OMG, 2003b).

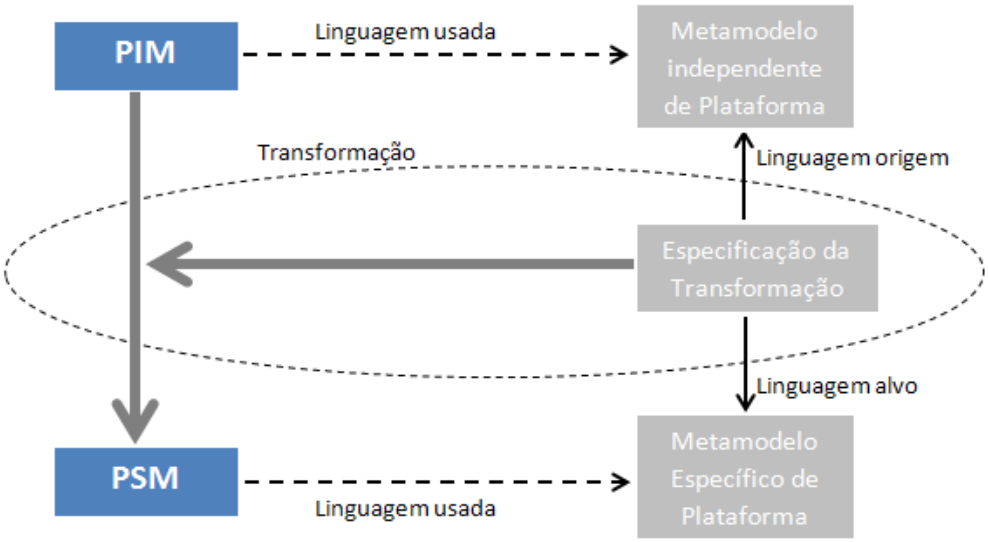


Figura 5 - Transformações de metamodelos. Adaptado de Weiss e Mouratidis (2008).

3.9 Transformações Modelo-Modelo

MDA define uma sequência de atividades para o desenvolvimento de software. Essas atividades são divididas em etapas, começando com o levantamento dos requisitos do sistema e pela elaboração do CIM, baseado nestes requisitos. A partir do CIM gera-se o PIM que é um modelo com alto nível de abstração, independente de qualquer tecnologia de implementação, onde todas as funcionalidades do sistema e restrições do negócio serão modelados (BASIN e DOSER, 2005). Uma transformação deve ser aplicada ao PIM para que se possa gerar um PSM. Esse modelo combina as especificações no PIM com os detalhes que especificam como o sistema usa um determinado tipo de plataforma (OMG, 2003b). Como o PIM é um modelo inicial, uma especificação MDA completa consiste de um PIM base, além de um ou mais PSMs e conjuntos de definições de interface, cada um descrevendo como o modelo base é implementado em uma plataforma de middleware diferente (OPENGROUP, 2007).

Esses modelos e suas melhorias são uma parte crítica da metodologia de desenvolvimento para situações que incluem refinamentos entre os modelos que representam diferentes aspectos do sistema, além de detalhes de um modelo, ou a conversão entre diferentes tipos de modelos. Essa perspectiva é vislumbrada como uma transformação que é a atividade de gerar um modelo alvo a partir de um

modelo fonte, de acordo com um mapeamento, formado por um conjunto de regras, que juntas descrevem como um modelo na linguagem fonte pode ser transformado em um modelo na linguagem alvo (KLEPPE, WARMER E BAST, 2007).

A OMG (2003) apresenta cinco abordagens para as transformações:

- Marcas - após a escolha da plataforma, cria-se, se não existir, um mapeamento para esta plataforma, o qual possui um conjunto de marcas que devem ser usadas para marcar elementos do PIM para que este seja transformado em um PSM.
- Metamodelo - O PIM é constituído usando a linguagem independente de plataforma especificada por um metamodelo. Escolhe-se uma plataforma e constrói-se, se não existir, uma especificação de uma transformação mapeando os metamodelos do PIM no PSM.
- Modelos utilizando tipos - segue o mesmo modelo da transformação anterior, entretanto utiliza tipos independentes de plataforma. Os elementos do PIM são subtipos desses tipos independentes de plataforma.
- Aplicação de gabaritos - podem ser usados em transformações de frameworks e transformações de modelos para mapear o PIM para tipos e gabaritos no PSM.
- União de modelos - essas transformações são baseadas na união de modelos, como, por exemplo, o uso de padrões.

As transformações utilizadas neste trabalho são baseadas em mapeamentos com metamodelos, uma vez que traz como vantagem ao processo o baixo custo, por ser uma ferramenta livre; e alta flexibilidade, por facilitar grandes alterações na transformação diretamente através da interface do editor de regras *Atlas Transformation Language* (ATL).

3.10 Aplicação das Transformações Modelo-Modelo

As transformações deste conjunto realizam a transformação de um modelo de entrada, em um modelo de saída. Estes modelos podem ser instâncias do mesmo metamodelo ou de metamodelos distintos. O tipo básico de transformação modelo-

modelo é a transformação PIM-PSM, entretanto, antes de gerar o modelo PSM, inúmeras transformações no modelo PIM podem ser necessárias. Portanto, os modelos manuseados por essas transformações podem ser nomeados modelo de entrada e saída, origem e destino ou ainda esquerda e direita para as transformações bidirecionais.

A especificação de transformação dentre modelos pode ser realizada por meio de uma linguagem genuinamente declarativa, imperativa, ou uma combinação desses dois tipos.

Uma linguagem declarativa apresenta relacionamentos entre variáveis através de funções ou regras de inferência, e um interpretador/ compilador dessa linguagem aplica um algoritmo fixo sobre essas relações para produzir um resultado (MAIA, 2006). No caso da declaração de transformações, uma especificação declarativa pode conter informação suficiente para descrever completamente uma transformação, seja unidirecional ou bidirecional (GARDNER *et al.*, 2002).

Uma linguagem imperativa permite a especificação explicita do estado de um sistema computacional. Qualquer linguagem de manipulação de dados, por exemplo, SQL, pode ser considerada uma linguagem imperativa (MAIA, 2006). Uma implementação imperativa pode explicitamente construir os elementos no modelo de destino (GARDNER *et al.*, 2002).

Em um cenário típico de desenvolvimento, modelos de alto nível de abstração (ex.: modelo de casos de uso) são refinados sucessivamente até o início da fase de implementação, onde os conceitos e funcionalidades representados nos modelos são codificados na linguagem de programação da plataforma escolhida. Estes refinamentos, na grande maioria dos casos, são caracterizados pela realização de tarefas repetitivas que podem resultar na inserção de inconsistências nos modelos. Justamente por isso, estes refinamentos podem ser automatizados através de transformações (MAIA, 2006).

4 PADRÕES DE SEGURANÇA

Neste capítulo destacam-se alguns conceitos, metodologias e técnicas empregadas com o objetivo de garantir a segurança da informação em sistemas computacionais.

4.1 Segurança baseada em modelos

Essencialmente, a segurança baseada em modelos pode gerar automaticamente regras técnicas de autorização ao analisar o aplicativo com todas as suas interações e aplicar os requisitos genéricos de segurança a ele (ROSADO, 2006).

A segurança baseada em modelos é um processo suportado por ferramentas, que envolve a modelagem de requisitos de segurança em um alto nível de abstração. A segurança é direcionada, principalmente, aos modelos funcionais do aplicativo para gerar automaticamente regras técnicas de autorização com baixa granularidade e contextuais (CUNHA, 2007).

Ainda segundo o autor, as necessidades de segurança de um sistema dependem fortemente do ambiente no qual o sistema será implantado. Além disso, o comportamento da informação com as restrições de segurança estão incluídos em um modelo padrão de segurança. O desenvolvedor pode usar esta informação para verificar se um projeto específico ou a implementação do padrão é consistente com as propriedades essenciais de segurança.

As organizações de segurança estão passando por uma mudança significativa na forma como elas percebem o seu próprio problema. De um modo geral, segurança é uma palavra-chave no desenvolvimento de software. Notoriamente, sua importância tem sido intensa, pois a maioria dos ataques a sistemas são baseados em vulnerabilidades causadas por aplicativos mal concebidos e desenvolvidos (CUNHA, 2007). Em decorrência dessa situação, Schumacher *et al.* (2006) propõem uma vasta coleção de padrões de segurança,

onde a abordagem baseia-se num conjunto de ativos em camadas que pode ser explorada por qualquer metodologia de desenvolvimento existente.

A utilização de segurança com MDA permite que segurança esteja integrada na modelagem de sistemas à medida que modelos de projeto são combinados com modelos de segurança e técnicas de geração automática de código são utilizadas para automatizar a construção de sistemas a partir desses modelos. Isto auxilia a criação de arcabouços de segurança flexíveis e garante que os requisitos de segurança sejam levados em consideração durante todas as fases do processo de desenvolvimento do sistema. Com isso, as falhas de segurança podem ser identificadas mais rapidamente no processo de desenvolvimento e a implementação é mantida consistente com a política de segurança modelada, além de poder ser migrada para novas plataformas (ROSADO *et al.*, 2006).

4.2 Segurança com Padrões

Aplicações de comércio eletrônico e outras similares no âmbito da internet possuem a segurança como requisito fundamental e apesar da grande variedade de arcabouços de segurança disponíveis, constantemente há notícias sobre vulnerabilidades e falhas de segurança em sistemas de software (FINK *et al.* 2006). Á medida que os arcabouços aparecem, eles se tornam rapidamente ultrapassados por diferentes motivos. Na maioria das vezes porque eles não oferecem um modelo arquitetural suficientemente flexível para acompanhar as necessidades do negócio, o qual está em desenvolvimento ou porque eles limitam o escopo da arquitetura de que o negócio necessita.

Devido à disparidade de requisitos de negócio de cada aplicação, arquitetos de software e analistas precisam cada vez mais criar seus próprios arcabouços e conservar suas integridades e coesão em relação às obrigações do negócio as quais eles atendem. Entretanto, na maior parte, os projetistas de sistemas não possuem experiência em segurança (CUNHA, 2007).

Além disso, existe uma ampla lacuna entre as soluções teóricas e o que é desenvolvido na área de segurança. Uma das dificuldades que colaboram para a elaboração de software com segurança fraca é este requisito ser raramente avaliado

nos estágios iniciais do desenvolvimento de software e ser descartado e relegado a segundo plano ao longo do mesmo, ocasionando problemas de segurança tanto na arquitetura da aplicação quanto na lógica implementada (FINK, KOCH e PAULS, 2006).

De acordo com os problemas relatados, são necessárias novas formas de desenvolver software seguro, baseadas não somente na aplicação das teorias existentes como na adoção de um processo de desenvolvimento que considere os requisitos de segurança como parte integral do projeto de construção de software (FERNANDEZ e SORGENTE, 2005). Além disso, o aumento da complexidade dos sistemas aliado à dinamicidade inerente às regras de negócio das aplicações atuais faz com que o processo de desenvolvimento de segurança necessite cada vez mais de arcabouços flexíveis e maior gerência de requisitos e mudanças.

Uma das abordagens que pode auxiliar arquitetos e projetistas a construir sistemas seguros consiste no uso de padrões de segurança. Contudo, para obter os benefícios da utilização destes padrões é indispensável saber quando e como utilizá-los a partir de suas definições, pois os desenvolvedores podem acabar introduzindo falhas de segurança ao longo do projeto. É necessário que exista analistas e programadores com alto grau de conhecimento de padrões de segurança e de estrutura de segurança. Esse conhecimento faz com o que o custo do desenvolvimento de sistemas aumente consideravelmente. Desta forma, uma abordagem orientada a modelo faculta o processo de desenvolvimento de sistemas seguros, promovendo uma implementação controlada dos padrões de projeto (CUNHA, 2007).

4.3 Padrão

Padrões foram inicialmente definidos como uma solução para um problema em um contexto (CHRISTOPHER, 1978). Essa definição é marcada como um erro de interpretação por Vlissides (1997), uma vez que o problema para o qual o padrão oferece uma solução deve ocorrer inúmeras vezes, não se restringindo a uma única vez. Vlissides enfatiza ainda, a necessidade de a solução ser relevante em condições que não são imediatas, além de não proporcionar a obrigatoriedade de ter

um nome para referenciar o padrão. Outro problema analisado pelo autor é a necessidade de um padrão ser uma solução comprovada e eficiente.

Embora critique a definição proposta por Alexander, ele não aponta uma definição própria, justificando que uma definição aceitável seria de difícil entendimento. Entende-se, deste modo, que um padrão é uma solução comprovada elaborado por pessoas experientes que já enfrentaram um problema várias vezes em determinadas instâncias de um contexto.

Um padrão sempre possui um valor agregado, uma verdade essencial que permite ao usuário identificar a qualidade embutida nele e, assim, sentir-se confortável e satisfeito com a aplicação para a resolução de seu problema. Os padrões fazem parte de uma estrutura, uma linguagem de padrões, que relaciona cada um deles a uma coleção e a um domínio. Ela descreve os cincos componentes (requisitos funcionais) que julga ser necessário em uma linguagem de padrões, bem como os requisitos não funcionais presentes em qualquer sistema, que fazem a diferença, pois dá a noção da “qualidade sem nome”, verdade essencial para a satisfação dos usuários.

Os requisitos funcionais devem capturar a prática, ou seja, um padrão deve descrever um problema que existe no mundo real, proporcionar abstração de uma qualidade de um conjunto de exemplos em um nível de entendimento imediato, prover exemplos que descrevam o real valor de um padrão no seu domínio, estruturar princípios para formar uma linguagem e, finalmente, encontrar uma forma de apresentação ideal para facilitar o entendimento dos conceitos e da motivação envolvidos (ROSADO *et al.*, 2001).

Os requisitos não funcionais são inevitavelmente relacionados aos requisitos funcionais. O primeiro requisito não funcional que padrões devem descrever soluções que não são óbvias, para tanto, é necessário experiência e pesquisa intensa no domínio. Um segundo requisito não funcional é o classificado como *insight*, ou seja, um padrão faz mais do que identificar uma solução, ele explica a necessidade dessa solução. E, finalmente, um padrão deve ter poder comunicativo para facilitar sua divulgação e aceitação (ROSADO *et al.*, 2001).

4.4 Padrão de Segurança

Padrões de segurança são soluções reutilizáveis aos problemas de segurança. Embora muitos padrões de segurança e técnicas para usá-los têm sido propostos, é complexo adaptá-los e integrá-los em cada fase do desenvolvimento de software (SCHUMACHER, 2003). Um padrão pode contribuir para solucionar problemas de má implementação de processos de segurança, como também para aumentar a eficiência de procedimentos de segurança já implementados. No contexto organizacional é observado o considerável esforço científico e acadêmico em desenvolver padrões que focalizem soluções para problemas relacionados com a proteção dos sistemas de informação (SI). Entretanto, a aplicação de padrões para o desenvolvimento de processos de gestão da segurança para SI ainda é limitada por parte das organizações.

