

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
INSTITUTO DE MATEMÁTICA
NÚCLEO DE COMPUTAÇÃO ELETRÔNICA

MILENE FIORIO DA CUNHA

ARCHIMDAS: UM ARCABOUÇO DE SEGURANÇA BASEADO EM
TRANSFORMAÇÕES DE MODELOS EM MDA

Rio de Janeiro

2007

Milene Fiorio da Cunha

ARCHIMDAS: UM ARCABOUÇO DE SEGURANÇA BASEADO EM
TRANSFORMAÇÕES DE MODELOS EM MDA

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Informática, Instituto de Matemática, Núcleo de Computação Eletrônica, Universidade Federal, do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Informática.

Orientador: Carlo Emmanoel Tolla de Oliveira, Ph.D.
Co-orientador: Paulo de Figueiredo Pires, D.Sc.

Rio de Janeiro
2007

Cunha, Milene Fiorio da.

ArchiMDAs: Um arcabouço de segurança baseado em transformações de modelos em MDA / Milene Fiorio da Cunha. - Rio de Janeiro, 2007.

xiv, 120 f.: il.

Dissertação (Mestrado em Informática) – Universidade Federal do Rio de Janeiro - UFRJ, Instituto de Matemática / Núcleo de Computação Eletrônica, 2007.

Orientador: Carlo Emmanoel Tolla de Oliveira, Paulo de Figueiredo Pires

1. MDA 2. Segurança 3. Controle de Acesso I. Oliveira, Carlo Emmanoel Tolla de (Orient.). II. Pires, Paulo de Figueiredo III. Universidade Federal do Rio de Janeiro. Instituto de Matemática. Núcleo de Computação Eletrônica. IV. Título.

Milene Fiorio da Cunha

ARCHIMDAS: UM ARCABOUÇO DE SEGURANÇA BASEADO EM
TRANSFORMAÇÕES DE MODELOS EM MDA

Dissertação de Mestrado apresentada ao
Programa de Pós-Graduação em Informática, Instituto de
Matemática, Núcleo de Computação Eletrônica,
Universidade Federal do Rio de Janeiro, como parte dos
requisitos necessários à obtenção do título de Mestre em
Informática.

Rio de Janeiro, _____ de _____ de _____

Aprovada por:

Carlo Emmanoel Tolla de Oliveira, Ph.D, UFRJ

Paulo de Figueiredo Pires, D.Sc, UFRN

Luiz Fernando Rust da Costa Carmo, Ph.D, UFRJ

Ivan Mathias Filho, D.Sc, PUC-RJ

Flávia Coimbra Delicato, D.Sc, UFRN

Rio de Janeiro
2007

A minha mãe, pela dedicação, exemplo, carinho e apoio em todos os momentos da minha vida.

AGRADECIMENTOS

A Deus pela vida e pelas oportunidades.

À minha mãe, pelo exemplo de vida e dignidade, por ter acreditado e investido em minha educação, pelo carinho e amor, que nunca me faltaram. Te amo!

Ao meu pai, que mesmo longe, sempre me apoiou e me incentivou nos momentos mais difíceis.

Ao meu namorado Cláudio 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, pela força e incentivo. Pela compreensão pelos momentos ausentes, pelos aniversários e festas de família que eu amo tanto e que não pude comparecer.

Aos meus afilhados, Gabriel e Jhonatha, pelas visitas que me trouxeram tantas alegrias nos momentos mais estressantes.

À minha irmã de coração Bianca, que vem me aturando cada vez mais nos últimos anos. Obrigada por agüentar meus momentos de mau humor e pelos momentos de diversão.

A todos meus amigos que me ajudaram de alguma forma e estiveram ao meu lado durante todos os momentos da minha vida. Obrigada pelo apoio e por entenderem o meu sumiço, principalmente no último ano. Todos vocês são parte importantíssima de minha vida. Muito obrigada!

Aos meus orientadores Carlo Emmanoel Tolla de Oliveira e Paulo de Figueiredo Pires, por terem me orientado, além da paciência, confiança e compreensão dos momentos difíceis pelos quais passei. Muito obrigada por compartilharem parte de seu conhecimento e suas idéias comigo, me mostrando o caminho da pesquisa.

Aos professores Flávia Coimbra Delicato, Ivan Mathias Filho e Luiz Fernando Rust da Costa Carmo por aceitarem participar desta banca, agregando valor inestimável a este trabalho.

À CAPES, pelo apoio financeiro ao desenvolvimento deste trabalho.

À Fundação COPPETEC, pelas oportunidades de crescimento profissional. Em especial, ao professor Blaschek, pelo apoio e solidariedade.

À UFRJ, pelo excelente ensino, por ter me tornado o profissional que sou hoje.

Ao NCE e ao Instituto de Matemática, por terem me aceito como aluna de mestrado e terem me proporcionado um dos maiores orgulhos da minha vida!

RESUMO

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.

Grande parte dos sistemas de informação depende de requisitos de segurança de alta qualidade. A qualidade de segurança pode ser provida através de um projeto arquitetural focado na geração automática da implementação. Neste trabalho, apresentamos um arcabouço de segurança orientado a modelo (MDA) que simplifica e flexibiliza a modelagem dos requisitos de segurança seguindo a arquitetura do sistema, provendo proteção em diferentes camadas de uma aplicação. A segurança é aplicada implicitamente no sistema através de transformação entre modelos e codificação automática, garantindo que a segurança não será violada em nenhum nível e não será suscetível a erros ou alteração do código.

ABSTRACT

CUNHA, Milene Fiorio da. ArchiMDAs: A MDA Transformation Based Security Framework. 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.

Most of Information systems depend on high quality security requirements. Quality of security can be provided through Architectural design enforced into implementation by automatic generation. In this work, we present a security framework based on Model Driven Architecture (MDA) that simplifies and becomes flexible security requirements modeling. This security framework is based on the system architecture and provides protection at different levels. Security is then applied implicitly to the whole system providing a high assurance that security measures are not violated at any level and are not susceptible to code error or tampering.

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
CORBA	Common Object Request Broker Architecture
CIM	Computation Independent Model
DAC	Discretionary Access Control
DSD	Dynamic Separation of Duty
EJB	Enterprise Java Bean
EMF	Eclipse Modeling Framework
GEF	Graphical Editing Framework
GOF	Gang of Four
GRASP	General Responsibility Assignment Software Patterns
J2EE	Java 2 Platform, Enterprise Edition
JAAS	Java Authentication and Authorization Service
HTTPS	HyperText Transfer Protocol Secure
MAC	Mandatory Access Control
MDA	Model Driven Architecture
MOF	Meta-Object Facility
OASIS	Advancing open standards for the information society
OMG	Object Management Group
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PIM	Platform Independent Model
PSM	Platform Specific Model
RBAC	Role based Access Control
SSD	Static Separation of Duty
UML	Unified Modeling Language
URL	Uniform Resource Locator
VBAC	View Based Access Control
XACML	eXtensible Access Control Markup Language
XMI	XML Metadata Interchange
XML	EXtensible Markup Language

LISTA DE FIGURAS

Figura 1. RBAC Básico (Fonte: GUERRA, PAICA, FERNANDES 2004).....	13
Figura 2. Relação entre modelos, linguagens e ferramentas de transformação (Fonte: KLEPPE, WARMER, BAST 2002 traduzida pela autora).....	25
Figura 3. A aplicação do padrão MDA (Fonte: KLEPPE, WARMER, BAST 2002 traduzida pela autora).....	28
Figura 4. Segurança dirigida a modelo (Fonte: BASIN, DOSER 2005)	32
Figura 5. Modelando um cenário de um banco (Fonte: BASIN, DOSER 2005)	33
Figura 6. Modelos MDA-VBAC (Fonte: FINK, KOCH, PAULS 2004)	35
Figura 7. Transformações em MDA com extensões de segurança.....	40
Figura 8. Modelo RBAC com Contexto (Fonte: GUERRA, PAICA, FERNANDES 2004) ..	44
Figura 9. RBAC com Restrição a Dados	45
Figura 10. Modelos de Segurança	48
Figura 11. Metamodelo de Segurança	48
Figura 12. Processo de Construção e Utilização do Arcabouço de Segurança ArchiMDAs ...	51
Figura 13. Arquitetura do AndromDA (Fonte: ANDROMDA 2007 traduzida pela autora) ..	56
Figura 14. Autenticação no JAAS	62
Figura 15. Autenticação no JAAS	63
Figura 16. Lógica de Autenticação	65
Figura 17. Implementação Autenticação	66
Figura 18. Lógica de autorização de vista.....	69
Figura 19. Lógica de Autorização dos elementos de vista.....	70
Figura 20. Lógica de autorização da camada de serviço.....	71
Figura 21. Lógica de adição de restrições.....	71
Figura 22. Lógica de adição das restrições na consulta	72
Figura 23. Implementação Autorização URL	73
Figura 24. Implementação autorização elementos da vista.....	74
Figura 25. Autorização camada de serviço	76
Figura 26. Modelo de Restrição	77
Figura 27. Verificação e Adição de restrições a dados.....	78
Figura 28. Adição Restrições a Consulta	79
Figura 29. Implementação do caso de uso Trocar Senha.....	80
Figura 30. Implementação do caso de uso sair do sistema	81
Figura 31. Diagrama de Casos de Uso.....	87
Figura 32. Diagrama de Classes de Domínio	90
Figura 33. Diagrama de Classes de Serviço.....	91

Figura 34. Exemplo autorização componentes de vista.....	93
Figura 35. Configuração do jboss.xml	94
Figura 36. Modelo de Dados após transformação	95
Figura 37. Tela de autenticação.....	98
Figura 38. Menu de líder.....	99
Figura 39. Menu Desenvolvedor	99
Figura 40. Tela de Troca de Senha	100

LISTA DE TABELAS

Tabela 1. Padrões de Segurança	47
Tabela 2. Perfil do Usuário no Projeto.....	96
Tabela 3. Restrições	96
Tabela 4. Restrições associadas aos Serviços por perfil	96
Tabela 5. Resultados	101

SUMÁRIO

1	INTRODUÇÃO.....	1
1.1	MOTIVAÇÃO.....	1
1.2	OBJETIVO.....	3
1.3	ORGANIZAÇÃO.....	4
2	FUNDAMENTOS DE SEGURANÇA EM AMBIENTES COMPUTACIONAIS.....	6
2.1	CONCEITO DE SEGURANÇA	6
2.2	POLÍTICA DE SEGURANÇA	7
2.3	VIOLAÇÕES DE SEGURANÇA	7
2.4	CONTROLE DE ACESSO.....	7
2.4.1.	Autenticação.....	8
2.4.2.	Autorização.....	9
2.4.3.	Auditoria.....	9
2.4.4.	Modelos de Segurança.....	9
2.4.4.1.	Modelo Discrecional.....	10
2.4.4.2.	Modelos Obrigatórios.....	11
2.4.4.3.	Modelos Baseados em Papéis (RBAC)	11
2.5	CONCLUSÃO.....	13
3	MODELAGEM DE SISTEMAS SEGUROS	15
3.1	PADRÕES DE PROJETO DE SEGURANÇA	16
3.1.1.	Padrões de Projeto	16
3.1.1.1.	Representação de padrões de projeto.....	18
3.1.1.2.	Metodologia de utilização de padrões de projeto.....	19
3.1.2.	Padrões de Projeto de Segurança	20
3.2	SEGURANÇA COM MDA.....	22
3.2.1.	Model Driven Architecture	23
3.2.1.1.	A utilização da UML na descrição dos modelos.....	26
3.2.1.2.	Mapeamento entre modelos	27
3.2.2.	Análise dos Trabalhos Existentes	31
3.3	CONCLUSÃO.....	36
4	DESCRIÇÃO DO ARCABOUÇO ARCHIMDAS.....	39
4.1	REQUISITOS DE SEGURANÇA	40
4.2	RBAC ESTENDIDO	43
4.2.1.	RBAC com contextos	43
4.2.2.	Restrições a dados.....	44
4.2.3.	Níveis de segurança.....	45
4.3	PADRÕES DE PROJETO APLICADOS AO ARCABOUÇO ARCHIMDAS	46
4.4	METAMODELO DE SEGURANÇA.....	47
4.5	ARQUITETURA E PROCESSO.....	50
4.6	CONCLUSÃO.....	54
5	IMPLEMENTAÇÃO.....	55
5.1	FERRAMENTA ANDROMDA.....	55
5.2	EXTENSÃO DO ANDROMDA	56
5.3	JAAS	60
5.3.1.	Autenticação no JAAS	60
5.3.2.	Autorização no JAAS.....	63
5.4	MÓDULO DE REGRAS DE SEGURANÇA.....	64
5.4.1.	Autenticação.....	64
5.4.1.1.	Lógica de Implementação	64
5.4.1.2.	Implementação.....	65
5.4.2.	Autorização.....	68
5.4.2.1.	Lógica de Implementação	69
5.4.2.2.	Implementação.....	72

5.4.3.	<i>Casos de uso Sair do Sistema e Trocar Senha</i>	80
5.5	MÓDULO ADMINISTRATIVO DE SEGURANÇA.....	81
5.6	CONCLUSÃO.....	83
6	ESTUDO DE CASO	85
6.1	DEFINIÇÃO DO ESTUDO	85
6.2	EXEMPLO DE APLICAÇÃO.....	86
6.2.1.	<i>Modelo</i>	87
6.2.2.	<i>Geração de Código</i>	92
6.2.2.1.	<i>Autenticação</i>	92
6.2.2.2.	<i>Autorização</i>	92
6.2.2.3.	<i>Casos de Uso Sair do Sistema e Trocar Senha</i>	94
6.2.3.	<i>Avaliação</i>	94
6.2.3.1.	<i>Casos de Teste</i>	98
6.2.4.	<i>Análise dos Resultados</i>	101
6.3	CONCLUSÃO.....	103
7	CONCLUSÃO.....	104

1 INTRODUÇÃO

1.1 MOTIVAÇÃO

Devido aos diversos tipos de ataques de quebra de segurança existentes atualmente, os sistemas computacionais estão necessitando cada vez mais de proteção para os dados e informações confidenciais. As organizações necessitam cada vez mais investir em implementações de sistemas com uma maior ênfase na proteção de informações e na restrição no uso de recursos e serviços computacionais, dando acesso apenas a pessoas autorizadas.

Porém, apesar de grande parte dos sistemas de informação possuir a segurança como requisito fundamental, constantemente há notícias sobre vulnerabilidades e falhas de segurança [FINK, KOCH, PAULS 2004].

Isto se deve a vários motivos, entre eles, podemos citar o aumento da complexidade de implementação dos mecanismos de controle de segurança devido a diversificação e evolução dos modos de ataque. Além disso, na maioria das vezes, os projetistas de sistemas não possuem experiência em segurança e acabam provocando problemas de segurança tanto na arquitetura da aplicação quanto na lógica implementada

Um outro problema é o fato do requisito de segurança ser raramente considerado nos estágios iniciais do desenvolvimento de software e acaba sendo delegado a segundo plano ao longo do mesmo. Além disso, existe uma grande lacuna entre as soluções teóricas e o que é de fato implementado na área de segurança [BASIN, DOSER 2005].

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 2000]. 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 projetos. Contudo, para obter os benefícios da utilização de padrões de projeto é necessário saber quando e como utilizá-los a partir de suas definições, pois os desenvolvedores podem acabar introduzindo falhas de segurança.

Há vários arcabouços de segurança disponíveis, entre eles, podemos citar JAAS [JAAS 2007], MISF [MISF 2007], Acegi [ACEGI 2007] e HDIV [HDIV 2007]. Todos esses

arcabouços representam uma solução de segurança independente do domínio da aplicação, feito normalmente por desenvolvedores experientes que possuem decisões de projetos e utilizam padrões de projeto.

Contudo, à medida que os arcabouços de segurança 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 que o negócio necessita. Devido à diversidade de requisitos de negócio de cada aplicação, arquitetos de software e analistas necessitam cada vez mais criar seus próprios arcabouços e manter suas integridades e coerência em relação às necessidades do negócio as quais eles atendem [BLAKLEY, HEALTH 2004].

Neste contexto, uma abordagem orientada a modelo (MDA) pode auxiliar o processo de desenvolvimento de sistemas seguros, promovendo uma implementação controlada dos padrões de projeto e, desta forma, garantindo aderência às diretrizes arquiteturais da organização e às questões de segurança.

MDA é uma abordagem de desenvolvimento de sistemas que se baseia na idéia da separação da especificação das funcionalidades de um sistema dos detalhes de implementação [OMG 2003]. Os três objetivos principais do MDA são portabilidade, interoperabilidade e reutilização através da separação arquitetural de interesses. Neste sentido, MDA define mecanismos para especificar um sistema independentemente da plataforma que o suporta, especificar plataformas, escolher uma plataforma para um sistema e transformar a especificação do sistema em uma plataforma específica. Com a separação de interesses, esta abordagem ajuda a focar nos aspectos de negócio da solução ao invés dos aspectos técnicos.

A idéia de “orientação a modelos” está fundamentada na utilização de modelos para direcionar o entendimento, projeto, construção, instalação e manutenção de sistemas. Apesar desta idéia ser antiga, a MDA tem se mostrado bastante promissora devido ao diferencial em que os modelos são transformados em código de forma automática e configurável. Desta forma, uma possível mudança de plataforma implica em alteração da especificação das transformações, enquanto que a especificação da solução se mantém intacta.

Esta abordagem recomenda a criação de modelos em três níveis de abstração na especificação de sistemas [KLEPPE, WARMER, BAST 2002]. O modelo de mais alto nível é o *Computation Independent Model* (CIM). Este modelo é uma visão do sistema do ponto de vista independente de computação e serve, principalmente, como meio de comunicação entre os especialistas de domínio e requisitos e os especialistas no projeto e construção do sistema.

O modelo no nível de abstração seguinte é chamado de *Platform Independent Model* (PIM). Este modelo já contém detalhes computacionais, mas ainda é independente de qualquer tecnologia de implementação. O modelo de mais baixo nível de abstração é chamado de *Platform Specific Model* (PSM). Este modelo já contém detalhes de uma plataforma específica e pode ser considerado como a implementação do sistema.

No âmbito de segurança, a utilização de MDA com segurança faz com que a segurança fique integrada na modelagem de sistemas de forma que o código de segurança é gerado automaticamente.

1.2 OBJETIVO

Segurança constitui um requisito implícito de qualidade, que deve garantir que o sistema está protegido em qualquer circunstância. O fator humano pode interferir na robustez de segurança em qualquer ponto, ao longo do processo de desenvolvimento, onde intervenção manual é requerida.

Algumas abordagens já utilizam MDA para desenvolver arcabouços de segurança. Nessas abordagens, a segurança depende da modelagem feita pelo analista ou projetista. Desta forma, a partir do momento em que a segurança do sistema é ativada através da modelagem da política de controle de acesso feita manualmente, ela se torna suscetível a erros humanos.

Além disso, o modelo de negócio é feito juntamente com o de segurança, o que pode provocar uma sobrecarga de atribuições ao desenvolvedor do sistema, elevando a complexidade do desenvolvimento e, conseqüentemente, 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.

Neste contexto, desenvolvemos uma abordagem que utiliza MDA para desenvolver arcabouços de segurança de controle de acesso focado nas regras de transformação entre modelos e não na modelagem da política de segurança.

O objetivo deste trabalho é apresentar 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 chamamos de módulo de regras de segurança e

outro responsável pela administração de segurança específica de cada aplicação, que chamamos de módulo administrativo de segurança.

O módulo de regras de segurança é responsável pela regras de autenticação e autorização dos artefatos da aplicação onde o processo de modelagem das regras de negócio não sofre alterações, o projetista deve focar somente no negócio, sem se preocupar com a segurança. Os requisitos de segurança serão acrescentados ao sistema de forma automatizada e ortogonal ao desenvolvimento das regras de negócio, através dos mecanismos oferecidos pelo ArchiMDAs.

Já o módulo administrativo de segurança representa o modelo de segurança da aplicação, que depende da política de controle de acesso da organização. Visando facilitar o desenvolvimento da aplicação, uma aplicação administrativa de segurança é gerada automaticamente considerando uma política de segurança padrão, de forma que só é necessário criar sua própria aplicação de segurança caso este não esteja de acordo com a política de segurança da organização.

Buscamos com o arcabouço ArchiMDAs simplificar o desenvolvimento de sistemas de software, aumentando a produtividade e a qualidade das especificações de segurança e com isso, aumentando consideravelmente a qualidade e a manutenibilidade dos sistemas construídos. O desacoplamento entre a implementação do negócio propriamente dito e a solução de segurança, facilita a evolução da solução fazendo com que ela não se torne obsoleta, principalmente pelo fato do código de segurança ser gerado automaticamente, através de transformações. O arcabouço foi planejado implementando os principais padrões de segurança e com isso garantindo formalmente ao cliente a segurança da aplicação.

1.3 ORGANIZAÇÃO

Este trabalho encontra-se organizado em sete capítulos. No capítulo 2, apresentamos os fundamentos de segurança em ambiente computacional, onde são descritos alguns conceitos e definições, como por exemplo, a definição de modelo de segurança e os principais modelos existentes. No capítulo 3, realizamos uma análise de modelagem de sistemas seguros, apresentando técnicas de integração de soluções de segurança nas fases de análise e projeto de sistemas de software. Entre essas técnicas, podemos destacar a utilização da abordagem de padrões de projeto e da abordagem orientada a modelo MDA, onde são descritos conceitos, definições e trabalhos realizados na área. No capítulo 4 apresentamos o projeto e a arquitetura do arcabouço. No capítulo 5 apresentamos a implementação do arcabouço. No capítulo 6

apresentamos um experimento de utilização do arcabouço. Finalmente no capítulo 7, descrevemos as conclusões e trabalhos futuros.

2 FUNDAMENTOS DE SEGURANÇA EM AMBIENTES COMPUTACIONAIS

A segurança em sistemas computacionais trata-se de uma disciplina que busca através de todos os seus conceitos, metodologias e técnicas empregadas, manter as propriedades de um sistema de forma que ele não possa ser alvo de possíveis ações danosas praticadas por entidades não autorizadas junto às informações e recursos nele existentes.

Neste capítulo iremos introduzir alguns desses conceitos, metodologias e técnicas empregadas com o objetivo de garantir a segurança da informação em sistemas computacionais. Na Seção 2.1, definimos o conceito de segurança. Na Seção 2.2, definimos o conceito de política de segurança. Na Seção 2.3, definimos o conceito de violações de segurança. Na Seção 2.4, definimos o conceito de controle de acesso e de seus processos e apresentamos os principais modelos de segurança de controle de acesso existentes na literatura. Na seção 2.5 concluímos este capítulo apresentando algumas considerações finais.

2.1 CONCEITO DE SEGURANÇA

A segurança de sistemas computacionais visa à proteção contra a indisponibilidade, vazamento da informação e a leitura ou modificação não-autorizada das informações. Além de garantir dados e informações, a segurança visa prevenir, detectar, conter e documentar eventuais ameaças aos sistemas computacionais [BISHOP 2003].

Existem hoje na literatura várias definições para segurança e, em sua grande maioria, elas incluem a necessidade de se manter no sistema um conjunto de propriedades [DENNING 1982]:

- **Confidencialidade:** a informação poderá ser acessada somente por usuários autorizados. Este acesso é restrito a usuários devidamente cadastrados para que impeça usuários não autorizados de terem acesso à informação;
- **Integridade:** prover a garantia que a informação deve ser retornada em sua forma original no momento em que foi armazenada, protegendo contra modificações não autorizadas;
- **Disponibilidade:** a informação ou sistema de computador deve estar disponível no momento em que a mesma seja requisitada.

2.2 POLÍTICA DE SEGURANÇA

A política de segurança pode ser definida como sendo o conjunto de regras e serviços que visam especificar como um sistema provê os seus recursos mantendo sempre as propriedades de confidencialidade, integridade e disponibilidade [SILVA 2005].

As políticas de segurança são classificadas em duas categorias em relação ao controle de acesso: as discricionárias e as obrigatórias. Nas discricionárias os acessos a cada recurso ou informação são manipulados sem restrição pelo proprietário ou responsável do sistema, baseando-se na idéia de que este proprietário deve determinar quem tem acesso a estas informações segundo a sua vontade (à sua discrição). Já nas obrigatórias, as autorizações de acesso são definidas através de um conjunto de regras que expressam algum tipo de organização envolvendo a segurança das informações no sistema como um todo [MACKENZIE, POTTINGER 1997], baseando-se em uma administração centralizada de segurança, a qual dita qual serão as regras de acesso à informação.

