UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
DEPARTAMENTO DE ELETRÔNICA E COMPUTAÇÃO
CURSO DE CIÊNCIA DA COMPUTAÇÃO

AGREGANDO PADRÕES DE SEGURANÇA EM TRANSFORMAÇÕES DE MODELOS MDA

TRABALHO DE GRADUAÇÃO

Luís Fernando Zaltron

Santa Maria, RS, Brasil
2011

AGREGANDO PADRÕES DE SEGURANÇA EM TRANSFORMAÇÕES DE MODELOS MDA

Por

Luís Fernando Zaltron

Trabalho de Graduação apresentado ao Curso de Ciência da Computação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de
Bacharel em Ciência da Computação.

Orientador: Profª Drª Lisandra Manzoni Fontoura

Co-orientador: Fábio Sarturi Prass

Trabalho de Graduação Nº334
Santa Maria, RS, Brasil
2011

Universidade Federal de Santa Maria
Centro de Tecnologia
Curso de Ciência da Computação

A Comissão Examinadora, abaixo assinada,
aprova o Trabalho de Graduação

AGREGANDO PADRÕES DE SEGURANÇA EM TRANSFORMAÇÕES DE MODELOS MDA

elaborado por
Luís Fernando Zaltron

como requisito parcial para obtenção do grau de
Bacharel em Ciência da Computação

COMISSÃO EXAMINADORA:

_____
Lisandra Manzoni Fontoura, Drª.
(Presidente/Orientador)

_____
Eduardo Kessler Piveta, Dr. (UFSM)

_____
Márcia Pasin, Drª. (UFSM)

Santa Maria, 16 de Dezembro de 2011.

RESUMO

Trabalho de Graduação
Curso de Ciência da Computação
Universidade Federal de Santa Maria

AGREGANDO PADRÕES DE SEGURANÇA EM TRANSFORMAÇÕES DE MODELOS MDA

AUTOR: LUÍS FERNANDO ZALTRON
ORIENTADOR: PROFª DRª LISANDRA MANZONI FONTOURA
CO-ORIENTADOR: FÁBIO SARTURI PRASS
Data e Local da Defesa: Santa Maria, 16 de dezembro de 2011

O aumento da automatização nos processos de desenvolvimento de sistemas vem ganhando espaço no contexto atual devido ao aumento da complexidade e tamanho dos mesmos. Com isto, cada vez mais surgem ferramentas baseadas na ideia de geração automática de código a partir de modelos. Outro aspecto que cada vez mais vem merecendo atenção é a segurança dos sistemas, principalmente os ligados à Internet. Com a intenção de proporcionar uma alternativa para unir estas tendências, este trabalho propõe o desenvolvimento de regras que possibilitem agregar padrões de segurança em modelos, por meio da abordagem *Model Driven Architecture* (MDA), através das transformações descritas por esta arquitetura.

Palavras-chave: MDA. Segurança. Transformação de modelos.

ABSTRACT

Undergraduate Final Work
Undergraduate Program in Computer Science
Federal University of Santa Maria

ADDING SECURITY PATTERNS IN MDA TRANSFORMATION MODEL

AUTHOR: LUÍS FERNANDO ZALTRON
ADVISOR: PROFª DRª LISANDRA MANZONI FONTOURA
CO-ADVISOR: FÁBIO SARTURI PRASS

The increased automation in the process of systems development is becoming more popular in the current context because of the increased complexity and size of them. With this increasingly arise tools based on the idea of automatic code generation from models. Another aspect that is receiving increasing attention is the security of systems, especially those connected to the Internet. With the intention of providing an alternative to link these trends, this paper proposes the development of rules that enable adding security patterns in models through the Model Driven Architecture (MDA), through the transformations described by this architecture.

Keywords: MDA. Security. Models transformation.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS

ADT - ATL Development Tools

ATL - Atlas Transformation Language

CIM - Computation Independent Model

EMF - Eclipse Modeling Framework

JMI - Java Metadata Interchange

MDA - Model Driven Architecture

MOF - Meta-Object Facility

OCL - *Object Constraint Language*

OMG - Object Management Group

PIM - Platform Independent Model

PSM - Platform Specific Model

QVT - Query/View/Transformation

UML - Unified Model Language

SUMÁRIO

1 INTRODUÇÃO

Este capítulo contextualiza o tema proposto neste trabalho além de abordar a motivação para o desenvolvimento, a descrição do problema, os objetivos que se deseja alcançar com o desenvolvimento deste trabalho e sua estrutura.

1.1 Contextualização

Os sistemas computacionais atuais tem se tornado cada vez mais complexos e repletos de detalhes técnicos, atrelado a isso está a necessidade cada vez maior de se manter os programas de software funcionando de maneira integrada, fornecendo informações uns para os outros pela rede. Essa evolução aumenta a complexidade do desenvolvimento de software uma vez que gera preocupação com compatibilidade com diferentes sistemas e versões. Outra preocupação proveniente dessa evolução está na segurança destas aplicações, uma vez que estas estão ligadas à rede (FRANKEL, 2003).

Model Driven Architecture (MDA) é uma alternativa proposta pela *Object Management Group* (OMG) para diminuir este problema proporcionando uma maneira de desenvolver sistemas de computação baseando-se em modelos. Com esta opção, além de facilitar o entendimento e a visualização dos passos do desenvolvimento do sistema, é possível abstrair peculiaridades do sistema como características de plataforma destino e facilitar a fidelização da documentação do sistema com o que está programado, uma vez que o código é gerado a partir de modelos através de processos de transformação de modelos que gradativamente diminuem a abstração destes modelos.

Outra possibilidade gerada com o uso de desenvolvimento dirigido por modelos é a de modelar padrões de segurança para problemas recorrentes, facilitando a agregação ao sistema destes padrões durante as transformações.

1.2 Problema abordado

Segundo Santos (2006), um dos problemas mais relevantes no contexto de desenvolvimento de software atual é a conciliação entre a produtividade no desenvolvimento e a consistência da documentação deste sistema. A falta de documentação, ou a deficiência de consistência da mesma, torna futuras alterações no código mais complicadas e demoradas já que o programador acaba tendo que adivinhar a finalidade de porções do sistema ou perder muito tempo analisando o código.

Conforme as afirmações de Casanas (2001), atualmente a informação tem se tornado cada vez mais vital para os negócios. Ao passo que os negócios são marcados por características como a dinamicidade, são poucas as organizações que não dependem de software de computador para manter suas informações disponíveis para o acesso de pessoas autorizadas. Com isso, vulnerabilidades de segurança podem causar sérios prejuízos podendo até mesmo levar estas organizações à falência.

Uma proposta interessante para solucionar estes problemas é a utilização de técnicas de desenvolvimento de sistemas como MDA agregando características de segurança aos modelos. Esta alternativa proporciona um aumento da coerência da documentação, uma vez que em MDA o código é gerado automaticamente a partir dos modelos, além de facilitar a implementação de padrões de segurança para resolver problemas de segurança recorrentes, promovendo assim ganhos em produtividade, segurança e qualidade de documentação aos sistemas desenvolvidos. As soluções de segurança abordadas neste trabalho são baseadas em Schumacher et al. (2006).

1.3 Objetivos

O objetivo deste trabalho é a elaboração de regras de transformação de modelos para MDA que possibilitem, de forma prática, a inserção de soluções de segurança aos modelos de sistemas computacionais. Estas regras devem ser

desenvolvidas para transformar modelos independentes de plataforma em modelos nesse mesmo nível de abstração, porém, agregando os padrões de segurança. Assim sendo, nas próximas transformações, para modelos de plataforma específica e código, o sistema manterá as características de segurança.

1.4 Organização do Texto

O texto está estruturado da seguinte maneira: no Capítulo 2 é apresentada uma revisão da bibliografia sobre MDA, além de detalhar o funcionamento das transformações de MDA e uma caracterização das principais linguagens utilizadas para transformação de modelos.

No Capítulo 3 são descritos conceitos de segurança além de definição de padrões de segurança e a função destes no desenvolvimento de sistemas, visando possibilitar o entendimento da proposta.

O Capítulo 4 apresenta a sequencia de atividades que podem ser usadas para elaboração de uma metodologia para agregar padrões de segurança através de transformações de modelos baseadas na arquitetura MDA. O Capítulo 5 descreve o estudo de caso de um sistema bancário utilizando durante o desenvolvimento a metodologia para agregar padrões de segurança proposta por este trabalho.

O Capítulo 6 encerra este trabalho com a conclusão e sugestões para trabalhos futuros.

2 MODEL DRIVEN ARCHITECTURE (MDA)

MDA é uma arquitetura de desenvolvimento de software proposta pela OMG que promove a codificação de sistemas utilizando modelos e linguagens de modelagem como se fossem linguagens de programação (OMG, 2003).