De acordo com Schumacher (2003), os padrões, sejam eles autônomos ou integrados em um sistema padrão ou linguagem padrão, devem ser apresentados de uma forma adequada. Uma boa descrição nos ajuda a compreender a essência de um padrão imediatamente, identificar qual é o problema e qual é a solução proposta. Uma boa descrição também nos fornece os detalhes necessários para implementá-lo, e para considerar as consequências de sua aplicação.

Este trabalho segue o formato descrito por Schumacher et al (2006) o qual relata os seguintes conceitos para descrição de um padrão:

- Nome - o nome e um breve resumo do padrão. Deve capturar a essência do padrão.
- Também conhecido como - enumera outros nomes para o padrão, incluindo nomes pelos quais outros podem ter referenciado na literatura.
- Exemplo - um exemplo do mundo real, demonstrando a existência do problema e da necessidade para o padrão.
- Contexto - situações nas quais o padrão pode ser aplicado. As condições gerais em que o problema ocorre e quais as forças que surgem são descritas.
- Problema - deve descrever as condições que motivam o uso do padrão. Esta seção descreve o contexto em que o padrão é aplicável, bem como a

explicação da sua motivação para o uso. Quando vários padrões possuem o mesmo problema básico, o problema para cada um deve fornecer o contexto mais detalhado que o diferencia do outro.

- Solução - define os conceitos de arquitetura independente de tecnologias específicas e estratégias de implementação. Deve se determinar claramente os conceitos, limites e relações entre os blocos da arquitetura.
- Estrutura - determina a especificação detalhada dos aspectos estruturais do padrão, utilizando-se notações apropriadas.
- Dinâmica - os cenários típicos que descrevem o comportamento de tempo de execução do padrão. Para os padrões estruturais, um diagrama da UML deve ser usado para descrever a arquitetura visual. Para padrões de nível de objeto, a UML pode ser utilizada para sua representação através do diagrama de objetos. Para padrões de procedimento, os diagramas de sequencia podem representar os intervenientes no processo descrito.
- Implementação - deverá proporcionar informações na forma de dicas detalhadas e técnicas. Deve identificar os erros mais comuns no uso desse padrão e fornecer ao leitor uma orientação para evitá-los. Deve adaptar a aplicação para atender suas necessidades, adicionando diferenças ou etapas mais detalhadas. Sempre que possível aplica-se fragmentos da UML para ilustrar uma possível implementação, descrevendo detalhes do problema.
- Exemplo resolvido - discussões sobre quaisquer aspectos importantes para a resolução do exemplo, ainda não tratados nos itens anteriores.
- Variantes - demonstra uma breve descrição das variantes ou especializações de um padrão.
- Usos conhecidos - citam-se exemplos que são conhecidos em uso real. Referências explícitas a produtos ou sistemas podem ser utilizadas.
- Consequências - descreve o eventual impacto da utilização do padrão em relação aos requisitos funcionais e não funcionais.
- Consulte também - apresenta referências a padrões que resolvem problemas similares e aos que ajudam a aperfeiçoar o padrão que está sendo descrito.

5 CONJUNTO DE PADRÕES DE SEGURANÇA

Neste capítulo é descrito o conjunto de padrões de segurança que é utilizado para gerar o modelo contendo a agregação e relacionamentos dos padrões escolhidos. Um padrão é uma solução para um problema em um contexto que é útil para o projeto, mas, surge um problema: há uma proliferação de padrões de segurança e, desta forma, está se tornando mais difícil selecionar quais devem ser aplicados, neste capítulo se descrevem os passos para se escolher os padrões de forma adequada.

5.1 A decisão para o uso dos padrões de segurança

Um sistema de software é elaborado para solucionar um problema. Durante o processo de desenvolvimento, o problema é primeiramente analisado e, em seguida, a solução para o problema é iterativamente refinado. Durante cada fase do ciclo de vida de software, a solução é vista de um nível diferente de abstração. A segurança é um requisito não funcional que deve ser definida a partir do nível conceitual. Assim, os padrões de segurança também existem em diferentes níveis de abstração. Na camada inferior, os padrões de segurança para uma plataforma específica, linguagem ou tecnologia são encontrados, que controlam uma ameaça de segurança para este contexto particular.

Padrões de segurança abstratos residem na camada mais alta, que controlam as ameaças genéricas, como a representação, o repúdio, reprodução de mensagens, a divulgação não autorizada de dados, etc. Nesse nível são padrões que serão aplicados de forma diferente, dependendo da tecnologia subjacente, da linguagem ou da plataforma em uso.

Pode-se pensar em construir um sistema de software como resolver um problema. Na fase de análise do desenvolvimento, procura-se fazer com que o problema seja sanado, não há a preocupação com aspectos de software, tais como a implementação e plataforma. Do ponto de vista da segurança, se deseja apenas

indicar que os mecanismos de segurança específicos são necessários, e não a sua implementação. Portanto, é necessário nesta fase um conjunto de padrões que definem os mecanismos de segurança abstratos. Esses padrões devem especificar apenas as características fundamentais do mecanismo ou serviços, não os aspectos específicos do software.

5.2 Modelo de padrões de segurança proposto

Segurança aplicada ao MDE fornece métodos e ferramentas para integrar segurança no processo de desenvolvimento. A ideia principal é o uso de abstração, estabelecendo modelos visuais que associam o projeto do sistema com o de segurança e emprega técnicas de geração automática de código para automatizar a elaboração de sistemas a partir desses projetos.

A criação de um modelo no nível PIM possui um nível de detalhamento abstrato a plataforma. Cada modelo deve instanciar um modelo que representa um domínio, uma área de conhecimento sobre a qual diversos modelos distintos podem ser construídos. Tendo como base os requisitos levantados, o primeiro passo foi a elaboração de um modelo a partir dos padrões de segurança. A fim de aplicar esses conceitos no desenvolvimento do modelo proposto, adotam-se os padrões de segurança classificados por Schumacher (2003) como *Access Control Models, System Access Control Architecture e Operating System Access Control*.

Access Control Models - apresenta padrões que especificam os controles de acesso como modelos orientados a objetos, padrões declarativos que podem ser usados como diretrizes para a construção de sistemas seguros. Existem ainda padrões que documentam a dinâmica de avaliar pedidos de acordo com as limitações definidas pelos modelos declarativos e, mostra padrões que ajudam a encontrar os direitos associados a funções em um modelo de controle de acesso baseado em função. Essa classificação tem-se os seguintes padrões:

- *Authorization* - descreve as regras que definem os acessos permitidos aos recursos

- *Role-Based Access Control* (RBAC) - uma extensão do padrão *Authorization* em que os direitos de acesso são dados aos papéis funcionais.
- *Multilevel Security* - baseada em níveis de isenção para determinar o acesso, descrevendo como categorizar as informações sensíveis e impedir a sua divulgação.
- *Reference Monitor* - em um ambiente computacional no qual os usuários ou processos realizam pedidos de dados ou recursos, esse padrão impõe restrições de acesso quando ocorre uma solicitação de recursos em uma entidade ativa.
- *Role Rights Definition* - auxiliam a encontrar os direitos associados com papéis em um modelo RBAC.

System Access Control Architecture - apresenta padrões de controle de acesso a nível arquitetônico. Há um padrão que mostra por que e como reunir os requisitos subjacentes para um sistema em consideração, a partir de um conjunto genérico de requisitos de controle de acesso. Contém, ainda, padrões que lidam com a arquitetura de sistemas de software a ser assegurado pelo controle de acesso. Essa classificação tem-se os seguintes padrões:

- *Access Control Requirements* - explica por que e como reunir os requisitos subjacentes para um sistema em consideração, a partir de um conjunto genérico de requisitos de controle de acesso.
- *Single Access Point* - define um ponto de entrada único para o sistema, e pode ser avaliado quando da aplicação da política de segurança desejada.
- *Check Point* - fornece serviços de segurança de controle de acesso para uma aplicação ou sistema.
- *Security Session* - verifica os direitos de um usuário de identidade e acesso para cada função do sistema.
- *Full Access with Errors* - usuários não devem ser capazes de exibir dados ou executar operações para as quais eles não têm permissões.
- *Limited Access* - orienta o desenvolvedor a apresentar apenas as funções atualmente disponíveis para um usuário, escondendo tudo o que ele não tem permissão.

Operating System Access Control - apresenta padrões para os serviços de controle de acesso e mecanismos destinados a sistemas operacionais que descrevem como estes controlam o acesso a recursos como espaços de memória de endereço e dispositivos de entrada e saída. Essa classificação tem-se os seguintes padrões:

- *Authenticator* - é o processo de identificar individualmente os clientes de suas aplicações e serviços.
- *Controlled Process Creator* - trata de como definir e conceder direitos de acesso apropriados para um novo processo.
- *Controlled Object Factory* - trata como especificar os direitos de processos com respeito a um novo objeto.
- *Controlled Object Monitor* - trata de como controlar o acesso através de um processo a um objeto.
- *Controlled Virtual Address Space* - trata de como controlar o acesso de processos para áreas específicas de seu espaço de endereço virtual de acordo com um conjunto de tipos de acesso pré-definidos.
- *Execution Domain* - define um ambiente de execução para os processos, indicando explicitamente todos os recursos em que um processo pode ser utilizado durante a sua execução, bem como o tipo de acesso aos recursos.
- *Controlled Execution Environment* - controla o acesso a todos os recursos do sistema operacional por processos, com base em autorizações de usuário, grupo ou papel.
- *File Authorization* - descreve como controlar o acesso a arquivos em um sistema operacional.

A escolha por esses padrões ocorreu pelo fato deles especificarem modelos de controle de acesso orientados a objetos, declarativas que possam ser usadas como diretrizes para a construção de sistemas seguros, formas de utilização para coletar os requisitos fundamentais de um sistema considerado a partir de um conjunto genérico de requisitos de acesso e mecanismos destinados a descrever como o sistema operacional controla o acesso a recursos.

A Figura 6 ilustra as características principais para formação do modelo para analisar as composições dos padrões de segurança. Em um primeiro momento,

cada padrão é convencionalmente especificado de forma declarativa, usando o formato descrito na seção 4.4 deste trabalho. Suas especificações são genéricas no sentido de que eles capturam boas práticas de projeto de uma forma independente de domínio. Essas representações declarativas, as quais constituem os modelos dos padrões de segurança, em seguida, são instanciadas em representações concretas de domínio específico.

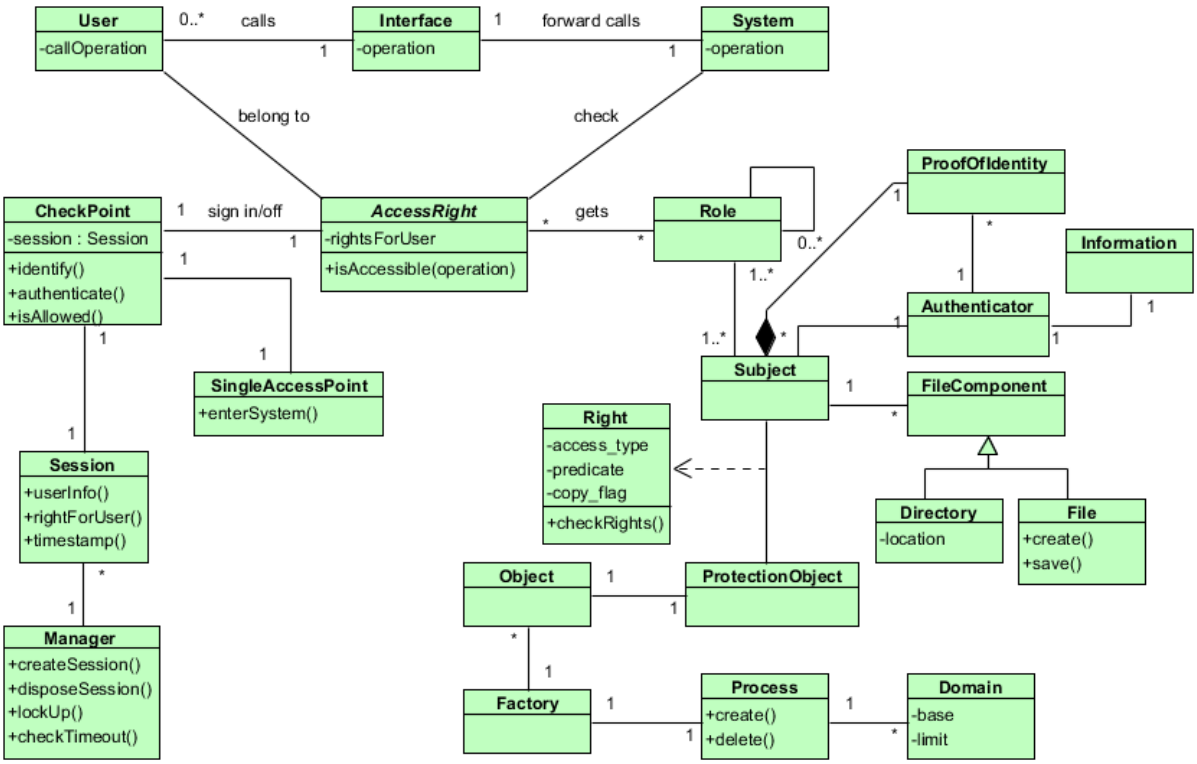


Figura 6 - Modelo baseado em padrões de segurança

A seguir são descritos os passos utilizados para os tipos de relações de padrões que aparecem no modelo proposto, tomando como exemplo os padrões *Authenticator* e *Authorization*, conforme observado na Figura 7. A relação entre esses dois padrões existe dentro de uma mesma camada de abstração.

Descreve-se agora o processo de unificação em mais detalhe. Na especificação UML, a unificação é definida por regras, restrições e transformações. As regras de transformações são agrupadas de acordo com tipos de elemento (por exemplo, as associações diferem das operações). Podem-se distinguir entre os três grupos seguintes:

- Combinar as regras definindo a igualdade. Se dois elementos são unificados, eles são considerados para representar o mesmo elemento e são unificados para em conjunto representar o mesmo elemento do modelo resultante. Por exemplo, na Figura 7, o *Subject* do padrão *Authorization* coincide com *Subject* do padrão *Authenticator*, e assim estas duas classes são unificadas para produzir a classe resultante. Em geral, dois elementos se correspondem se eles têm o mesmo nome. Algumas exceções incluem operações, bem como o mesmo nome, e associações, que devem ter correspondência entre tipos de associação.
- As restrições definem as pré-condições sobre a unificação. Se um par de *receiving*/pacotes unificados violam qualquer das restrições, marca-se a unificação como não é válida.
- Transformações definem as pós-condições da fusão. Descrevem como resolver os conflitos quando dois elementos correspondentes são unificados recursivamente (por exemplo, se duas classes correspondentes têm atributos diferentes, ou dois atributos correspondentes têm multiplicidades diferentes).