2.3 VIOLAÇÕES DE SEGURANÇA

As regras definidas pela política de segurança determinam as entidades que serão autorizadas e responsáveis pelas ações executadas sobre todas as informações mantidas no sistema. Dependendo do nível de aplicação desta política pode-se caracterizar como sendo entidade um usuário, um processo ou ainda uma máquina dentro de uma rede de computadores. Uma entidade que ganha acesso a recursos de um sistema computacional de forma ilícita, violando assim a política de segurança, é denominada de intruso.

As violações de segurança em sistemas computacionais se traduzem como sendo a arte de burlar de alguma forma a política de segurança [SILVA 2005].

2.4 CONTROLE DE ACESSO

“O controle de acesso está relacionado diretamente ao acesso concedido. A função desse controle é garantir que o acesso seja feito somente dentro dos limites estabelecidos.” [CARUSO, STEFFEN 1999]. O controle de acesso da segurança do computador inclui a autenticação para saber quem possui acesso ao sistema, a autorização para saber tudo que um usuário do sistema pode realizar dentro dele e a auditoria que torna possível rastrear e descobrir quem é o responsável por determinada ação realizada no sistema, sendo essa ação benéfica ou maléfica. Existem modelos de controle de acesso com políticas distintas: controle de acesso discricionário, controle de acesso obrigatório e controle de acesso baseado em

papéis.

O controle de acesso discricionário especifica o modo de acesso a um objeto, no caso leitura, escrita ou execução, baseado na identidade do usuário, grupos de usuários e objetos de sistemas. Neste caso um usuário que possui acesso a uma informação pode transferir a outro usuário o acesso a esta, de modo que este outro usuário poderá acessá-la e ainda repassar a terceiros, sem precisar ser autorizado pelo verdadeiro dono da informação [SANDHU, SAMARATI 1994]. No controle de acesso obrigatório, o usuário e o objeto de sistema pertencem a um nível de segurança [SANDHU, SAMARATI 1994].

No controle de acesso obrigatório, o usuário não possui mais credenciais para poder repassar aos outros usuários privilégios de acessos, isso é feito somente pelo sistema de controle de acesso. Cada informação possui um nível de segurança necessário para ser acessada, então um usuário deve possuir um nível igual ou maior ao que é requerido para poder acessar a informação.

O controle de acesso baseado em papéis determina que cada usuário possua o seu papel dentro do sistema, sendo assim, só poderá acessar informações as quais o seu papel o possibilita. O papel é um conjunto de particularidades que um usuário pode assumir. [SANDHU, SAMARATI 1994]. Vários usuários podem possuir o mesmo papel.

2.4.1. AUTENTICAÇÃO

A identificação e autenticação fazem parte de um processo de dois passos que determina quem pode acessar determinado sistema. Durante a identificação o usuário diz ao sistema quem ele é (normalmente através de um nome de usuário). Durante a autenticação a identidade é verificada através de uma credencial (uma senha, por exemplo) fornecida pelo usuário.

“A autenticação é essencial para a segurança dos sistemas, ao validar a identificação dos usuários, concedendo-lhes a autorização para o acesso aos recursos.” [NAKAMURA, GEUS 2002]. Sistemas que possuem acesso limitado utilizam mecanismos de autenticação para identificar os usuários que já estejam previamente cadastrados no sistema e que podem acessá-lo.

A maneira mais usada para realizar essa autenticação é por meio de dados previamente conhecidos pelo usuário, como por exemplo, o *login* e a senha, que inseridos em um formulário são enviados para o mecanismo de autenticação vigente, onde se os dados coincidirem com os armazenados no banco de dados, o usuário poderá receber a autorização para acessar os recursos de que dispõem. Outro modo de autenticação é por algo que o usuário

possui como *smartcard* ou outro objeto de identificação no sistema que só ele possua. Também por algo que o usuário tenha fisicamente, como impressão digital, geometria das mãos, reconhecimento da retina, reconhecimento da íris e mesmo a voz [NAKAMURA, GEUS 2002].

2.4.2. AUTORIZAÇÃO

A autorização define quais direitos e permissões um determinado usuário do sistema possui. Após o usuário ser autenticado o processo de autorização determina o que ele pode fazer no sistema.

A autorização é um serviço usado na área computacional para a proteção não só de informações, como também de arquivos, funcionalidades de aplicações, dispositivos do computador e etc. Segundo [NAKAMURA, GEUS 2002] o mecanismo de autorização é a permissão de acesso dada pelo sistema a um recurso protegido somente se o usuário ou processo que esteja querendo acessá-lo possuir a autorização ou permissão necessária para fazê-lo. Em um exemplo simples, um usuário após ter passado com sucesso pelo mecanismo de autenticação poderá por meio das permissões que possui acessar as informações e os recursos do sistema. O usuário deverá ser parado pelo mecanismo de autorização se tentar acessar um recurso que não possua a permissão necessária.

2.4.3. AUDITORIA

A auditoria é uma referência à coleta da informação relacionada à utilização, pelos usuários, dos recursos de um sistema. Esta informação pode ser utilizada para gerenciamento, planejamento, cobrança e etc. A auditoria em tempo real ocorre quando as informações relativas aos usuários são trafegadas no momento do consumo dos recursos. Na auditoria em *batch* as informações são gravadas e enviadas posteriormente. As informações que são tipicamente relacionadas com este processo são a identidade do usuário, a natureza do serviço entregue, o momento em que o serviço se inicia e o momento do seu término.

2.4.4. MODELOS DE SEGURANÇA

Os modelos de segurança são formas de descrever as políticas de segurança, determinando o comportamento de entidades administradas pela política e também as regras que definem a evolução desta política.

Quando alguns modelos matemáticos formais de segurança são utilizados para descrever políticas de autorização, os mesmos permitem de alguma maneira, verificações de

que a política é coerente, e servem como guia para implementações de esquemas de autorização, correspondentes às especificações contidas no modelo [LANDWEHR 1981].

Os modelos de segurança correspondem a descrições mais detalhadas do comportamento de um sistema. Eles atuam sempre segundo regras de uma política de segurança estabelecida. Estes modelos são representados na forma de um conjunto de entidades e relacionamentos [GOGUEN, MESAJUER 1982]. Na literatura é comum encontrarmos os modelos divididos em três tipos básicos [SANDHU *et al.* 1996; OSBORN, SANDHU, MUNAWER 2000]:

- Baseados em identidade ou discricionários;
- Baseados em regras ou obrigatórios;
- Baseados em papéis.

2.4.4.1. MODELO DISCRICIONÁRIO

No modelo discricionário, os direitos de acesso aos recursos ou informações são especificados para cada sujeito, pelo proprietário da informação ou recurso. As requisições para utilização de recursos são analisadas por um mecanismo de segurança discricionário e o acesso é concedido de acordo com as regras de autorização [LANDWEHR 1981; AMOROSO 1994].

O modelo discricionário se baseia em conceder e retirar privilégios. O controle poderá ser centralizado, isto é, a autorização é administrada por um controle central, normalmente o administrador do sistema. Podem também ter um controle descentralizado de maneira que o proprietário da informação possui o direito de conceder ou retirar os privilégios.

A adoção deste modelo para aplicação em políticas de segurança tem a vantagem de torná-las flexíveis, fazendo com que tais políticas possam ser utilizadas por vários tipos de sistemas e aplicações comerciais e industriais e, conseqüentemente, atualmente serem as mais utilizadas. Vários modelos foram propostos e implementados em vários sistemas, porém certos requisitos de segurança podem não ser totalmente satisfeito, como no caso em que um usuário com permissão de leitura a determinada informação poderá transferi-la para um objeto de outro usuário que não possui esta permissão. As políticas de controle de acesso discricionárias não permitem classificar os objetos e sujeitos no que diz respeito à confidencialidade. Com isto, não é possível criar uma política que possa restringir os privilégios de acesso a informações confidenciais com base na classificação dos sujeitos, independentemente de especificações determinadas pelo administrador. Este tipo de política é importante quando se deseja limitar a autoridade do administrador, protegendo informações confidenciais, muitas vezes vitais, como no caso dos sistemas militares. Mas sua aplicação

pode ser bem mais abrangente, sendo fundamental em qualquer sistema onde a confidencialidade das informações for importante.

2.4.4.2. MODELOS OBRIGATÓRIOS

Os Modelos obrigatórios têm em seu esquema de autorização um conjunto de regras que expressam um tipo de organização envolvendo a segurança das informações no sistema como um todo. O modelo obrigatório prevê que os usuários, objetos e recursos do sistema sejam rotulados de modo que esses rótulos dos objetos seguem uma classificação específica enquanto os usuários de acesso possuem níveis de habilitação. Os controles que determinam as autorizações de acesso são baseados numa comparação da habilitação do usuário com a classificação do objeto [AMOROSO 1994].

Os modelos obrigatórios que caracterizam as políticas obrigatórias baseiam-se em uma administração centralizada de segurança, a qual dita regras de acesso à informação. As mesmas definem regras e estruturas válidas no âmbito de um sistema, normalmente especificando algum tipo de política multinível. As políticas multiníveis estão baseadas em algum tipo de classificação ao qual estão submetidos os acessos dos sujeitos aos objetos. Elas visam 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.

2.4.4.3. MODELOS BASEADOS EM PAPÉIS (RBAC)

Os modelos Baseados em Papéis (RBAC) têm como objetivo intermediar o acesso dos usuários à informação, com base nas atividades que são por eles desenvolvidas no sistema. A idéia central é que o usuário desempenhe diferentes papéis 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, no RBAC, as permissões são conferidas aos papéis e os usuários são autorizados a exercer papéis. O controle de acesso baseado em papéis facilita a gerência de autorização, porque quando o usuário muda de atribuição – sendo, portanto, desassociado de um papel e assumindo um outro – a manutenção das permissões dos papéis não sofre mudanças. Em geral, o RBAC trabalha com o princípio do mínimo privilégio, um usuário ativa apenas o subconjunto de papéis que precisa para executar uma operação, e esta ativação pode ou não estar sujeita a restrições.

Um modelo unificado denominado RBAC-NIST foi criado para tentar padronizar as várias tendências que têm surgido em modelos baseado a papéis [SANDHU, FERRAILOLO,

KUHN 2000]. Esta família de modelos RBAC-NIST se apresenta estratificada a partir do modelo RBAC básico, evoluindo para outros modelos da família – RBAC Hierárquico, RBAC com Restrições e RBAC Simétrico.

O RBAC básico pode ser visto na Figura 1. Ele implementa a infra-estrutura básica composta por quatro entidades principais: U (Usuários), P (Papéis), A (Autorizações) e S (Sessões). O relacionamento UP caracteriza uma relação muitos para muitos entre usuários e papéis, o que mostra que um usuário pode possuir simultaneamente mais de um papel. O relacionamento PA, também muitos para muitos, é que configura as autorizações que um determinado papel deve possuir. As sessões se relacionam com apenas um usuário por vez, mas permite a ativação de qualquer papel que estiver na relação UP com aquele usuário [FERRAIOLO, BARKLEY, KUHN 1999].

É importante ressaltar que o modelo também caracteriza a autorização como um relacionamento entre os objetos para os quais o acesso é protegido e as formas de se acessar ou modificar esses objetos. Mas apesar disto, o padrão deixa em aberto a interpretação de usuário, papel e autorização para ser especificado em modelos mais detalhados.

O RBAC hierárquico acrescenta ao RBAC básico uma hierarquia entre papéis, que define uma relação de ordem entre os papéis, de forma a explicitar melhor as linhas de autoridade e responsabilidade de uma organização.

O RBAC com Restrições introduz as restrições entre papéis, com a intenção de evitar a concentração de poder nas mãos de um único usuário, visando minimizar a possibilidade de fraude ou dano acidental. Ele está calcado no princípio do mínimo privilégio (suporta a separação de tarefas: o usuário só utiliza os direitos necessários para uma dada tarefa), impondo desta forma restrições na ativação de papéis.

O RBAC Simétrico inclui suporte à revisão de associações permissão-papel, o que torna possível identificar as permissões associadas a um papel e os papéis que possuem determinadas permissões. Ele representa o modelo mais completo, constituindo a união do RBAC hierárquico com o RBAC com restrições.

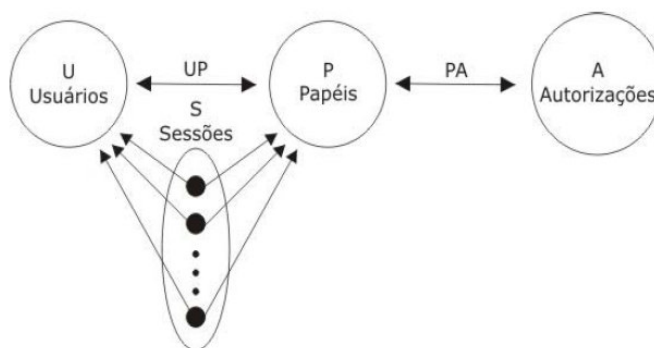


Figura 1. RBAC Básico (Fonte: GUERRA, PAICA, FERNANDES 2004)

2.5 CONCLUSÃO

Neste capítulo foram abordados os conceitos de segurança, onde se leva o entendimento que para se ter um sistema computacional seguro, este tem que atender a algumas propriedades básicas. Os modelos de segurança são formas de descrever as políticas de segurança, determinando o comportamento de entidades administradas por regras, as quais definem uma política de segurança.

Ao longo dos anos, diversos modelos de segurança foram propostos, baseando-se nas mais variadas premissas e visando atender a diferentes aspectos de segurança.

Os modelos discricionário e obrigatório possuem diversas diferenças, onde o modelo obrigatório impõe regras mais rígidas e incontornáveis, mas são menos utilizados em sistemas comerciais, devido ao fato de serem mais difíceis as suas implementações.

A grande vantagem do RBAC em relação aos sistemas de controle de acesso que o precederam, está na administração das autorizações de acesso. Nos modelos anteriores, as autorizações eram atribuídas diretamente aos usuários, o que, de acordo com o número de usuários envolvidos, dificulta a administração das autorizações, tornando esta abordagem pouco escalável. No modelo RBAC, as autorizações de acesso são associadas a um papel e estes papéis são atribuídos aos usuários. Dessa forma, a administração do controle de acesso se torna mais simples e viável para sistemas com grande número de usuários, trazendo inclusive benefícios financeiros em sua utilização.

Outras vantagens que podemos destacar dos modelos baseados em papéis são a separação de tarefas, o princípio do mínimo privilégio e a possibilidade de incorporar, de alguma forma, a hierarquia natural das organizações ao controle de acesso. Nenhum dos outros modelos fornece esta importante facilidade ao administrador de segurança. Devido às

suas características, o modelo baseado em papéis é uma forte tendência na área de segurança de controle de acesso em sistemas computacionais. Nesta dissertação, iremos utilizar como base o modelo baseado em papéis.

No próximo capítulo, serão apresentadas técnicas de integração dos conceitos e soluções de segurança na modelagem de sistemas, incluindo a aplicação de padrões de projeto de segurança e o novo paradigma de desenvolvimento orientado a modelos denominado MDA (*Model Driven Architecture*).

3 MODELAGEM DE SISTEMAS SEGUROS

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, KOCH, PAULS 2004]. À 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 diversidade de requisitos de negócio de cada aplicação, arquitetos de software e analistas necessitam cada vez mais criar seus próprios arcabouços e manter suas integridades e coerência em relação às necessidades do negócio as quais eles atendem. Contudo, na maioria das vezes, os projetistas de sistemas não possuem experiência em segurança.

Além disso, existe uma grande lacuna entre as soluções teóricas e o que é de fato implementado na área de segurança. Um dos problemas que contribuem para a construção de software com segurança fraca é o fato de este requisito ser raramente considerado nos estágios iniciais do desenvolvimento de *software* e ser relegado a segundo plano ao longo do mesmo, provocando problemas de segurança tanto na arquitetura da aplicação quanto na lógica implementada.

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 2000]. 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 projetos. Contudo, para obter os benefícios da utilização de padrões de projeto é necessário saber quando e como utilizá-los a partir de suas definições, pois os desenvolvedores podem acabar introduzindo falhas de segurança. É necessário que haja analistas e programadores com alto grau de conhecimento de padrões de

projeto de segurança e de arcabouços de segurança e esse conhecimento faz com o que o custo do desenvolvimento de sistemas aumente consideravelmente.

Neste contexto, uma abordagem orientada a modelo pode auxiliar o processo de desenvolvimento de sistemas seguros, promovendo uma implementação controlada dos padrões de projeto e, desta forma, garantindo aderência às diretrizes arquiteturais da organização e às questões de segurança.

Este capítulo apresenta técnicas de integração de soluções de segurança nas fases de análise e projeto de sistemas de software. Na Seção 3.1, definimos os principais conceitos de padrões de projeto e apresentamos os principais trabalhos existentes de padrões de projeto de segurança. Na Seção 3.2, definimos os principais conceitos de MDA e apresentamos os principais trabalhos existentes de Segurança com MDA, e na Seção 3.3 concluímos este capítulo apresentando algumas considerações finais.

3.1 PADRÕES DE PROJETO DE SEGURANÇA

Padrões de projeto foram introduzidos como uma forma de identificar e apresentar soluções a problemas recorrentes na programação orientada a objetos. Joseph Yoder e Jeffrey Barcalow [YODER, BARCALOW 1997] foram uns dos primeiros a utilizar esta abordagem como uma tentativa de criar arcabouços de segurança bem projetados. O uso de padrões nesse contexto justifica-se, pois enquanto muitos problemas de segurança são novos ou complicados, uma grande parte de outros problemas são bem conhecidos e possuem soluções bem estabelecidas. No espírito do provérbio “É melhor ensinar a pescar do que dar o peixe”, é melhor explicar como utilizar padrões de projeto para criar arquiteturas de segurança próprias a utilizar arquiteturas prontas que não atendem as necessidades do negócio e que muitas vezes o arquiteto e o projetista têm que modificar para fazer o melhor acoplamento possível.

3.1.1. PADRÕES DE PROJETO

Os padrões de projeto de software, também muito conhecidos pelo termo original, *Design Patterns*, descrevem soluções para problemas recorrentes no desenvolvimento de sistemas de *software* orientados a objetos. Essas soluções são desenvolvidas e conhecidas por especialistas, tornando-se padrões por serem reutilizadas várias vezes em vários projetos, além de terem eficácia comprovada. Os padrões de projeto visam facilitar a reutilização de soluções de *design* - isto é, soluções na fase de projeto do software, sem considerar reutilização de código.

O conceito de padrão de projeto foi criado na década de 70 pelo arquiteto Christopher Alexander. Em seus livros *Notes on the Synthesis of Form*, *The Timeless Way of Building* [ALEXANDER 1979] e *A Pattern Language* [ALEXANDER 1978], ele estabelece que um padrão deve possuir, idealmente, as seguintes características:

- **Encapsulamento:** um padrão encapsula um problema/solução bem definido. Ele deve ser independente, específico e formulado de maneira a ficar claro onde ele se aplica;
- **Generalidade:** todo padrão deve permitir a construção de outras realizações a partir dele;
- **Equilíbrio:** quando um padrão é utilizado em uma aplicação, o equilíbrio estabelece a razão, relacionada com cada uma das restrições envolvidas, para cada passo do projeto. A forma de encontrar este equilíbrio pode ser uma análise racional que envolva uma abstração de dados empíricos, uma observação da aplicação de padrões em artefatos tradicionais, uma série convincente de exemplos e uma análise de soluções ruins ou fracassadas;
- **Abstração:** os padrões representam abstrações da experiência empírica ou do conhecimento cotidiano;
- **Abertura:** um padrão deve permitir a sua extensão para níveis mais baixos de detalhamento;
- **Combinatoriedade:** os padrões são relacionados hierarquicamente. Padrões de alto nível podem ser compostos ou relacionados com padrões que endereçam problemas de nível mais baixo.

Além da definição das características de um padrão, Alexander definiu o formato que a descrição de um padrão deve ter. Ele estabeleceu que um padrão deve ser descrito em cinco partes:

- **Nome:** descrição da solução, mais do que do problema ou do contexto;
- **Exemplo:** uma ou mais figuras, diagramas ou descrições que ilustrem um protótipo de aplicação;
- **Contexto:** descrição das situações sob as quais o padrão se aplica;
- **Problema:** descrição das forças e restrições envolvidas e como elas interagem;
- **Solução:** relacionamentos estáticos e regras dinâmicas descrevendo como construir artefatos de acordo com o padrão, frequentemente citando variações e formas de ajustar a solução segundo as circunstâncias. Inclui referências a outras soluções e o relacionamento com outros padrões de nível mais baixo ou mais alto.

Partindo do trabalho de Alexander, profissionais de software começaram a incorporar esses princípios na criação das primeiras documentações de padrões de projetos como um guia para desenvolvedores iniciantes. O resultado prático foi a publicação de *Design Patterns: Elements of Reusable Object-Oriented Software* [GAMMA *et al.* 1995]. Os autores desse livro são Eric Gamma, Richard Helm, Ralph Johnson e John Vlissides, conhecidos como a "Gangue dos Quatro" (*Gang of Four*) ou simplesmente "GoF". Este livro é a referência principal no assunto para a comunidade de software. Nele são descritos 23 padrões que foram baseados na experiência dos autores.

Posteriormente, vários outros livros tratando do tema foram publicados, como *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development* [LARMAN 2004], que introduziu um conjunto de padrões conhecidos como GRASP (*General Responsibility Assignment Software Patterns*).

Dentre as principais vantagens da utilização de padrões de projetos estão:

- Capturar o conhecimento e a experiência de especialistas em projeto de software;
- Especificar abstrações que estão acima do nível de classes ou objetos isolados ou de componentes;
- Definir um vocabulário comum para a discussão de problemas e soluções de projeto;
- Facilitar a documentação e manutenção da arquitetura do software;
- Auxiliar o projeto de arquiteturas mais complexas;
- Fornecer soluções que já foram testadas e aprovadas;
- Tornar o sistema mais fácil de entender e manter.

3.1.1.1. REPRESENTAÇÃO DE PADRÕES DE PROJETO

A representação de um padrão de projeto deve conter no mínimo o nome (identificação), o problema (quando o uso do padrão pode ser interessante), a solução (como aplicar o padrão de projeto) e as conseqüências (perdas e ganhos ao se aplicar o padrão). Não há um formato único e padronizado para representar padrões de projeto. Com isso, diferentes formatos têm sido utilizados por diferentes autores. Porém, alguns formatos se tornaram mais conhecidos que outros e, conseqüentemente, se tornaram padrões na documentação de novos padrões. Um exemplo de formato comumente utilizado é o da gangue dos quatro, descrito em [GAMMA *et al.* 1995], que contém:

- **Nome:** um nome descritivo e único que ajuda a identificar e referenciar um padrão;
- **Intenção:** uma descrição do objetivo do padrão e a razão para utilizá-lo;
- **Também conhecido como:** outros nomes para o padrão;

- **Motivação:** um cenário contendo um problema e um contexto onde esse padrão possa ser utilizado;
- **Aplicabilidade:** situações onde este padrão possa ser utilizado, o contexto do padrão;
- **Estrutura:** uma representação gráfica do padrão. Diagramas de classe e diagramas de interação podem ser utilizados para este propósito;
- **Participantes:** uma listagem das classes e objetos utilizados no padrão e seus papéis no projeto;
- **Colaboração:** uma descrição de como classes e objetos utilizados no padrão interagem entre si;
- **Conseqüências:** uma descrição dos resultados, efeitos causados pela utilização do padrão;
- **Implementação:** uma descrição de uma implementação do padrão;
- **Código Exemplo:** uma ilustração de como o padrão pode ser utilizado em uma linguagem de programação;
- **Usos conhecidos:** exemplos de utilizações reais do padrão.

3.1.1.2. METODOLOGIA DE UTILIZAÇÃO DE PADRÕES DE PROJETO

Em [GAMMA *et al.* 1995] há um passo a passo para utilizar um padrão de projeto efetivamente:

1. Dedique atenção particular na aplicabilidade e nas conseqüências do padrão para ter certeza de estar utilizando o padrão correto para o seu problema;
2. Entenda perfeitamente as classes e os objetos no padrão e como eles se relacionam entre si;
3. Estude o código como forma de entender como implementar o padrão;
4. Defina as classes, declare as interfaces, estabeleça as heranças, os relacionamentos e defina as variáveis de instancia. Identifique classes existentes na aplicação que o padrão irá afetar e modifique-as de forma correta;
5. Defina nomes para as operações de forma específica à aplicação. Utilize as responsabilidades e colaborações associadas a cada operação como um guia;
6. Implemente as operações garantindo as responsabilidades e colaborações do padrão.