Segundo a OMG (2003), a modelagem de sistemas em MDA acontece de acordo com três pontos de vista com diferentes níveis de abstração: *Computation Independent Model* (CIM), o *Platform Independent Model* (PIM) e o *Platform Specific Model* (PSM). Uma descrição mais detalhada destes modelos é encontrada na Seção 2.2.

MDA baseia-se na transformação entre modelos com diferentes níveis de abstração, ou seja, transformações de modelos sem características de plataformas específicas em código fonte de para uma plataforma específica, logo com nível de abstração bem inferior.

2.1 Problemas na maneira tradicional de desenvolvimento de software

Durante a execução de processos de software são criados vários diagramas e artefatos que visam auxiliar o entendimento do sistema e servem como documentação do sistema desenvolvido, mas em muitas situações esses diagramas se tornam figuras sem muita relação com o sistema desenvolvido (WARMER, 2003). A desatualização dos diagramas acontece porque na maioria dos casos, os desenvolvedores são pressionados pelos curtos prazos e alto número de requisições de mudanças no software, e a atualização da documentação acaba recebendo uma prioridade menor devido ao fato de não ter uma consequência imediata no funcionamento do sistema.

O problema é que esta atitude pode gerar complicações na hora de resolver *bugs*, principalmente quando há remanejamento da equipe responsável pelo software, pois a documentação relativa ao sistema não terá fidelidade com o código.

Outro ponto importante para o desenvolvimento de software é a necessidade de se manter a compatibilidade com as versões mais atuais das tecnologias utilizadas, porém essa compatibilidade pode em alguns casos custar à reestruturação total do programa. Essa compatibilidade se faz necessária, pois os programas raramente trabalham isoladamente (WARMER, 2003), principalmente os com acesso via *web* que sofrem ainda mais com as constantes atualizações das tecnologias.

2.2 Processo de desenvolvimento MDA

O processo de desenvolvimento de software com MDA utiliza diferentes modelos com diferentes níveis de abstração: CIM, PIM e PSM.

As atividades de desenvolvimento são iniciadas pelo levantamento de requisitos e posteriormente é gerado o CIM de acordo com os requisitos. O CIM, também chamado de modelo de domínio (OMG, 2003), explora uma visão centrada no ambiente e nos requisitos do sistema, outros detalhes como estruturas usadas não são descritos neste nível. O CIM ajuda a comunicar com precisão as funcionalidades do sistema e os requisitos mapeados neste nível devem ser mapeados para os modelos seguintes.

Através de um processo normalmente manual, é realizada uma transformação do CIM para gerar o PIM. O PIM enfoca na representação das operações do sistema e esconde os detalhes da plataforma de maneira que seja possível utilizar quaisquer plataformas para gerar os modelos do próximo nível sem necessitar de alterações neste PIM.

Com a geração do PIM, pode-se partir para a transformação do PIM em PSM. Essa transformação é realizada utilizando regras de mapeamento que descrevem como transformar elementos entre modelos. Através deste processo, a transformação do PIM é realizada mapeando cada elemento para o seu elemento correspondente no PSM, logo, estas regras são específicas para a plataforma que o PSM especifica e são essas regras que adicionam ao modelo as peculiaridades da plataforma, reduzindo o grau de abstração do modelo.

As regras de transformação necessárias para esta, são destinadas a plataformas específicas. O grau da abstração reduzida com essa transformação pode ser tão grande a ponto de gerar o próprio código, e o fato desta transformação ser a responsável pela inserção das características da plataforma destino faz com que ela seja considerada a transformação mais importante de MDA.

2.3 Transformações e Mapeamentos do MDA

As transformações de modelos em MDA são o ponto chave desta arquitetura uma vez que é através deste processo que é feito o refinamento dos modelos até a codificação do sistema. Nas seções subsequentes, são explorados os tipos de mapeamentos e transformações descritos pela OMG (2003) além de uma breve descrição de algumas das linguagens de mapeamento de modelos mais usadas.

2.3.1 Mapeamentos

O ponto chave de MDA é o mapeamento, pois é através dele são transformados os diferentes modelos de MDA. Os mapeamentos são utilizados para fazer as transformações entre PIM e PIM, PIM e PSM, PSM e PSM.

O mapeamento de PIM para PIM é utilizado com o intuito de filtrar o modelo sem acrescentar informações sobre a plataforma. Essas transformações são utilizadas para refinar o modelo existente.

O mapeamento de PIM para PSM é o mapeamento utilizado para reduzir o nível de abstração da entrada, agregando ao PIM características específicas da plataforma de destino. O mapeamento é feito através de regras de mapeamento específicas para mapear modelos para a plataforma de destino desejada. O mapeamento somente pode ser aplicado quando o modelo PIM estiver refinado o suficiente para ser adaptado para uma plataforma específica.

O mapeamento de PSM para PIM serve para gerar o modelo independente de plataforma através de um modelo de plataforma específica quando não se tem o

modelo mais abstrato. No caso ideal esse mapeamento ira resultar no PIM do sistema.

O último mapeamento, de PSM para PSM é similar ao mapeamento de PIM para PIM, pois também tem o objetivo de refinar e filtrar o modelo já existente.

A Figura 1 exibe a sequência entre os mapeamentos previstos por MDA.

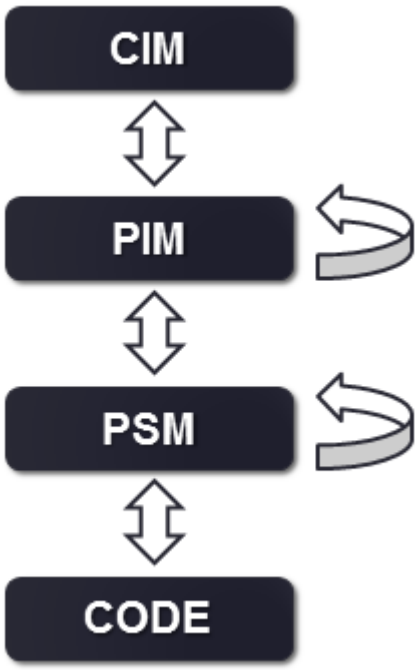


Figura 1 - Transformações MDA.

2.3.2 Linguagens de transformação de modelos

Esta seção traz uma breve abordagem sobre linguagens de transformação de modelos juntamente com a diferença básica entre elas que é a maneira como definem as regras de transformação entre os modelos.

2.3.2.1 Java Metadata Interchange (JMI)

A especificação JMI tolera uma infraestrutura independente de plataforma que permite o gerenciamento da criação, armazenamento, acesso, descoberta e troca de metadados. JMI é baseado no *Meta Object Facility* (MOF) especificado pela OMG. O padrão MOF é um conjunto de artefatos básicos descritos com o uso de *Unified Modeling Language* (UML), que permitem a criação de modelos de qualquer tipo de metadados, os metamodelos (DIRCKZE, 2002).

JMI define interfaces padronizadas para permitir a busca e o acesso aos metadados. Também permite que esses metadados sejam acessados, consultados e manipulados em tempo de *design* e em tempo de execução. Através de *Extensible Markup Language* (XML), usando a especificação de *XML Metadata Interchange* (XMI), permite também a troca de metadados e metamodelos.

Esta linguagem adquiriu grande popularidade devido à similaridade e facilidade de integração com a linguagem Java que é muito usada.

2.3.2.2 Query/View/Transformation (QVT)

QVT é um padrão de modelos definido pela OMG que foi definido com o intuito de acompanhar o avanço nas pesquisas de MDA e para ser compatível com as recomendações como MOF, *Object Constraint Language* (OCL), UML e outros (OMG, 2008).

QVT pode ser vista como uma linguagem de domínio específico para transformações possibilitando a construção e manutenção de transformações de forma simples, com alto poder de expressão e estruturas avançadas.

Segundo sua especificação, QVT tem uma natureza híbrida declarativa imperativa, com a porção declarativa sendo dividida em uma arquitetura de dois níveis, *relations* e *core* (OMG, 2008). A Figura 2, extraída de (OMG, 2008), ilustra a relação entre os metamodelos QVT.

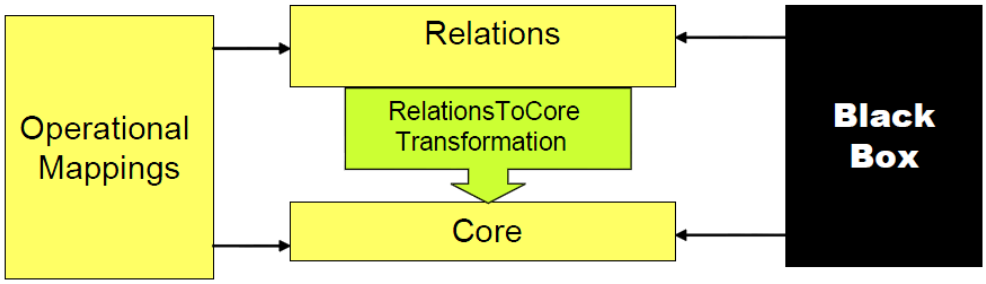


Figura 2 - Relação entre metamodelos QVT (OMG, 2008).