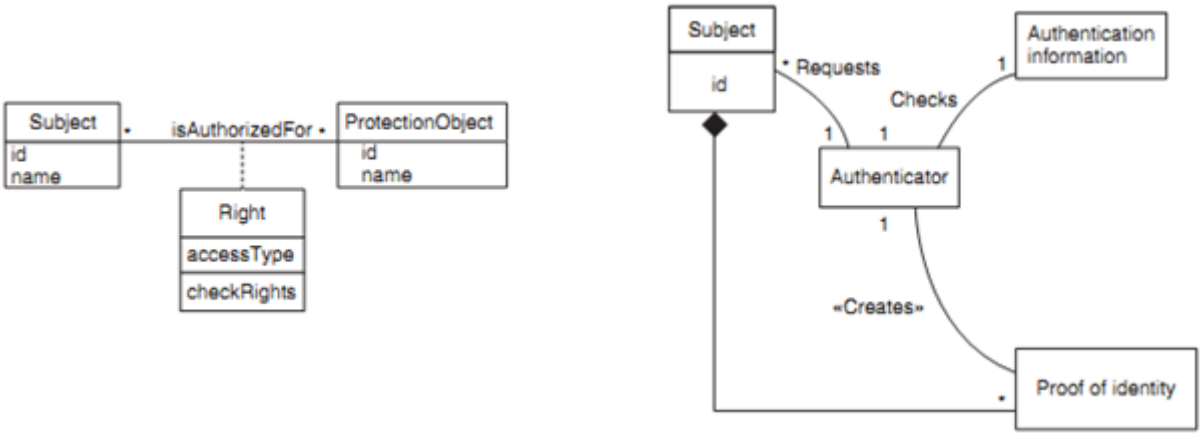


Figura 7 - Padrões *Authorization* e *Authenticator*, respectivamente. Disponível em Schumacher *et al.* (2006).

Desta forma são unificados os padrões inicializando pela classe *Subject* por ser comum aos padrões. Posteriormente tenta-se identificar demais classes semelhantes e por meio de restrições são rechaçadas as possibilidades. As demais

classes são então transformadas e adaptadas ao modelo destino, conforme visualizado no Figura 8.

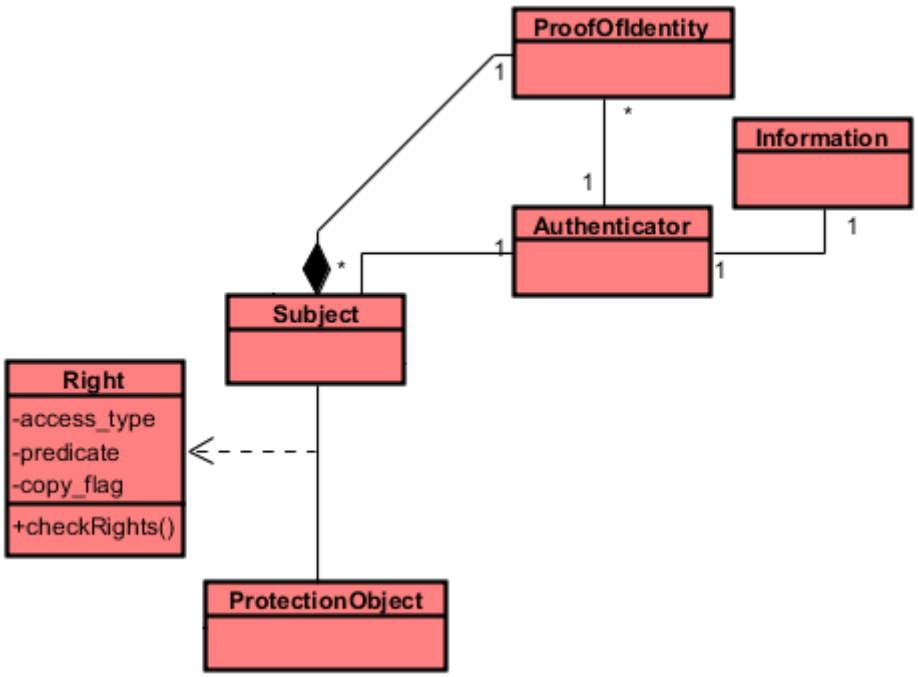


Figura 8 - Unificações dos Padrões *Authorization* e *Authenticator*.

Os padrões de segurança adotados têm como objetivo intermediar o acesso dos usuários à informação, com base nas atividades que são por eles desenvolvidas no sistema. A ideia central é que o usuário desempenhe diferentes papéis (*Roles*) em um sistema. Um papel pode ser definido como um conjunto de atividades e responsabilidades associadas a um determinado cargo ou função em uma organização. Assim, as permissões são conferidas aos papéis e os usuários são autorizados a exercer tais papéis. O controle de acesso facilita a gerência de autorização, pois quando o usuário muda de atribuição – sendo, portanto, desassociado de um papel e assumindo outro – a manutenção das permissões dos papéis não sofre mudanças. Em geral, trabalha-se com o princípio do mínimo privilégio, onde um usuário ativa apenas o subconjunto de papéis que precisa para executar uma operação, e esta ativação pode ou não estar sujeita a restrições.

A *Interface* representa uma classe do sistema que está associada a um caso de uso, que é representado pela classe *System* que representa qualquer artefato

que possua acesso controlado, ou seja, pode ser um caso de uso, uma tela, um campo ou botão que pertence a uma determinada tela, uma operação, etc.

Os modelos de segurança apresentados acima são a entrada para uma transformação de modelo para modelo, cujo resultado é um modelo de plataforma de destino que está de acordo com um metamodelo que representa uma tecnologia alvo. O código de segurança da aplicação final é gerado a partir desse modelo. Haverá um modelo de plataforma de destino, e, portanto, um modelo de projeto de segurança e um modelo de implementação de segurança, para cada aspecto de segurança considerada para a geração de software.

Os mecanismos que fazem parte do controle de acesso implementam políticas que permitem ao usuário atribuir direitos de acesso sobre seus recursos computacionais de acordo com a sua necessidade. Tais mecanismos baseiam-se na ideia de que o proprietário da informação deve determinar quem terá acesso a essa informação. O controle permite que os dados sejam livremente copiados de objeto para objeto, de modo que, mesmo que o acesso aos dados originais seja negado, pode-se obter acesso a uma cópia. Porém, se o usuário não atribuir corretamente estes direitos, ou mesmo se o fizer permitindo acesso de cópia a outros sujeitos, a disseminação de suas informações no sistema não pode ser controlada. O controle de acesso não impõe nenhuma restrição à disseminação de direitos e à própria evolução da matriz de acesso.

Nesse contexto, pode-se perceber que a grande mudança ocorre quando a segurança é incluída, pois nesse momento, na transformação do PIM para o PSM, os novos requisitos de segurança requerem que um novo PSM seja gerado incluindo os padrões de segurança. Consequentemente, a transformação de PSM para código também é alterada devido ao fato da criação de toda infraestrutura de segurança do sistema. Sendo assim, esse modelo contribui na integração de segurança durante o desenvolvimento do sistema, simplificando o desenvolvimento, aumentando a produtividade e a qualidade das especificações de segurança e, com isso, acrescentando consideravelmente a qualidade e a manutenibilidade dos sistemas construídos. A separação entre a implementação do negócio e a solução de segurança, facilita a evolução da solução fazendo com que ela não se torne obsoleta, principalmente pelo fato da segurança poder ser gerada automaticamente.

5.3 Arquitetura e processo

O *framework* de segurança é dividido em dois módulos, um responsável pelo conjunto de padrões de segurança e outro pelas regras de transformação. O módulo de conjunto de padrões de segurança representa o modelo de segurança do cliente, que independe da política de controle de acesso da organização. Visando facilitar o desenvolvimento da aplicação, uma relação de segurança é gerada automaticamente considerando uma política de segurança padrão, de forma que só é necessário que a aplicação tenha que criar sua própria aplicação.

O módulo de regras de transformação é responsável pelas regras adaptadas pelos padrões escolhidos. A segurança é aplicada implicitamente no sistema através de transformação entre modelos e codificação automática. Assim, o processo de modelagem das regras de negócio não sofre alterações, o engenheiro deve focar somente no negócio, sem se preocupar com a segurança. Os requisitos de segurança serão acrescentados ao sistema de forma automatizada a partir das regras de negócio, através dos mecanismos oferecidos pelo *framework*.

Pode-se considerar o módulo de regras de padrões de segurança como o núcleo dessa abordagem, pois ele contém todas as regras de transformação definidas para aplicação dos padrões selecionados, através dos requisitos de segurança levantados inicialmente.

A segurança é gerada automaticamente através de transformações entre modelos. O fato de ser gerado automaticamente provê uma liberdade ao arquiteto de segurança à medida que ele pode alterar o que é gerado, a qualquer o momento, sem influenciar ou prejudicar o desenvolvimento do sistema. Em caso de troca de plataforma, o arquiteto apenas precisa implementar a mesma solução para outra plataforma, ou seja, mapear as regras de transformação já definidas para a plataforma escolhida.

5.4 Execução e definição de transformações

Nesta seção se discute os aspectos relacionados com a execução de definições de transformação ATL. Primeiro, apresenta-se um algoritmo em

pseudocódigo para a execução de transformações. Em seguida, discutem-se algumas questões de otimização e são fornecidas informações sobre a rescisão e determinismo de transformações ATL.

O algoritmo da Listagem 1 apresenta os conceitos básicos da semântica da linguagem procedural ATL. Este algoritmo é utilizado pelo mecanismo de transformação. Assume-se que as definições de transformação possam conter combinações e chamadas às regras. O algoritmo apresenta apenas a execução de regras. A manipulação de *helpers* não está incluída.

```
1. execute called rule marked as entrypoint
2.
3. --Combinar regra ao padrão
4. foreach standard rule R {
5.   --Um padrão candidato é um conjunto de elementos que
6.   -- correspondem aos tipos do padrão de uma regra
7.   foreach candidate pattern C of R {
8.   evaluate the guard of R on C
9.     if guard is true then
10.       create target elements in target pattern of R
11.       create TraceLink for R, C, and target elements
12.     else
13.       discard C
14.     endif
15.   }
16. }
17.
18. --Aplicar regra ao padrão
19. foreach TraceLink T {
20.   R = the rule associated to T
21.   C = the matched source pattern of T
22.   P = the created target pattern of T
23.   --Inicializar elementos no padrão alvo
24.   foreach target element E of P {
25.     --Iniciar cada característica de E
26.     foreach binding B declared for E {
27.       expression = initialization expression of B
28.       value = evaluate expression in the context of C
29.       featureValue = resolve value
30.       set featureValue to corresponding feature of B
31.     }
32.   }
33.   execute action block of R in the context of C and T
34.   --Blocos Imperativos podem executar qualquer navegação em C
35.   -- ou T e qualquer ação de T.
36.   --Para realizar operações somente válido.
37. }
38. execute called rule marked as endpoint
```

Listagem 1 – Algoritmo para execução de transformações.

Este algoritmo começa com a execução de uma regra opcional chamada como ponto de entrada. Esta regra, por sua vez, pode invocar outras chamadas de regras. Em seguida, o algoritmo prossegue com a execução das regras do padrão correspondente no programa de transformação (algumas delas podem conter um bloco de ação). A adequação de regras e aplicação é separada. Isto não é absolutamente necessário. No entanto, é mais simples para descrever e aplicar um algoritmo de duas fases porque cada *TraceLink* está disponível após o primeiro estágio. Por isso, a facilidade de resolver os elementos-alvo previsto na segunda etapa. Elementos-alvo poderiam, no entanto, serem inicializados como eles são criados. Isto faria com que o algoritmo de resolução seja mais complexo, uma vez que algumas inicializações poderão ser adiadas até o fim da transformação.

Esse algoritmo não supõe qualquer ordem, correspondência de regra, criação de elementos-alvo para uma correspondência, inicialização alvo elementos para um *TraceLink* e inicialização de características de um elemento destino. A ação de Bloquear (se houver) deve, no entanto, ser executado após ter sido aplicado a parte declarativa da regra. Isso facilita a tarefa do programador já que o padrão-alvo está em um estado um pouco previsível (usado apenas quando as construções declarativas não são suficientes).

De fato as restrições de ordem podem ser realizadas de forma menos restritivas. Por exemplo, todas as características do elemento de destino realmente podem ser inicializadas em qualquer ordem, enquanto o algoritmo dado reforça que todas as características de um determinado elemento destino são inicializadas em blocos e que todos os elementos alvo de uma *TraceLink* são inicializados em bloco. Isto não é absolutamente necessário.

A especificação de transformação pode ser dividida em duas partes lógicas. A primeira parte executa a decoração do modelo de fonte e a segunda parte contém as regras de transformação real. Desde que a especificação da transformação seja declarativa, não há ordem de execução explícita entre essas partes.

A parte de decoração é composta por um conjunto de atributos *helpers*. Eles são atributos adicionais de elementos de origem, modelos atribuídos durante a transformação.

Atributos *helpers* são usados para as seguintes tarefas:

- Para determinar os atributos e associações para cada classe, incluindo os herdados pela superclasse.

- Para determinar o “pai” mais alto de uma determinada classe. Este *helper* é utilizado quando um atributo/associação tem como tipo uma classe não persistente cuja classe pai é persistente;
- Para nivelar hierarquias de herança com a raiz de uma classe persistente. Neste caso todos os atributos e associações das classes diretas e indiretas estão unidas e unificadas com a classe raiz persistente.
- Para nivelar as características (atributos ou associações de saída) de classes, realizando o algoritmo para lidar com os atributos de tipos não primitivos.

Os atributos e associações de saída de uma classe são tratados de maneira uniforme, referindo a eles como *features*. Para cada recurso que aplicar as regras para derivar colunas da tabela. Se o recurso é do tipo primitivo uma única coluna será criado. Se o recurso é de uma classe não persistente, em seguida, que resulta em um conjunto de características primitivas com acúmulo recursivo de nomes. Se o recurso é da classe persistente (ou o pai de topo é uma classe persistente), em seguida, obtêm-se as características principais desta classe.

O objetivo do *helper* é associar um conjunto de características primitivas para cada classe da qual colunas são diretamente derivadas. Uma vez que algumas destas características são resultadas de um nivelamento, eles são de fato uma sequência de recursos derivados de acordo com o algoritmo de *drill-down*, chamados para rastrear uma sequência. Portanto, associa-se um conjunto de rastreamentos para cada classe e criar uma coluna de todos os rastreamentos de uma classe persistente. Os nomes das colunas são formados como uma concatenação dos nomes das funcionalidades nos rastreamentos.