Estes passos são apenas um guia pra começar. Com o tempo, cada pessoa desenvolve sua própria maneira de trabalhar com padrões de projeto.

Padrões de projeto podem parecer abstratos à primeira vista, porém eles se tornam mais concretos à medida que são utilizados. Uma vez que o vocabulário dos padrões de

projeto se torna conhecido, a interação se torna mais precisa e rápida com outras pessoas que utilizam este vocabulário. Por exemplo, é melhor dizer "esta é uma instância do padrão Visitor" do que "este é um código que varre a estrutura e realiza chamadas de retorno, sendo que alguns métodos devem estar presentes, para serem chamados em uma ordem particular e de uma determinada forma".

Padrões de projeto podem aumentar ou diminuir a capacidade de compreensão de um projeto ou de uma implementação. Eles podem diminuir a capacidade de compreensão ao complicar o projeto e/ou diminuir o desempenho enquanto que podem aumentar a capacidade de compreensão ao melhorar a modularidade, separando melhor os conceitos, e simplificando a descrição.

É preciso saber quando e como utilizar padrões de projeto de forma que um padrão de projeto só seja utilizado quando a flexibilidade e os benefícios que ele oferece forem realmente necessários.

3.1.2. PADRÕES DE PROJETO DE SEGURANÇA

Como dito anteriormente, atualmente, existe uma grande lacuna entre as soluções teóricas e o que é de fato implementado na área de segurança. Neste contexto, a abordagem de padrões de projeto pode e deve ser aplicada à segurança como forma de preencher essa lacuna promovendo várias vantagens, entre elas:

- Novatos podem atuar como especialistas;
- Especialistas em segurança podem identificar, nomear e discutir problemas e soluções de modo mais eficiente;
- Problemas podem ser resolvidos de uma forma estruturada;
- Dependências de componentes podem ser identificadas e consideradas de forma apropriada.

Padrões de segurança devem ser utilizados quando há um problema específico em um contexto específico e quando se deseja desenvolver uma arquitetura para resolver este problema.

Yoder e Barcalow [YODER, BARCALOW 1997] foram uns dos primeiros a adaptar a abordagem de padrões de projeto a segurança da informação onde apresentaram os seguintes padrões:

- **Ponto de Acesso único:** prover um módulo de segurança e uma forma de autenticação;

- **Ponto de verificação:** organizar pontos de verificação de segurança e suas repercussões;
- **Papéis:** organizar usuários com privilégios de segurança similares;
- **Sessão:** localizar informação global em um ambiente multi-usuário;
- **Visão completa com erros:** prover uma visão completa aos usuários, exibindo exceções quando necessário;
- **Visão limitada:** permitir que os usuários visualizem somente aquilo a que eles têm acesso.

Atualmente, há vários trabalhos relacionados ao assunto de padrões de projeto de segurança. Dentre eles, podemos citar um guia técnico de padrões de segurança [BLAKLEY, HEALTH 2004] publicado pelo Opengroup [OPENGROUP 2007] que contém a definição de alguns padrões de segurança que foram separados em duas categorias.

A primeira categoria contém padrões de segurança que facilitam a construção de sistemas que estão sempre disponíveis, ou seja, que provêm acesso ininterrupto aos serviços e recursos que eles oferecem aos usuários. Como exemplo, podemos citar os padrões Sistema com Ponto de Restauração (*Checkpointed System*), que visam estruturar um sistema de modo que seu estado possa ser recuperado e restaurado a um estado válido caso haja falha em algum componente. Outro exemplo, é o padrão Sistema Tolerante a falha (*Comparator-Checked Fault-Tolerant System*) que visa estruturar um sistema de forma que uma falha independente em um componente seja detectada rapidamente e que não cause falha no sistema inteiro.

A segunda categoria contém padrões de segurança que facilitam a construção de sistemas que protegem recursos contra uso não autorizado, divulgação ou modificação. Como exemplo, podemos citar os padrões Sistema Protegido (*Protected System*), que visa estruturar um sistema de modo que todo acesso feito aos recursos seja mediado por um guardião que reforce uma política da segurança. Outro exemplo é o Proxy Seguro (*Secure Proxy*) que define o relacionamento entre dois guardiões de duas instâncias de Sistema Protegido no caso onde uma instância está totalmente contida dentro da outra.

Outro trabalho na área foi feito por [FERNANDEZ, PAN 2001], onde são discutidos três padrões que correspondem a modelos de segurança: Autorização, Controle de Acesso Baseado em Papéis e Segurança Multi-nível. O padrão Autorização representa o modelo de matriz de acesso e visa descrever autorizações por entidades computacionais ativas (sujeitos) a recursos passivos (objetos). O padrão Controle de Acesso Baseado em Papéis visa descrever a atribuição de permissões a usuários de acordo com seus papéis em uma

instituição. Já o padrão Segurança Multi-nível 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, entre outros.

Darrel M. Kienzle and Matthew C. Elder construíram um repositório de padrões de segurança com vinte e seis padrões e três mini-padrões. O foco desses padrões está na aplicação de segurança na web. Os padrões estão disponíveis em [KIENZLE, ELDER 2002]. O relatório final do projeto contendo resumo de todos os padrões esta disponível em [KIENZLE *et al.* 2002].

Vários livros foram publicados, entre eles, podemos citar [SCHUMACHER 2003], [SCHUMACHER *et al.* 2005] e [STEEL, NAGAPPAN, LAI 2005]. Este último foi escrito por um grupo da SUN [SUN 2007] e oferece um conjunto de padrões de segurança para aplicações J2EE, serviços Web e gerência de identidade. Já em [SCHUMACHER *et al.* 2005], encontra-se a descrição de vários padrões de segurança separados por tipos, entre esses tipos, podemos citar: Padrões de identificação e autorização, Padrões de modelo de controle de acesso e Padrões de projeto de segurança para aplicações internet.

3.2 SEGURANÇA COM MDA

MDA (*Model Driven Architecture*) é uma abordagem de desenvolvimento de sistemas que se baseia na idéia da separação da especificação das funcionalidades de um sistema dos detalhes de sua implementação [OMG 2003]. Os três objetivos principais da abordagem MDA são portabilidade, interoperabilidade e reutilização através da separação arquitetural de interesses. Neste sentido, MDA define mecanismos para: (i) especificar um sistema independentemente da plataforma que o suporta; (ii) especificar plataformas; (iii) escolher uma plataforma para um sistema; e (iv) transformar a especificação do sistema em código uma plataforma específica. Com a separação de interesses, esta abordagem ajuda a focar nos aspectos de negócio da solução ao invés dos aspectos técnicos.

A idéia de “orientação a modelos” está fundamentada na utilização de modelos para direcionar o entendimento, projeto, construção, instalação e manutenção de sistemas. Apesar de esta idéia ser antiga, a MDA tem se mostrado bastante promissora devido ao diferencial em que os modelos são transformados em código de forma automática e configurável. Desta forma, uma possível mudança de plataforma implica alteração da especificação das transformações, enquanto que a especificação da solução se mantém intacta.

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.

3.2.1. MODEL DRIVEN ARCHITECTURE

O objetivo da engenharia de software é oferecer uma estrutura de processos, métodos e ferramentas para o desenvolvimento de software. Melhorar a produtividade, a manutenibilidade, a integração e a qualidade são alguns dos objetivos a serem alcançados em um ambiente de desenvolvimento de software. Desde o surgimento da engenharia de software, muitos métodos, processos e ferramentas foram propostos para alcançar esses objetivos.

O desenvolvimento de sistemas orientado a modelos (*Model Driven Development*) (MDD) [KLEPPE, WARMER, BAST 2002] é um exemplo. Ele é uma abordagem de desenvolvimento de sistemas onde, ao longo do ciclo de vida do projeto, os modelos são refinados progressivamente até conterem informações suficientes para geração de código.

Um modelo é uma representação simplificada de algum conceito, com o objetivo de observação, manipulação e entendimento sobre tal conceito [MELLOR *et al.* 2004]. No desenvolvimento de *software*, modelos são criados com o objetivo de diminuir a complexidade inerente ao desenvolvimento.

Antes do *MDD*, os modelos eram elaborados com o objetivo de facilitar a comunicação entre os desenvolvedores e como uma espécie de rascunho para o projeto de construção do *software* [MELLOR *et al.* 2004]. Com o *MDD*, os modelos agora necessitam de uma maior precisão para serem parte direta do processo de produção do *software*, isto é, estes modelos são artefatos de entrada para a construção do sistema através da execução de transformações.

O principal objetivo dessa abordagem é tornar a lógica de aplicação independente da tecnologia da plataforma utilizada. Uma plataforma pode ser definida como um conjunto de subsistemas e tecnologias que provêem serviços coerentes através de interfaces e padrões de

utilização. Os sistemas desenvolvidos para a plataforma utilizam estes serviços, sem que seja necessário o conhecimento dos detalhes inerentes à implementação dos mesmos. Um exemplo de plataforma disponível atualmente no mercado é o *JEE (Java Enterprise Edition)* [JAVA 2007] e o . [NET 2007].

A Arquitetura Orientada a Modelos (*Model Driven Architecture*) é uma instanciação do MDD. Ela constitui uma solução de desenvolvimento de sistemas independente de tecnologia. Essa abordagem é baseada em padrões definidos pelo *Object Management Group* (OMG) [OMG 2003] e a linguagem de modelagem de sistemas adotada é a *Unified Modeling Language* (UML) [UML 2007].

A noção central do MDA é criar modelos em diferentes níveis de abstração estabelecendo uma ligação entre eles até chegar à implementação do sistema. Alguns destes modelos são independentes de plataforma, enquanto outros são voltados a uma plataforma específica.

A abordagem MDA recomenda a criação de modelos em três níveis de abstração na especificação de sistemas:

- **Modelo Independente de Computação** (*Computation Independent Model - CIM*): é a representação do sistema do ponto de vista independente de computação. Esse modelo foca nos requisitos do sistema, onde os detalhes de estrutura e processamento estão ocultos ou ainda indeterminados. Esse modelo é também conhecido como modelo de domínio. Serve, principalmente, como meio de comunicação entre os especialistas de domínio e requisitos e os especialistas no projeto e construção do sistema;
- **Modelo Independente de Plataforma** (*Platform Independent Model*) (PIM): é um modelo de alto nível de abstração que representa as funcionalidades de um sistema, com foco nas regras de negócio, utilizando-se de uma especificação que não dependa de plataforma. O PIM é transformado em um ou mais modelos PSM;
- **Modelo Específico de Plataforma** (*Platform Specific Model*) (PSM): é a complementação do Modelo Independente de Plataforma com detalhes tecnológicos específicos da plataforma de implementação do sistema.

MDA permite desenvolvimento incremental e iterativo, já que os mapeamentos entre modelos podem ser aplicados mais de uma vez. Isto permite expressar cada aspecto do sistema no nível de abstração apropriado mantendo os vários modelos em sincronia. Um mapeamento entre modelos assume um ou mais modelos como entrada e produz um modelo como saída. As regras definidas no mapeamento determinam as transformações entre os modelos.

A transformação de um modelo independente de plataforma em um modelo específico para uma plataforma é feita através da execução de definições de transformação descritas em uma linguagem de transformação. Uma definição de transformação é um conjunto de regras de mapeamento que descrevem como um modelo pode ser mapeado para um modelo alvo. Estes modelos podem estar especificados na mesma linguagem ou em linguagens distintas. Uma regra de mapeamento é uma descrição de como uma ou mais estruturas definidas em uma linguagem origem podem ser transformadas para uma ou mais construções na linguagem alvo. Uma ferramenta de transformação executa uma definição de transformação e gera um PSM a partir de um PIM. A Figura 2 mostra como os modelos, linguagens e ferramentas estão relacionados.

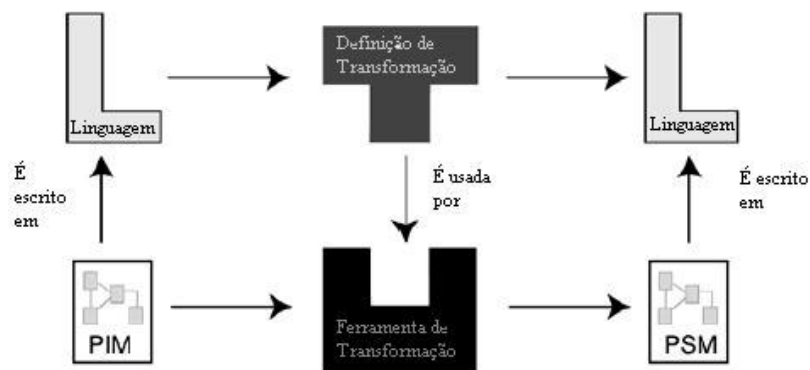


Figura 2. Relação entre modelos, linguagens e ferramentas de transformação (Fonte: KLEPPE, WARMER, BAST 2002 traduzida pela autora)

Analogamente, um PSM pode ser mapeado para código fonte utilizando uma ferramenta de transformação. Neste caso, é possível também a utilização de uma ferramenta de automação que não seja guiada por transformações. O mapeamento é trivial porque o modelo específico para uma plataforma possui toda informação necessária para a geração do código.

Os sistemas são projetados com um nível de abstração onde detalhes específicos de plataformas são escondidos. O projeto abstrato permite mais flexibilidade e agilidade caso mudanças de tecnologias ocorram, já que os requisitos do sistema são descritos de maneira formal e por isso são facilmente recuperados.

A separação entre a arquitetura do sistema e a implementação protege o investimento realizado no sistema caso ocorram mudanças na tecnologia. A inserção de detalhes tecnológicos é feita através da transformação de um modelo independente de plataforma para um modelo dependente de plataforma, onde essa transformação pode ser feita tanto manualmente como automaticamente.

As ferramentas de geração automática devem observar os elementos do modelo e aplicar a eles boas práticas de programação, como padrões de projeto. A aplicação dessas boas práticas garante a qualidade do artefato gerado. Outros requisitos para essas ferramentas são a possibilidade de modificação do artefato gerado e a permanência do código inserido pelo desenvolvedor. Assim, essas ferramentas evitarão que o modelo seja abandonado durante a fase de desenvolvimento do sistema devido a problemas relacionados ao prazo de entrega do projeto.

3.2.1.1. A UTILIZAÇÃO DA UML NA DESCRIÇÃO DOS MODELOS

Atualmente, a UML (*Unified Modelling Language*) [UML 2007] é a notação mais utilizada na modelagem de sistemas orientados a objetos. A UML é uma linguagem gráfica para visualizar, especificar, construir e documentar os artefatos de sistemas orientados a objetos. Ela é basicamente composta de elementos e diagramas. Os elementos compõem o modelo do sistema e os diagramas são uma forma de visualizar apenas um subconjunto destes elementos. Através dos diagramas, podemos ter diferentes visões de um mesmo modelo.

A UML é uma linguagem adequada para a especificação de modelos, pois provê mecanismos para criação de projetos precisos suficiente para dirigir a geração de código e de projetos abstratos suficiente para permanecer independente de plataforma. Além disso, a UML tem sido largamente utilizada e aceita como linguagem padrão para modelagem de sistemas.

A UML é independente de tecnologia, entretanto a representação de detalhes dependentes de plataforma pode ser feita através da utilização de um Perfil UML [KLEPPE, WARMER, BAST 2002]. Perfis são conjuntos de extensões para UML, que especializam um modelo para uso em um domínio particular. Os mecanismos de extensão mais utilizados são os estereótipos e os valores etiquetados (*tagged values*).

Estereótipos definem novos tipos de elementos de modelo ampliando a semântica dos tipos existentes ou de classes em um meta-modelo UML. A notação consiste do nome do estereótipo escrito entre os sinais de maior e menor, por exemplo, <<nomeDoEstereótipo>>. A notação é anexada a um elemento do modelo, que ganha o significado descrito pelo estereótipo. Anexar um valor etiquetado a um elemento do modelo é uma maneira de explicitamente definir uma propriedade para esse elemento. Um valor etiquetado é um par nome-valor. A notação correspondente é {*nome=valor*} com o valor sendo atribuído ao nome. Esses mecanismos são utilizados para marcar os elementos do PSM e tornam o modelo específico para plataforma.

Os elementos de uma determinada plataforma e a maneira como eles se relacionam são descritos no perfil, facilitando a modelagem e permitindo checagem do modelo. A UML 2.0 permite, através da criação de perfis, que vendedores ou usuários finais configurem suas ferramentas de UML 2.0 para suportar explicitamente vários estilos de arquiteturas. Além disso, nesta versão da UML passou a existir uma maior integração com o MOF [MOF 2002].

O MOF possibilita a definição de linguagens de modelagem e permite a construção de ferramentas para a definição dessas linguagens. Adicionalmente, o MOF especifica um conjunto padrão de mecanismos de acesso e gerenciamento de modelos que foram criados utilizando qualquer linguagem descrita em um modelo MOF. A UML 2.0 define um padrão para o acesso e o gerenciamento de modelos UML baseados no meta-modelo MOF para UML.

Estes mecanismos de MOF são genéricos e podem ser realizados de maneiras diferentes. Atualmente, o OMG padronizou mapeamentos em XML, Java, e CORBA®. O mapeamento XML é chamado *XML Metadata Interchange (XMI)*, e é usado tipicamente para transferência de metadados entre ferramentas *MOF-compliant*. Os mapeamentos Java e CORBA resultam em APIs que são úteis para trocar a informação entre ferramentas de uma maneira mais granular.

A existência de mecanismos padrões possibilita a integração entre as ferramentas MDA, permitindo ao usuário o intercâmbio entre ferramentas. Antes do MOF, as ferramentas de cada vendedor usavam mecanismos proprietários, este era um problema tanto para os usuários, que gostariam de utilizar várias ferramentas no processo de desenvolvimento, como também para os fabricantes, porque se esforçavam para adicionar ferramentas novas ao conjunto de ferramentas existentes, para integrar suas ferramentas com as de um outro vendedor, ou para integrar os tipos diferentes de meta-dados usados em um projeto. Através dos seus mapeamentos, o MOF soluciona estes problemas, permitindo que ferramentas compartilhem informação sobre modelos UML.

3.2.1.2. MAPEAMENTO ENTRE MODELOS

Uma transformação é o mapeamento de um modelo origem para um modelo alvo seguindo as regras especificadas na definição da transformação. As definições são descritas em uma linguagem de transformação e são executadas por ferramentas apropriadas.

Na fase de geração automática dos artefatos, o modelo origem é mais abstrato do que o modelo alvo. Na fase de criação de modelos a partir dos artefatos existentes, o modelo

origem é mais refinado do que o modelo alvo. Abstração pode ser definida por supressão de detalhes irrelevantes para o contexto e refinamento é um antônimo para abstração.

Detalhes podem ser adicionados ao longo do desenvolvimento, através da criação de novos modelos cada vez mais refinados [OMG 2003]. Dessa forma o modelo alvo de uma transformação pode se tornar o modelo origem para uma outra, isto porque, o termo independência de plataforma pode ter significado diferente dependendo do ponto de vista. Exemplo, um modelo de componentes distribuídos gera um modelo de componentes CORBA, que em um segundo momento serve de entrada para uma transformação que gera um modelo de componentes CORBA específicos de uma determinada linguagem de programação. Este comportamento pode ser observado na Figura 3.

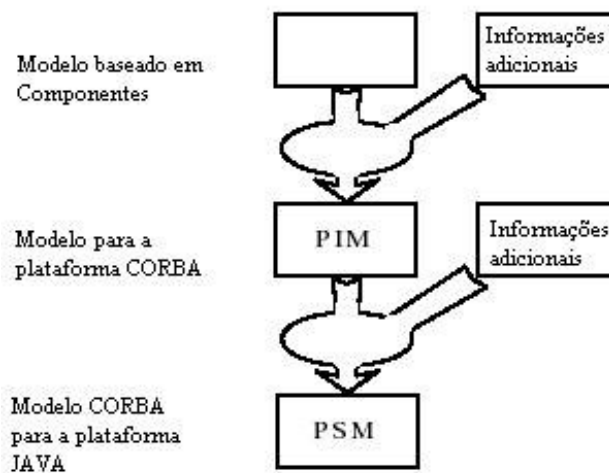


Figura 3. A aplicação do padrão MDA (Fonte: KLEPPE, WARMER, BAST 2002 traduzida pela autora)

O framework MDA não define como as transformações devem ser realizadas, apenas indica, em idéias gerais, quatro métodos de transformações de modelo [OMG 2003]. Esses métodos são explicados a seguir.

- Transformação manual: praticamente idêntico à forma como o projeto de software vem sendo realizado até então, onde decisões são tomadas para criar um projeto que proporcione uma implementação em conformidade com os requisitos previamente definidos. A contribuição da MDA, neste caso, vem da distinção explícita de um modelo independente de plataforma, com o resultado da transformação que é um modelo específico de uma plataforma.
- Transformação de PIM preparado com perfil: um modelo PIM é manipulado por um especialista, que realiza marcações utilizando um perfil UML independente de

plataforma. Posteriormente, o PIM é transformado para um PSM expresso por um segundo perfil UML, este específico de plataforma.

- Transformação utilizando padrões e marcações: padrões podem ser utilizados para definir mapeamentos. Marcações correspondem a elementos desses padrões. Os elementos de um PIM são marcados e transformados de acordo com o padrão estabelecido no mapeamento, produzindo um PSM. Uma outra possibilidade é utilizar regras especificando que todos os elementos no PIM, que seguem um determinado padrão, serão transformados em instâncias de um outro padrão, no PSM.
- Transformação automática: existem contextos onde um PIM pode conter toda a informação necessária para a implementação, ou seja, o PIM é completo em relação a sua classificação, estrutura, invariantes, pré e pós-condições. Nestes casos, o desenvolvedor pode especificar o comportamento diretamente no modelo, utilizando uma linguagem de ações. Essa linguagem faz com que o PIM seja computacionalmente completo e automaticamente transformado para o código-fonte do programa.

Com relação às plataformas, as transformações podem ser classificadas em três tipos básicos:

- PIM–PIM: são transformações realizadas sobre os modelos independentes de plataforma, sem aplicar conceitos de plataforma (e.g.: aplicação de padrões de projeto);
- PIM–PSM: aplicam conceitos da plataforma a um modelo independente de plataforma ou tecnologia (e.g.: transformação de um modelo de classes em um modelo de componentes na arquitetura EJB);
- PSM–PSM: realizam um refinamento no modelo específico de plataforma. Um exemplo é a aplicação dos padrões de projeto específicos para a plataforma J2EE [SUN 2007].

Com essas transformações, um modelo pode ser desenvolvido, refinado através de transformações PIM-PIM, convertido para uma plataforma (PIM-PSM) e refinado ainda mais, em direção à implementação do software. Em um outro momento, caso exista a necessidade de gerar a implementação desde modelo em uma outra plataforma, o modelo inicial (PIM) pode sofrer outras transformações PIM-PSM para a nova plataforma, e assim por diante.

As transformações podem ser especificadas através da definição de mapeamentos entre os modelos transformados. Um mapeamento MDA provê especificações para

transformação do PIM para o PSM de uma plataforma particular. O modelo da plataforma determina a natureza do mapeamento.

O framework MDA define três categorias básicas de mapeamentos, são elas:

- **Mapeamentos de Tipos do Modelo:** esta categoria inclui mapeamentos definidos de acordo com os tipos dos elementos dos modelos. Um mapeamento entre PIM e PSM, é descrito a partir dos tipos existentes no PIM e os tipos existentes no PSM. Numa transformação entre modelos de classes da UML e um modelo relacional, um possível mapeamento poderia ser definido entre o tipo classe e o tipo tabela relacional. Com isso, todas as classes seriam mapeadas para tabelas;
- **Mapeamentos de Instâncias do Modelo:** nesta categoria, os elementos mapeados são identificados através de marcações que indicam como a transformação será realizada. Na transformação de um PIM para um PSM, o mapeamento pode definir uma marcação que representa um conceito do modelo PSM, e que poderá ser aplicada aos elementos do modelo PIM, para informar como o elemento marcado deve ser transformado. A diferença para a categoria anterior está no fato de elementos que são instâncias de um mesmo tipo poderem ser mapeados de forma diferente, de acordo as marcações realizadas.
- **Mapeamentos Combinados entre Tipos e Instâncias:** esta categoria envolve mapeamentos que combinam as características dos dois tipos anteriores. Assim, um mapeamento entre tipos dos modelos, pode ser, de certa forma, configurado através de marcações que indicam características presentes no PSM que não podem ser definidos através dos tipos presentes no PIM.