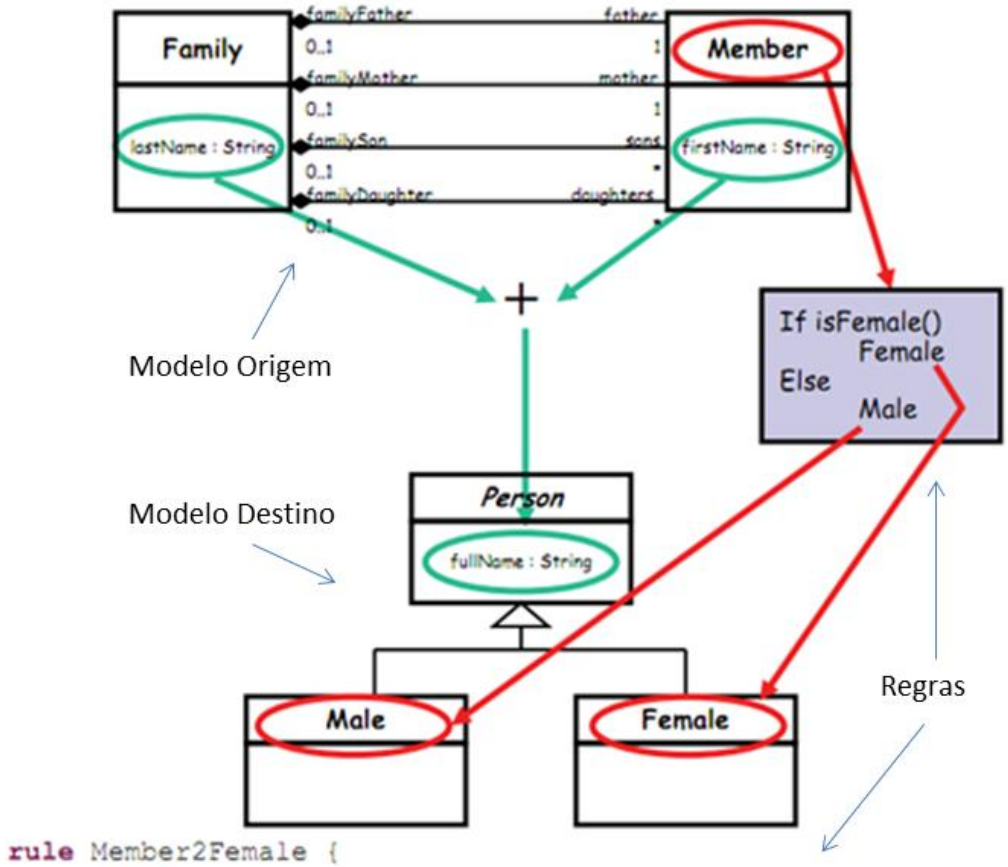
A linguagem *Relations* tem o papel de especificar os relacionamentos entre os modelos, enquanto a linguagem *Core* serve para definir a semântica da linguagem de relacionamentos. Já a Operational Mapping serve para estender a porção declarativa com elementos imperativos como laços e condições.

2.3.2.3 Atlas Transformation Language (ATL)

ATL é uma linguagem híbrida já que possui partes imperativas e declarativas, foi feita para suportar as transformações de modelos de MDA. Os programas escritos em ATL são compostos por regras que definem como elementos de um modelo origem são mapeados para um modelo destino.

Para o desenvolvimento em ATL, foi criada uma IDE juntamente ao Eclipse, a *ATL Development Tools* (ADT), que usa *Eclipse Modeling Framework* (EMF) para editar os modelos. Além disso, foi desenvolvido também um editor com *syntax highlighting* e uma extensão que permite a depuração dos códigos ATL (ATLAS, 2011).

Um exemplo de ATL é mostrado na Figura 3 extraída de (ALLILAIRE, 2007).




```
rule Member2Female {
      from
            s : Families!Member (s.isFemale())
      to
            t : Persons!Female (
                  fullName <- s.firstName + ' ' + s.familyName
            )
}
```

Figura 3 - Exemplo de transformação e código ATL.

Na Figura 3, são ilustrados os modelos de origem, *Family e Member*, e de destino, *Person, Male e Female*. É ilustrado também, a regra ATL usada para transformar *Member* em *Female*, na qual é possível observar que a partir de cada objeto *Member* que retorne verdadeiro da função que verifica se é de sexo feminino, é criado um modelo do tipo *Female* que recebe como atributo *fullName* a concatenação dos atributos *firstName* e *familyName* do objeto *Member* de origem.

3 SEGURANÇA E PADRÕES

No tempo das informações armazenadas em papéis, garantir a segurança destas, significava garantir acesso apenas de pessoas autorizadas através de um controle físico de acesso ao local onde estavam armazenadas. Porém com o contexto atual onde as informações são armazenadas em meio digital e necessitam ser disponibilizadas pela rede e internet, com isso a preocupação com segurança acaba aumentando. Nos sistemas computacionais atuais, a tarefa de mantê-los em segurança está longe de ser fácil, uma vez que constantemente são descobertas novas vulnerabilidades e falhas de implementação em sistemas, que quando exploradas por pessoas mal intencionadas podem gerar grandes prejuízos para as organizações (ROSEMANN, 2002).

Para reduzir ao máximo a incidência destas falhas, existem mecanismos como políticas de segurança, modelos, padrões e regras que visam sempre aumentar a segurança do software e diminuir o impacto causado por uma possível invasão.

3.1 Normas de segurança

Normas de segurança definem diversas políticas de segurança como controle de acesso, processo de registro de usuário, gestão de privilégios de usuários, gestão de senhas de usuários, processo de revisão de direitos de acesso, regras para o uso de senhas, etc. Essas normas tem o objetivo de padronizar soluções de segurança visando detalhar criar um conjunto de itens a serem considerados para garantir que os sistemas não estejam vulneráveis tanto fisicamente quando logicamente (CORDERO, 2010).

3.2 Padrões de segurança

O desenvolvimento e utilização de padrões é uma alternativa que vem ganhando força no desenvolvimento de sistemas quando a preocupação é agregar segurança. Isto acontece à medida que se percebe que não é sempre que um novo projeto significa o surgimento de novas preocupações com segurança que necessitem de soluções inovadoras. Usualmente as necessidades de segurança dos sistemas são similares e isto acaba fazendo com que a equipe de desenvolvimento utilize soluções parecidas com as utilizadas anteriormente para suprir estas necessidades (SCHUMACHER, 2006).

Com isso surge a possibilidade da padronização de necessidades de segurança e suas respectivas soluções, pois uma vez que seja criada uma descrição detalhada da necessidade encontrada e a metodologia utilizada para suprir esta, surge uma facilidade em reutilizar a mesma solução com poucas adaptações em necessidades semelhantes. Esta descrição auxilia também a utilização destes padrões até mesmo por outras equipes de desenvolvimento, o que proporciona maior ganho de segurança, uma vez que o padrão descrito está consolidado, e agilidade no processo de desenvolvimento proporcionado pela facilidade de uso destes.

Schumacher et al. (2006) define padrões de segurança para solucionar estes problemas recorrentes. Esta definição é feita utilizando-se de critérios como: exemplo problema, definição do contexto, problema abordado, solução, estrutura, implementação, exemplo de aplicação, usos conhecidos, consequências e alternativas.

Como exemplo de definição temos os padrões *Authenticator* e *Authorization* que são definidos por Schumacher et al. (2006) como nos itens seguintes:

3.2.1 Authenticator (Autenticador)

O padrão de autenticação é indicado para solucionar o problema de verificação de que o usuário realmente é quem diz ser. Como esta verificação

deverá ser feita, por exemplo biometria ou senha, é uma questão fora do escopo deste trabalho, pois o enfoque deste é aplicar a estrutura do padrão de segurança, os detalhes de implementação ficam restritos a níveis mais baixos de abstração como código.

Os sistemas operacionais autenticam os usuários ao iniciar e o que acontece após isto é de responsabilidade do usuário autenticado, entretanto para algumas operações mais específicas é necessário que o usuário confirme sua autenticação. No caso de alguns sistemas como os que manipulam informações restritas ou controles financeiros, faz-se necessário uma nova autenticação para garantir que o sujeito que está solicitando acesso, seja ele usuário ou outro sistema, realmente pode acessar este sistema.

Para suprir a necessidade de autenticação, Schumacher et al. (2006) define um padrão para autenticação o qual é estruturado conforme a Figura 4.