A ideia por trás desta funcionalidade é gerar rastreamento de características além de algumas informações adicionais sobre os rastreamentos. A estrutura que detém esta informação é uma tupla com os seguintes campos:

- *trace* - contém o rastreamento como uma sequência de recursos;
- *isPrimary* - indica se o rastreamento gera uma coluna principal;
- *isForeignKey* - indica se o rastreamento termina com uma característica de um tipo persistente;
- *ForeignKeyColumns* - se o campo *isForeignKey* for verdadeiro, então este campo contém uma sequência de rastreamentos que serão utilizados para

produzir as colunas da chave estrangeira. Esses rastreamentos são cópias dos rastreamentos da coluna principal da classe persistente para o qual a chave estrangeira é gerada.

A lógica é começar com uma lista inicial de recursos para uma classe e construir uma sequência de tuplas em conformidade com a estrutura descrita através da aplicação de nivelamento de forma recursiva. Se a classe é uma classe persistente, a lista inicial de recursos é a união dos valores de atributos e associações da classe. Se for é uma classe não persistente, que não herda de uma classe persistente, a lista inicial é a união de valores de todas as associações e atributos.

Em seguida inicializa-se o *slot* da tabela. O valor desse *slot* é uma coleção de todas as colunas da tabela. Colunas são criadas a partir de marcas que não representam chaves estrangeiras. O valor da expressão de inicialização é um conjunto de rastreamentos, ou seja, uma série de sequências de elementos do modelo fonte. Portanto, este valor deve ser resolvido de acordo com o algoritmo de resolução ATL. A resolução exige encontrar uma regra que transforma o valor da expressão em elementos do modelo alvo. Assim, neste recurso de inicialização, tem-se uma invocação implícita de uma regra de transformação, que transforma rastreamentos em colunas.

Além disso, o *slot chave* contém todas as colunas primárias da tabela. Colunas primárias são um subconjunto do conjunto de todas as colunas da tabela e, também são geradas pela regra. Da mesma forma que o *slot* anterior, tem-se uma invocação implícita da regra, ou seja, a mesma regra pode ser acionada várias vezes sobre a mesma fonte.

5.5 Método e ferramenta de transformação genérica

Nesta seção, apresenta-se em detalhes uma ferramenta que executa a união do modelo de origem com o modelo de padrões de segurança, descrito na Seção 5.1 e um método para a definição e verificação das regras de transformação aplicáveis ao modelo destino. Deve, portanto, ser aplicável a dois modelos, um suposto modelo de origem e outro modelo alvo da transformação ou o refinamento,

definido ou obtido a partir de uma grande variedade de ferramentas ou quando o engenheiro de segurança intervir manualmente em modelos.

São definidas duas técnicas para escrever uma transformação, e mais precisamente as restrições sobre a evolução dos elementos entre o modelo de origem e o modelo de destino. Restrições são expressas em um único contexto, é necessário, então, ser capaz de manipular ambos os modelos a partir de um único contexto.

A especificação da operação de transformação para este problema deve conectar a operação de transformação para um elemento do modelo e especificá-lo em ATL. O modelo antes da transformação é modelo de origem e, uma vez que a transformação tem sido executada, o modelo modificado pela operação é o modelo alvo. Dentro da pós-condição, pode-se referir aos elementos do modelo alvo e do modelo de código através do operador *@pre*. Assim, a pré-condição permite que a parte relativa ao modelo de origem possa ser escrita e a pós-condição contenha as outras duas partes (no modelo de destino e sobre a evolução do elemento). Como exemplo dessa interface, pode-se observar a Listagem 2.

```
1. contexto ModelBase::addAllInterfaces()
2. pre: --restrições ao modelo de fonte
3. post:
4. --restrições ao modelo de destino e
5. --a evolução dos elementos entre modelos de origem e destino
6. allClasses -> size() = allClasses@pre -> size() and
7. ...
```

Listagem 2 – Algoritmo para aplicação da transformação.

A operação de transformação é chamada *addAllInterfaces* e está anexada ao *ModelBase* do meta-elemento. A pós-condição da operação verifica, particularmente, se o número de classes após a transformação é a mesma de antes. Os elementos unidos antes da transformação (do modelo de origem) e elementos após a transformação (do modelo alvo) podem ser manipulados.

No contexto da plataforma Eclipse/EMF, foi desenvolvida uma ferramenta que realiza, automaticamente, a concatenação de modelos. A ferramenta tem como entrada o modelo de origem e o modelo destino. Em uma primeira etapa, adiciona-se ao modelo uma classe *ModelReference* que contém um atributo *string* chamado

modelName. Todas as classes são modificadas para especializar esta nova classe. Em uma segunda etapa, a ferramenta adiciona todos os elementos do modelo de origem e todos os elementos do modelo destino em um terceiro modelo (unificado). Durante esta etapa, cada elemento é marcado através do atributo *modelName* como "fonte" ou "destino", dependendo do modelo a que pertence. Como saída, a ferramenta retorna o modelo modificado e o modelo global contendo todos os elementos de ambas as fontes e modelos de alvo com indicação da sua origem.

Como evidenciado, a transformação é composta de três partes: as limitações do modelo de origem, as restrições do modelo de destino e as restrições sobre a evolução dos elementos entre o modelo de origem e o modelo destino. As duas primeiras partes são simples de expressar, é necessário definir invariantes associadas com o modelo, o que gera um caminho comum. Para o exemplo de adição de interface, o modelo origem é qualquer tipo de diagrama de classes sem restrição em particular, pois, nenhuma restrição sobre o modelo origem deve ser definida. Para o modelo destino, cada classe do diagrama tem que implementar pelo menos uma interface.

Uma regra ATL pode então ser usada para verificar se esta restrição é validada pelo modelo destino. A parte relacionada à evolução entre os dois modelos é mais complexa para se estabelecer, pois está associada com o modelo global, que concatena o modelo de origem e o de destino dentro de um único modelo. Em seguida, é necessário determinar referências sobre o modelo de origem e o de destino e seus elementos. Isso é realizado com todos os elementos do modelo global, marcando-os para indicar a qual modelo eles pertencem. Do mesmo modo, é necessário expressar que um elemento em um modelo é equivalente a outro elemento sobre o novo modelo.

No mapeamento entre elementos de origem e de modelos, o objetivo principal sobre a evolução dos elementos entre a origem e os modelos de destino, consiste em verificar que cada classe do modelo destino implementa, diretamente ou por meio de sua interface, os mesmos métodos da classe equivalente no modelo fonte. Para essa validação, é necessário determinar qual sua classe equivalente. De uma forma mais geral, é necessário ser capaz de definir que um determinado elemento em um modelo tem ou não um elemento do mesmo tipo, e ser capaz de obter este elemento no mapeamento existente.

Esses mapeamentos entre os elementos são implementados em ATL por uma série de funções. Dependendo da complexidade do modelo, o número de mapeamentos necessários entre os elementos e o número de funções pode ser elevado e cansativo para se realizar manualmente.

Este problema é facilmente evitável por meio de uma ferramenta dedicada capaz de gerar automaticamente estas funções para um dado modelo. Ou seja, essas funções são sempre baseadas na mesma estrutura, verificando o mapeamento de atributos e de referência de um elemento em uma maneira transitiva. Para obter uma lista de referência ou atributo, o mapeamento dos elementos da lista é verificado um por um. Para sanar esse problema a ferramenta implementada analisa qualquer modelo e propõe para o *designer* a livre escolha, para cada tipo de elemento do modelo, o tipo de mapeamento desejado. A ferramenta gera, então, todas as funções de mapeamento ATL. Na Figura 9 pode ser visualizada a ferramenta no contexto das escolhas de mapeamento para o elemento de classe.

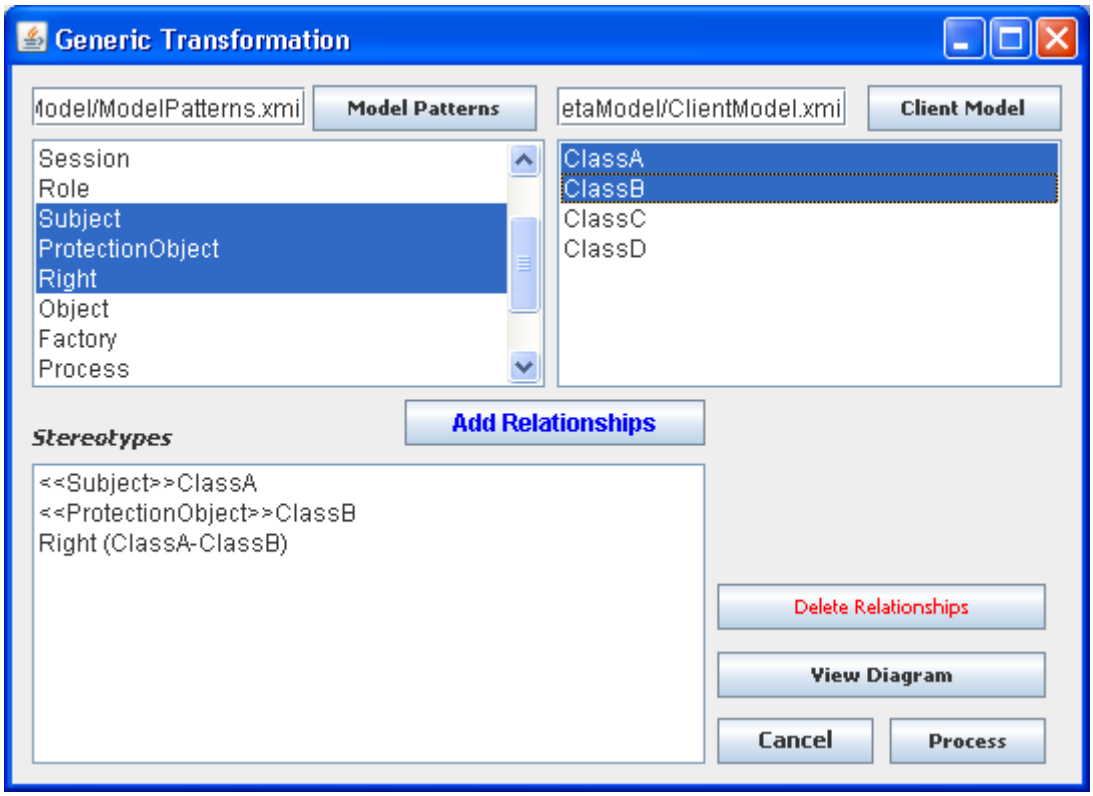


Figura 9 - Ferramenta para unificação de modelos e transformações genéricas

Esta ferramenta de transformação não somente verifica e define os métodos que são mantidos para cada classe, mas também assegura que elementos chaves não sejam modificados durante a transformação. Ou seja, a transformação é validada se todas as classes são conservadas com o mesmo nome e com a mesma lista de atributos. Também se verifica se todos os métodos permanecem sem modificação de suas assinaturas (através das chamadas dos métodos). Isso garante um escopo bem definido para as modificações manuais do modelo que porventura ocorram.

5.6 Aplicação do modelo a transformações

Para aplicar o modelo desenvolvido adotam-se as transformações modelo-modelo do processo MDA. Esta categoria realiza a transformação de um modelo de entrada (UML), em um modelo de saída (XMI - *XML Metadata Interchange*). Uma analogia pode ser realizada entre transformações gerais e classes genéricas (ou *templates*) utilizadas em linguagens orientadas a objetos. Assim, enquanto os parâmetros das classes genéricas estão ligados em tempo de projetos, os tipos variáveis em regras de transformação são substituídos somente em tempo de execução (YOSHIOKA *et al.*, 2004).

O objetivo é demonstrar como as regras de transformações da abordagem MDA podem ser especializadas para que seja chamada, neste trabalho, de *framework*. Nesta abordagem, as linguagens de modelagem de uso geral e funções de transformação são estendidas pelas regras de transformação para integrar a segurança no processo de desenvolvimento (SELIC, 2005). Estas medidas são usadas para especificar as propriedades de segurança do sistema de destino e gerar mecanismos de segurança correspondente.

A fim de descrever as regras necessárias para a geração do *framework* proposto, utiliza-se como linguagem de desenvolvimento a ATL, que é uma linguagem para a realização de transformação de modelos para modelos no contexto de MDA. A Figura 10 ilustra as etapas de concepção e utilização do *framework* com os requisitos do modelo de padrões de segurança, onde se tem os seguintes passos:

1. O modelo baseado em segurança mostrado na Figura 6, descreve um esquema geral para a integração dos requisitos de padrões de segurança em modelos de classe.
2. Apresenta uma modelo de classes de um sistema na qual se deseja inserir segurança.

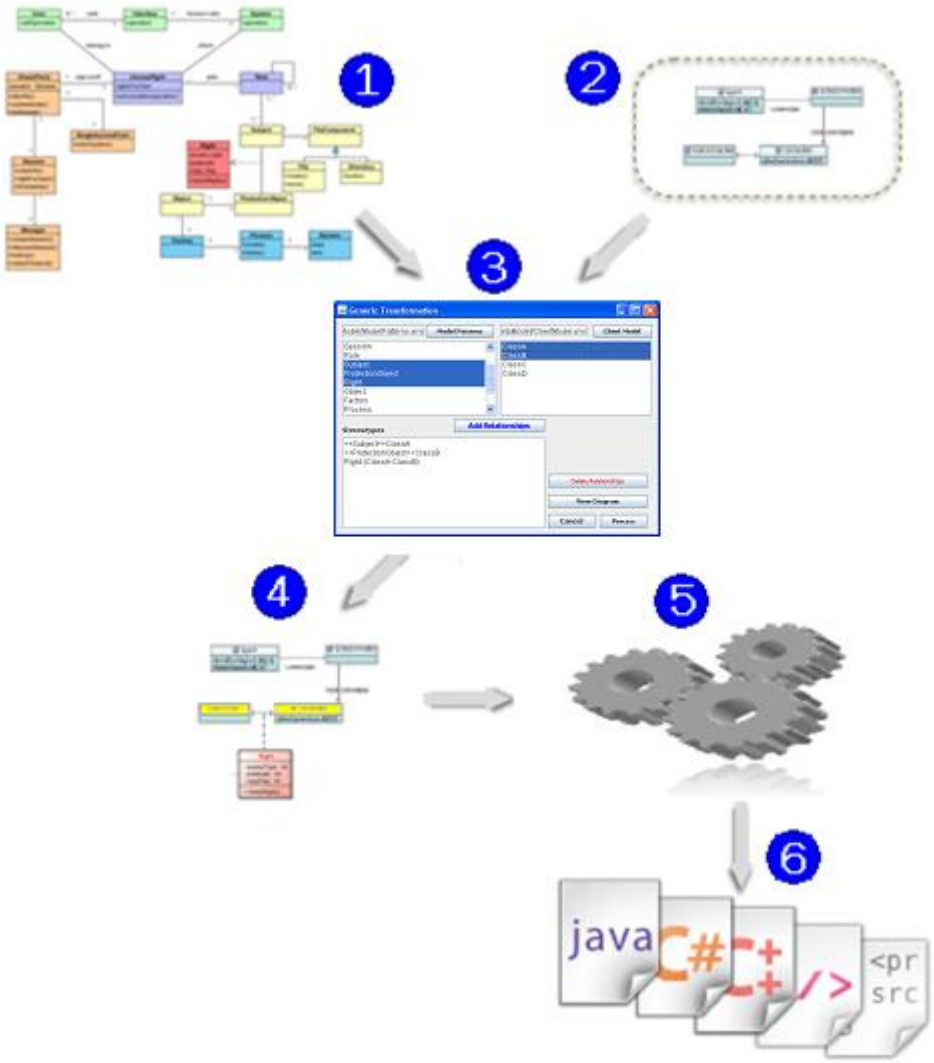


Figura 10 - Etapas de aplicação do *framework*.