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 (e.g.: todos os elementos do tipo classe da UML para elementos do tipo tabela em um meta-modelo relacional). Com isso, a transformação é sempre executada de forma determinística e guiada por informações independentes de plataforma [OMG 2003]. No entanto, o engenheiro pode ter a necessidade de utilizar as marcações para adicionar informações ao modelo, que serão utilizadas no processo de transformação, por exemplo, para especificar características não funcionais que não podem ser informadas através dos tipos existentes no PIM.

Uma outra distinção importante é a categorização com relação aos artefatos transformados. Existem transformações que manipulam exclusivamente modelos, conhecidas

como transformações modelo-modelo. Já outras transformações podem gerar o código-fonte a partir de um modelo de entrada, definidas como transformações modelo-texto.

3.2.2. ANÁLISE DOS TRABALHOS EXISTENTES

Algumas abordagens já utilizam MDA para desenvolver arcabouços de segurança. Em [BASIN, DOSER 2005], é apresentada uma abordagem MDA para engenharia de segurança, chamada segurança dirigida a modelo (*Model Driven Security*). Esta abordagem utiliza uma linguagem de modelagem baseada na UML chamada SecureUML [BASIN, DOSER 2002], que integra controle de acesso baseado em papéis (*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 infra-estruturas de controle de acesso.

Segurança dirigida a modelo provê métodos e ferramentas para integrar segurança no processo de desenvolvimento. A idéia 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 modelo de projeto e segurança (*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.

A Figura 4 representa a abordagem de segurança dirigida a modelo, onde o modelo do projeto é combinado com o modelo de segurança e automaticamente transformado na arquitetura do sistema. O modelo do projeto é representado com um digrama de classes que é enriquecido com novos elementos de modelagem representando papéis e permissões. Esta combinação pode ser utilizada para definir uma política de controle de acesso de uma aplicação.

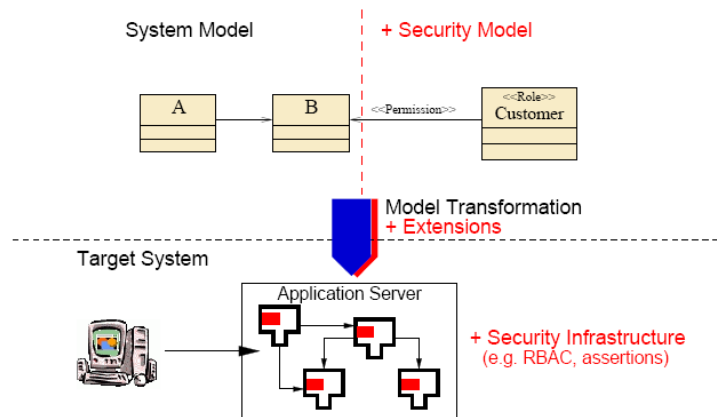


Figura 4. Segurança dirigida a modelo (Fonte: BASIN, DOSER 2005)

Em [BASIN, DOSER 2005] foi construída uma ferramenta que suporta a modelagem da política de controle de acesso e que gera infra-estruturas de segurança que são compatíveis com diferentes padrões para sistemas baseados em componentes como J2EE/EJB e .NET. A utilização desta ferramenta promove algumas vantagens, como por exemplo, a simplificação da especificação da política de segurança, identificação das falhas de segurança durante o processo de desenvolvimento, a implementação pode ser mantida de forma consistente com a política de segurança modelada e implementações podem ser migradas para novas plataformas mudando as regras de transformações utilizadas.

A Figura 5 exemplifica a modelagem de um sistema de acordo com a linguagem SecureUML através de um diagrama de classes. O exemplo dado é uma aplicação bancária simplificada. As entidades no exemplo são: *Account* (conta), que contém os atributos *owner* (dono) e *balance* (*saldo*) assim como métodos *withdraw* (retirar) e *desposit* (depositar). As contas, seus atributos e seus métodos representam os recursos que necessitam proteção. Além disso, há três papéis, que formaliza tipos diferentes de usuários: *Customer* (clientes), *Employee* (empregados) e *Manager* (gerentes). Neste contexto, a política de segurança será:

(P1) Cada empregado pode ler informação associada com todas as contas assim como criar novas contas. Além disso, ele pode alterar o dono de uma conta;

(P2) Em adição às permissões dos empregados, cada gerente pode excluir contas;

(P3) Cada cliente pode ler todas as informações associadas a suas próprias contas.

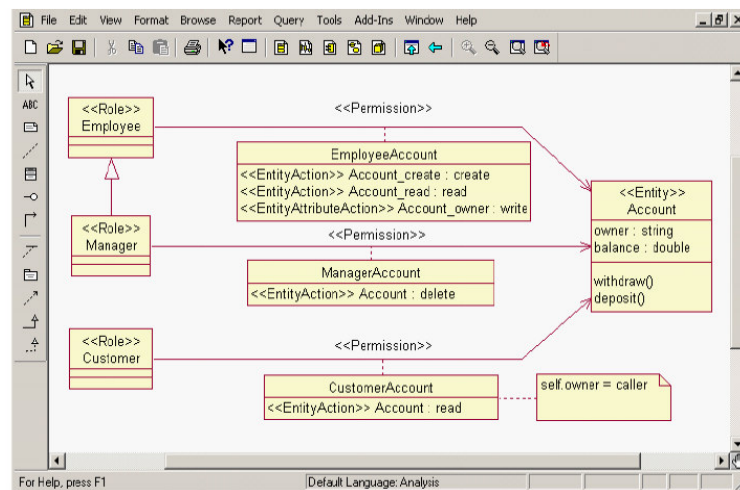


Figura 5. Modelando um cenário de um banco (Fonte: BASIN, DOSER 2005)

Classes são utilizadas para definir a entidade *Account* (Conta) com seus atributos e métodos e para definir os papéis *Customer* (Cliente), *Employee* (Empregado) e *Manager* (Gerente). *Manager* (Gerente) é um sub papel de *Employee* (Empregado), o que significa que ele herda todas as permissões de *Employee* (empregado). As classes do meio conectam os papéis aos recursos de forma a modelar a política de controle de acesso. A permissão de um *Employee* (Empregado) para acessar um *Account* (Conta) é definida como uma associação com o estereótipo `<<Permission>>` entre as duas classes. Para restringir essa permissão, a associação é aplicada à classe *EmployeeAccount* (ContaEmpregado). Um atributo da classe de associação especifica o tipo de acesso ao recurso protegido.

No exemplo, (P1) é modelado da seguinte forma: a classe *EmployeeAccount* (ContaEmpregado) contém três atributos. Dois deles contêm o estereótipo *EntityAction* e possuem os tipos *create* (criar) e *read* (ler), respectivamente. Isto formaliza que um empregado possui o direito de criar novas contas e ler. O terceiro atributo na classe de associação *EmployeeAccount* (ContaEmpregado) expressa que um empregado pode alterar o valor do atributo *owner* (dono) de uma conta.

(P2) é modelado como uma herança entre papéis e definindo uma permissão adicional com o tipo de ação *delete* (deletar) entre *Manager* (gerente) e *Account* (Conta).

Para modelar (P3), é necessário formalizar que cada cliente pode ler contas, mas somente as contas que ele é dono. A permissão de ler contas é feita da mesma forma que (P1). Para expressar a restrição de dono, é necessário modelar uma *constraint*, que nesse caso, é especificada utilizando OCL (*Object Constraint Language*), que faz parte da UML. Neste

exemplo, *CustomerAccount* (contaCliente) é restringida através da *constraint* `self.owner = caller`.

Como na primeira abordagem apresentada [BASIN, DOSER 2005], em [FINK, KOCH, PAULS 2004] é apresentada uma abordagem MDA para desenvolver políticas de controle de acesso em sistemas distribuídos. Os modelos são expressos como modelos MOF [MOF 2002] enriquecidos por perfis UML. MOF é um padrão de meta linguagem definido pela OMG a partir do qual onde outras linguagens de modelagem podem ser especificadas. A vantagem do MOF é fazer com que a definição de meta-modelos seja independente do domínio da aplicação e prover um conjunto de conceitos conciso e único para a definição de meta-modelos. Além disso, múltiplos meta-modelos podem ser gerenciados pelo MOF e relações entre eles podem ser utilizadas como base para uma transformação entre modelos.

Em [FINK, KOCH, PAULS 2004], um modelo de controle de acesso é especificado por um meta-modelo independente de domínio e de plataforma. Este meta-modelo pode ser instanciado em um domínio de aplicação. Além disso, o meta-modelo pode ser refinado para um meta-modelo específico de plataforma (CORBA, J2EE, SOAP). A combinação de ambos resulta em um modelo específico de plataforma contendo uma aplicação de controle de acesso com a implementação específica de alguma plataforma.

O modelo de controle de acesso utilizado é o VBAC (*View-Based Access Control*) [BROSE 2001], que é uma extensão do RBAC para sistemas distribuídos. VBAC é um modelo de controle de acesso feito para suportar o projeto e gerenciamento de políticas de controle de acesso em sistemas orientados a objeto. A principal característica do VBAC é ter uma visão dos direitos de acesso, que são as permissões ou negações para operações de objetos distribuídos. Visões são atribuídas aos principais, isto é, a sujeitos individuais ou papéis, e um principal possui acesso a uma operação de um objeto se ele possui uma visão do objeto com a permissão para chamar a operação. O principal não possui acesso se a operação está explicitamente negada em outra visão deste objeto que está disponível ao perfil ou se nenhuma permissão for achada.

O processo de desenvolvimento é feito da seguinte forma: o analista cria o modelo independente de plataforma VBAC (VBAC PIM). Após a fase de especificação, é necessário escolher a tecnologia que será utilizada para a implementação. Os modelos independentes de plataforma são mapeados para os modelos específicos de plataforma. Para mapear os modelos relacionados ao controle de acesso, o VBAC-PIM é compilado para o VBAC-PSM (Modelo específico de plataforma VBAC). Por ultimo, o VBAC-PSM é utilizado para gerar a infraestrutura. Esses passos podem ser visualizados na Figura 6.

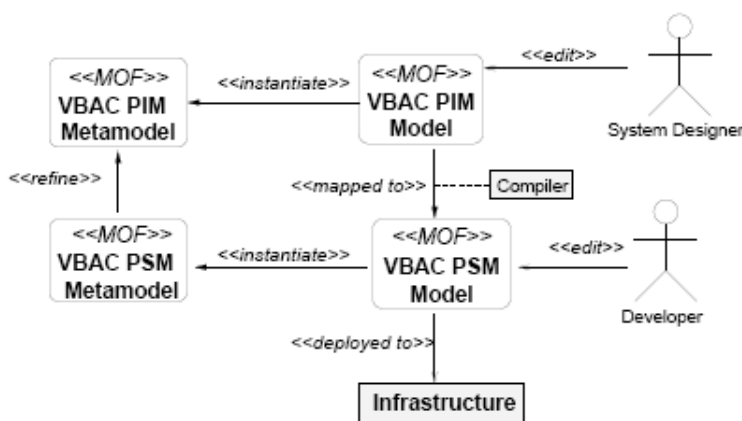


Figura 6. Modelos MDA-VBAC (Fonte: FINK, KOCH, PAULS 2004)

Em [JIN 2006], é apresentada um arcabouço que utiliza a abordagem MDA juntamente com perfis UML para construir aplicações RBAC e especificações de segurança para sistemas distribuídos. Além disso, é mostrado um estudo de caso exemplificando como os perfis UML apresentados podem ser utilizados nas fases iniciais do sistema gerando automaticamente as especificações de segurança do sistema no formato XACML (*eXtensible Access Control Markup Language*).

XACML [GODIK, MOSES 2003] foi desenvolvida pelo OASIS [OASIS 2007] e descreve um formato para a definição de políticas de acesso em XML. O padrão XACML define linguagens de marcação que permitem especificar políticas de segurança, requisições e respostas para decisões de controle de acesso, permitindo a organizações utilizarem essas políticas para controlar acesso a conteúdos e informações protegidas.

O funcionamento proposto por essa especificação baseia-se em duas figuras principais: PEP (*Policy Enforcement Point*) e PDP (*Policy Decision Point*). O PEP é a entidade responsável por formular a solicitação de acesso. O PEP, dessa forma, envia a requisição ao PDP, que é responsável por avaliar a requisição. O PDP localiza as políticas aplicáveis e, baseado na avaliação das mesmas, informa a decisão de acesso, autorizando ou não a requisição.

Em fevereiro de 2005, a OASIS aprovou o padrão RBAC XACML [RBAC XACML 2004] que define um perfil para o uso do XACML junto com os requisitos do RBAC. Este padrão especifica quatro tipos de políticas para construir uma solução RBAC:

- Permission <PolicySet> ou PPS : PPS é uma coleção de permissões associadas a um papel;
- Role <PolicySet> ou RPS: RPS conecta um papel aos seus PPS correspondentes que contêm as permissões atuais associadas ao papel;

- Role Assignment <Policy> ou <PolicySet>: este tipo é opcional. É utilizado para responder a questão se um sujeito tem permissão para fazer parte de um papel;
- HasPrivilegeOfRole <Policy>: este tipo é opcional. É utilizado para responder a questão se um sujeito possui privilégios associados ao papel.

O processo de desenvolvimento é feito da seguinte forma: o desenvolvedor cria o modelo independente de plataforma (RBAC XACML PIM) baseado nos requisitos de controle de acesso da aplicação. Depois, o RBAC XACML PIM é transformado no modelo específico de plataforma (RBAC XACML PSM) que é utilizado para gerar os arquivos da infra-estrutura de segurança. Após a geração automática, o desenvolvedor implementa as partes que faltam, compila e testa o sistema. Neste trabalho, foi utilizado o pacote SunXACML, que foi projetado e desenvolvido pela empresa *Sun Microsystems*, é uma API que implementa a especificação XACML. Essa biblioteca é constituída por um conjunto de classes Java que interpretam a linguagem XACML. O Sun XACML facilita o desenvolvimento de PDPs e PEPs, fornecendo rotinas úteis para a utilização destas funções.

Em [JIN 2006], uma ferramenta de modelagem RBAC XACML foi projetada e implementada como um componente acoplável do Eclipse [ECLIPSE 2007] e utiliza os arcabouços EMF [EMF 2007] e GEF [GEF 2007]. Esta ferramenta foi elaborada para demonstrar a abordagem MDA proposta. Ela permite que o usuário crie modelos visuais para aplicações RBAC desde as fases iniciais do sistema e utilize técnicas de geração de código para automatizar a construção de sistemas a partir desses modelos.

Esta ferramenta também provê: (1) um editor visual que permite o usuário visualizar e editar modelos RBAC XCAML graficamente; (2) transforma e gera automaticamente arquivos de segurança no formato XACML; (3) cria um modelo EMF para usuário, papel e recursos nos modelos RBAC XACML e (4) valida automaticamente o modelo para evitar erros no projeto.

3.3 CONCLUSÃO

Neste capítulo, vimos que 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 2000].

Vimos que uma das abordagens que pode auxiliar arquitetos e projetistas a construir sistemas seguros consiste no uso de padrões de projetos. Apresentamos as definições de padrões de projeto e descrevemos os principais trabalhos de padrões de projeto de segurança.

Contudo, para obter os benefícios da utilização de padrões de projeto é necessário saber quando e como utilizá-los a partir de suas definições, pois os desenvolvedores podem acabar introduzindo falhas de segurança.

Vimos que uma abordagem orientada a modelo pode auxiliar o processo de desenvolvimento de sistemas seguros, promovendo uma implementação controlada dos padrões de projeto e, desta forma, garantindo aderência às diretrizes arquiteturais da organização e às questões de segurança.

Neste capítulo apresentamos uma visão geral sobre a teoria e os conceitos relacionados à Arquitetura Orientada por Modelos (MDA) e apresentamos os principais trabalhos de MDA com segurança presentes na literatura.

Todos os trabalhos apresentados até o momento, permitem a definição de políticas de controle de acesso utilizando perfis UML, onde os modelos são modelos de permissões. Porém, o fato da segurança do sistema ser ativada através da modelagem da política de controle de acesso feita manualmente, faz com que a segurança fique suscetível a erros humanos à medida que o fator humano interfere na robustez de segurança em qualquer ponto onde intervenção manual é requerida.

Nesse contexto, concluímos que seria necessário desenvolver uma abordagem de MDA com segurança que fosse focada nas regras de transformação entre modelos e não na modelagem da política de segurança. Esta nova abordagem será apresentada no próximo capítulo.

Em todas essas abordagens a definição de políticas de controle de acesso é feita utilizando-se perfis UML que estendem a UML com elementos específicos que representam modelos de permissões. Nessas abordagens, o modelo do 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.

Um dos problemas com esse tipo de abordagem é que o fator humano pode interferir na robustez de segurança em qualquer ponto onde intervenção manual é requerida. Desta forma, a partir do momento em que a segurança do sistema é ativada através da modelagem da política de controle de acesso feita manualmente, ela se torna suscetível a erros humanos.

Além disso, a junção dos dois modelos, o de negócio e o de segurança, pode provocar uma sobrecarga de atribuições ao desenvolvedor do sistema, elevando a complexidade do desenvolvimento e, conseqüentemente, 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.

4 DESCRIÇÃO DO ARCABOUÇO ARCHIMDAS

No Capítulo 3, apresentamos algumas abordagens que integram requisitos de segurança no desenvolvimento orientado a modelos (MDA). Em todas essas abordagens a definição de políticas de controle de acesso é feita utilizando-se perfis UML que estendem a UML com elementos específicos que representam modelos de permissões. Nessas abordagens, o modelo do 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.

Um dos problemas com esse tipo de abordagem é que o fator humano pode interferir na robustez de segurança em qualquer ponto onde intervenção manual é requerida. Desta forma, a partir do momento em que a segurança do sistema é ativada através da modelagem da política de controle de acesso feita manualmente, ela se torna suscetível a erros humanos.

Além disso, a junção dos dois modelos, o de negócio e o de segurança, pode provocar uma sobrecarga de atribuições ao desenvolvedor do sistema, elevando a complexidade do desenvolvimento e, conseqüentemente, 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.

Visando tratar desses problemas, desenvolvemos uma abordagem que utiliza MDA para desenvolver arcabouços de controle de acesso focado nas regras de transformação entre modelos e não na modelagem da política de segurança (Figura 7).

O arcabouço de segurança ArchiMDAs é dividido em dois módulos, um responsável pelas regras de segurança, que chamamos de **módulo de regras de segurança** e outro responsável pela administração de segurança específica de cada aplicação, que chamamos de **módulo administrativo de segurança**.

O módulo de regras de segurança é responsável pelas regras de autenticação e autorização dos artefatos da aplicação. 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 projetista deve focar somente no negócio, sem se preocupar com a segurança. Os requisitos de segurança serão acrescentados

ao sistema de forma automatizada e ortogonal ao desenvolvimento das regras de negócio, através dos mecanismos oferecidos pelo ArchiMDAs.

Já o módulo administrativo de segurança representa o modelo de segurança da aplicação, que depende da política de controle de acesso da organização. Visando facilitar o desenvolvimento da aplicação, uma aplicação administrativa de segurança é gerada automaticamente considerando uma política de segurança padrão, de forma que só é necessário criar sua própria aplicação caso este não esteja de acordo com a política de segurança da organização.

As próximas seções detalham esta abordagem e estão organizadas da seguinte forma: a Seção 4.1 apresenta os requisitos de segurança que foram levantados para a construção do arcabouço de segurança ArchiMDAs; a Seção 4.2 descreve o modelo de segurança denominado RBAC estendido que será utilizado; a Seção 4.3 apresenta os padrões de projeto aplicados ao arcabouço ArchiMDAs de acordo com os requisitos levantados; a Seção 4.4 descreve o metamodelo de segurança que foi construído a fim de atender os requisitos levantados; a Seção 4.5 apresenta a arquitetura e o processo de construção de arcabouços de segurança de acordo com a abordagem apresentada. A Seção 4.6 encerra o capítulo, apresentando algumas considerações finais.

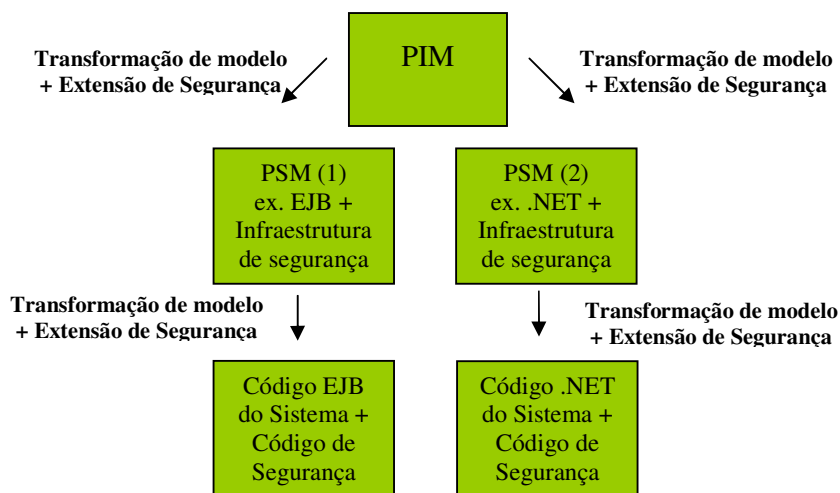


Figura 7. Transformações em MDA com extensões de segurança

4.1 REQUISITOS DE SEGURANÇA

A segurança referente ao controle de acesso é composta dos processos de autenticação, autorização e auditoria. Neste contexto o controle de acesso pode ser entendido como a habilidade de permitir ou negar a utilização de um objeto (uma entidade passiva, como um sistema ou arquivo) por um sujeito (uma entidade ativa, como um indivíduo ou um processo).

A autenticação faz parte de um processo de dois passos que determina quem pode acessar determinado sistema. Durante a identificação o usuário diz ao sistema quem ele é (normalmente através de um nome de usuário). Durante a autenticação a identidade é verificada através de uma credencial (uma senha, por exemplo) fornecida pelo usuário.

A autorização define quais direitos e permissões o usuário do sistema possui. Após o usuário ser autenticado o processo de autorização determina o que ele pode fazer no sistema.

A auditoria registra o que o usuário fez, é uma referência à coleta da informação relacionada à utilização, pelos usuários, dos recursos de um sistema. Esta informação pode ser utilizada para gerenciamento, planejamento, cobrança e etc.

Neste trabalho, procuramos criar um arcabouço responsável pela segurança no qual se refere ao controle de acesso para os processos de autenticação e de autorização, não levando em consideração o processo de auditoria.

Os requisitos utilizados para a construção do arcabouço tiveram origem na literatura e na prática de desenvolvimento de sistemas comerciais. Abaixo são listados os requisitos que foram encontrados durante a pesquisa bibliográfica [SANDHU, SAMARATI 1994; YODER, BARCALOW 1997; SCHUMACHER *et al.* 2005]:

1. Deve haver um ponto único de acesso ao sistema (*single-sign-on*);
2. Cada usuário deve possuir um nome de usuário (*login*) único e uma senha que serão utilizados em sua autenticação;
3. Para qualquer tentativa de acesso a recursos, o sistema deve verificar se o usuário está autenticado. Caso não esteja, o usuário deve ser encaminhado a realizar sua autenticação;
4. Os usuários devem ser organizados em papéis;
5. Cada usuário pode possuir vários papéis e estes podem estar ativos ao mesmo tempo e não necessariamente deve haver uma hierarquia entre eles;
6. Cada papel deve possuir um conjunto de permissões que representam acessos a recursos do sistema;
7. Deve ser possível identificar o usuário autenticado, assim como suas informações, em qualquer parte do sistema;
8. Deve ser possível identificar informações comuns a todos os usuários autenticados em qualquer parte do sistema;
9. Os recursos devem ser protegidos contra acesso não autorizado de forma que toda requisição seja avaliada para saber se ela possui ou não permissão para acessar o recurso. A princípio, o usuário terá uma visão completa do sistema, podendo visualizar

todas as opções. Porém, ao tentar acessar alguma operação a qual ele não possui acesso, o sistema deverá emitir um erro;

10. Deve haver um mecanismo que contém a lógica implementada das regras de segurança e que seja acessado a cada requisição de forma a avaliar a política de segurança corretamente.