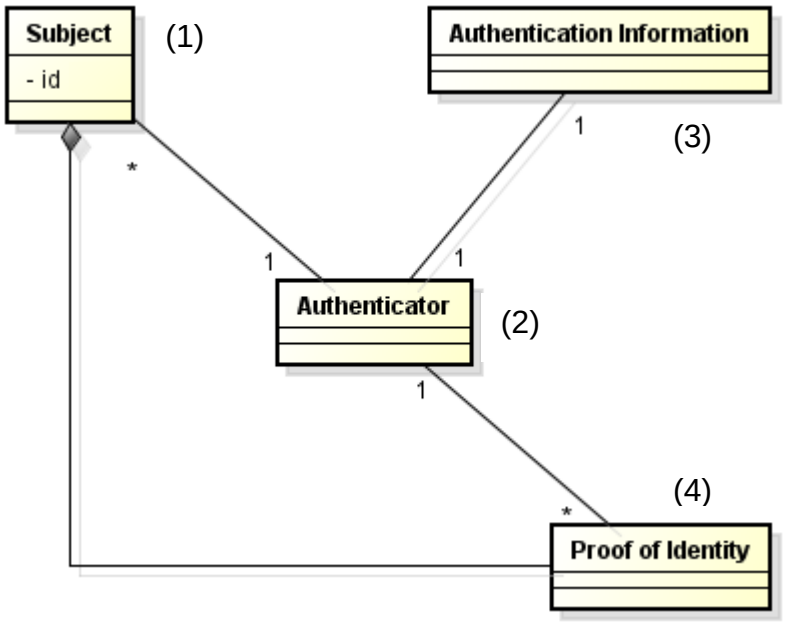


Figura 4 - Diagrama de classe do padrão de autenticação.

Na Figura 4, a classe *Subject* (1) representa o sujeito que será autenticado para acessar o sistema, geralmente um usuário, a classe *Authenticator* (2) recebe a solicitação de autenticação e aplica o protocolo de autenticação utilizando a informação para autenticação armazenada em *Authentication Information* (3). Caso a

autenticação tenha sucesso é gerado um objeto da classe *Proof of Identity* (4) que pode ser explicito como um *token* ou implícito.

3.2.2 Authorization (Autorização)

O padrão de autorização tem como objetivo solucionar problemas relativos ao controle de acesso a informações ou recursos, ou seja, é o responsável por permitir, ou não, o acesso do sujeito autenticado no sistema a um ou mais recursos do mesmo.

Este padrão deve ser aplicado às porções do sistema que necessitam de controle de acesso diferenciado de acordo com cada sujeito ou grupo de sujeitos. Esta aplicação implica permissões diferentes como apenas leitura, leitura e edição, para cada porção do sistema.

Para solucionar problemas de autorização, Schumacher et al. (2006) propõe o padrão modelado na Figura 5.

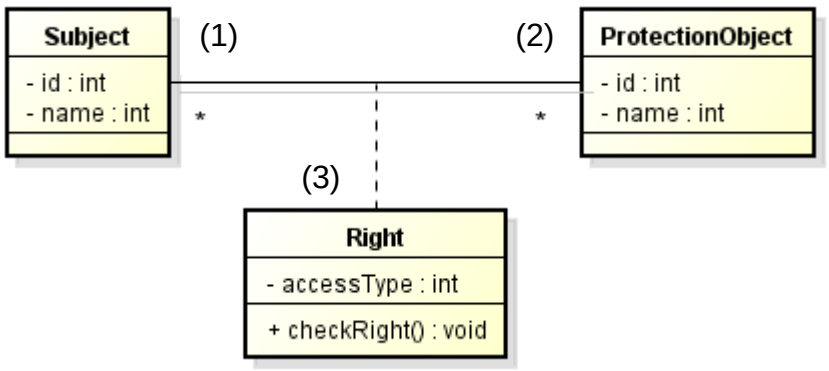


Figura 5 - Diagrama de classe do padrão de autorização.

Na Figura 5, a classe *Subject* (1) representa o sujeito que deseja acessar o recurso representado pela classe *ProtectionObject* (2). A associação entre *Subject* (1) e *ProtectionObject* (2) define a classe *Right* (3) que descreve o tipo de acesso permitido deste *Subject* (1) para este *ProtectionObject* (2).

Outro ponto a ser observado na aplicação do padrão de autorização é a necessidade de ter aplicado anteriormente alguma forma de autenticação, pois uma vez que o sujeito que deseja acessar o recurso não seja autenticado, o sistema o sistema deixa uma possibilidade de um sujeito passar-se por outro que pode ter permissões de acesso diferenciadas.

3.3 Modelagem de sistemas seguros

Um problema encontrado ao tentar tornar sistemas seguros está na adaptação do sistema para agregar segurança, uma vez que a adaptação é mais trabalhosa e menos satisfatória que a segurança agregada desde as etapas iniciais de desenvolvimento do software.

Como uma alternativa viável, surge o desenvolvimento orientado a modelos, pois essa abordagem facilita o entendimento dos padrões e da integração deles ainda na etapa de modelagem do software, garantindo assim maior aderência aos padrões. Deste modo, a tarefa de agregar segurança ao sistema pode tornar-se mais confiável, uma vez que possibilita gerar o código do sistema todo após agregar modelos de segurança ao sistema.

A proposta de integrar padrões de segurança ao desenvolvimento de sistemas com MDA objetiva simplificar o conhecimento necessário para trabalhar com padrões de segurança além de possibilitar a correta aplicação destes padrões. Assim, é possível construir regras de transformações de modelos MDA que agreguem estes padrões de segurança aos modelos do sistema de forma simples e rápida, proporcionando de maneira pouco complicada um ganho com segurança.

4 ATIVIDADES PARA AGREGAÇÃO DE PADRÕES DE SEGURANÇA EM TRANSFORMAÇÕES DE MODELOS

Este capítulo descreve um conjunto de atividades que pode ser utilizada para agregar padrões de segurança em transformações baseadas em MDA, e tem como objetivo identificar as alterações no desenvolvimento de software tradicional para o desenvolvimento de software orientado a modelos. Não é objetivo descrever uma metodologia completa, mas sim um conjunto de atividades relacionadas diretamente ao objetivo de agregar de padrões em transformações de modelos. A Figura 6 sumariza as atividades proposta pela metodologia.

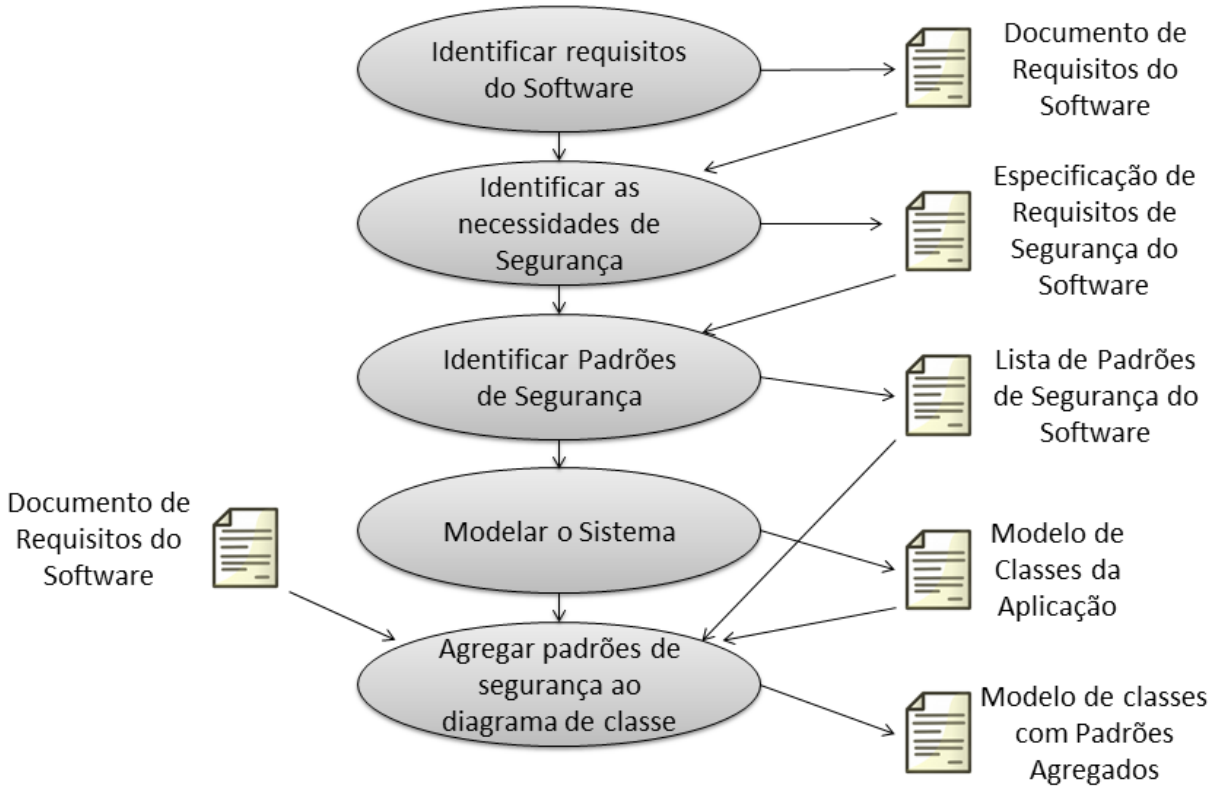


Figura 6 - Atividades propostas.