3. A ideia principal é adicionar linguagens de modelagem de segurança específicas aos recursos que se deseja proteger. Nesta etapa, especialistas em segurança devem determinar as dependências entre os padrões referindo-se a seção do modelo de padrões de segurança. Além disso, existe a possibilidade de que a dependência poderia ser

verificada atendendo o problema e o contexto do padrão aplicado, e também para o contexto, a solução, e as consequências de todos os demais padrões. Desta forma, o sistema de transformação (em ATL) utiliza um modelo de entrada, descrito como diagrama de classe e, gera um modelo de saída (XMI).

Cada padrão de segurança é aplicado de forma iterativa, fazendo uma transformação no modelo, ou seja, o desenvolvedor seleciona o(s) padrão(ões) de segurança e as entradas do modelo e os parâmetros e o sistema transforma o modelo e as saídas do modelo com o padrão de segurança escolhido. Para fazer a transformação de PIM para PSM, devem ser tomadas decisões de projeto. Tais decisões podem ser tomadas durante a etapa de projeto, que ajusta os requisitos de engenharia na implementação. Esta é uma forma útil, porque estas decisões são tomadas no contexto de um projeto de implementação específico.

4. Apresenta as mesmas propriedades, atributos, funções, etc., do item 2, descrito acima, agregado pelos padrões de segurança selecionados. Para cada relacionamento criado tem-se a marcação de um estereótipo do padrão escolhido. Como exemplo toma-se o padrão de *Authorization* onde se adiciona o estereótipo *ProtectionObject* para a classe que correspondente ao papel do objeto de proteção para ser utilizado pelo desenvolvedor como um argumento de entrada. Adiciona-se, ainda, um estereótipo para a classe que corresponde ao papel *Right* e o relacionamento entre as classes *Subject* e *Right*. Nesta etapa o modelo é completo quanto à classificação, estrutura, invariantes e pré/pós-condições. O desenvolvedor pode especificar os requisitos diretamente no modelo, usando uma linguagem de ação. Isto faz do PIM computacionalmente completo, ou seja, este PIM contém todas as informações necessárias para produzir um código de programa. Neste contexto não é necessário que o desenvolvedor veja o PSM, nem é necessário que seja adicionada informação ao PIM.
5. A partir desta etapa e através de uma ferramenta de livre escolha do usuário, tais como AndroMDA, xUML, OptimalJ (MAIA, 2006), ocorre a interpretação o modelo diretamente ou transforma o modelo

diretamente no código-fonte do programa. É a partir do PSM que será gerado o código fonte, onde o trabalho de gerá-lo fica a cargo de uma ferramenta que lê o PSM e escreve o código.

6. Nesta fase tem-se a geração do código fonte para uma linguagem previamente escolhida.

6 ESTUDO DE CASO

O objetivo deste capítulo é apresentar a forma como foram dirigidos os experimentos para a avaliação da abordagem proposta neste trabalho. O estudo de caso tem como principal objetivo validar o *framework* adicionado ao modelo do cliente e verificar as reais vantagens dessa abordagem. Além disso, verificam-se algumas das modificações necessárias ao modelo de entrada para as transformações, devido a requisitos específicos no modelo do cliente.

6.1 Definição do Estudo

O cenário de aplicação implementado a fim de demonstrar a utilização do *framework* proposto neste trabalho é o de comércio eletrônico. A escolha deste cenário deve-se ao fato da sua popularidade na atualidade, bem como a sua importância para a sobrevivência das empresas no mercado atual. Sua implantação tem se tornado um grande diferencial entre concorrentes, uma vez que visa comodidade para o cliente e redução de custos para as empresas.

Sistemas de comércio eletrônico, em geral, incluem inúmeros artefatos: clientes, servidor *web*, servidor de negócios, sistema de gerenciamento de banco de dados, *gateway* de pagamento, dentre outros. Um cliente geralmente utiliza um navegador para se conectar ao servidor *web*. Este, então, se comunica aos demais aplicativos que podem conectar qualquer outro serviço, como o sistema de gerenciamento de banco de dados para uso da informação ou o *gateway* de pagamento para obter informações de pagamento. Na Figura 11 pode ser visualizado o diagrama de classe da parte de fechamento de um pedido ou compra.

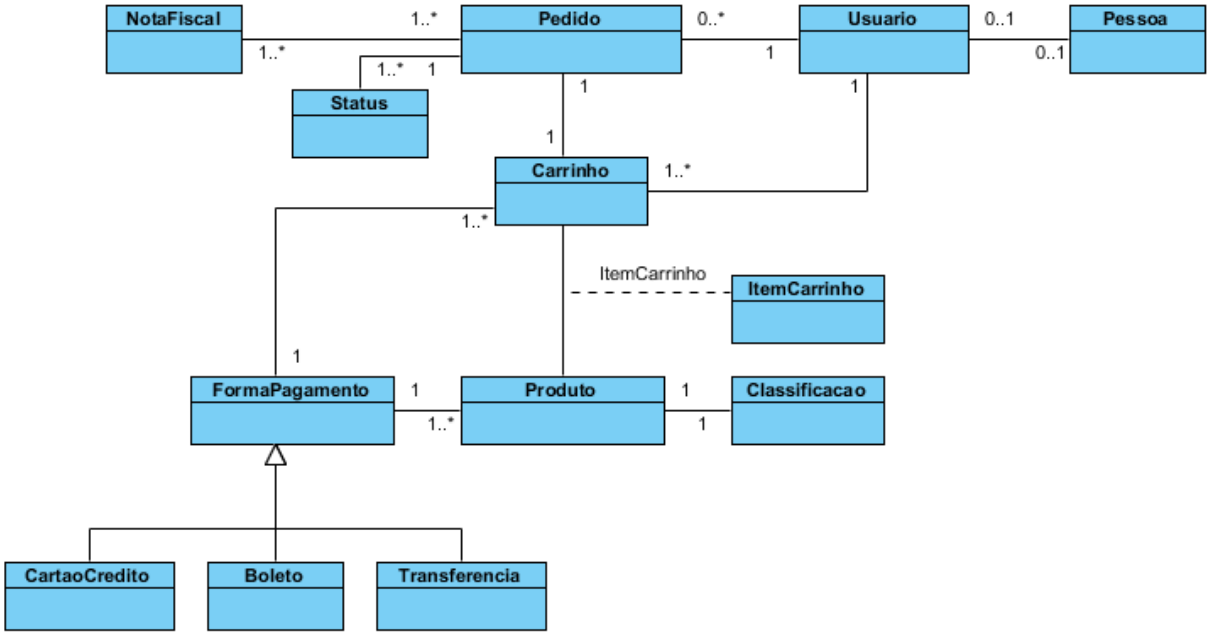


Figura 11 - Diagrama de classe do estudo de caso.

6.2 Padrões aplicados

Ameaças à segurança tornaram-se quase uma rotina nos dias de hoje, portanto, era de se esperar que, no ambiente da internet, isso não fosse diferente. No ambiente virtual também é necessário se proteger de ocorrências como a clonagem do site, o roubo de senhas, o acesso a informações sigilosas trocadas entre o site e o visitante, entre outras.

A partir da descrição do sistema devem ser identificadas as necessidades de segurança a serem aplicadas ao projeto. Caso alguma necessidade de segurança passe despercebida, poderá causar uma falha de segurança no sistema possibilitando o uso indevido das suas funcionalidades.

Partindo para a parte do pagamento via cartão, que é o meio mais prático de pagamento no *e-commerce*, também é o que desperta maior receio por parte do consumidor *on-line*. O risco de expor os dados do cartão a terceiros é a grande preocupação de muitos clientes que, mesmo possuindo o cartão, preferem fazer pagamentos usando o boleto bancário ou transferência *on-line*, o que demanda mais trabalho para quem compra e para quem vende, além de resultar em maior demora na entrega da mercadoria adquirida.

As administradoras de cartão vêm aprimorando os sistemas de segurança da informação no processo de pagamento *on-line*, de forma a tornar o uso do cartão o mais seguro possível, o que é plenamente justificável, uma vez que a segurança na internet é fundamental para maior confiança do consumidor nas compras *on-line* e, consequentemente, para o lucro dessas empresas nesse canal de comercialização.

É provável que, da mesma forma que não se tem um mundo totalmente seguro, nunca se tenha uma internet da mesma forma segura. No entanto, se cada engenheiro de segurança se preocupar com a segurança de sua loja virtual e tomar medidas preventivas como as abordadas a seguir, minimizará consideravelmente os riscos, levando-os para um nível de segurança plenamente aceitável.

Segundo a descrição do software de comércio eletrônico, foram identificadas necessidades e características para suprir as vulnerabilidades do sistema no que tange o processo de pagamento de um pedido. Para suprir estas necessidades de segurança do sistema e com o intuito de aplicar o *framework* proposto, foram selecionados alguns padrões e seus detalhamentos para exemplificação das funcionalidades de segurança.

Single Access Point - fornece um módulo de segurança e uma forma de se logar no sistema. Ao limitar a entrada de aplicativo para um único ponto, impede que os usuários obtenham acesso a outras entradas no sistema para visualizar ou editar os dados. Fornece um local seguro para validar os usuários e coletar informações globais. Tem-se um sistema que precisa ser acessado por usuários e é necessário ser capaz de controlar como se interage com ele.

Check Point - encapsula o algoritmo para uma política de segurança da organização e da política de segurança pode ser alterada em um único lugar. É um local de segurança crítica, onde a segurança deve ser absolutamente forçada. Este padrão é dividido em duas partes conceituais: autenticação e autorização. A autenticação é realizada quando um usuário começa a usar o sistema e autorização é necessária sempre que um usuário tenta usar uma operação dita segura.

Session - titular de todas as variáveis que precisam ser compartilhado por vários objetos. Ele é utilizado em torno de objetos que precisam de seus valores a qualquer instante. Algumas dessas informações estão relacionadas à segurança, tais como o papel do usuário e privilégios.

Authenticator - é o processo de identificar individualmente os clientes de suas aplicações e serviços. Estes podem ser usuários finais, outros serviços, processos

ou computadores. Em termos de segurança, os clientes autenticados são chamados de diretores. É ainda considerado um recurso de segurança primária, pois os mecanismos usados para autenticação frequentemente influenciam os mecanismos utilizados para permitir outros recursos de segurança, tais como a confidencialidade dos dados e autenticação da origem dos dados.

Authorization - é o processo que gerencia os recursos e as operações que o cliente autenticado tem acesso permitido. Os recursos incluem arquivos, bases de dados, tabelas, linhas e assim por diante, com nível de sistema de recursos, tais como chaves de registro e dados de configuração. As operações incluem a realização de transações como a compra de um produto, transferência de dinheiro de uma conta para outra, ou o aumento de notação de crédito do cliente.

6.3 Execução de transformações modelo-modelo

Ao utilizar o aplicativo para aplicar o *framework*, o usuário pode elaborar o modelo da aplicação na ferramenta *CASE* de sua preferência, desde que a ferramenta escolhida possibilite a exportação e importação de modelos em formato XMI. Para o exemplo demonstrado neste estudo de caso utilizou-se a *AstahCommunity*® (ASTAH, 2011).

Para realizar a transformação, os elementos do modelo de entrada (PIM) que serão mapeados em elementos do modelo de padrões de segurança (também um PIM), devem ser marcados de acordo com os critérios definidos pelos buscadores, definidos na seção 5.4, no momento da definição da transformação. Essa marcação pode ser realizada na ferramenta CASE de origem ou através do aplicativo deste estudo (Figura 12).

Através do aplicativo, o usuário pode selecionar elementos (classes) do modelo de destino e seus respectivos relacionamentos com o conjunto de padrões de segurança, para a aplicação dos estereótipos. A vantagem da utilização deste *plug-in* está na facilidade proporcionada ao usuário, que pode selecionar múltiplos elementos para serem marcados de uma só vez.

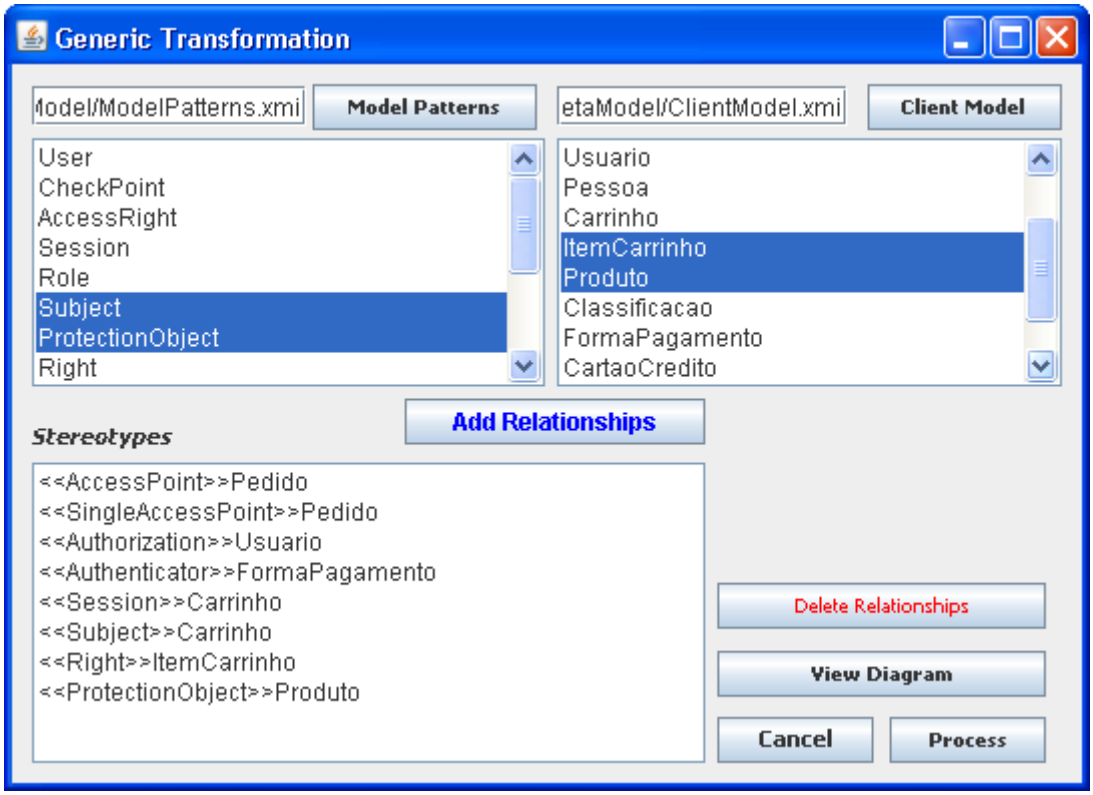


Figura 12 - Uso da ferramenta desenvolvida para realização mapeamentos sobre modelos UML.