Além dos requisitos obtidos através de pesquisa bibliográfica, levamos em conta na construção do arcabouço ArchiMDAs outros requisitos levantados a partir de experiência prática em desenvolvimento de sistemas. Esses requisitos são listados a seguir:

11. Um usuário pode estar associado a um contexto de uso – e.g. um usuário pode estar associado a uma filial ou unidade ou departamento de uma empresa. A interpretação desse contexto depende da aplicação. Caso o usuário pertença a um contexto, o usuário deve possuir um ou mais perfis dentro desse contexto;
12. Deve ser possível associar restrições a dados. Essas restrições devem ser estabelecidas com base em fatores dinâmicos ou estáticos no momento em que uma solicitação de acesso é realizada, isto é, uma restrição pode ser estática ou dinâmica;
13. A autorização deve ser feita por níveis, ou seja, a aplicação sendo dividida em camadas deve possuir mecanismos de autorização que controle o acesso a recursos em cada camada.

De acordo com os requisitos 4, 5 e 6, podemos perceber que o modelo de segurança utilizado será o modelo de controle de acesso baseado em papéis – RBAC, descrito no capítulo 2. Esse modelo determina que cada usuário possua o seu papel dentro do sistema, podendo acessar somente as informações as quais o seu papel o possibilita. O papel é um conjunto de particularidades que um usuário pode assumir. [SANDHU, SAMARATI 1994] onde vários usuários podem possuir o mesmo papel.

Porém, esse modelo não se mostra suficiente para alguns requisitos de segurança necessários a sistemas de informação atuais.

Podemos citar como alguns desses requisitos, a autonomia das divisões no que diz respeito ao controle de acesso (Requisito 11), a restrição aos dados (Requisito 12) e o controle de acesso feito por níveis (Requisito 13).

Visando atender esses requisitos, neste trabalho propomos uma extensão do modelo RBAC que denominamos de RBAC Estendido que será descrito na próxima Seção.

4.2 RBAC ESTENDIDO

O RBAC estendido é baseado no modelo RBAC com contexto [GUERRA, PAICA, FERNANDES 2004] de forma a atender o requisito 11 e possui extensões, propostas nessa dissertação, para acomodar os requisitos 12 e 13.

4.2.1. RBAC COM CONTEXTOS

O modelo RBAC com contexto foi proposto em [GUERRA, PAICA, FERNANDES 2004]. O objetivo do modelo RBAC com contextos é se adequar a um tipo de sistema utilizado por várias divisões dentro da mesma organização, tornando a administração de autorizações e papéis mais simples e mais intuitiva para o usuário. Por outro lado, não se deseja perder em funcionalidade, de forma a tornar o sistema de controle de acesso inadequado em relação aos requisitos de segurança.

O primeiro passo para tornar o sistema de controle de acesso adaptado ao tipo de aplicação é a introdução de uma nova entidade além das quatro já existentes no RBAC. Essa nova entidade reflete as divisões existentes dentro da organização, para as quais se deseja a separação da atribuição de papéis, pois essa atribuição somente será válida dentro de um contexto. Por este motivo, a esta nova entidade foi dado o nome de Contexto (C), e na Figura 8 pode-se perceber como ela se relaciona com as outras entidades. A interpretação do contexto também vai depender da aplicação, mas algumas das interpretações feitas em sistemas já implementados são as seguintes: filiais, unidades e departamentos. Dessa forma, assim como foi feito com as outras entidades do RBAC, a interpretação para os contextos será deixada em aberto.

Como a idéia dessa dissertação é prover um arcabouço de segurança flexível e genérico, nesse novo modelo, a entidade contexto é opcional. Isto é, uma sessão é composta por um usuário, obrigatoriamente, um contexto, se necessário, e um ou mais papéis ativos ao mesmo tempo, sem que haja necessariamente uma hierarquia entre eles. Para que um usuário ative um novo contexto em uma sessão, é necessário que ele desative o contexto anterior caso exista. Dessa forma, o que existe é uma troca de contextos.

Com a utilização dos contextos é possível realizar facilmente a implementação da autonomia desejada para as divisões da organização no que diz respeito à administração de papéis.

Para os usuários, o acúmulo de papéis fica mais claro, pois os papéis são atribuídos por contexto, algo que acaba sendo mais intuitivo para o mesmo: “Eu possuo o papel de

Administrador na Filial A e o papel de Supervisor na Filial B”. Pode-se dizer que o grande ganho do modelo RBAC com contextos, do ponto de vista do usuário, é realizar uma simplificação que atende os requisitos de segurança, ao mesmo tempo em que facilita a compreensão do sistema.

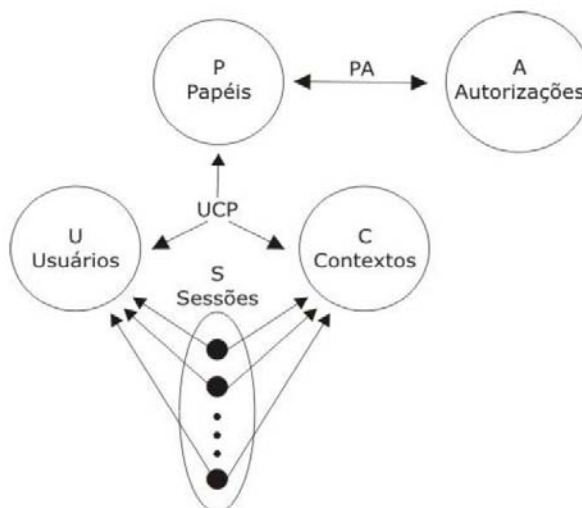


Figura 8. Modelo RBAC com Contexto (Fonte: GUERRA, PAICA, FERNANDES 2004)

O impacto da facilidade de implementação se reflete em menor tempo de desenvolvimento, menor custo e, conseqüentemente, em melhor nível de manutenibilidade. Com os contextos, as autorizações também ficam mais fáceis de serem definidas. As autorizações são o resultado do relacionamento entre os objetos protegidos e as formas de acesso ao mesmo. Os contextos ajudam na definição de quais são os objetos para os quais uma determinada autorização está sendo concedida.

4.2.2. RESTRIÇÕES A DADOS

As restrições existentes no modelo RBAC são restrições entre papéis, com a intenção de evitar a concentração de poder nas mãos de um único usuário, visando minimizar a possibilidade de fraude ou dano acidental.

Além de inserir a informação de contexto, adicionamos restrições a dados. Essas restrições devem ser estabelecidas com base em fatores dinâmicos ou estáticos no momento em que uma solicitação de acesso é realizada, isto é, uma restrição pode ser estática ou dinâmica.

Uma restrição é definida em termos de variáveis ambientais que, quando avaliadas, resultam em informações sobre o usuário corrente, data/hora do acesso, informações de rede

(local do acesso) e outras que podem ser livremente programadas e incorporadas para especificação de regras de acesso mais complexas.

Por exemplo, considere a hipótese em que um conjunto de usuários pertencentes a um papel só pode alterar dados de itens cuja procedência indique o Brasil. Entretanto o mesmo conjunto de usuários possui acesso à consulta de todos os itens. Uma operação que recupere um conjunto de itens da base de dados e que seja usado por serviços de consulta e manutenção de item será executado sob contextos diferentes. Quando a recuperação dos dados ocorrer sob o contexto do serviço de manutenção de item, a restrição de procedência do Brasil deverá ser incluída na consulta ao banco. Quando a recuperação ocorrer sob o contexto da consulta, não existirá nenhuma restrição quanto a procedência do item e nenhuma restrição adicional será incluída na consulta submetida à base de dados. Este é um exemplo de restrição estática.

Uma restrição dinâmica depende de fatores dinâmicos, como data, hora, informações do usuário corrente, entre outras. Por exemplo, considere a hipótese em que em um sistema de gerenciamento de projetos e processos, um usuário só possa incluir alocações em tarefas destinadas a ele. Neste caso, uma restrição dinâmica será adicionada na consulta submetida à base de dados na consulta de tarefas.

Para tornar o sistema de controle de acesso adaptável foi introduzido mais uma entidade no RBAC. Esta nova entidade, denominada Restrição (R), reflete as restrições que podem ser adicionadas aos dados. O relacionamento da entidade Restrição com as outras entidades do modelo RBAC pode ser visto Figura 9.

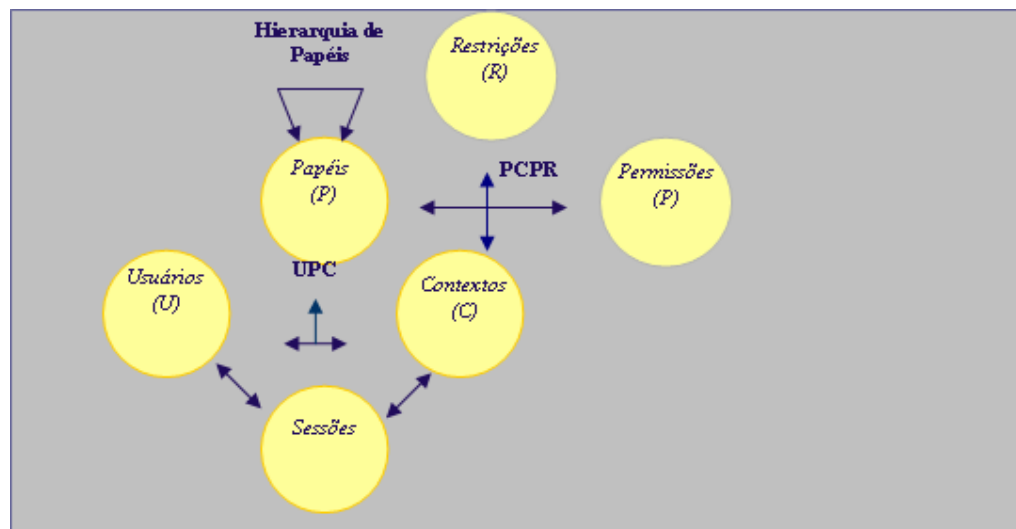


Figura 9. RBAC com Restrição a Dados

4.2.3. NÍVEIS DE SEGURANÇA

Além da inserção de contextos e de restrições a dados, neste trabalho, o controle de acesso é organizado em três níveis: (i) vista; (ii) serviço e (iii) dados. O nível vista visa controlar o acesso aos elementos gráficos que compõem a camada de interface com o usuário, o nível de serviço foca no controle de acesso aos serviços disponibilizados pelos componentes que implementam as regras de negócio da aplicação, e o nível de dados atua sobre as informações persistentes da aplicação. Esses níveis têm o objetivo de tratar as especificidades dos elementos de cada camada de uma aplicação que segue o modelo arquitetural MVC [MVC 2007].

Por padrão, a partir do momento em que a segurança é ativada, todos os elementos da vista são acessíveis a todos os usuários, enquanto que as funções de acesso ao sistema, que chamamos de nível de serviço, são, a priori, não acessíveis a todos os usuários. Isto é, o usuário possui uma visão completa do sistema, porém, ao tentar acessar qualquer função do sistema, uma exceção é levantada.

O controle de acesso é feito primeiramente no nível de vista. Na vista, a autorização não é apenas uma decisão “sim” ou “não” para uma pergunta do tipo: “o usuário X com perfil Y pode acessar o recurso W do sistema?” ou no caso de haver contexto, “o usuário X com perfil Y no contexto Z pode acessar o recurso W do sistema?”. A vista pode ser totalmente configurada de acordo com as permissões do usuário de forma que o usuário possa visualizar apenas o que ele tem acesso, passando de visão completa com erros a uma visão limitada de acordo com as permissões de cada usuário. Por exemplo, um elemento de menu seria instanciado apresentando somente as opções que o usuário pode acessar, o mesmo ocorrendo com as transições da interface (botões e links). No caso dos campos da interface, estes podem aparecer de forma somente leitura, escrita ou invisível de acordo com as permissões do usuário.

Após passar pela vista, o controle de acesso é feito no nível de serviço, que engloba as funções de acesso ao sistema e em seguida, no nível de dados apresentado na Seção 4.2.2.

4.3 PADRÕES DE PROJETO APLICADOS AO ARCABOUÇO ARCHIMDAS

Como forma de atender aos requisitos levantados pesquisamos padrões de projetos de segurança que pudessem auxiliar a construção do arcabouço de segurança ArchiMDAs. A Tabela 1 descreve os padrões utilizados, suas intenções e os requisitos que eles atendem.

Tabela 1. Padrões de Segurança

Padrão de Segurança	Intenção	Requisitos
Ponto de Acesso único	Prover um módulo de segurança e uma forma de autenticação	1, 2
Ponto de verificação	Organizar pontos de verificação de segurança e suas repercussões	2, 3, 9
Papéis	Organizar usuários com privilégios de segurança similares	4, 5, 6
Sessão	Localizar informação global em um ambiente multi-usuário	7, 8
Visão completa com erros	Prover uma visão completa aos usuários, exibindo exceções quando necessário	9
Visão limitada	Permitir que os usuários visualizem somente aquilo a que eles têm acesso	9
Sistema protegido	Estruturar o sistema de forma que todos os acessos a recursos sejam intermediados por um guardião que reforça uma política de segurança	3, 9, 10
Política	Isolar a lógica da política de segurança em um componente discreto do sistema	3, 10
Descritor do sujeito	O controle de acesso de um sujeito, isto é, uma entidade (humana ou programa) a diferentes recursos depende dos atributos desse sujeito. Descritor do sujeito provê acesso aos atributos de um sujeito e facilita gerencia e proteção a esses atributos	7
RBAC Estendido	RBAC com contexto, restrições a dados e níveis de segurança	4, 5, 6, 11, 12, 13

4.4 METAMODELO DE SEGURANÇA

A abordagem MDA recomenda a criação de modelos em três níveis de abstração na especificação de sistemas: Modelo Independente de Computação (CIM), Modelo Independente de Plataforma (PIM) e Modelo Específico de Plataforma (PSM).

Como o CIM é um modelo independente de computação e o foco do presente trabalho é a implementação de requisitos de segurança em sistemas computacionais, esse nível de modelo não é utilizado nessa dissertação.

Cada modelo deve instanciar um metamodelo que representa um domínio, uma área de conhecimento sobre a qual diversos modelos distintos podem ser construídos. A Figura 10 demonstra como esses modelos e metamodelos se relacionam.

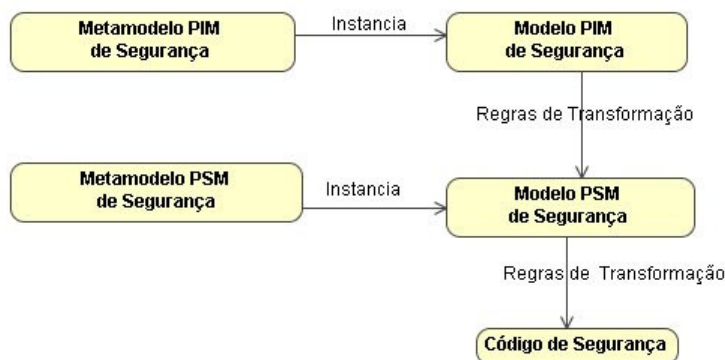


Figura 10. Modelos de Segurança

Tendo como base os requisitos levantados na Seção 4.1, o primeiro passo na construção do arcabouço ArchiMDAs foi a elaboração de metamodelos que na abordagem MDA significou construir um metamodelo PIM de segurança. Esse metamodelo é uma representação UML da parte estrutural dos padrões de projeto descritos na Seção anterior. Esse metamodelo pode ser visto na Figura 11.

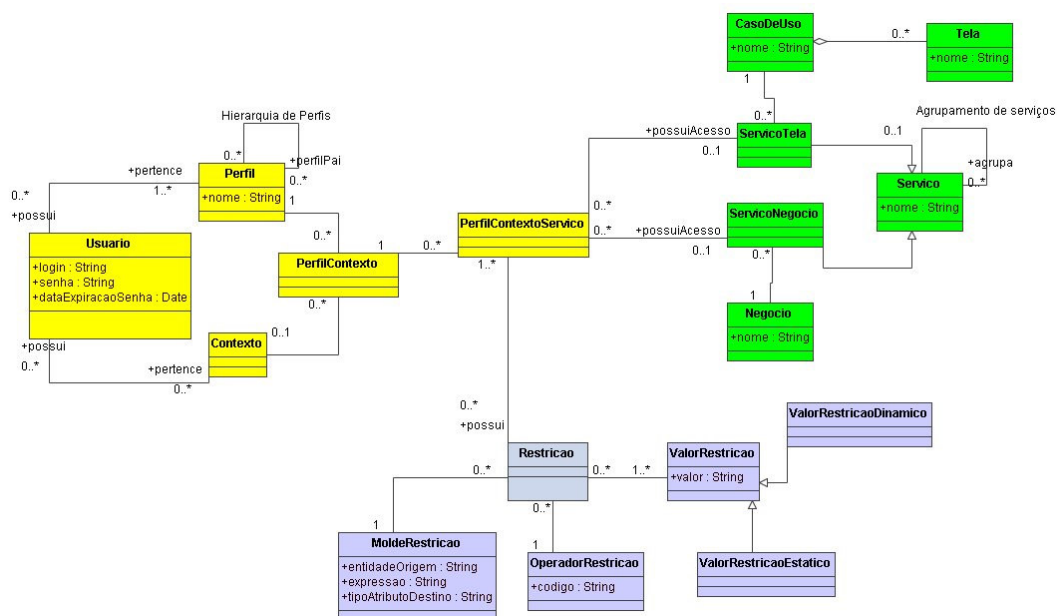


Figura 11. Metamodelo de Segurança

Neste metamodelo se encontram as entidades do RBAC estendido.

Usuario representa uma entidade do sistema, como uma pessoa ou um processo.

Cada usuário pode possuir um ou mais perfis, que é representado pela entidade **Perfil**.

Um perfil representa um papel, ou seja, o conjunto de atividades e responsabilidades associadas a um determinado cargo ou função em uma organização. Um perfil pode herdar de

outros perfis, essa herança é identificada através da auto associação da entidade Perfil que recebeu o nome de *Hierarquia de Perfis*.

Cada usuário pode pertencer a contextos, que são representados pela entidade *Contexto*, onde contextos representam as divisões existentes dentro da organização, para as quais se deseja a separação da atribuição de perfis.

A entidade *PerfilContexto* representa o usuário em um determinado contexto caso ele exista, de forma que um perfil é obrigatório e um contexto pode existir ou não.

Um *PerfilContexto* está associado a *PerfilContextoServico* que por sua vez está associado a *Servico*. Um *PerfilContextoServico* representa o serviço que um determinado perfil em um determinado contexto, caso exista, possui permissão de acesso.

Um serviço representa qualquer artefato que possua acesso controlado, ou seja, um serviço pode ser um caso de uso, uma tela, um campo ou botão que pertence a uma determinada tela, uma operação, etc. Um *ServicoTela* representa os serviços relacionadas a camada de vista. Cada *ServicoTela* está associado a um caso de uso, que é representado pela entidade *CasoDeUso* e cada caso de uso pode possuir telas, que são representadas pela entidade *Tela*. Um *ServicoNegocio* representa os serviços relacionados a camada de controle que representam as funções de acesso ao sistema.

As restrições a dados são representadas pela entidade *Restricao* e estão associadas a entidade *PerfilContextoServico*, ou seja, um serviço que um determinado perfil em um determinado contexto, caso exista, possui permissão de acesso pode possuir restrições aos dados que este serviço manipula.

Uma restrição, por sua vez, possui um molde restrição que é representado pela entidade *MoldeRestricao*. Um molde restrição indica o dado e o atributo deste dado onde a restrição é atribuída. Além de molde restrição, uma restrição possui um operador restrição que é representado pela entidade *OperadorRestricao*. Um operador restrição indica se a restrição será ‘igual’, ‘diferente’, ‘menor que’, ‘maior que’, ‘Em’ (IN) ou ‘Não Em’ (NOT IN). Por último, uma restrição possui um ou mais valores restrições que são representados pela entidade *ValorRestricao*. Um valor restrição representa o valor da restrição que junto com o operador restrição e um molde restrição formam uma restrição. Uma restrição pode possuir mais de um valor restrição pois no caso do operador restrição for ‘Em’ ou ‘Não Em’, uma restrição pode possuir vários valores.

Um valor restrição pode ser estático ou dinâmico. A entidade *ValorRestricaoEstatico* representa um valor restrição estático enquanto que a entidade *ValorRestricaoDinamico* representa um valor restrição dinâmico. Quando estático, o valor utilizado na restrição é o

próprio valor, enquanto que quando dinâmico o valor representa uma operação que será invocada em tempo de execução e o valor utilizado será o retorno dessa operação.

Para facilitar a compreensão da formação de uma restrição, voltaremos ao exemplo dado na Seção 4.2.2, onde temos a hipótese de que um conjunto de usuários pertencentes a um perfil só pode alterar dados de itens cuja procedência indique o Brasil. Considere que o perfil é Desenvolvedor, que não será utilizado contexto, que o dado onde será acrescido uma restrição é o atributo id da entidade Pais e que neste caso, o id possui valor 1 (um). Para tanto, deverá haver uma restrição associada a um *PerfilContextoServico* formado pelo perfil Desenvolvedor e por um serviço de negócio que representa a operação que altera dados de itens.

A restrição por sua vez, será formada por um molde restrição com entidade origem Pais e expressão id, um operador restrição do tipo ‘igual’ e um valor restrição que indique o id do Brasil, que nesse exemplo, é 1 (um). Com isso, quando for feita a consulta para retornar os itens no contexto de manutenção de itens, a restrição será acrescida da consulta de forma que o resultado da consulta só conterá itens cuja procedência indique o Brasil.

4.5 ARQUITETURA E PROCESSO

O arcabouço de segurança ArchiMDAs é dividido em dois módulos, um responsável pelas regras de controle de acesso, que denominamos módulo de regras de segurança, e outro responsável pela administração de controle de acesso, que denominamos de módulo administrativo de segurança.

O módulo de regras de segurança é responsável pelas regras de autenticação e autorização dos artefatos da aplicação. 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 projetista deve focar somente no negócio, sem se preocupar com a segurança. Os requisitos de segurança serão acrescentados ao sistema de forma automatizada e ortogonal ao desenvolvimento das regras de negócio, através dos mecanismos oferecidos pelo ArchiMDAs.

Já o módulo administrativo de segurança representa o modelo de segurança da aplicação, que depende da política de controle de acesso da organização. Visando facilitar o desenvolvimento da aplicação, uma aplicação administrativa 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 caso este não esteja de acordo com a política de segurança da organização.

O processo está sendo representado por um diagrama de atividades onde cada raia possui as atividades executadas por um determinado cargo dentro do desenvolvimento de sistemas utilizando a abordagem proposta.

Abaixo serão descritas as responsabilidades e tarefas de cada cargo.

Arquiteto de Negócio

O arquiteto de negócio desenvolve a arquitetura MDA de negócio criando extensões dos perfis UML, regras de transformação e metamodelo específicos para a criação do modelo de negócio. Estas atividades fazem parte de um processo normal de desenvolvimento, ou seja, elas não são específicas da proposta apresentada nessa dissertação. Todos os artefatos desenvolvidos nessa atividade são bastante reutilizáveis, ou seja, o trabalho realizado pelo arquiteto de negócio pode ser reaproveitado no desenvolvimento de vários sistemas. Em situações onde os artefatos não se adequem, o arquiteto pode estender o que já foi feito reaproveitando o que for possível.

Arquitetura de Segurança

O arquiteto de segurança desenvolve arquitetura MDA de segurança criando extensões dos perfis UML, regras de transformação e metamodelo específicos de segurança. A partir desses artefatos, são desenvolvidos o módulo de regras de segurança e o módulo administrativo de segurança que serão aplicados na transformação entre modelos.

Podemos considerar o módulo de regras de segurança como o núcleo dessa abordagem, pois ele contém todas as regras de transformação definidas através dos requisitos de segurança levantados inicialmente. Já o módulo administrativo de segurança é responsável pela geração da aplicação padrão de segurança do sistema. Esta aplicação é uma aplicação administrativa de segurança. Ela contém os casos de uso responsáveis pelo cadastro dos artefatos de segurança, como por exemplo, perfis, usuários e permissões.