Nas subseções seguintes, uma breve descrição de cada atividade da metodologia, comparando-a com o desenvolvimento tradicional[1] de software.

4.1 Identificação dos Requisitos do Software

Inicialmente, é necessário identificar os requisitos do software que será desenvolvido. Por meio de técnicas de levantamento de requisitos, tais como: entrevista com os *stakeholders* do sistema, questionários, análise de documentos, entre outras; é possível identificar os requisitos do novo sistema e elaborar uma descrição textual do sistema a ser desenvolvido, normalmente chamada de *Documento de Requisitos do Software*.

Essa descrição textual pode evoluir para modelo de casos de uso e especificações de casos de uso. Essa atividade não tem maiores alterações em relação ao desenvolvimento tradicional de software.

4.2 Identificação das necessidades de segurança

A identificação das necessidades de segurança da aplicação em desenvolvimento é um passo importante para a proposta deste trabalho de graduação, pois a partir desta identificação que serão selecionados os padrões de segurança a serem adicionados ao sistema que, posteriormente, serão transformados em código. A identificação das necessidades de segurança é realizada com base no *Documento de Requisitos do Software.*

Como a segurança é um fator de suma importância nos sistemas desenvolvidos atualmente, deve-se dedicar grande atenção a esta etapa do desenvolvimento, pois se uma necessidade de segurança passar despercebida nesta etapa poderá ocasionar uma falha de segurança no software.

[1] Neste trabalho convencionou-se chamar de desenvolvimento tradicional, o desenvolvimento de software que não utiliza a abordagem MDA.

Os requisitos de segurança para o sistema podem ser documentados em um documento de *Especificação de Requisitos de Segurança do Software*.

4.3 Identificar Padrões de Segurança

Após identificar as necessidades de segurança é necessário eleger padrões de segurança que possam suprir essas necessidades.

Padrões de segurança materializam mecanismos básicos ou abordagens que fornecem meios de proteger a confidencialidade, integridade e disponibilidade das informações (SCHUMACHER, 2006). Padrões são úteis para satisfazer requisitos de segurança. A utilização de padrões de segurança tem como objetivo reusar experiências bem-sucedidas para desenvolvimento de software seguro, facilitando a modelagem deste tipo software.

Os catálogos de padrões mais conhecidos são: Schumacher et al. (2006), Rosado (2006), Romanosky (2003), Kienzle e Elder (2002). Nesse trabalho optou-se por utilizar os padrões de segurança propostos por Schumacher et al. (2006).

Com base em catálogos de padrões, é necessário identificar os padrões que satisfazem os requisitos de segurança documentados na *Especificação de Requisitos de Segurança do Software*. Como resultado da execução desta atividade tem-se uma *Lista de Padrões de Segurança*.

4.4 Modelagem do sistema

A modelagem do sistema é um passo natural do processo de desenvolvimento de sistemas, principalmente se este seguir a arquitetura MDA que tem os modelos como artefato principal do desenvolvimento.

A etapa de modelagem do sistema inclui diversos diagramas com a finalidade de demonstrar corretamente as funcionalidades e requisitos do sistema a ser desenvolvido.

Essa atividade não apresenta modificações em relação ao desenvolvimento tradicional de software. Ao final da execução desta atividade será obtido o *Diagrama de Classes da Aplicação*.

4.5 Agregação de Padrões de Segurança ao Modelo de Classes

O diagrama utilizado neste trabalho para agregar padrões de segurança é o diagrama de classes que representa aspectos de dados do sistema, logo, deve-se desenvolver o diagrama de classes do sistema em desenvolvimento e dos padrões de segurança escolhidos, caso sua descrição não inclua os diagramas de classes.

Após a criação dos diagramas de classes do sistema e dos padrões a serem integrados devem ser desenvolvidas as regras de transformações que irão agregar os padrões de segurança ao sistema.

4.6 Desenvolvimento das regras de transformação

Para que a metodologia proposta possa ser aplicada é necessário que, anteriormente, as regras de transformação sejam desenvolvidas para os padrões que podem ser incorporados na modelagem do sistema durante a execução da metodologia proposta.

Há diversas linguagens que possibilitam descrever regras de transformação para a arquitetura orientada a modelos, algumas destas foram brevemente descritas na Seção 2.3.2. Para este trabalho foi escolhida a linguagem ATL por ser de fácil utilização, ter compatibilidade com o Eclipse IDE e estar em crescente uso no meio acadêmico.

Os elementos envolvidos no processo de transformação utilizando ATL podem ser visualizados na Figura 7.

Figura 7 - Funcionamento de transformações em ATL.

Na Figura 7 é possível observar que para efetuar transformações com ATL é necessário um metamodelo de entrada que define a estrutura do modelo de entrada, um metamodelo de saída no qual o modelo de saída deve ser baseado e as regras de transformações escritas em ATL. A diferença entre um modelo e um metamodelo é que o metamodelo identifica os elementos que podem ser definidos no modelo e esse é baseado no domínio da aplicação que se deseja modelar. O modelo de entrada deve ser descrito utilizando a linguagem *XML Metadata Interchange* (XMI) e o modelo gerado pela regra de transformação também é descrito em XMI.

5 ESTUDO DE CASO – SISTEMA BANCÁRIO

Como demonstração da metodologia para agregação de padrões de segurança em transformações de modelos, este capítulo apresenta um estudo de caso detalhando os passos para agregar padrões de segurança a um modelo de um sistema bancário. Nas seções seguintes serão exibidos os procedimentos e artefatos desenvolvidos para gerar um modelo de sistema resultante com padrões de segurança agregados.

5.1 Descrição da aplicação

O sistema abordado por este estudo é um sistema bancário que deve possibilitar o cadastro dos clientes e dos empregados do banco. Os clientes devem ser cadastrados para que as informações deles estejam salvas no sistema. Já os empregados devem ser também cadastrados no sistema para que tenha uma restrição de acesso ao sistema apenas para empregados do banco.

Além disso, o sistema deve conter o cadastro das contas dos clientes, sejam elas do tipo conta corrente ou poupança, e deve armazenar um histórico da conta composto por transações bancárias.

É importante ressaltar que como se trata de um sistema de controle bancário, assunto que exige extrema segurança e sigilo é necessário que apenas alguns funcionários do banco possam acessar de modo mais livre, como possibilidade de edição, os dados dos clientes e respectivas suas contas. O sistema deve verificar se o funcionário que está utilizando o sistema tem permissão para acessar as informações de cada cliente.

5.2 Necessidades de segurança

Esta seção deve receber grande atenção, pois é o momento de identificar, a partir da descrição do sistema, quais as necessidades de segurança deste. As necessidades identificadas nesta etapa serão supridas por algum padrão de segurança também selecionado nesta etapa. Caso alguma necessidade de segurança passe despercebida, poderá causar uma falha de segurança no sistema possibilitando o uso indevido das funcionalidades do sistema.

Segundo a descrição deste software bancário, foi identificada a necessidade de algum método de autorização dos funcionários do banco para acessarem as informações dos clientes, uma vez que nem todos os funcionários devem ter permissão para acessar e modificar as informações de todos os clientes.

Com esta característica, surge também a necessidade de autenticar os funcionários do banco ao acessarem o sistema. Esta funcionalidade faz-se necessária para garantir que o funcionário que está utilizando o sistema seja realmente quem se diz ser e não uma pessoa mal intencionada tentando invadir o sistema fazendo-se passar por um funcionário autorizado a acessar as informações.

Para suprir estas necessidades de segurança do sistema, foram selecionados dois padrões descritos por Schumacher et al. (2006), os quais são: *Authenticator* e *Authorization*. Uma breve descrição destes padrões foi exibida nas Seções 3.2.1 e 3.2.2, respectivamente.

Por meio destes padrões deve ser possível autenticar os funcionários do banco que estão utilizando o sistema e verificar a autorização de cada funcionário para acessar ou modificar os dados de cada cliente do banco.

5.3 Modelagem do sistema

Nesta etapa do desenvolvimento é necessário que seja feito o diagrama de classes da aplicação que posteriormente será utilizado como entrada para as transformações que irão agregar os padrões de segurança selecionados.

O diagrama de classes exibido na Figura 8 foi elaborado para atender os requisitos do sistema.