Após relacionar os padrões, o engenheiro de software deve aplicar a sua solução proposta para o software. A solução proposta por um padrão de segurança é composta por descrições textuais e gráficas, descritos na seção 5.2. As descrições gráficas são geralmente de alto nível de modelos, como modelos UML, que mostram a estrutura da solução. Na maioria dos casos, estes modelos servem como um modelo para aplicação do padrão, e o engenheiro de software substitui os elementos do padrão com os elementos do software que está a ser concebido.

Após a configuração, a execução da transformação se inicia e as regras de transformação instanciam cada mapeamento, seus mecanismos e realiza a busca por elementos que satisfaçam os critérios definidos nos buscadores. O usuário pode pré-visualizar cada elemento do modelo de entrada e os mapeamentos instanciados para cada um desses elementos, para tanto se deve clicar no botão “*ViewDiagram*”. Na Figura 13 pode ser observada a pré-visualização da execução direta da transformação. Todos os mapeamentos da classe podem ser visualizados na árvore de pré-visualização, onde o usuário pode desmarcar algum mapeamento

indesejado. Completada a operação, os elementos podem ser editados pelo *designer*, visualizado na Figura 14 em destaque na cor laranja claro, através das características da ferramenta de sua escolha, para completar o *design* do software, desde que e ferramenta escolhida tenha suporte ao XMI.

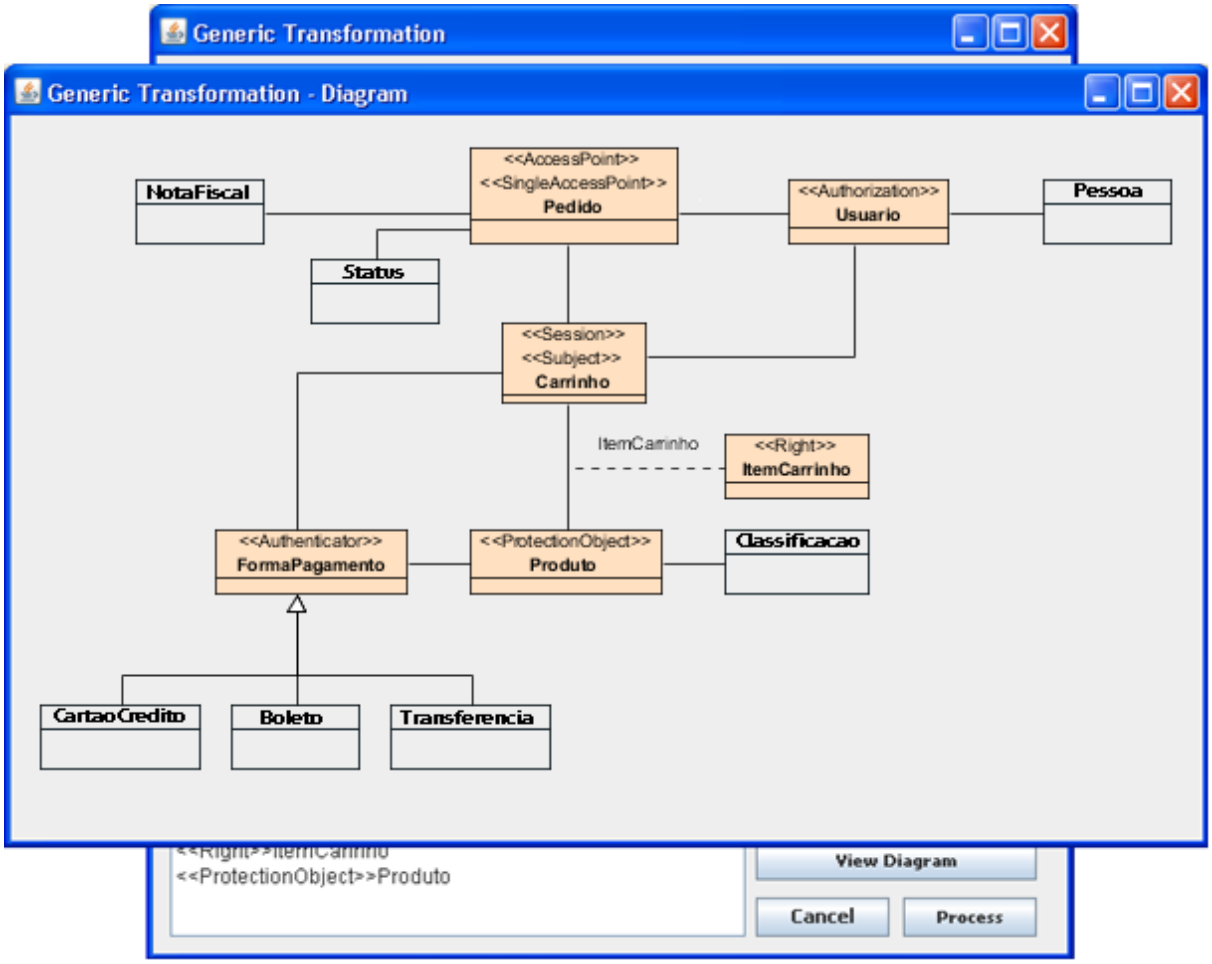


Figura 13 - Pré-visualização da execução da transformação.

Ao clicar no botão "*Process*" ocorre transformação do modelo PIM desenvolvido pelo cliente, com base no conjunto de padrões de segurança descritos na seção 5.2, associados à plataforma da linguagem de programação Java. O processo está dividido em duas etapas. A primeira etapa é a geração da especificação de transformação utilizando apenas a linguagem ATL e a segunda etapa é a geração de especificação de transformação de modelos utilizando a

técnica agrupamento de modelos, onde a ferramenta desenvolvida e a linguagem ATL são utilizadas conjuntamente.

Na criação das regras de transformação existentes tomou-se como base o os padrões de segurança usando ATL para seu detalhamento. Cada uma das regras de transformação consistem em uma pré-condição, um argumento e uma operação. A pré-condição é a suposição da aplicação do padrão de segurança. A definição de uma pré-condição varia de acordo com a presença da existência de um padrão sob o qual pode ser aplicado uma dependência de um padrão de segurança.

Se nenhum padrão no qual a aplicação dos padrões de segurança depende da existência, especialistas de segurança devem definir qual parte corresponde ao papel do padrão de segurança no modelo. Se tal padrão existe, especialistas em segurança devem dispor uma marca no modelo que indica que o padrão existente é uma pré-condição para a aplicação do novo padrão. Essa etapa suporta aplicações consecutivas de padrões de segurança, considerando a dependência entre padrões, definindo a saída da transformação e o modelo anterior como uma pré-condição para a transformação subsequente do modelo.

Por exemplo, para descrever as regras de transformação ao aplicar um padrão de *Authorization*. A pré-condição é que a classe com o estereótipo descrito <<*Authenticator*>> existe. Este estereótipo indica onde o padrão *Authenticator*, que o padrão *Authorization* depende, é aplicado no modelo.

O argumento é um nome de classe que corresponde ao papel do objeto de proteção do padrão de *Authorization*. A operação tem os seguintes passos.

- Adicionar o estereótipo *Authorization*.*ProtectionObject* para a classe que corresponde ao papel de proteção de objeto (classe Carrinho).
- Adicionar a classe que corresponde ao papel *Authorization*.*Right* (classe ItemCarrinho).
- Adicionar a relação entre a classe que corresponde ao papel *Authorization*.*Subject* (classe Produto) e a classe que corresponde ao papel *Authorization*.*Right*.
- Adicionar a relação entre a classe que corresponde ao papel *Authorization*.*ProtectionObject* e a classe que corresponde ao papel *Authorization*.*Right*.

- Remover a relação entre a classe que corresponde ao papel de *Authorization*.*ProtectionObject* e a classe que corresponde ao papel *Authorization*.*Subject*.

Se neste cenário de transformação as relações de correspondência e condição entre os elementos pertencentes ao modelo de destino são semelhantes a alguns dos padrões de segurança descritos no modelo de origem, as regras de transformação serão construídas por estes padrões.

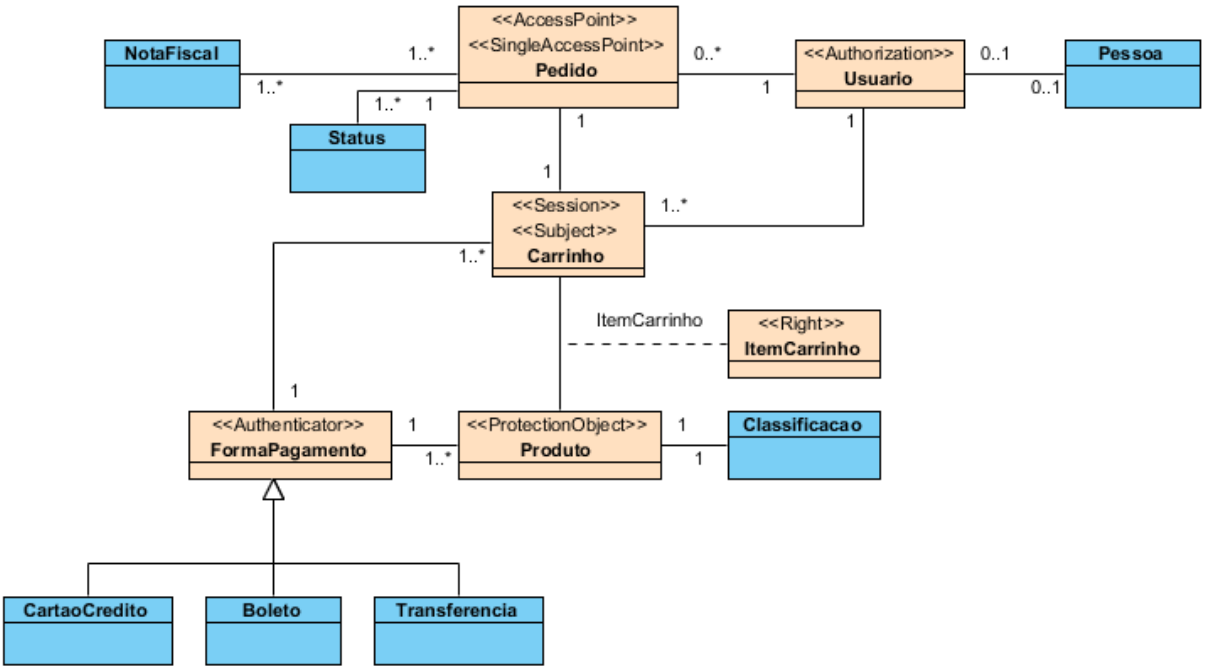


Figura 14 - Diagrama de classe com adição de Padrões de Segurança aplicados para a finalização de um pedido.

Em cada etapa do processo, a regra de transformação suporta a função definida pelo usuário, que pode criar novos passos para adaptá-las através de algumas operações compostas entre as regras de transformação existentes. A fim de melhorar a eficiência de utilização das regras de transformação, há a necessidade de um mecanismo para controlar a ordem extração e qualidade das relações de correspondência. A definição da estratégia de extração é utilizada para implementar o mecanismo da próxima etapa, que consiste em construir não apenas às regras de transformação, mas também as regras da nova transformação, as quais são definidas de acordo com a análise dessas relações de correspondência e condições de transformação, sendo enviadas de forma iterativa ao conjunto de

regras de transformação. Finalmente, através dessas relações de correspondência se constroem as regras de transformação para adaptação do modelo definido pelo usuário.

As alterações realizadas podem ser visualizadas também em outros diagramas da UML, como o exemplo da Figura 15 onde se tem o diagrama de sequência para o fechamento de um pedido de compra de mercadorias. As etapas descritas neste diagrama também sofrem inferências dos padrões aplicados na etapa anteriormente descrita. Na Figura 16 pode ser observada a inserção dos estereótipos vinculados a cada fase do processo descrito.