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. Além disso, o fato de ser gerado automaticamente e sem intervenção humana garante a qualidade do código e conseqüentemente do sistema.

As atividades feitas pelo arquiteto de segurança não fazem parte de um processo tradicional de desenvolvimento, elas são específicas da proposta apresentada nessa

dissertação. A partir de uma equipe de especialistas de segurança, são levantados requisitos de segurança que são acoplados no arcabouço de segurança pelos arquitetos de segurança. Todas as extensões dos perfis UML, regras de transformação e metamodelo específicos de segurança criados podem e devem ser reaproveitados na construção de novos sistemas.

Uma das funções realizadas nessa dissertação foi a de Arquiteto de Segurança, pois desenvolvemos o arcabouço ArchiMDAs de acordo com a proposta apresentada com o objetivo de fornecer uma solução de segurança que pode ser reaproveitada ou estendida caso os requisitos das aplicações alvo sejam distintos dos requisitos atendidos pelo arcabouço.

Desenvolvedor do Negócio

O desenvolvedor de negócio continua modelando o PIM de negócio como de costume e utilizando a arquitetura MDA de negócio criada pelo arquiteto de negócio para realizar as transformações de modelo de PIM para PSM e de PSM para código. Os desenvolvedores não precisam tomar nenhuma atitude em relação à segurança pois esta será aplicada de forma transparente a ele. Estas atividades fazem parte de um processo normal de desenvolvimento, ou seja, elas não são específicas da proposta apresentada nessa dissertação.

Gerente de Segurança

Cada organização possui sua própria política de segurança, de forma que a aplicação padrão gerada pelo módulo administrativo de segurança pode não se encaixar no modelo de segurança de um determinado sistema. Nesse caso, o gerente de segurança deve desenvolver uma aplicação de segurança específica para o sistema de acordo com o metamodelo definido pela arquitetura de segurança criada pelo arquiteto de segurança e fazer a ligação dessa aplicação com o módulo de regras de segurança.

Caso a aplicação padrão gerada pelo módulo administrativo de segurança sirva para o sistema específico, o gerente de segurança apenas precisa indicar no modelo de negócio, através de estereótipos, as entidades do modelo que representam as entidades do RBAC com contextos, ou seja, usuário, papel, autorização e contexto.

Gerente de Implantação

O gerente de implantação é a pessoa responsável pela ativação de segurança. Ou seja, é ele que, através do PIM modelado pelo desenvolvedor de negócio, da aplicação padrão ou específica de segurança do sistema e do módulo de regras de segurança, gera o PSM da aplicação com segurança e em seguida, o código contendo todos os artefatos de segurança gerados que é colocado em produção.

A segurança pode ser excluída ou incluída (*switch on-off*) na geração do sistema de forma configurável sem interferir no trabalho dos desenvolvedores, facilitando o

desenvolvimento e a configuração do sistema. Essa exclusão ou inclusão da segurança pode ser feita por níveis, onde é possível escolher em que níveis a segurança será ativada.

Administrador do Sistema

Após a aplicação estar instalada, o administrador de segurança realiza o cadastro dos artefatos de segurança, como o cadastro de usuários, perfis e permissões. Se esses cadastros não forem feitos, o sistema irá barrar qualquer acesso, pois o usuário não conseguira se autenticar e com isso não terá acesso a nenhum serviço.

A segurança é dinâmica, isto é, ela é manipulada em cima de permissões do usuário, e essas permissões podem ser criadas ou removidas dinamicamente. Por exemplo, se as permissões são agrupadas por perfis, é possível criar ou remover perfis e associar permissões aos perfis sem que seja necessário alterar o modelo de negócio. Dessa forma, o modelo não se torna poluído, continuando a ter apenas o negócio propriamente dito.

4.6 CONCLUSÃO

Neste capítulo, descrevemos o arcabouço ArchiMDAs. Para isso, apresentamos os requisitos de segurança levantados para sua construção.

De acordo com os requisitos levantados, podemos perceber que o modelo de segurança utilizado seria o modelo de controle de acesso baseado em papéis – RBAC. Porém, esse modelo não se mostrou suficiente para alguns requisitos de segurança necessários a sistemas de informação atuais. Como exemplo, a autonomia das divisões no que diz respeito ao controle de acesso, a restrição aos dados e o controle de acesso feito por níveis.

Visando atender esses requisitos, neste trabalho propomos uma extensão do modelo RBAC que denominamos de RBAC Estendido.

Como forma de atender aos requisitos levantados pesquisamos padrões de projetos de segurança que pudessem auxiliar a construção do arcabouço de segurança ArchiMDAs. Apresentamos uma relação de padrões utilizados neste trabalho, suas intenções e os requisitos que eles atendem.

Tendo como base os requisitos levantados, o primeiro passo na construção do arcabouço ArchiMDAs foi a elaboração de metamodelos que na abordagem MDA significou construir um metamodelo PIM de segurança. Esse metamodelo é uma representação UML da parte estrutural dos padrões de projeto pesquisados.

Além disso, apresentamos a arquitetura e o processo de criação e utilização do arcabouço ArchiMDAs. No próximo capítulo, iremos apresentar sua implementação.

5 IMPLEMENTAÇÃO

No capítulo anterior, descrevemos o arcabouço ArchiMDAs e seu processo de criação e utilização. Para apoiar a utilização da abordagem na prática, apresentamos, neste capítulo, a implementação do arcabouço ArchiMDAs.

Como o arcabouço é baseado em MDA, foi necessário escolher uma ferramenta que seguisse o paradigma MDA, compatível com a abordagem adotada. Dentre as varias ferramentas existentes, optamos por estender a ferramenta AndroMDA [ANDROMDA 2007].

Para implementar a abordagem proposta, foram necessárias algumas adaptações no conjunto de transformações originais do AndroMDA que serão apresentadas ao longo desse capítulo.

Este capítulo está dividido da seguinte forma: a Seção 5.1 apresenta a ferramenta utilizada e alterada nesta implementação; a Seção 5.2 descreve o que foi estendido na ferramenta; a Seção 5.3 descreve a arquitetura de autenticação e autorização do JAAS; a Seção 5.4 descreve a lógica de implementação e a implementação propriamente dita da autorização e da autenticação gerada pelo módulo de regras de segurança assim como os casos de uso Trocar de Senha e Sair do Sistema; a Seção 5.5 descreve a implementação da aplicação de segurança gerada pelo módulo administrativo de segurança; a Seção 5.6 encerra o capítulo, apresentando algumas considerações finais.

5.1 FERRAMENTA ANDROMDA

Dentre as várias ferramentas existentes, optamos pela AndroMDA [ANDROMDA 2007] para desenvolver o arcabouço. Essa escolha se deu a partir de alguns critérios como: possuir grande número de usuários, maturidade, suporte eficaz a usuários e desenvolvedores, possuir código aberto, e ser extensível. Uma breve comparação entre algumas ferramentas MDA existentes pode ser encontrada em [MAIA 2006].

Devido ao fato de utilizarmos a versão 3.1 da ferramenta, os modelos podem ser feitos em praticamente qualquer ferramenta de modelagem que suporte UML 1.4 [UML 2007] e XMI [XMI 2007]. A ferramenta de modelagem utilizada neste projeto foi Magic Draw [MAGIC DRAW 2007].

A Figura 13 representa o funcionamento do AndroMDA. Os modelos UML são feitos em ferramentas de modelagem, que os exportam no formato XMI. A ferramenta então, com base no XMI e nos *templates* de transformação, inicia a geração dos artefatos.

As transformações do AndroMDA são baseadas em cartuchos. Um cartucho é um conjunto de *templates*, escritos em Velocity [VELOCITY 2007], para uma plataforma específica.

Hibernate [HIBERNATE 2007] (um framework de persistência objeto-relacional) e Enterprise Java Beans (EJB) [EJB 2007] (uma arquitetura de componentes que simplifica o processo de construção de aplicações corporativas em Java) são alguns exemplos de cartuchos disponíveis no AndroMDA. O AndroMDA sabe qual cartucho deve ser aplicado a partir dos estereótipos e da configuração de cada projeto. Alguns estereótipos pré-definidos são Entity e Service. Classes estereotipadas como Entity podem ser transformadas em classes e descritores Hibernate, por exemplo, caso essa seja a plataforma escolhida. Nesse caso, o AndroMDA gera todos os artefatos necessários para aplicações de cadastro (classes e arquivos de mapeamento). Classes estereotipadas como Service podem ser transformadas em Session Beans (que é um tipo de componente EJB), por exemplo, caso essa seja a escolhida. Nesse caso, o AndroMDA gera apenas o esqueleto de código e os descritores de instalação.

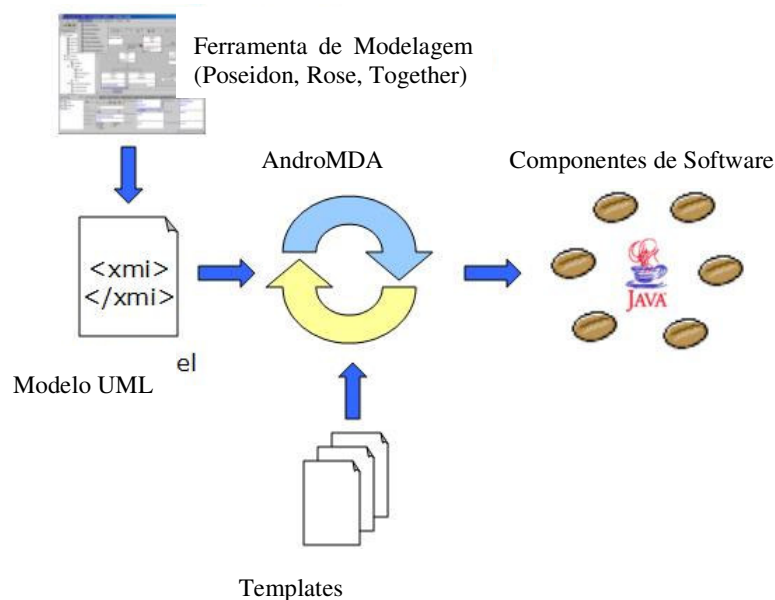


Figura 13. Arquitetura do AndroMDA (Fonte: ANDROMDA 2007 traduzida pela autora)

5.2 EXTENSÃO DO ANDROMDA

Para implementar a abordagem proposta, foram necessárias algumas adaptações no conjunto de transformações originais do AndroMDA.

Com o intuito de garantir que o requisito de segurança por níveis fosse atendido, a implementação do arcabouço ArchiMDAs foi feita seguindo o padrão de projeto arquitetural

MVC (*Model-View-Controller*), que pode ser traduzido para Modelo-Visão-Controlador e consiste num padrão organizacional em camadas. O Modelo representa os objetos ou dados de um aplicativo. A Visão é a representação deste modelo em formato de apresentação para o cliente, com interface apropriada para a entrada de informações. O Controlador, por sua vez, implementa a interatividade, através do processamento das ações tomadas pelo cliente e atualização do modelo.

Como o AndroMDA já possui cartuchos disponíveis para Struts [STRUTS 2007], EJB e Hibernate, optamos por utilizar na camada de visão o arcabouço Struts, que foi desenvolvido com o objetivo de facilitar o desenvolvimento de aplicações para Web. Na camada de serviços, EJB (*Enterprise Java Beans*), que tem como objetivo promover uma arquitetura de objetos distribuídos, provendo um ambiente para o gerenciamento de componentes. E na camada de persistência, Hibernate, que é um arcabouço de mapeamento objeto-relacional em Java.

Nesse contexto, estendemos esses cartuchos de forma que quando segurança estivesse ativada, eles gerassem além do código específico de cada arcabouço, o código de segurança. Dessa forma, os padrões de segurança são gerados através das regras de transformação adicionadas em cada cartucho. Além disso, para que os padrões de segurança funcionassem, uma série de padrões de projeto foram aclopados na geração do código. Como exemplo, podemos citar o padrão interceptador, que promove um controle central interceptando chamadas a métodos e incluindo verificação de regras de segurança.

Como dito anteriormente, as regras de transformação de segurança são criadas e acrescentadas nos cartuchos pelos arquitetos de segurança, de forma que a qualidade das regras de transformação é garantida pela qualidade do código gerado e pela facilidade de alteração do que é gerado.

Além de estender a ferramenta AndroMDA, utilizamos e estendemos algumas classes do JAAS (*Java Authentication and Authorization Service*) que é uma interface portátil que possibilita que se faça autenticação de autorização de clientes em Java. A escolha de utilização desse arcabouço foi devido ao fato dele ser conhecido e largamente utilizado. Além do JAAS, como o servidor de aplicação utilizado foi o JBoss [JBoss 2007], também estendemos algumas classes do JBossSX [JBossSX 2007] que é um *framework* de segurança do JBoss baseado no JAAS. O JBossSX utiliza somente as capacidades de autenticação do JAAS.

Algumas classes são geradas representando o núcleo responsável pelas regras de segurança. Essas classes são:

- **PrincipalImpl**

Representa o Principal que será associado ao Subject, que é uma entidade ou cliente que solicita o acesso ao sistema. Estende a classe SimplePrincipal do JBossSX. Essa classe representa a identidade do Subject. Ela contém os perfis, o cliente que representa o Subject e as restrições.

- **LoginCallbackHandler**

Implementa CallbackHandler e é responsável pela comunicação com o usuário.

- **LoginModuleImpl**

Estende a classe DatabaseServerLoginModule do JBossSX que é um módulo de login que espera que haverá um banco de dados contendo uma tabela representando o usuário, uma representando o perfil e uma relação entre essas tabelas. O LoginModuleImpl é responsável por criar e associar PrincipalImpl ao Subject.

- **ControleAcesso**

Classe abstrata que contém todos os métodos invocados nos módulos de segurança criados nesse trabalho.

- **ControleAcessoImpl**

Estende a classe ControleAcesso e contém a implementação de todos os métodos declarados abstratos na classe ControleAcesso. Esta classe representa a ligação entre o módulo de regras de segurança e o módulo administrativo de segurança. Os métodos declarados abstratos são os métodos que dependem da política de segurança da organização. Esses métodos devem ser implementados pelo gerente de segurança caso a organização não utilize a implementação padrão gerada pelo módulo administrativo de segurança. A implementação desses métodos deve retornar instancias das classes contidas no núcleo de classes geradas pelo módulo de regras de segurança. Esses métodos são:

- `public abstract boolean precisaTrocarSenha(Usuario usuario)`
Verifica se o usuário necessita trocar senha
- `public abstract accessControl.Operador getOperador(java.lang.String login)`
Recupera o operador de acordo com o login
- `public abstract HashMap<String, java.util.Collection<Autorizacao>> listaServicos()`

Lista todos o serviços em um map onde a chave é o nome do serviço e o valor é a coleção de autorizações que possui acesso ao serviço. Uma autorização representa um PerfilContexto, ou seja, possui o perfil, o contexto e o tipo de

permissão (leitura e/ou escrita) para os serviços que representam elementos da vista.

- public abstract boolean trocaSenha(Usuario usuario, String senhaAtual, String novaSenha)

Realiza troca de senha do usuario

- public abstract HashMap<String, Collection<Restricao>> getRestricoes(Operador operador, Servico servico)

Recupera restrições do serviço

- public Collection<accessControl.Autorizacao> buscaPerfisServico(Servico servico)

Recupera as autorizações que possuem acesso a um determinado serviço

- SecurityProxyImpl

Classe que é configurada na geração do arquivo de configuração jboss.xml de forma que toda invocação a alguma operação a algum serviço EJB passa antes por esse proxy para realizar o controle de acesso

- ActionServletImpl

Classe responsável por invocar o método listaServicos() do ControleAcessoImpl com o objetivo de listar os serviços associados a suas autorizações ao iniciar o sistema. Essa listagem é adicionada a classe ServicosSingleton

- ServicosSingleton

Classe responsável por armazenar os serviços associados a suas autorizações. Essa classe implementa o padrão de projeto Singleton e suas informações podem ser acessadas em qualquer parte do sistema.

Além disso, para que o arcabouço ArchiMDAs funcionasse, foi necessário construir regras de transformação onde classes que instanciam o metamodelo de segurança (Figura 11) fossem geradas. Essas classes são utilizadas durante todo o processo de autenticação e autorização do ArchiMDAs de forma que a implementação dos métodos da classe ControleAcessoImpl devem retornar instâncias dessas classes. Essas classes são:

- Operador

Representa um usuário

- Perfil

Representa um Perfil que é o papel do usuário

- Autorizacao

Representa uma autorização que pode ser formada por um perfil, um contexto e um tipo de permissão, que pode ser leitura ou escrita

- Servico

Representa um serviço que possui acesso controlado. Os serviços podem ser os elementos da vista, que formam a interface com o usuário e funções de acesso ao sistema

- Contexto

Representa um contexto

- Restricao

Representa uma restrição a um dado. Uma restrição possui um `OperadorRestricao`, um `MoldeRestricao` e um ou mais `ValorRestricao`

- `OperadorRestricao`

Representa o operador da restrição

- `MoldeRestricao`

Representa um molde de restrição que indica onde a restrição será aplicada

- `ValorRestricao`

Representa o valor da restrição

- `ValorRestricaoEstatico`

Representa o valor da restrição quando este é estático

- `ValorRestricaoDinamico`

Representa o valor da restrição quando este é dinâmico

Nas próximas seções, iremos descrever a arquitetura de autenticação e autorização do JAAS além da lógica de implementação e a implementação propriamente dita da autorização e da autenticação gerada pelo ArchiMDAs apresentando com mais detalhes as extensões realizadas no AndroMDA.

5.3 JAAS

A API (*Application Program Interface*) JAAS (*Java Authentication and Authorization Service*) foi desenvolvida pela Sun Microsystems com o propósito de facilitar a implementação de uma aplicação com controle de acesso. A API de segurança do Java JAAS se propõe a fornecer um mecanismo de autenticação que seja independente da aplicação, e também realizar a implementação de um mecanismo de autorização.

5.3.1. AUTENTICAÇÃO NO JAAS

O JAAS se propõe a resolver o problema de programar um sistema com o mecanismo de autenticação independente da aplicação. A forma com que a autenticação é implementada no JAAS é chamada de “plugável”, pois as atualizações ou mudanças que o programador desejar realizar no mecanismo de autenticação serão possíveis sem a necessidade de fazer alterações na aplicação [SUN 2007].

Um programador de aplicações Java que não utiliza JAAS implementa o mecanismo de autenticação dentro da aplicação, desta maneira a manutenção ou mesmo atualização desse mecanismo são trabalhosas, pois é necessário modificar a aplicação e compilá-la outra vez. Segundo [SUN 2007] com a API JAAS as modificações podem ser realizadas em tempo de execução da aplicação, eliminando assim a possibilidade do programador inserir novos problemas na aplicação ou parar o funcionamento da mesma, causando transtorno aos usuários do sistema.

O programador pode implementar um outro módulo de autenticação e “plugá-lo” na aplicação, retirando o mecanismo de autenticação que estava sendo usado antes ou adicionando o novo. O funcionamento básico da autenticação no JAAS é ilustrado a seguir na Figura 14.

Os passos da Figura 14 serão descritos abaixo:

1. O cliente instancia um novo contexto de login passando o nome da configuração de login e um CallbackHandler que é utilizada para a comunicação com o cliente. O LoginContext é uma classe fornecida pelo container. Ela é responsável por coordenar o processo de login;
2. O contexto de login instancia um novo objeto Configuration, que deve ser escrito pelo desenvolvedor. Esse objeto sabe o tipo de autenticação que o sistema irá usar;
3. O contexto de login pede ao objeto Configuration a lista de mecanismos de autenticação que serão utilizados de acordo com o nome da configuração de login passado. Essa lista é definida em um arquivo de configuração do *login* que especifica o módulo de autenticação a ser invocado para cada aplicação. Caso o nome da configuração de login passado não exista, a configuração other é utilizada com padrão;
4. O objeto Configuration retorna uma lista de mecanismos de autenticação. Cada um desses é chamado de módulo de login (LoginModule). Um módulo de login sabe como contactar um provedor de segurança específico e autenticar de uma maneira proprietária;
5. O contexto de login instancia os módulos de login. Pode-se ter muitos módulos de login se for necessário fazer autenticação através de vários provedores de segurança;

6. O contexto de login inicializa os módulos de login;
7. O código cliente tenta logar chamando o método login() no contexto de segurança;
8. O contexto de segurança delega a chamada ao método login() aos módulos de login, já que somente os módulos de login sabem como executar a autenticação. Os módulos de login (escritos pelo desenvolvedor) autenticam o cliente usando um modo proprietário;
9. Os Módulos de Login recebem um Subject, que é uma entidade ou cliente que solicita o acesso ao sistema.

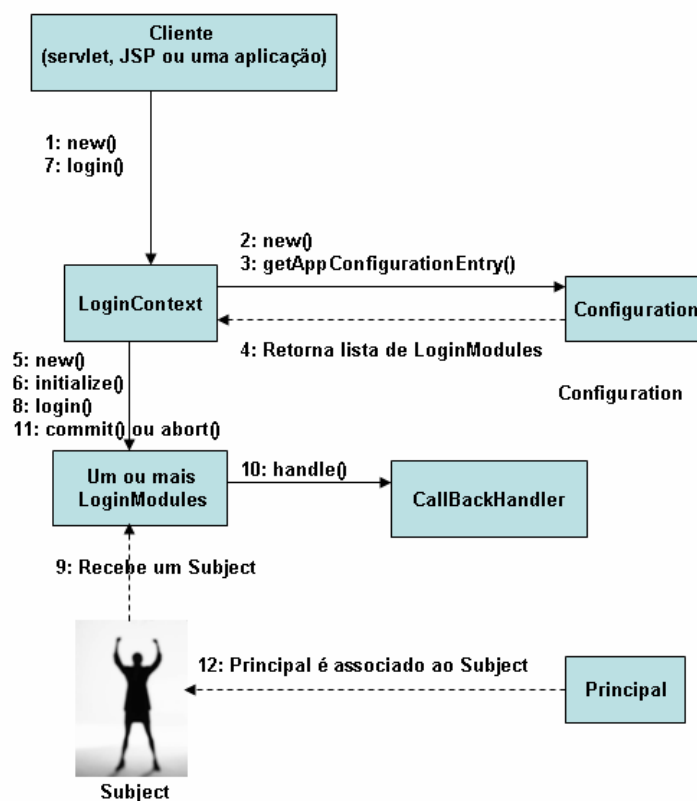


Figura 14. Autenticação no JAAS

10. Para realizar a autenticação, os módulos de login invocam o método `handle()` do `CallbackHandler` que é responsável pela comunicação com o usuário. O `CallbackHandler` requisita ao usuário um nome de login e uma senha.
11. Se o login for feito com sucesso, então é indicado aos módulos de login que eles devem fazer o `commit` (`commit()`). Eles também podem abortar (`abort()`) se a autenticação falhar.
12. Se for autenticado com sucesso, um `Principal` é associado ao `Subject`. O `Principal` é uma identidade do `Subject` no sistema.

5.3.2. AUTORIZAÇÃO NO JAAS

Na estrutura de segurança do Java já existe, desde antes do JAAS, um mecanismo de autorização que é muito centralizado no código. Isto é, esse mecanismo controla o acesso aos recursos protegidos especificando as permissões para o código da aplicação, não importando quem que esteja executando-o. Com JAAS a autorização não verifica mais apenas o código que está sendo executado, ou seja, agora também é possível verificar quem está executando o código que possui os objetos protegidos [SUN 2007].

Assim, a autorização com JAAS resolve a deficiência da estrutura de segurança do Java. O JAAS faz com que a autorização seja mais completa, pois ela controla o acesso a um objeto protegido verificando quem que está tentando acessá-lo. Assim, os direitos de acesso para os objetos protegidos podem ser concedidos ou negados de duas maneiras: para o código da aplicação que esteja tentando acessá-los; e também para o usuário ou entidade do sistema que esteja tentando executar os recursos protegidos. O funcionamento básico da autorização no JAAS é detalhado a seguir na Figura 15.