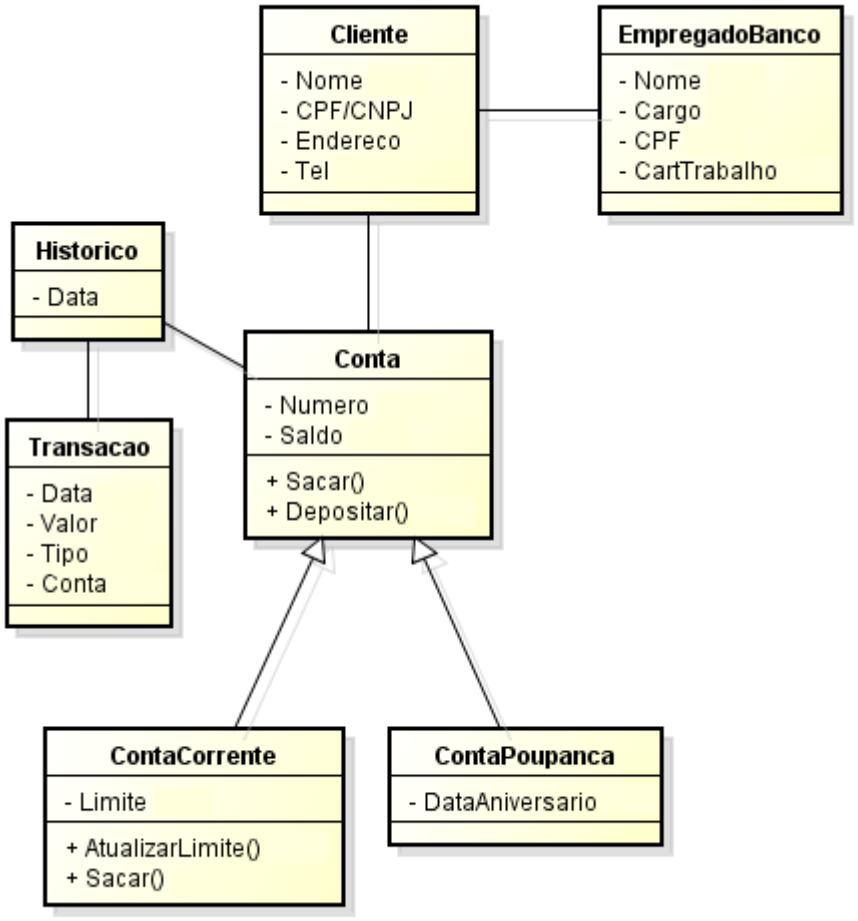


Figura 8 - Diagrama de classes do sistema bancário.

No diagrama da Figura 8 são representadas as classes do sistema descrito. É possível verificar a estrutura de armazenamento das informações dos empregados do banco que irão acessar com diferentes permissões os dados armazenados nos objetos da classe Cliente e suas respectivas contas com históricos de transações.

Através deste diagrama foram identificadas as necessidades de segurança estão em torno da classe que armazena as informações dos empregados do banco e a classe que armazena as informações do cliente. Portanto com o objetivo de focar apenas na porção do diagrama de classes que irá sofrer modificações com a agregação de padrões de segurança, o diagrama da Figura 8 foi simplificado através

da eliminação das classes que não seriam atingidas pelas transformações. Esta redução permite uma melhor visualização do efeito das transformações propostas por este trabalho. O diagrama de classes resultante da simplificação do diagrama da Figura 8 é ilustrado na Figura 9.

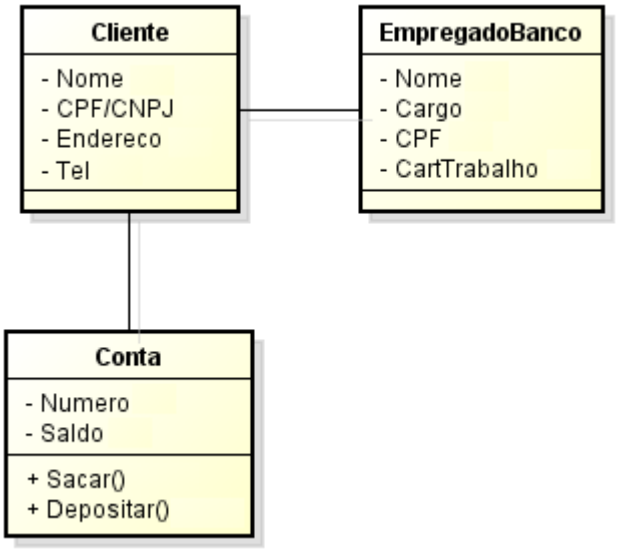


Figura 9 - Diagrama de classes do sistema bancário (versão simplificada).

No diagrama da Figura 9 foram preservadas apenas as classes mais importantes para este estudo de caso. A classe de armazenamento das informações dos empregados do banco, dos clientes e informações das contas dos clientes.

5.4 Agregação de Padrões de Segurança ao Diagrama de Classes do Sistema

Conforme identificação das necessidades de segurança descritas na seção 5.2 foram selecionados para suprir os padrões de autenticação e autorização. Neste estudo, para tornar mais clara a ação das transformações e a separação dos dois padrões selecionados, a agregação dos padrões foi dividida em duas etapas.

Primeiramente foi agregado ao modelo do sistema o padrão de autenticação (*Authenticator Pattern*) e, posteriormente, foi agregado o padrão de autorização (*Authorization Pattern*).

Como a etapa de agregação foi dividida em duas etapas, fez-se necessária a criação de duas transformações. Porém, o modelo de saída da primeira transformação pode ser aproveitado como entrada para a transformação seguinte.

Nas seções seguintes, serão detalhadas estas duas transformações e bem como as regras escritas em ATL que deram origem a estas.

5.4.1 Transformação para agregar autenticação

Esta etapa tem o objetivo de agregar o padrão *Authenticator* (definido na seção 3.2.1) ao modelo simplificado visto na Figura 9. Para isto, foram desenvolvidas regras que possibilitam a criação das classes necessárias para a aplicação correta do padrão durante a transformação.

Para a aplicação do padrão, foi identificado que o objeto a ser autenticado é o objeto da classe EmpregadoBanco, logo, esta classe deve receber o papel da classe *Subject* do padrão de autenticação exibido na Figura 4. Após agregar o padrão ao modelo do sistema, a transformação deve gerar um modelo de acordo com o metamodelo da Figura 10.

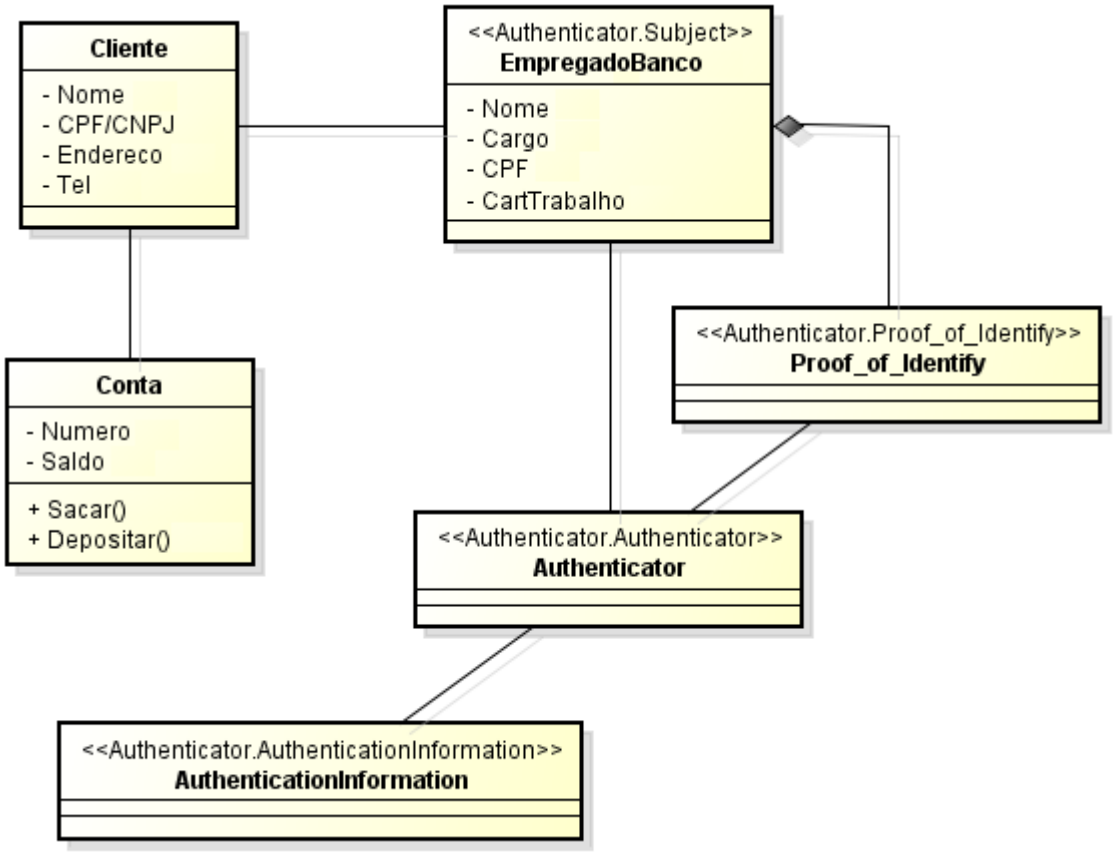


Figura 10 - Metamodelo simplificado do sistema com padrão de autenticação agregado.