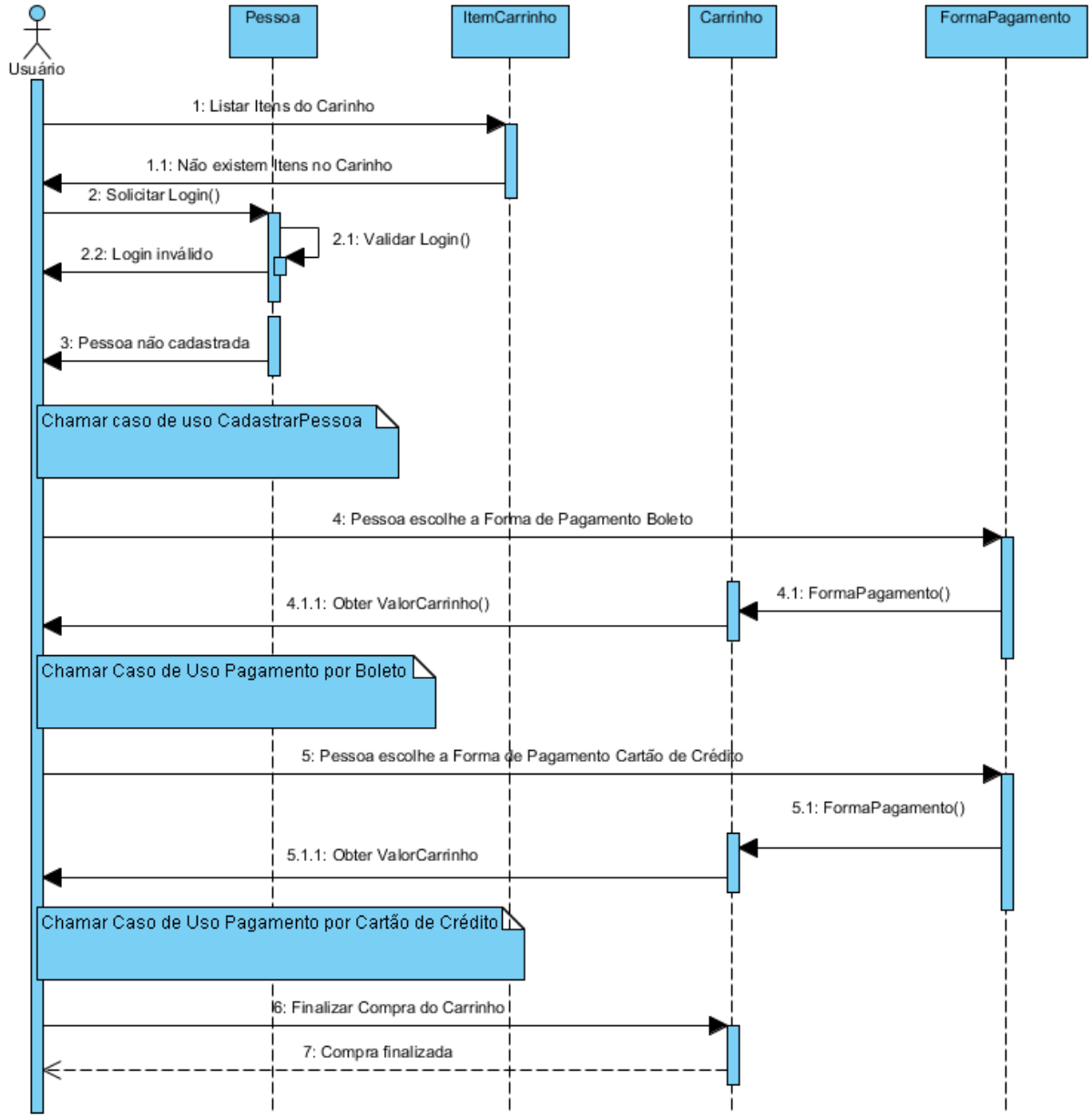


Figura 15 - Diagrama de sequencia para finalizar uma compra.

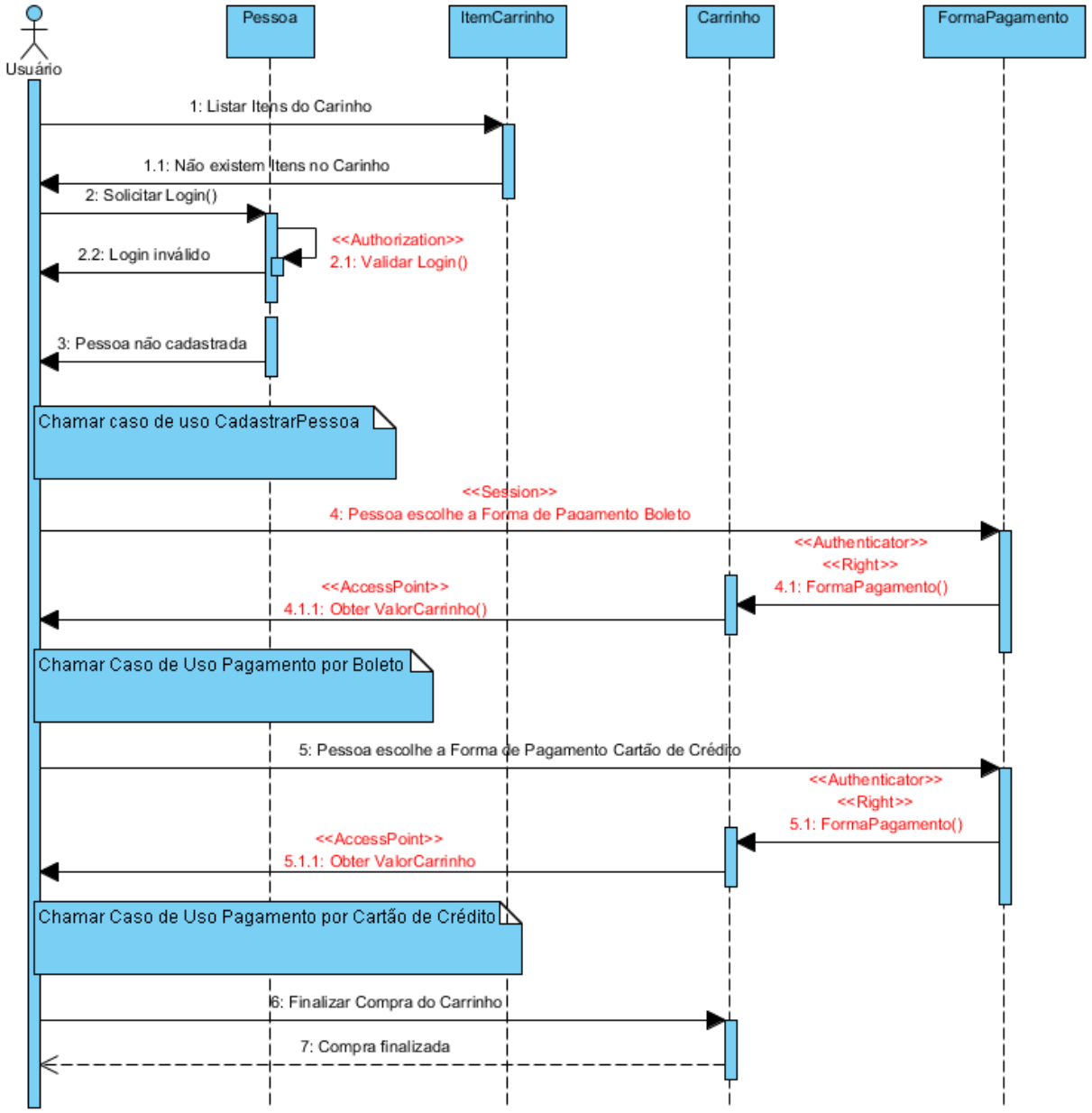


Figura 16 - Diagrama de sequência para finalizar uma compra com adição dos padrões de segurança.

7 COMPARAÇÃO COM TRABALHOS CORRELATOS

7.1 Trabalhos correlatos

Algumas abordagens utilizam MDA para desenvolver estruturas de segurança. Kienzle, Elder, Tyree e Edwards-Hewitt (2006) apresentam uma abordagem MDA para engenharia de segurança que utiliza uma linguagem de modelagem baseada na *Unified Modeling Language* (UML), que integra o Role-*Based Access Control* (RBAC) no processo de desenvolvimento de software dirigido a modelo. A informação de segurança integrada nos modelos UML é utilizada para gerar infraestruturas de controle de acesso. Segurança com MDA provê métodos e ferramentas para integrar segurança no processo de desenvolvimento. A ideia chave é o uso de abstração, construindo modelos visuais que integram o projeto do sistema com o projeto de segurança e utiliza técnicas de geração automática de código para automatizar a construção de sistemas a partir desses projetos. O modelo do projeto é combinado com o modelo de segurança, criando um novo tipo de modelo, chamado de *Security Design Model*. Neste modelo, as políticas de segurança se referem aos elementos do modelo do sistema, como componentes, objetos de negócio, métodos, atributos, etc.

O trabalho de Kienzle, Elder, Tyree e Edwards-Hewitt (2006) é especificamente concebido para controle de acesso e propriedades de autorização. Uma diferença é que o trabalho proposto em referência é mais generalizado para analisar tipos variáveis de propriedade de segurança, desde que ela possa ser expressa desde a modelagem do sistema. Finalmente, se está interessado na segurança de validação da propriedade em especificações abstratas de modelos e não em investigar a geração de código a partir desses modelos. Pode ser possível a utilização de transformações do modelo para refinar o PIM para o PSM e parcialmente gerar o código para o sistema.

Fernandez e Sorgente (2005) propõem o uso de três padrões que correspondem a modelos de segurança: *Authorization*, *RBAC* e *Multilevel Security*. Os métodos permitem que os usuários definam modelos usando a notação UML e

usam a *Object Constraint Language* (OCL) para especificar as restrições. O padrão *Authorization* representa o modelo de matriz de acesso e visa descrever autorizações por entidades computacionais ativas (sujeitos) a recursos passivos (objetos). O padrão *RBAC* visa descrever a atribuição de permissões a usuários de acordo com seus papéis em uma instituição. Já o padrão *Multilevel Security* visa ajudar na decisão de acesso em um ambiente com classificações de segurança, ou seja, ambientes onde informações e documentos possuem níveis de segurança, como por exemplo, secreto, confidencial, etc.

Cunha (2007) apresenta um arcabouço de controle de acesso orientado a modelo onde a segurança é introduzida no sistema através de transformação entre modelos e codificação automática. Desta forma, os requisitos de segurança das aplicações geradas pelo arcabouço se tornam menos suscetíveis a erros ou alteração do código, garantindo a qualidade do sistema gerado. O arcabouço de segurança denominado ArchiMDAs é dividido em dois módulos, um responsável pelas regras de segurança, que é chamado de módulo de regras de segurança e outro responsável pela administração de segurança específica de cada aplicação, denominado de módulo administrativo de segurança. O módulo de regras de segurança é responsável pelas regras de autenticação e autorização dos artefatos da aplicação onde o processo de modelagem das regras de negócio não sofre alterações. O módulo administrativo de segurança representa o modelo de segurança da aplicação, que depende da política de controle de acesso da organização.

Apesar dos trabalhos de Fernandez e Sorgente (2005) e Cunha (2007) consentirem a definição de políticas de controle de acesso utilizando UML, usando modelos de permissões, a segurança do sistema é ativada através da modelagem da política de controle de acesso realizada manualmente. Além disso, a junção dos dois modelos, o de negócio de o de segurança, pode provocar uma sobrecarga de atribuições ao desenvolvedor do sistema, elevando a complexidade do desenvolvimento e, consequentemente, aumentando o risco de incidência de falhas de segurança. Outro ponto negativo é a poluição do modelo de negócio, o que dificulta sua compreensão e manutenção.

Nesta dissertação, o modelo de projeto é combinado com o modelo de segurança, isto é, o PIM de negócio é enriquecido com novos elementos de modelagem representando a política de controle de acesso de uma aplicação.

Assim, o PIM passa a ter elementos de modelagem representando papéis, permissões, etc. Na Tabela 2 podem ser observadas as comparações e divergências desses trabalhos.

Tabela 2 - Comparação entre os trabalhos correlatos

Trabalho	*Patterns*	*Modelo UML*	*MDE*	*Regras de transformação*	*Foco*
A Pattern Language for Secure Operating System Architectures	3 padrões: Autorização, Controle de Acesso Baseado em Papéis e Segurança Multinível.	Não	Não	Não	Sistema Operacional Solaris 10
ArchiMDAs: Um arcabouço de segurança baseado em transformações de modelos	RBAC estendido com a criação de 10 padrões de segurança	Sim	Sim	Sim	Áreas administrativas
Security Patterns Repository, version1.0	26 padrões e 3 mini padrões	Não	Não	Não	Aplicações Web
Trabalho proposto	**26 padrões**	**Sim**	**Sim**	**Sim**	**Qualquer aplicação**

8 CONCLUSÕES, CONTRIBUIÇÕES E PERSPECTIVAS FUTURAS

8.1 Conclusões

Este trabalho apresentou uma forma de transformação de modelos com aplicação de um conjunto de padrões de segurança, motivados pela propriedade desta apresentar uma importante independência em relação a linguagens de transformação de modelos. A técnica permite que sejam estabelecidas ligações, com tipo definido, entre elementos de dois modelos.

Uma das motivações para a realização deste trabalho foi a possibilidade de facilitar a busca de padrões de segurança e a aplicação dos mesmos nas fases iniciais de desenvolvimento. As dificuldades encontradas na utilização de padrões serviram de entrada para a especificação dos requisitos e posterior modelagem do repositório de padrões de segurança.

Apresenta-se técnicas de integração de soluções de segurança nas fases de análise e projeto de sistemas de software, tais como a utilização de padrões de segurança e de uma abordagem orientada a modelos como forma de auxiliar arquitetos e projetistas a construir sistemas seguros.

Segurança compõe um requisito implícito de qualidade, que deve garantir que o sistema esteja protegido em qualquer circunstância. O fator humano pode intervir na segurança em qualquer ponto, ao longo do processo de desenvolvimento, onde a intervenção manual é requerida. Nessas abordagens, a segurança depende da modelagem realizada pelo analista ou engenheiro. Assim, a partir do momento em que a segurança do sistema é ativada através da modelagem da política de controle de acesso realizada manualmente, ela se torna suscetível a erros humanos.

Além disso, a junção dos dois modelos, o de negócio de o de segurança, pode provocar uma sobrecarga de atribuições ao desenvolvedor do sistema, elevando a complexidade do desenvolvimento e, consequentemente, aumentando o risco de incidência de falhas de segurança. Outro ponto negativo é a poluição do modelo de negócio, o que dificulta sua compreensão e manutenção.

Através do desenvolvimento de regras de transformações, construiu-se uma ferramenta de transformação simples e eficiente. Todo o desenvolvimento teve por base a integração de novos utilizadores na definição de transformações de modelos e com isso se colabora também para a evolução do próprio desenvolvimento orientado a modelos.

Por meio do estudo de caso, pode-se verificar que a metodologia para aplicação de padrões de segurança em transformações de modelos traz vantagens como uma maior facilidade na aplicação de padrões de segurança, uma vez que, as regras, depois de codificadas, podem ser reutilizadas; o tempo para implementação de padrões de segurança em aplicações pode ser minimizado, uma vez que parte do código é gerado automaticamente através de regras implementadas.

Para a realização e implementação das regras de transformação, foi necessário uma adaptação visando o suporte da meta-modelagem proposta. Com a adaptação para o suporte de diagramas de classe e objetos UML, foram escritos *templates* em ATL para unificação dos modelos de origem e destino. A aplicação das regras de transformação, a partir de uma modelagem UML, foi completamente integrada ao ambiente de desenvolvimento escrita em um Java.

O *framework* permite que o desenvolvimento do software se torne independente da ferramenta ao permitir que as ferramentas adotem seus próprios estilos de modelagem, organização e configurações de modelos e seus elementos, necessitando apenas que se utilize o padrão XMI para exportação de dados.