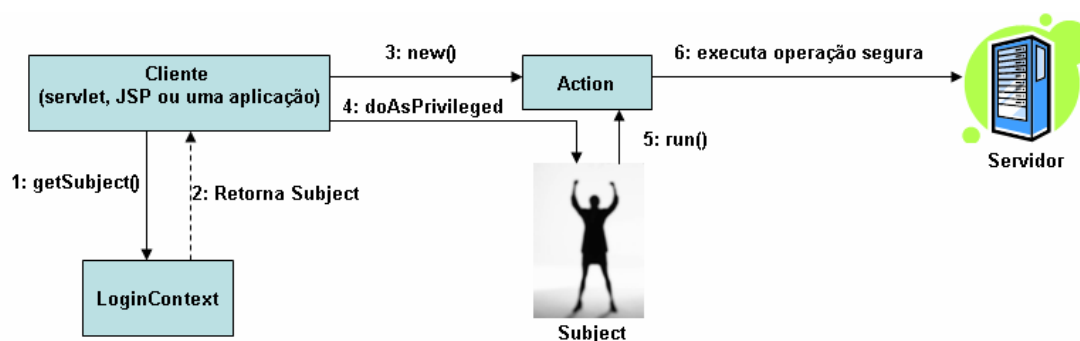


Figura 15. Autenticação no JAAS

Os passos da Figura 15 serão descritos abaixo:

1. O cliente invoca o Subject autenticado;
2. Um novo objeto Subject é retornado para o código cliente. Esse objeto Subject representa alguém (ou algo) que foi autenticado. Esse objeto pode ser usado para executar operações seguras;
3. O código cliente instancia um novo objeto Action. Esse objeto é uma instância de uma classe escrita pelo desenvolvedor. Esse objeto sabe como executar uma operação que deve ser executada de forma segura;
4. O desenvolvedor diz ao objeto Subject para fazer (do) a ação como (as) o objeto Subject, por isso o nome doAs() do método;
5. O objeto Subject chama o método run() do objeto Action;

6. O objeto Action executa a sua operação e o contexto de segurança do usuário logado é automaticamente propagado junto com a chamada do método. O contexto de segurança é passado para o servidor, o servidor pode agora executar o processo de autorização.

5.4 MÓDULO DE REGRAS DE SEGURANÇA

O módulo de regras de segurança é responsável pela regras de autenticação e autorização dos artefatos da aplicação. Para que ele funcione, é necessário que o gerente de segurança indique no modelo a entidade que representa o usuário e a entidade que representa o contexto, caso exista. Os estereótipos utilizados são <<User>> e <<Context>> respectivamente.

Como dito anteriormente, neste trabalho, realizamos o controle de acesso por níveis de aplicação, onde se encaixam os elementos da vista, que formam a interface com o usuário e funções de acesso ao sistema. Além da inserção de contextos e de restrições a dados. Porém, o controle pode ser ativado por níveis (*switch on / off*) através de arquivos de configuração gerados. Ou seja, o gerente de segurança pode escolher os níveis em que deseja ativar a segurança.

Nas próximas seções, iremos descrever a lógica de implementação e a implementação propriamente dita da autenticação e da autorização do módulo de regras de segurança desenvolvido.

5.4.1. AUTENTICAÇÃO

Utilizamos os serviços de segurança quando um dos requisitos para o desenvolvimento de uma aplicação seja o acesso seguro aos componentes de negócio. Um dos mecanismos utilizados para realizar o controle de acesso é a autenticação.

Como já dito anteriormente, a autenticação é o processo que verifica a identidade e a autenticidade de um cliente em um sistema durante qualquer tentativa de acesso, ou seja, ela verifica se o cliente é realmente quem ele diz ser. Neste trabalho, alteramos a implementação padrão do AndroMDA para que os artefatos necessários para realizar a autenticação de um cliente fossem gerados automaticamente. Abaixo, iremos apresentar a lógica de implementação utilizada e o código de autenticação gerado.

5.4.1.1. LÓGICA DE IMPLEMENTAÇÃO

A lógica de implementação utilizada para realizar a autenticação pode ser vista na Figura 16.

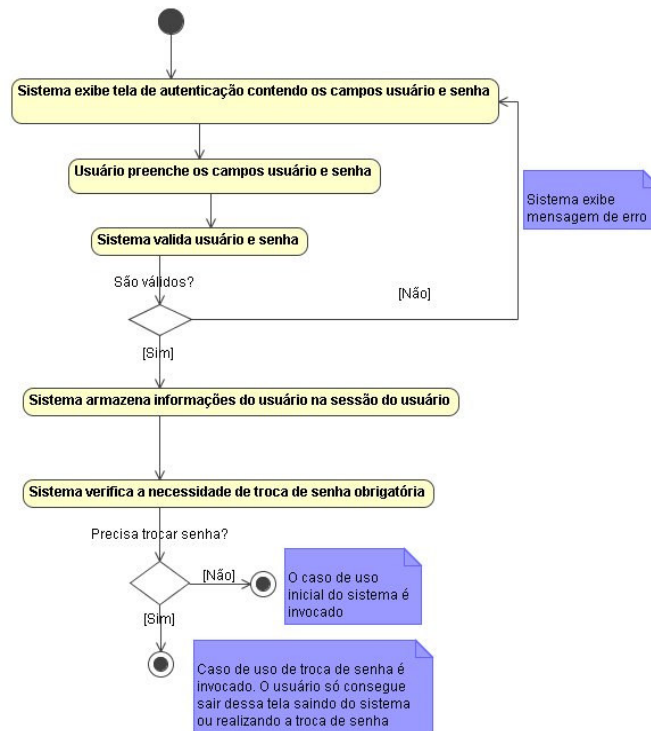


Figura 16. Lógica de Autenticação

A lógica de implementação da autenticação pode ser descrita da seguinte forma:

1. O sistema exibe uma tela de autenticação contendo os campos usuário e senha;
2. O usuário preenche os campos usuário e senha;
3. O sistema valida usuário e senha;
 - 3.1. Se forem válidos, sistema armazena as informações do usuário na sessão;
 - 3.1.1. Sistema verifica a necessidade de troca de senha obrigatória;
 - 3.1.1.1. Se for necessário, o usuário é redirecionado para a tela de troca de senha e só consegue sair dessa tela saindo do sistema ou realizando a troca de senha;
 - 3.1.1.2. Se não for necessário, o usuário é redirecionado para a tela principal do sistema;
 - 3.2. Se não forem válidos, a tela de autenticação é mostrada com o erro específico.

5.4.1.2. IMPLEMENTAÇÃO

O código gerado e as classes envolvidas para realizar a autenticação podem ser vista na Figura 17. Para realizar a implementação, alteramos as regras de transformação do AndroMDA para que os casos de uso de entrar no sistema, sair do sistema e trocar senha fosse gerados automaticamente. O caso de uso entrar no sistema realiza a autenticação do cliente a partir do

preenchimento dos campos usuário e senha. O caso de uso de sair do sistema desconecta o cliente do sistema. O caso de uso trocar senha realiza a troca de senha do cliente. Essa troca de senha pode ser obrigatória caso o cliente tenha que trocar de senha para acessar os recursos do sistema ou pode ser aleatória caso o cliente deseje trocar de senha por vontade própria. Em todas as telas geradas referentes a esses casos de uso, utilizamos o protocolo HTTPS (*HyperText Transfer Protocol Secure*) para garantir a segurança das informações transmitidas, ou seja, para evitar que essas informações transmitidas entre o cliente e o servidor sejam visualizadas por terceiros.

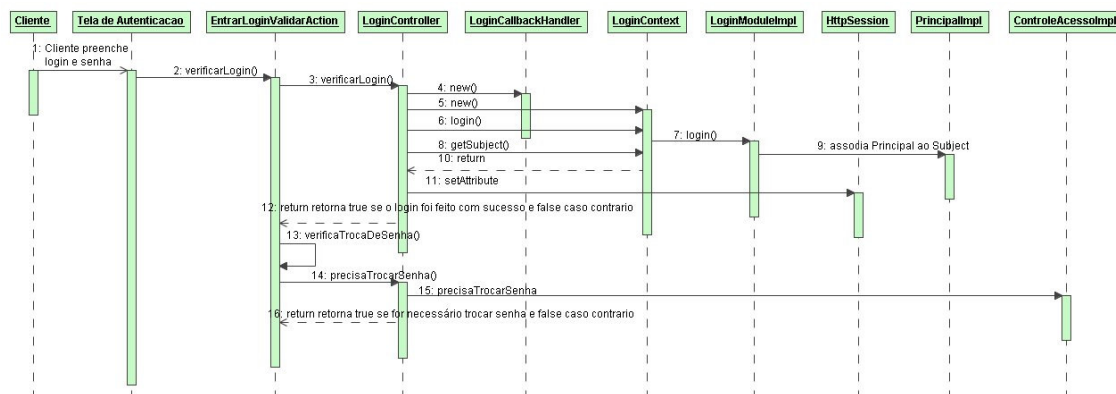


Figura 17. Implementação Autenticação

O código gerado pode ser descrito da seguinte forma:

1. Uma tela de autenticação é gerada contendo os campos login e senha. O cliente preenche esses campos e invoca a validação dos campos preenchidos;
2. De acordo com a arquitetura Struts, para cada ação ou solicitação a ser implementada deve existir uma Action que faz o papel do controlador interagindo com a camada de negócio. No AndromDA, cada caso de uso necessita de uma classe controladora que possui as implementações dessas ações ou solicitações feitas pelo cliente. Com isso, os Actions repassam a responsabilidade para os controladores. Para validar o login e senha preenchidos na tela de autenticação, uma classe chamada `EntrarLoginValidarAction` é gerada e o método `verificarLogin()` é invocado;
3. A action `EntrarLoginValidarAction` delega a responsabilidade para o `LoginController` invocando o método `verificarLogin()`;
4. O `LoginController` inicializa o `LoginCallbackHandler` passando como parâmetros o login e senha preenchidos;
5. O `LoginController` inicializa o `LoginContext` que é uma classe fornecida passando como parâmetros o identificador da aplicação como nome da configuração de login e a instância criada do `LoginCallbackHandler`;

6. O LoginController invoca o método login() do LoginContext para realizar a autenticação;
7. O LoginContext delega a chamada ao método login() ao módulo de login LoginModuleImpl que é responsável pela autenticação;
8. O LoginModuleImpl associa um PrincipalImpl ao Subject;
9. O LoginController invoca o método getSubject;
10. O LoginContext retorna o Subject autenticado;
11. O LoginController adiciona o Subject como atributo na sessão;
12. O LoginController retorna true caso a autenticação tenha sido realizada com sucesso e false caso tenha ocorrido algum erro;
13. Caso tenha ocorrido algum erro durante a autenticação, a tela de autenticação é exibida informando o erro ocorrido, como por exemplo, usuário ou senha inválido. Caso a autenticação tenha ocorrido com sucesso, o método verificaTrocaDeSenha() é invocado;
14. A action EntrarLoginValidarAction delega a responsabilidade para o LoginController invocando o método precisaTrocarSenha();
15. O LoginController invoca o método precisaTrocarSenha() da classe ControleAcessoImpl;
16. Caso necessite trocar senha, o caso de uso de troca de senha é invocado. Esse caso de uso é gerado pelo módulo de regras de segurança. Caso não necessite trocar senha, o caso de uso inicial do sistema é invocado.

Como estamos utilizando o servidor de aplicação JBoss, para que a autenticação funcione, é necessário configurar um arquivo chamado login-config.xml no JBoss acrescentando uma política de aplicação para a aplicação que está sendo construída. Um exemplo pode ser visto abaixo:

```
<application-policy name="testeSeguranca">
  <authentication>
    <login-module code="org.jboss.security.ClientLoginModule"
flag="required"></login-module>
    <login-module code="accessControl.LoginModuleImpl" flag="required">
      <module-option name="dsJndiName">java:/DefaultDS</module-option>
      <module-option name="unauthenticatedIdentity">guest</module-option>
      <module-option name="principalClass">accessControl.PrincipalImpl</module-
option>
```

```

<module-option name="hashAlgorithm">md5</module-option>
<module-option name="hashEncoding">hex</module-option>
<module-option name="principalsQuery">
    select senha from PESSOA where login =?</module-option>
<module-option name="rolesQuery">
    Select r.nome, 'Roles'
    from PERFIL r, PESSOA u
    where u.PERFIL_FK = r.ID
    and u.login = ?
</module-option>
</login-module>
</authentication>
</application-policy>

```

A descrição das propriedades encontra-se abaixo:

- dsJndiName: O nome JNDI do DataSource do banco contendo as tabelas da aplicação de segurança. Se não for especificado será utilizado java:/DefaultDS;
- principalsQuery: Query que retorna a senha de acordo com o login passado. Se não for especificado, será utilizada a query: select Password from Principals where PrincipalID=?;
- rolesQuery: Query que retorna os perfis do usuário autenticado. Se não for especificado, será utilizada a query: select Role, RoleGroup from Roles where PrincipalID=?;
- unauthenticatedIdentity: O nome do principal que deve ser utilizado com requests que não contém informação de autenticação. Deve ser utilizado o guest;
- hashAlgorithm: O nome do algoritmo java.security.MessageDigest utilizado para criptografar a senha. Deve ser utilizado o md5;
- hashEncoding: O formato da string utilizado no hashAlgorithm. Deve ser base64 ou hex. O default é base64. Deve ser utilizando o hex;
- principalClass: Especifica uma classe de implementação do Principal. Deve indicar o PrincipalImpl que é gerado pelo módulo de segurança.

5.4.2. AUTORIZAÇÃO

Como dito anteriormente, a autenticação é o processo que verifica a identidade de um cliente. Por sua vez, um outro mecanismo utilizado para realizar o controle de acesso é a autorização

que verifica se o cliente previamente autenticado possui permissão para executar uma operação desejada. Por este motivo, a autenticação sempre precede a autorização.

5.4.2.1. LÓGICA DE IMPLEMENTAÇÃO

- **Camada de Vista**

Na camada de vista, todos os componentes de vista podem possuir acesso controlado. Esses componentes incluem botões, links ou campos em uma determinada tela, um caso de uso ou uma terminada tela de um caso de uso.

A lógica de implementação da autorização da vista ao tentar acessar alguma URL pode ser vista na Figura 18.

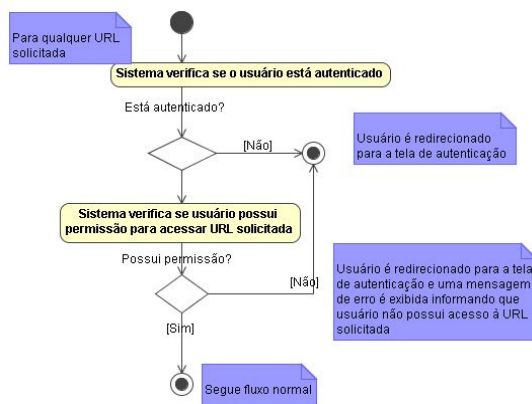


Figura 18. Lógica de autorização de vista

A lógica de implementação da autorização da vista ao tentar acessar alguma URL pode ser descrita da seguinte forma:

1. Para qualquer URL solicitada, o sistema verifica se o usuário que está tentando acessar a URL está autenticado no sistema;
2. Caso o usuário esteja autenticado, o sistema verifica se o usuário possui acesso a URL solicitada;
 - 2.1. Caso possua, o fluxo normal é seguido e a URL solicitada é executada;
 - 2.2. Caso não possua, o usuário é redirecionado para a tela de autenticação e um erro é exibido na tela informando que o usuário não possui acesso.

A lógica de implementação de implementação de algum elemento de tela como botões, links ou campos pode ser vista na Figura 19:

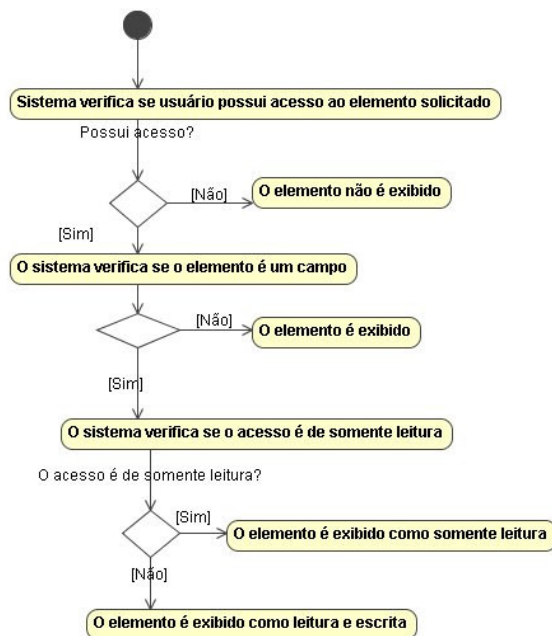


Figura 19. Lógica de Autorização dos elementos de vista

A lógica de implementação da autorização dos elementos de tela pode ser descrita da seguinte forma:

1. Para cada elemento de tela, o sistema verifica se usuário possui acesso ao elemento solicitado;
 - 1.1. Caso Não possua, o elemento não é exibido;
 - 1.2. Caso possua, o sistema verifica se o elemento é um campo;
 - 1.2.1. Caso não seja, o elemento é exibido normalmente;
 - 1.2.2. Caso seja, o sistema verifica se o acesso é de somente leitura;
 - 1.2.2.1. Caso seja, o elemento é exibido como somente leitura;
 - 1.2.2.2. Caso não seja, o elemento é exibido normalmente de forma que o usuário possa ler e editar o elemento.

- **Camada de Serviço**

Na camada de serviço todas as operações possuem acesso controlado. A lógica de implementação pode ser vista na Figura 20.

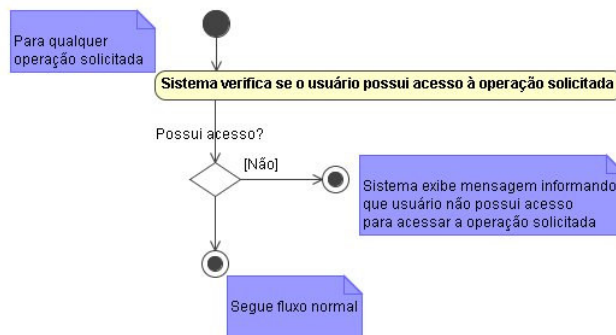


Figura 20. Lógica de autorização da camada de serviço

A lógica de implementação das operações da camada de serviço pode ser descrita da seguinte forma:

1. Para cada operação solicitada, o sistema verifica se usuário possui acesso à operação solicitada;
 - 1.1. Caso Não possua, o sistema exibem mensagem informando que o usuário não possui acesso à operação solicitada;
 - 1.2. Caso possua, o fluxo normal é seguido.

- **Camada de Dados**

Na camada de dados, o controle de acesso não é feito do tipo possui ou não acesso. O controle é feito adicionando restrições aos dados de acordo com a operação que está sendo executada e a verificação dessas restrições durante uma consulta de acordo com os perfis do usuário e do dado que está sendo manipulado. A lógica de implementação de adição de restrições pode ser vista na Figura 21.

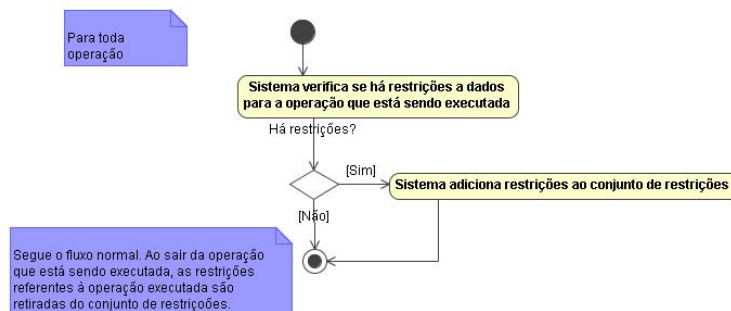


Figura 21. Lógica de adição de restrições

Essa lógica pode ser descrita da seguinte forma:

1. Para cada operação solicitada, o sistema verifica se há restrições a dados para a operação que está sendo executada;
 - 1.1. Caso possua, o sistema adiciona essas restrições ao conjunto de restrições ativas;
 - 1.2. Caso não possua, o fluxo normal é seguido.

Ao terminar a execução de uma determinada operação, as restrições referentes à operação, caso existam, são retiradas do conjunto de restrições ativas.

Após adicionar as restrições ao conjunto de restrições, deve haver um mecanismo que altere a consulta que está sendo realizada de forma que as restrições sejam adicionadas a consulta de acordo com o dado que está sendo manipulado e com os perfis do usuário. Essa lógica pode ser vista na Figura 22.

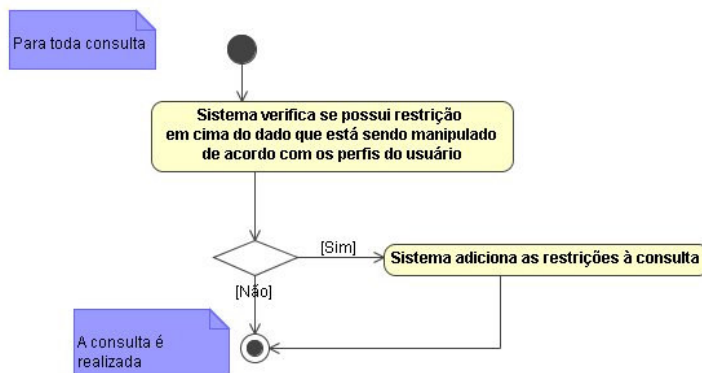


Figura 22. Lógica de adição das restrições na consulta

Essa lógica pode ser descrita da seguinte forma:

1. Para toda consulta realizada, o sistema verifica se há restrições para o dado que está sendo manipulado de acordo com os perfis do usuário;
 - 1.1. Caso haja, o sistema adiciona as restrições à consulta que será realizada;
 - 1.2. Caso não haja, a consulta é realizada.

5.4.2.2. IMPLEMENTAÇÃO

Para implementar a autorização, definimos o seguinte: todos os métodos de um serviço EJB e todos os componentes de tela (botões, campos, url, casos de uso) são considerados serviços. Cada serviço possui um identificador que é gerado automaticamente, mas que pode receber um identificador no modelo através do valor etiquetado `@andromda.service.security.code`.

A autorização é feita em cima de uma lista de serviços onde cada serviço contém o conjunto de autorizações que possui acesso a ele. Esta listagem é utilizada para verificar a permissão de um determinado usuário a um serviço e por padrão, é recuperada quando a aplicação é iniciada, através da invocação do método `listaServicos()` da classe `ControleAcessoImpl` de forma que a lista é colocada em um Singleton chamado `ServicosSingleton` que também é gerado. Porém, o gerente de segurança pode preferir criar essa lista por demanda, de forma que à medida que um serviço for solicitado, essa listagem é

criada e incrementada através da invocação do método `buscaPerfisServico()` da classe `ControleAcessoImpl` e colocado no Singleton.

- **Camada de Vista**

Como dito anteriormente, na camada de vista, todos os componentes de vista podem possuir acesso controlado. Esses componentes incluem botões, links ou campos em uma determinada tela, um caso de uso ou uma terminada tela de um caso de uso.

Na cama de vista, para realizar a autorização a URL, todas as Actions do Struts são geradas estendendo a Action `LoginAction` de forma que passos de verificação da autorização são realizados antes da solicitação ser realmente invocada. Esses passos podem ser vistos na Figura 23.