No diagrama da Figura 10 é possível perceber que foram adicionadas três novas classes, a classe *Authenticator* recebe a solicitação de autenticação e aplica o protocolo de autenticação utilizando a informação para autenticação armazenada em *Authentication Information*. Caso a autenticação tenha sucesso é gerado um objeto da classe *Proof of Identity* que funciona como o registro da autenticação.

Para gerar este diagrama foi necessário codificar regras na linguagem ATL para mapear as classes existentes e criar as classes correspondentes ao padrão de segurança aplicado. A parte do código responsável pela criação das novas classes pode ser vista na Figura 11.

```
1  rule EmpregadoBanco{
2       from
3               eb_ini: diagrama_inicial!EmpregadoBanco
4       to
5               eb_fin: diagrama_intermediario!EmpregadoBanco(
6                       Nome <- eb_ini.Nome,
7                       Cargo <- eb_ini.Cargo,
8                       CPF <- eb_ini.CPF,
9                       CartTrabalho <- eb_ini.CartTrabalho
10                      ),
11              cta_aui: diagrama_intermediario!AuthenticationInformation(),
12              cta_aut: diagrama_intermediario!Authenticator(),
13              cta_poi: diagrama_intermediario!Proof_of_Identify()
14 }
```

Figura 11 – Código para criação das classes do padrão de autenticação.

Na Figura 11 é exibida uma regra de transformação escrita em ATL que tem como nome EmpregadoBanco e é chamada automaticamente pelo motor de transformações ATL sempre que no modelo de entrada conter um objeto da classe EmpregadoBanco definida no metamodelo de entrada com nome diagrama_inicial. A ação desta regra é mapear as informações da classe EmpregadoBanco do diagrama_inicial para a classe EmpregadoBanco do diagrama_intermediário que é o metamodelo de saída desta transformação. Além disso, esta regra cria as três classes definidas pelo padrão *Authenticator*.

5.4.2 Transformação para agregar autorização

Nesta seção, o padrão a ser agregado ao modelo do sistema é o *Authorization* (descrito na seção 3.2.2). Este padrão deve ser agregado ao metamodelo do sistema resultante da transformação que agregou o padrão de autenticação, ou seja, o metamodelo de entrada para esta transformação é o demonstrado na Figura 10.

Para agregar de maneira correta o padrão de autorização é necessário identificar qual sujeito deve ser autorizado a acessar qual recurso. No contexto do sistema bancário, é possível identificar que a informação a ser acessada é a

armazenada pela classe Cliente que receberá o papel da classe *ProtectionObject* do padrão de autorização modelado na Figura 5. Já o papel da classe *Subject* é atribuído à classe EmpregadoBanco, uma vez que os empregados do banco devem ter autorização para acessar informações dos clientes.

Com essa atribuição de papéis é possível chegar ao metamodelo de saída desta transformação que é ilustrado na Figura 12.

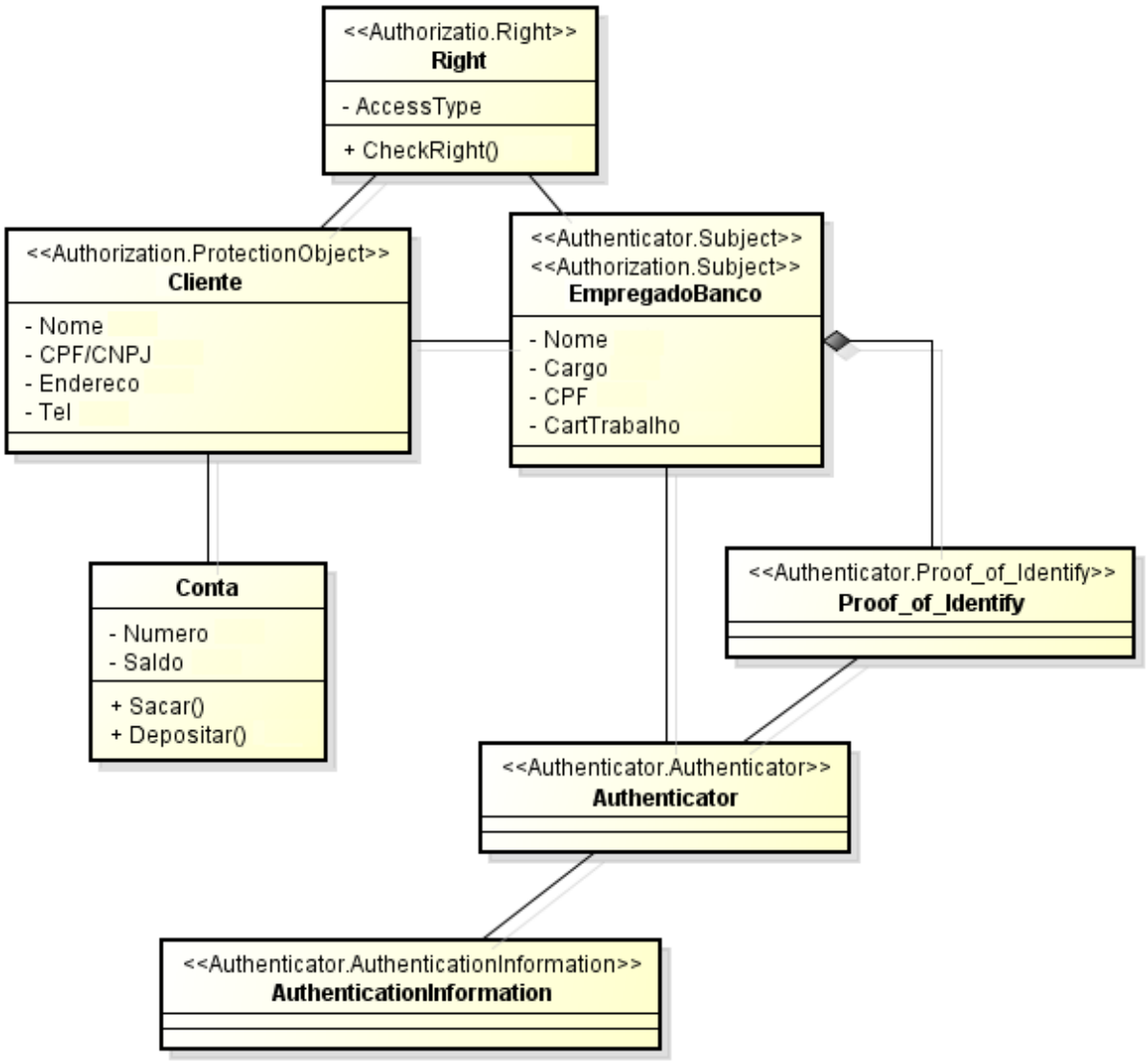


Figura 12 - Metamodelo simplificado do sistema com os padrões agregados.

Na Figura 12 é possível observar que foi criada a classe *Right* responsável pela verificação de autorização para acesso dos empregados do banco aos dados dos clientes.

Para executar esta transformação foram elaboradas regras de mapeamento utilizando a linguagem ATL. Essas regras transferem as informações das classes do metamodelo de entrada para o metamodelo de saída além de criar a classe *Right* de acordo com o metamodelo de saída da transformação.

A parte do código responsável pela criação da nova classe pode ser vista na Figura 13.

```
1 rule EmpregadoBanco{
2       from
3               eb_ini: diagrama_intermediario!EmpregadoBanco
4       to
5               eb_fin: diagrama_final!EmpregadoBanco(
6                       Nome <- eb_ini.Nome,
7                       Cargo <- eb_ini.Cargo,
8                       CPF <- eb_ini.CPF,
9                       CartTrabalho <- eb_ini.CartTrabalho
10                      ),
11              ri_fin: diagrama_final!Right()
12 }
```

Figura 13 - Código para criação da classe *Right*.

5.5 Resultados

Após as transformações propostas neste estudo, é possível observar a agregação dos padrões de segurança ao sistema visando satisfazer as necessidades de segurança do mesmo, pois ao agregar os padrões de segurança para autenticação e autorização foi possível satisfazer as necessidades de segurança para o sistema bancário.

A partir do modelo completo da aplicação mostrado na Figura 8 foi possível, após as duas transformações propostas, chegar a um modelo de sistema com padrões agregados que pode ser visto na Figura 14.