Verificou-se ainda uma facilidade de criação e manutenção da fase de modelagem no que tange o acompanhamento da evolução do processo usando a estratégia de implementar transformações através de ATL. Isto se justifica, pois o desenvolvedor não precisa se familiarizar com um ambiente novo de desenvolvimento, mas sim estudar as estruturas da linguagem e o conjunto de padrões de segurança utilizado. Além disso, cada necessidade de se modificar a segurança do processo torna-se mais fácil, mais flexível e reutilizável dar manutenção nas regras em ATL, pois a ferramenta proposta se encarrega da transformação automaticamente. Entretanto, vale ressaltar também, que nestes casos é necessário o conhecimento aprofundado da estrutura do conjunto de padrões de segurança proposto.

O XMI permite o intercâmbio de metadados através da linguagem XML. A partir do modelo proposto, dois artefatos podem ser gerados através do XMI. O

primeiro deles é a representação em XML do modelo, descrevendo todas as propriedades de todos os seus elementos. O segundo artefato armazena suas informações estruturais e pode ser usado para a validação de modelos que sejam instâncias destes respectivos modelos.

8.2 Contribuições

A técnica proposta permitiu aplicações automáticas de padrões de segurança que são dependentes uns dos outros pela criação de métodos que descrevem as regras de transformação, marcando o ponto no modelo em que foi transformado.

O resultado do experimento indicou que o uso da transformação não diminui a qualidade dos modelos e tampouco aumenta o tempo de sua criação.

A ferramenta implementada permite que os padrões de segurança possam ser escolhidos e executados diretamente de dentro desta, ou permite a sua utilização de forma independente de ferramenta de livre escolha, pois apresenta uma interface através da qual os modelos podem ser importados em XMI, transformados e exportados para o mesmo *template*. Além disso, as transformações são executadas na ferramenta de forma não-destrutiva, ao preservar informações existentes no modelo de destino.

A ferramenta consente que o engenheiro possa visualizar previamente os padrões de segurança que serão executados pelas regras de transformações, permitindo que ele possa não proceder à execução de algum mapeamento indesejado. Isso possibilita uma maior interatividade do engenheiro no processo de execução da transformação, de acordo com seus objetivos.

Como limitações dessa dissertação têm-se que as transformações modelo-modelo só podem ser definidas de forma unidirecional a partir de uma única especificação. Com isso, as transformações não podem ser reversíveis, impossibilitando a sua execução tanto para a recuperação de informações independentes de plataforma, quanto para a manutenção do sincronismo entre modelos de diferentes níveis de abstração.

A ferramenta implementada é capaz apenas de manipular modelos UML. Apesar da abordagem ser perfeitamente aplicável a outros modelos baseados em

metamodelos, por questões práticas, a implementação ficou restrita ao metamodelo UML.

8.3 Perspectivas futuras

Como trabalho futuro está previsto um estudo de caso mais detalhado, envolvendo uma aplicação de maior complexidade, alinhado com um maior mapeamento de padrões de segurança disponíveis no *framework* definido.

Realizar alterações na ferramenta proposta de forma que o engenheiro de software possa realizar alterações nas regras de transformação através do uso de um editor gráfico orientado pela sintaxe e semântica; e a criação de um metamodelo para o domínio do rastreamento das instâncias dos metamodelos de origem para o de destino.

Realizar um maior mapeamento de padrões de segurança, fazendo o uso de outras classificações descritas por Schumacher (2003) ou a adição de outros padrões descritos na literatura.

Criar funcionalidades de forma a permitir que a ferramenta proposta possa permitir a geração de um modelo no nível PSM e posteriormente este seja o modelo de origem para que se tenha a geração automática para o código-fonte.

REFERÊNCIAS BIBLIOGRÁFICAS

APPOLINÁRIO, F. **Dicionário de metodologia científica**: um guia para a produção do conhecimento científico. 2ª ed., São Paulo: Atlas, 2004.

ASTAH. Astah Community. Disponível em: < http://astah.net/editions/community>. Acesso em: 15 out. 2011.

ATKISON, C., & KÜHNE, T. **Reducing accidental complexity in domain models.** Journal on Software and Systems Modeling. pp. 345-359. DOI: 10.1007/s10270-007-0061-0, 2008.

BARROS, A. J., LEHFELD, N. A. **Fundamentos de Metodologia: um Guia para a Iniciação Científica**. 2ª ed. São Paulo: Makron Books, 2000.

BASIN, D.; DOSER, J. **Model Driven Security: from UML Models to Access Control Infrastructures**. In 5th International School on Foundations of Security Analysis and Design, FOSAD. 2005.

BÉZIVIN, J. **Model Driven Engineering**: An Emerging Technical Space. In: Generative and Transformational Techniques in Software Engineering, LNCS 4143, pp.36-64, 2006.

BROWN, A. W. **Model Driven Architecture**: Principles and practice. Software and Systems Modeling, v. 3, n. 4, p. 314–327, 2004.

CHRISTOPHER, Alexander. **A Pattern Language.** Oxford Press, Oxford, R. Unido, 1978.

CUNHA, Milene Fiorio da. **ArchiMDAs**: Um arcabouço de segurança baseado em transformações de modelos em MDA. Rio de Janeiro, 2007. Dissertação (Mestrado em Informática) - Instituto de Matemática - Núcleo de Computação Eletrônica, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2007.

DEN HAAN, Johann. **MDE - Model Driven Engineering - reference guide**. The Enterprise Architect Building an Agile Enterprise. Disponível em: http://www.theenterprisearchitect.eu/archive/2009/01/15/mde---model-driven-engineering----reference-guide. 2009.

DJURIC, D., GAŠEVIC, D., DEVEDŽIC, V. **The Tao of Modeling Spaces**. in Journal of Object Technology, vol. 5. No. 8, November-December, pp. 125-147. 2006.

ESCALONA, M. J., KOCH, N. **Metamodelling the Requirements of Web Systems**. Lecture Notes in Business Information Processing. 2nd International Conference on Web Information Systems and Technologies, INSTICC, pp. 310-317, 2006.

FERNANDEZ, E. B., PAN, R. **A pattern language for security models**. Conference on Pattern Languages of Programs PloP. Dept. of Computer Science & Engineering, Florida Atlantic University, pp. 1-13. 2001.

FERNANDEZ, E., SORGENTE, T., **A pattern language for secure operating system architectures**. Proceedings of the 5th Latin American Conference on Pattern Languages of Programs, Campos do Jordao, Brazil, August 16-19, 2005.

FINK, T.; KOCH, M.; PAULS, K. **An MDA approach to Access Control Specifications Using MOF and UML Profiles.** In Proceedings 1st International Workshop on Views On Designing Complex Architectures. Vol. 142, pp. 161-179. 2006.

FOWLER, M.; KOBRYN, C.. **UML Essencial**: Um breve guia para a linguagem-padrão de modelagem de objetos. 3ª ed., Bookman. 2005.

GARDNER, T., GRIFFIN, C., KOEHLER, J., HAUSER, R. **A review of OMG MOF 2.0 Query/Views/Transformations Submissions and Recommendations towards the final Standard**. 1st International Workshop on Metamodeling for MDA, pp. 147-161. 2003.

GIL, Antônio Carlos. **Como elaborar projetos de pesquisa**. 3ª ed. São Paulo: Atlas, 2000.

HAAN, J. D. **Multi-tenancy and Model Driven Engineering, necessary assets of a Platform-as-a-Service**. Acesso em 18 de Fevereiro de 2011, disponível em The Enterprise Architect - Building and Agile Enterprise: http://www.theenterprisearchitect.eu/archive/2010/04/27/multi-tenancy-and-model-driven-engineering-necessary-assets-of-a-platform-as-a-service. 2010.

HAFIZ, Munawar; ADAMCZYK, Paul; JOHNSON, Ralph E. **Organizing Security Patterns**, IEEE Software, July/August pp. 52 – 60. 2007.

HULL, M. E., JACKSON, K., & DICK, J. **Requirements Engineering**. 2ª ed., Springer. 2005.

JOUAULT, Frederic; KURTEV, Ivan. **On the Architectural Alignment of ATL and QVT**. Proceedings of ACM Symposium on Applied Computing (SAC 06), Model Transformation Track, Dijon, Bourgogne, France. 2006.

KIENZLE, Darrell M.; ELDER, Matthew C.; TYREE, David; HEWITT, James Edwards. **Security Patterns Repository Version 1.0**. 2006. Disponível em http://www.scrypt.net/~celer/securitypatterns. Acessado em 15 de Maio de 2011.

KLEPPE, A.; WARMER J.; BAST, W. **MDA Explained** - The Model Driven Architecture: Practice and Promise. Addison-Wesley. 2003.

LARMAN, C. **Applying UML and Patterns**: An Introduction to Object-Oriented Analysis and Design and the Unified Process. ISBN: 0131489062. 2004.

LOPES, D.; HAMMOUDI , S.; BEZIVIN, J.; JOUAULT, F. **Mapping Specific ation in MDA**: from Theory to Practice. In: Proceedings of the First International Conference on Interoperability of Enterprise Software and Applications (INTEROP-ESA'2005). p. 253 – 264, 2005.

MAIA, Natanael Elias Nascimento. **ODYSSEY-MDA**: Uma abordagem para a transformação de modelos. Dissertação submetida ao corpo docente da Coordenação dos Programas de Pós-Graduação de Engenharia da Universidade Federal do Rio de Janeiro. 2006.

MATULA, M., **NetBeans Metadata Repository**, NetBeans Community. 2003.

MEDEIROS, Elias Ricken de. **NovaStudio**: Code generator by using MDA approach. Trabalho de conclusão apresentado ao curso de Ciência da Computação da Universidade Federal do Rio Grande do Sul (UFRGS). Disponível em http://hdl.handle.net/10183/18530. 2009.

MELLOR, S. J. **MDA Distilled - Principles of Model Driven Architecture**. Addison-Wesley. 2004.

MILES, Russ. **Learning UML 2.0** [Paperback]. O'Reilly Media, 1ª ed. 2006.

MORENO, N.; VALLECILLO, A.. **Towards Interoperable Web Engineering Methods**. Accepted for publication at the Journal of the American Society for Information Science and Technology (JASIST), 2008.

OBJECT MANAGEMENT GROUP (OMG). **Common Warehouse Metamodel (CWM) Specification**, Version 1.1, Object Management Group, OMG document formal/2003-03-02. 2003a.

OBJECT MANAGEMENT GROUP (OMG). **MDA guide version 1.0.1. OMG**. Disponível em <http://www.omg.org/docs/omg/03-06-01.pdf>. Acesso em: 15 maio 2010. 2003b.

OPENGROUP 2007. Disponível em: http://www.opengroup.org/, acesso em junho de 2011.

ROSADO, D. G.; GUTIERREZ, C.; FERNANDEZ-MEDINA, E.; PIATTINI, M. **Security patterns related to security requirements**. E. Fernandez-Medina e M. Inmaculada (Eds.) Security in Informaiton Systems: Proceedings of the 4th International Workshop on Security in Information Systems. Setúbal, Portugal: INSTICC Press. 2006.

ROSSI, G.; PASTOR, O.; SCHWABE, D.; OLSINA, L. **Web Engineering**: Modelling and Implementing Web Applications. Springer Human-Computer Interaction Series, 2007.

SCHMIDT, Douglas C. **Cover Feature Model-Driven Engineering**. IEEE Computer, Vol. 39, N° 2, pag. 25-31. 2006.

SCHUMACHER, Markus. **Security Engineering With Patterns**: Origins, Theoretical Models, and New Applications, Springer Berlin, Heidelber. 2003.

SCHUMACHER, Markus; FERNANDEZ, Eduardo B.; HYBERTSON, Duane; BUSCHMANN, Frank; SOMMERLAD, Peter. **Security Patterns - Integrating Security and Systems Engineering**. ISBN: 0-470-85884-2. John Wiley & Sons. 2006.

SCHUSTER, Werner. **What's a Ruby DSL and what isn't?**. InfoQ.com Disponível em: <http://www.infoq.com/news/2007/06/dsl-or-not>. Acesso em: 05 jun. 2010.

SELIC, B. **The pragmatics of model-driven development**. IEEE Software, v. 20, n. 5, p. 19–25. 2003. Tariq N. A.; Akhter N. Comparison of Model Driven Architecture (MDA) based tools. 13th Nordic Baltic Conference (NBC). 2005.

SILVA, E. L. D.; MENEZES, E. M. **Metodologia da pesquisa e elaboração de dissertação**. 3ª ed., Florianópolis: Laboratório de Ensino a Distância da UFSC, 121 pp. 2001.

SIQUEIRA, Fábio Levy. **Transformação de um modelo de empresa em um modelo de casos de uso segundo os conceitos de engenharia dirigida por modelos**. Tese de doutorado. Universidade de São Paulo (USP) - São Paulo - SP. 2011.

SOMMERVILLE, Ian. **Engenharia de Software**. 8ª ed.; Pearson Addison-Wesley. 2007.

SÓRIA, F. G. **Implantação do CMMI**: metodologia baseada na abordagem por processos. Dissertação de Mestrado. 1ª ed., Curitiba, v. Pontifícia Universidade Católica do Paraná, 2006.

SUN. **Core J2EE Patterns**: Patterns index page. Disponível em: <http://java.sun.com/blueprints/corej2eepatterns/Patterns/index.html>. Acesso em: 10 mai. 2006.

VALDERAS, P.; PELECHANO, V.; PASTOR, O.. **A Transformational Approach to Produce Web Application Prototypes from a Web Requirements Model**. International Journal of Web Engineering and Technology (IJWET), Vol. 3, nº 1, pp. 4-42, 2007.

VLISSIDES, J. **Patterns**: The Top Ten Misconceptions. Object Magazine, 1997.

WAZLAWICK, R. S. **Uma Reflexão sobre a Pesquisa em Ciência da Computação à Luz da Classificação das Ciências e do Método Científico**. Revista de Sistemas de Informação da FSMA, n. 6, p. 3-10, 2010.

WEISS, M.; MOURATIDIS, H. **Selecting Security Patterns that Fulfill Security Requirements**. In: 16th IEEE International Requirements Engineering Conference pages 169–172. IEEE Computer Society. 2008.

WIMMER, M.; SCHAUERHUBER, A.; SCHWINGER, W.; KARGL, H.. **On the Integration of Web Modeling Languages**: Preliminary Results and Future Challenges. Workshop on Model-driven Web Engineering (MDWE 2007), held in conjunction with ICWE, Como, Italy, 2007.

YIN, Robert K. **Estudo de caso** - **planejamento e métodos**. (2Ed.). Porto Alegre: Bookman. 2001.

YOSHIOKA, N., HONIDEN, S., FINKELSTEIN, A. “**Security Patterns: A Method for Constructing Secure and Efficient Inter-Company Coordination Systems**”, In: 8th IEEE Intl Enterprise Distributed Object Computing Conf (EDOC). 2004.