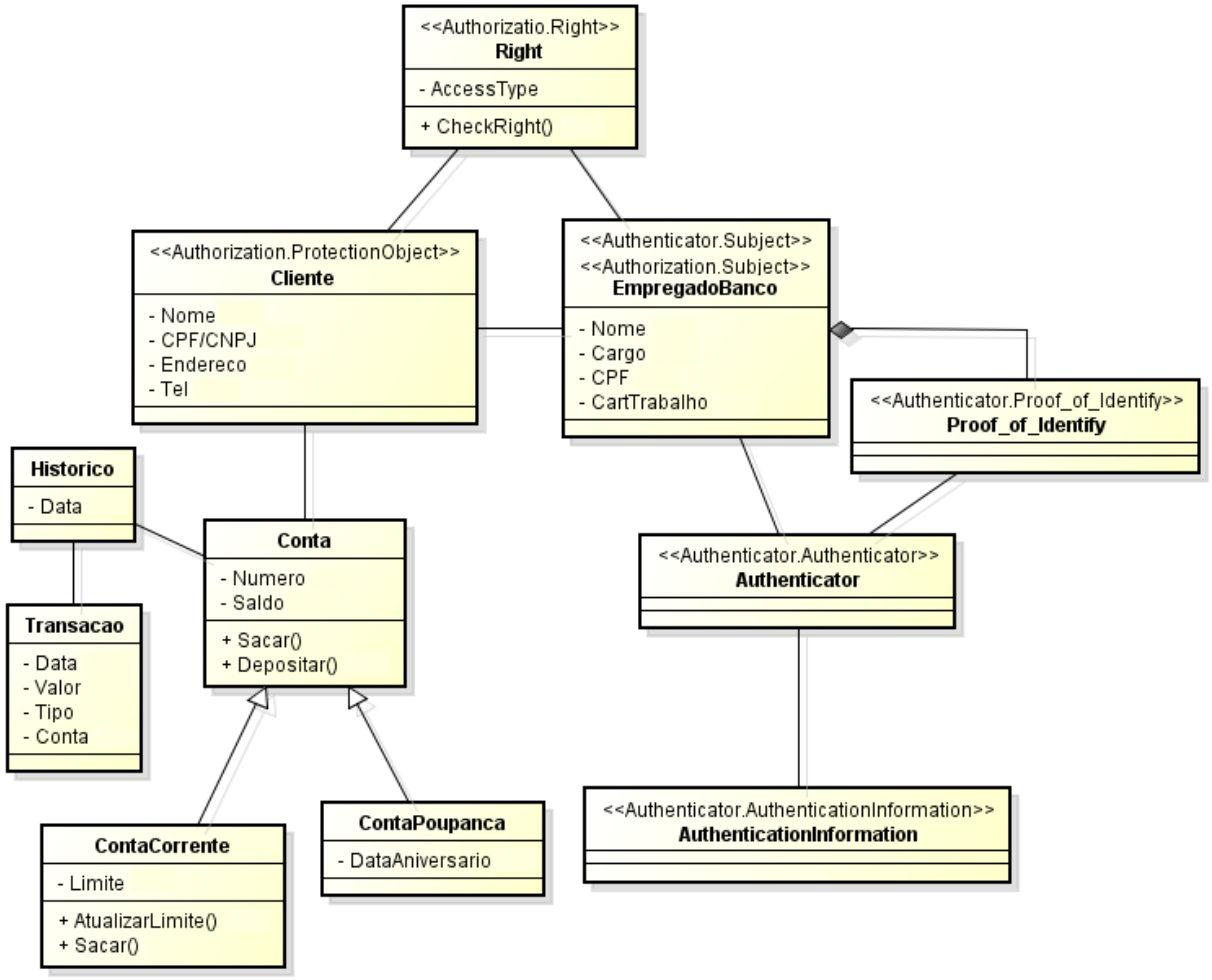


Figura 14 - Diagrama final do sistema bancário (versão completa).

Com isso foi possível gerar um modelo de sistema bancário com padrões de segurança consolidados que atendam aos requisitos de segurança identificados na seção 5.2. Portanto o objetivo das transformações propostas, que é agregar padrões de segurança que possam suprir as necessidades do sistema, foi alcançado.

6 CONCLUSÃO

A metodologia proposta descreve como padrões de segurança podem ser aplicados durante transformações em MDA para satisfazer requisitos de segurança identificados para um dado sistema. Algumas atividades da metodologia proposta não apresentam modificações significativas em relação ao desenvolvimento tradicional de software. Novas atividades foram propostas com o objetivo de agregar padrões de segurança na abordagem MDA, tais como: *Identificar Necessidades de Segurança, Identificar Padrões de Segurança e Agregar Padrões de Segurança ao Diagrama de Classes*.

Optou-se por usar *Atlas Transformation Language* devido a esta ser de fácil utilização, ter compatibilidade com o Eclipse IDE e estar em crescente uso no meio acadêmico.

Outro ponto importante observado foi a facilidade de readaptar estas transformações para outros sistemas uma vez que já existam regras implementadas anteriormente, gerando assim ganho em tempo de desenvolvimento.

Por meio do estudo de caso, pode-se verificar que a metodologia para aplicação de padrões de segurança em MDA tem como vantagens: uma maior facilidade na aplicação de padrões de segurança, uma vez que, as regras, após codificadas podem ser reusadas com pequenas ou nenhuma modificação, o tempo para implementação de padrões de segurança em aplicações do usuário pode ser minimizado, uma vez que parte do código é gerado automaticamente, maior correteza na implementação dos padrões, pois após a consolidação das regras, essas podem ser reusadas.

Com a metodologia descrita neste trabalho, é possível criar regras para agregar padrões de segurança em sistemas com quaisquer outros objetivos além de servir de base para trabalhos futuros que possam agregar estas transformações à sequencias de transformações de MDA para possibilitar a geração de código-fonte com padrões de segurança agregados a partir de modelos de sistemas sem soluções de segurança.

Outra possibilidade de trabalho futuro é a criação de um software auxiliar que possibilite a adaptação das regras de transformação de forma genérica.

REFERÊNCIAS

ALLILAIRE, F; JOUAULT, F. (2007). **"Families to Persons" A simple illustration of model-to-model transformation**. Disponível em < http://www.eclipse.org/m2m/atl/doc/ATLUseCase_Families2Persons.pdf>. Acessado em: 15 set. 2011.

ATLAS. **ATL: The ATLAS Transformation Language**. Disponível em <www.eclipse.org/m2m/atl/doc/ATL_PresentationSheet.pdf>. Acessado em: 20 out. 2011.

CORDERO, E. Z. (2010). **Proposta de um Guia com Boas Práticas de Segurança da Informação.** Trabalho Final de Graduação - Unifra.

DIRCKZE, R. (2002) **Java™ Metadata Interface(JMI) Specification**. Version 1.0 Disponível em <http://download.oracle.com/auth/otn-pub/jcp/7868-jmi-1.0-fr-spec-oth-JSpec/jmi-1_0-fr-spec.pdf?e=1319410979&h=68def31ed2b4797f1ff770e870925ff9>. Acessado em: 20 out. 2011.

FRANKEL, D. (2003). **Model Driven Architecture. Applying MDA to Enterprise Computing.** Indianapolis: Wiley Publishing, 2003.

KIENZLE, D. M; ELDER, M. C. (2002). **Security Patterns for Web Application Development**. Final Technical Report, Univ. of Virginia.

OMG. (2003). **MDA guide version 1.0.1.** Disponível em < http://www.omg.org/cgi-bin/doc?omg/03-06-01.pdf >. Acesso em: 18 out. 2011.

OMG. (2008). **Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification**. Disponível em <http://www.omg.org/spec/QVT/1.0/PDF>. Acessado em 19 out.2011.

ROSADO, D. G. (2006). **A Study of Security Architectural Patterns**. In: Proceedings of the First International Conference on Availability, Reliability and Security (ARES'06).

ROSEMANN, D. (2002). **Software para Avaliação da Segurança da Informação de uma empresa conforme a Norma NBR ISO/IEC 17799**. Trabalho de Conclusão

de Curso - Universidade Regional de Blumenau. Disponível em < http://campeche.inf.furb.br/tccs/2002-II/2002-2douglasrosemannvf.pdf>. Acessado em 24 out. 2011.

ROMANOSKY, S. (2003). **Operational security patterns**. In: EuroPLoP. Disponível em <http://hillside.net/europlop/europlop2003/papers/WritingGroup/WG4_RomanoskyS.doc> acesso em: 01 dez. 2011.

SANTOS, A. L. F. (2006). **Análise e Aplicação do MDA para Contrução de Sistemas de Software.** Trabalho de Conclusão de Curso - Universidade do Vale do Itajaí. Disponível em < http://siaibib01.univali.br/pdf/Andre%20Luiz%20Fernandes%20dos%20Santos.pdf>. Acessado em: 18 out. 2011.

SCHUMACHER, M. et al. (2006). **Security Patterns. Integrating Security and Systems Engineering.**

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