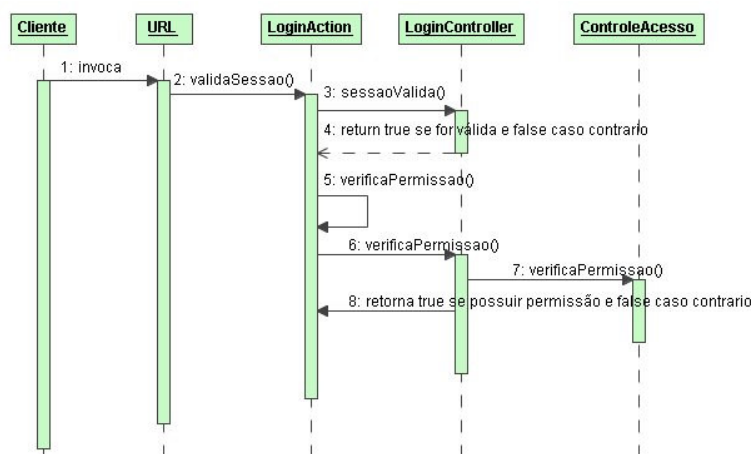


Figura 23. Implementação Autorização URL

Esses passos podem ser descritos da seguinte forma:

1. O cliente invoca qualquer URL do sistema;
2. O método `validaSessao()` do `LoginAction` é invocado;
3. O `LoginAction` delega para a classe controladora `LoginController` a responsabilidade de verificar se já existe algum `Subject` previamente autenticado no sistema invocando o método `sessaoValida()`;
4. O `LoginController` retorna `true` se houver um `Subject` autenticado no sistema e `false` caso contrário. Caso não haja, o caso de uso de login é invocado e a tela de autenticação exibida de forma que o cliente possa se autenticar;
5. Caso haja um `Subject` autenticado, o `LoginAction` invoca o método `verificaPermissao()`;

6. O `LoginAction` delega para a classe `LoginController` a responsabilidade de verificação se o cliente possui permissão para acessar a URL requisitada invocando o método `verificaPermissao()`;
7. Por sua vez, a classe `LoginController` delega para a classe `ControleAcesso` a responsabilidade de verificação se o cliente possui permissão de acesso invocando o método `verificaPermissao()`. Como dito anteriormente, a classe `ControleAcesso` possui todas as implementações dos métodos utilizados nos módulos criados neste trabalho. No caso dos serviços de vista, por padrão, todos os clientes possuem acesso a todos os serviços de vista. Um serviço de vista é negado a partir do momento que o administrador de segurança do sistema cadastra alguma autorização associada ao serviço de vista;
8. Caso o cliente possua permissão de acesso a URL solicitada, o fluxo normal é seguido, caso contrário, o caso de uso de login é invocado e a tela de autenticação exibida informando que o usuário não possui acesso à URL solicitada.

Para controlar o acesso aos componentes de vista, criamos uma *tag lib* security que possui uma *tag* chamada *containsPermission* que recebe o identificador do serviço e verifica se o usuário autenticado possui acesso a este serviço. Alteramos a geração dos jsp's para que cada serviço da tela (botão, link, campo) invocasse essa *tag lib*. Além disso, alteramos a geração do `menu.jsp` para que só fossem mostrados as opções que cliente tivesse acesso, ou seja, ao entrar no sistema, o cliente só visualizará as funcionalidades no qual ele possui acesso.

A implementação da autorização dos componentes de vista pode ser vista na Figura 24.

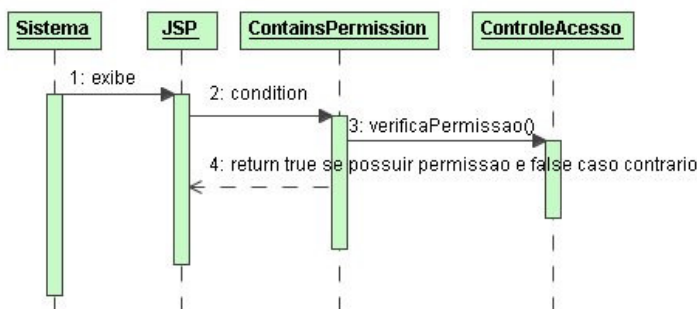


Figura 24. Implementação autorização elementos da vista

Os passos da implementação são descritos da seguinte forma:

1. O sistema invoca uma JSP para ser exibida;

2. Para cada campo, link ou botão pertencente a uma JSP, é invocado o método `condition()` da classe `ContainsPermission` referente a tag `containsPermission` da `tab lib security`;
3. A classe `ContainsPermission` delega para classe `ControleAcesso` a responsabilidade de verificar se o cliente possui acesso ao serviço solicitado através da invocação do método `verificaPermissao()`;
4. Caso o serviço não seja um campo, ele apenas pode ser ou não exibido de acordo com a permissão retornada. Caso contrário, ele pode ser exibido somente leitura ou não de acordo com o tipo de permissão.

- **Camada de Serviço**

A especificação J2EE define um modelo de segurança baseado em papéis para EJBs e componentes web. Ela permite usar um conjunto de regras de seguranças suportadas pelo servidor de aplicações. Basicamente existem dois tipos de segurança: as declarativas e programáticas. A segurança declarativa é aquela onde a configuração dos serviços de segurança é especificada utilizando um descritor padrão XML e o servidor de aplicações gerencia a segurança de acordo com o que foi definido nos descritores de implantação. Na segurança programática o código de acesso aos recursos é escrita junto com a lógica de negócio do componente.

No nosso caso, não queríamos nenhum dos tipos das seguranças fornecidas pela especificação J2EE pois não queríamos que a segurança estivesse descrita em XML por ser muito trabalhoso e custoso. Além disso, não queríamos que o código de segurança fosse escrito junto com a lógica de negócio, já que isso não deve ser tarefa do desenvolvedor.

Segundo a especificação EJB, para cada componente devemos criar suas interfaces local e/ou remota, sendo que sua implementação não implementa diretamente estas interfaces. Para evitar a detecção de inconsistências entre a implementação e a interface apenas em tempo de *deployment* e execução, uma solução é utilizar o padrão de projeto EJB chamado *Business Interface* [MARINESCU 2002]. Esta interface de negócios define as assinaturas de todos os métodos que o componente expõe. Ambas as interfaces, local e remota, e o componente implementam esta interface, possibilitando checagem de consistência em tempo de compilação. Este padrão de projeto combinado com um outro chamado *Service Locator* [ALUR *et al.* 2001], responsável pela localização do componente, oferece ao cliente maior transparência de localização. Estes dois padrões de projeto citados foram incorporados às transformações EJB.

Para implementar o controle de acesso aos serviços EJB, criamos um proxy de segurança que implementa a interface SecurityProxy pertencente ao JBossSX. Esse proxy é responsável por interceptar todas as chamadas a operações de serviços EJB e fazer a verificação se o cliente que está invocando a operação possui ou não permissão para acessar a operação.

A implementação de autorização aos serviços EJB pode ser vista na Figura 25.

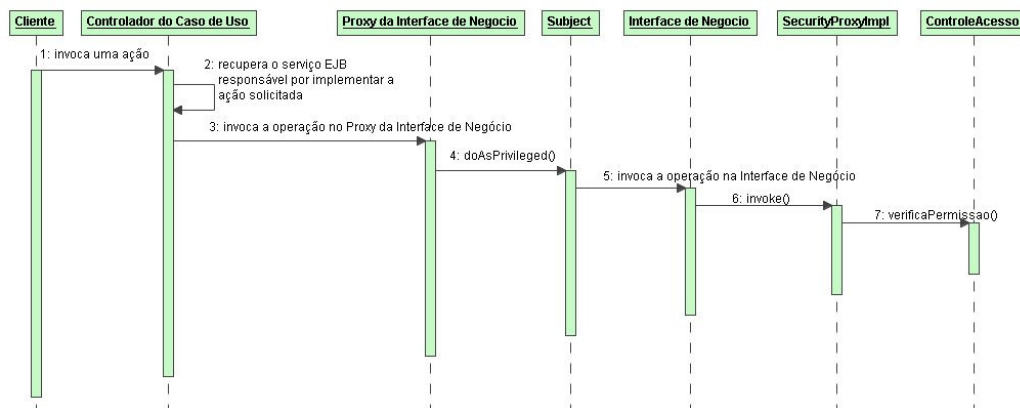


Figura 25. Autorização camada de serviço

Os passos da implementação são descritos da seguinte forma:

1. O cliente invoca uma ação;
2. O controlador do caso de uso recupera o serviço EJB responsável pela implementação da ação. O controlador não precisa saber se a segurança está ativa ou não no sistema, pois a forma que ele recupera o serviço EJB é independente de segurança;
3. O controlador do caso de uso invoca a operação ao serviço EJB. Neste momento, caso a segurança esteja ativa no sistema, a operação é invocada ao proxy da interface de negócio referente ao serviço EJB. Caso contrário, a operação é invocada à interface de negócio. Neste caso, iremos considerar que a segurança está ativada;
4. O proxy da interface de negócio instancia um novo objeto PrivilegedExceptionAction e invoca o método doAsPrivileged do Subject;
5. O objeto Subject invoca o método run() do objeto PrivilegedExceptionAction. O método run() contém a invocação à operação da interface de negócio. Com isso, o contexto de segurança do usuário logado é automaticamente propagado junto com a chamada da operação;
6. Um proxy de segurança chamado SecurityProxyImpl é configurado na geração do arquivo de configuração jboss.xml de forma que toda invocação a alguma operação a algum serviço EJB passa antes por esse proxy;

7. O SecurityProxyImpl delega a responsabilidade de verificação se o cliente possui ou não permissão para acessar a operação solicitada para a classe ControleAcesso invocando o método verificaPermissao(). Caso o cliente possua permissão, o fluxo é seguido e o cliente acessa à operação. Caso contrário, uma exceção é lançada informando que o cliente não possui permissão para acessar a operação solicitada.

- **Camada de Dados**

O controle de acesso aos dados é feito adicionando automaticamente restrições às consultas a serem realizadas. Essas restrições são adicionadas de acordo com o perfil do cliente e o tipo do dado que está sendo consultado.

Uma restrição é formada por um MoldeRestrição, um OperadorRestrição e um ou mais ValorRestricao que pode estático ou dinâmico. O relacionamento entre esses elementos pode ser visto na Figura 26.

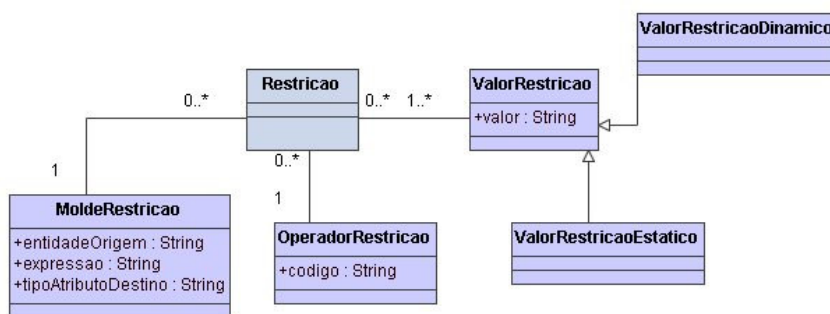


Figura 26. Modelo de Restrição

Um MoldeRestrição informa os dados da entidade que possui restrição. O parâmetro entidadeOrigem representa o caminho da classe da entidade que possui restrição, como `br.gov.defesa.sgdc.gerencia.controleMensagens.cd.DadoEmpresaPaisImpl` por exemplo, O parâmetro expressão, representa o atributo da entidade que terá restrição, com por exemplo, `valor`. Este parâmetro pode representar uma navegação, como por exemplo, `numeroIdentificacaoItem.procedenciaCatalogacao,valor`. O parâmetro tipoAtributoDestino representa o tipo do atributo que terá restrição, por exemplo, `java.lang.String`.

Um operadorRestricao representa o tipo de restrição. Seus valores atuais são: IN para igual, GE para maior ou igual, GT para maior que, LE para menor ou igual, LT para menor que e NOTIN para não igual.

Um valorRestrição representa o valor da restrição que pode ser estático ou dinâmico. No estático, o valor é fixo, enquanto que no dinâmico, o valor depende de algum

dado dinâmico, como por exemplo, o cliente autenticado ou a hora do sistema. Para isso, o valor deve ser cadastrado com uma String que representa um método que será invocado na classe ControleAcessoImpl e retornará o valor atual da restrição.

Como exemplo de uma restrição, suponha que estamos realizando uma consulta do tipo “select * from” em um tipo de dado que possua restrição em um determinado contexto com moldeRestricao com entidadeOrigem br.gov.defesa.sgdc.gerencia.controleMensagens.cd.DadoEmpresaPaisImpl, expressão valor e tipoAtributoDestino java.lang.Integer; um OperadorRestricao com código IN e dois valorRestricaoEstatico com valores 2 e 3 respectivamente, a consulta resultante seria “select * from br.gov.defesa.sgdc.gerencia.controleMensagens.cd.DadoEmpresaPaisImpl where valor in (2,3).”

A implementação da verificação e adição de restrições aos dados podem ser vista na Figura 27.

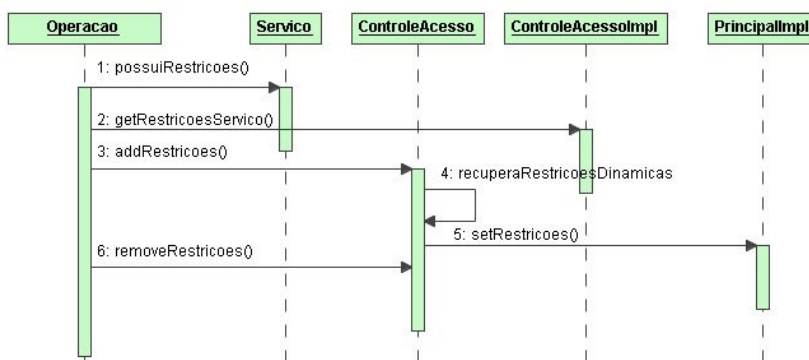


Figura 27. Verificação e Adição de restrições a dados

Os passos da implementação são descritos da seguinte forma:

1. Como cada operação de serviço EJB é considerada um serviço onde o acesso é controlado, na implementação de uma operação é gerado código para verificar se há restrições a dados para este serviço. Com isso, para cada operação, o método possuiRestricoes() do Serviço é invocado;
2. Caso o serviço possua restrições, método getRestricoesServico() é invocado para recuperar as restrições a dados da operação que está sendo executada;
3. O método addRestricoes() do ControleAcesso é invocado;
4. Antes de adicionar as restrições ao Principal, as restrições Dinâmicas são recuperadas e transformadas em Restrições Estáticas;

5. As restrições são adicionadas ao Principal através da chamada do método `setRestricoes()` do `PrincipalImpl`. As restrições são adicionadas em um Map onde a chave o perfil e valor a coleção de restrições referentes ao perfil;
6. Ao finalizar a implementação da operação, o método `removeRestricoes()` do `ControleAcesso` é invocado para remover as restrições do conjunto de restrições do `PrincipalImpl`.

A implementação da adição de restrições às consultas a serem realizadas pode ser vista na Figura 28.

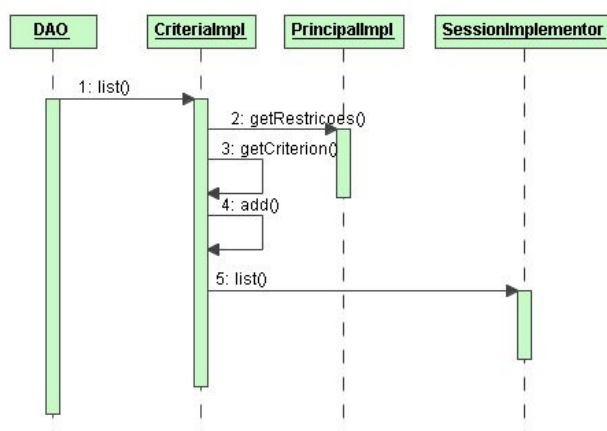


Figura 28. Adição Restrições a Consulta

Os passos da implementação são descritos da seguinte forma:

1. Um DAO (*Data Access Object*) é um padrão para persistência de dados que permite separar regras de negócio das regras de acesso a banco de dados. No nosso caso, para cada entidade modelada é gerado um DAO que é responsável por todas as funcionalidades de bancos de dados. Para realizar uma consulta, o método `list()` do `CriteriaImpl` é invocado. Essa classe é uma classe específica do hibernate que mapeia funcionalidades de SQL em Objetos e o método `list()` é responsável por realizar a consulta;
2. A classe `CriteriaImpl` do Hibernate foi alterada para acrescentar as devidas restrições na consulta a ser realizada. Para isso, o `PrincipalImpl` é recuperado e as suas restrições através do método `getRestricoes()`. As restrições são retornadas em forma de um map onde a chave é o perfil e o valor as restrições de acordo com o perfil;
3. Para todos os elementos do map cuja chave pertence ao conjunto de perfis do cliente, é verificada se alguma restrição é referente ao dado que está sendo manipulado. Para cada restrição referente ao dado que está sendo manipulado é invocado o método

getCriterion() para transformar a restrição em um Criterion que no hibernate, representa uma restrição a ser adicionada na consulta a ser realizada;

4. O Criterion é adicionado à Criteria que representa a consulta a ser realizada;
5. O método list() do SessionImplementor é invocado para realizar a consulta, que agora se encontra alterada devido à adição das restrições ao dado.

5.4.3. CASOS DE USO SAIR DO SISTEMA E TROCAR SENHA

Independentemente da utilização ou não do módulo administrativo de segurança, três casos de uso são gerados: entrar no sistema, sair do sistema e trocar de senha. O caso de uso de entrar no sistema foi apresentado durante a explicação da implementação da autenticação do arcabouço de regras de segurança. Já o caso de uso sair do sistema é responsável pela desconexão do usuário no sistema e o caso de uso trocar senha é responsável pela troca de senha do usuário.

Como visto anteriormente, o caso de uso de trocar senha pode ser invocado durante a autenticação do cliente dependendo do retorno do método precisaTrocarSenha() da classe ControleAcessoImpl. Caso o retorno seja positivo, consideramos que a troca de senha é obrigatória e o usuário não poderá acessar nenhum outro caso de uso enquanto não realizar a troca de senha.

Porém, o caso de uso trocar senha pode ser invocado a qualquer momento pelo usuário, após sua autenticação, caso este deseje trocar de senha por vontade própria.

A Figura 29 apresenta a implementação do caso de uso Trocar de senha:

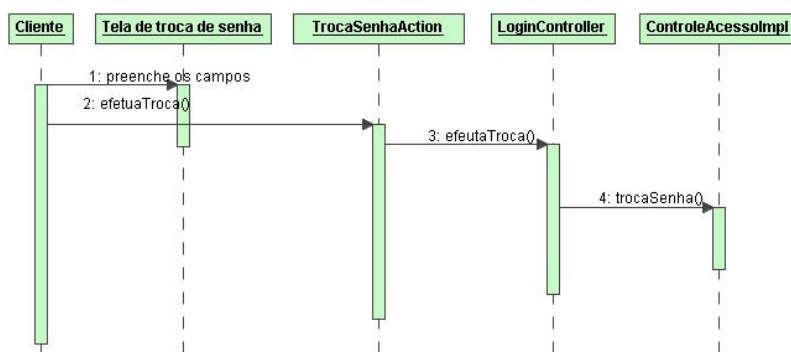


Figura 29. Implementação do caso de uso Trocar Senha

Os passos da implementação podem ser descritos da seguinte forma:

1. Quando o caso de uso Trocar senha é invocado, uma tela de troca de senha é exibida contendo os campos usuário, senha atual, nova senha e confirmação de nova senha onde o campo usuário já vem preenchido. O usuário para conseguir realizar a troca de

senha deve preencher todos os campos onde nova senha deve ser igual a confirmação de nova senha e a senha atual deve ser a senha do usuário;

2. Ao preencher os campos, cliente invoca o método `efetuaTroca()` do `TrocarSenhaAction`;
3. O `TrocarSenhaAction` invoca o método `trocaSenha()` do `LoginController`;
4. O `LoginController` delega a implementação da troca de senha pro `ControleAcessoImpl` invocando o método `trocaSenha()`. Se a troca de senha for realizada com sucesso, é exibida uma mensagem informando que a troca foi realizada com sucesso. Caso contrário, é exibida uma mensagem informando o erro ocorrido.

O caso de uso sair do sistema também pode ser acessado a qualquer momento pelo usuário quando este desejar sair do sistema. A implementação do caso de uso sair do sistema pode ser visto na Figura 30.

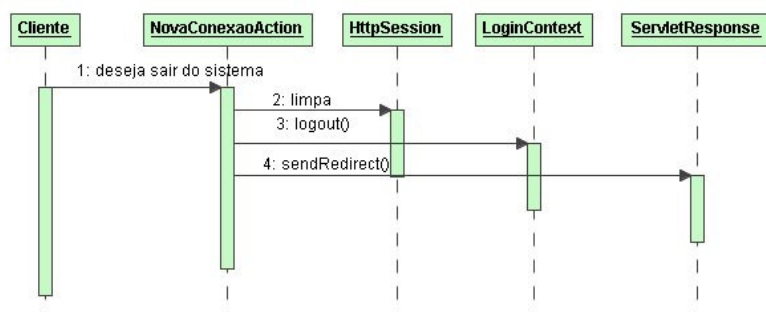


Figura 30. Implementação do caso de uso sair do sistema

Os passos da implementação podem ser descritos da seguinte forma:

1. O cliente invoca o caso de uso sair do sistema;
2. A sessão do usuário é limpa;
3. O método `logout()` é invocado do `LoginContext`;
4. A tela de autenticação é invocada.

5.5 MÓDULO ADMINISTRATIVO DE SEGURANÇA

Como dito anteriormente, o módulo administrativo de segurança representa o modelo de segurança da aplicação, que depende da política de controle de acesso da organização. Visando facilitar o desenvolvimento da aplicação, o módulo administrativo de segurança gera uma aplicação de gerência de controle de acesso padrão automaticamente, de forma que só é necessário que a aplicação tenha que criar sua própria aplicação caso a gerada não esteja de acordo com a política de segurança da organização. A aplicação de segurança gerada pelo

módulo administrativo de segurança é baseada no metamodelo que pode ser visto na Figura 11.

Ele é responsável por gerar todas as entidades referentes ao controle de acesso excluindo Usuario e Contexto, caso exista. O gerente de segurança deve informar no modelo de negócio a entidade que representa o usuario e a entidade que representa o Contexto através dos estereótipos <<User>> e <<Contexto>> respectivamente.

Os casos de uso gerados pelo módulo administrativo de segurança são:

- Cadastrar Perfil

Neste caso de uso, serão gerados todos os artefatos necessários para realizar o cadastro de perfil que inclui criação, filtragem, alteração e remoção.

- Cadastrar Contexto

Neste caso de uso, serão gerados todos os artefatos necessários para realizar o cadastro de contexto que inclui criação, filtragem, alteração e remoção.

- Cadastrar Usuário

Neste caso de uso, serão gerados todos os artefatos necessários para realizar o cadastro de usuário que inclui criação, filtragem, alteração e remoção.

- Associar Usuário a Perfil Contexto

Neste caso de uso, o usuário poderá associar um usuário a um perfil e a um contexto criando um PerfilContexto além de poder alterar, remover e filtrar as instâncias de PerfilContexto.

- Associar Perfil Contexto a Funcionalidades

Neste caso de uso, o usuário poderá associar um perfil contexto a funcionalidades do sistema criando um PerfilContextoServico além de poder alterar, remover e filtrar as instâncias de PerfilContextoServico

- Cadastrar Restrição nos dados

Neste caso de uso, serão gerados todos os artefatos necessários para realizar o cadastro de restrições nos dados que inclui criação, filtragem, alteração e remoção. No cadastro de Restrições, também serão feitos os cadastros de ValorRestricao e MoldeRestricao.

- Associar Restrição nos Dados a Perfil Contexto Serviço

Neste caso de uso, o usuário poderá associar uma restrições a um PerfilContextoServico.

Como dito anteriormente, a Classe ControleAcessoImpl estende a classe ControleAcesso e contém a implementação de todos os métodos declarados abstratos na classe ControleAcesso. Esta classe representa a ligação entre o módulo de regras de segurança e a aplicação de segurança. Os métodos declarados abstratos são os métodos que dependem da política de segurança da organização. Esses métodos devem ser implementados pelo gerente de segurança caso a organização não utilize a implementação padrão gerada pelo módulo administrativo de segurança.

Para auxiliar o cadastro da gerencia de controle de acesso, todos os serviços são listados em um script de forma que o administrador de segurança apenas precise rodar o script para inserir os serviços no banco de dados ou alterá-lo caso o que seja gerado precise de alguma modificação. Os serviços são agrupados por casos de uso que agrupam outros casos de uso, métodos de serviços e componentes de interface. Além disso, o controle de acesso a operações de serviços EJB é feito por fachadas, ou seja, se um serviço acessa outro serviço, só é necessário verificar o acesso ao primeiro serviço. Por isso, não é necessário inserir todos os serviços EJB no banco.

5.6 CONCLUSÃO

Neste capítulo apresentamos uma implementação do arcabouço ArchiMDAs. Como o arcabouço é baseado em MDA, foi necessário escolher uma ferramenta que seguisse o paradigma MDA, compatível com a abordagem adotada. Dentre as varias ferramentas existentes, optamos por estender a ferramenta AndroMDA.

Para atender o requisito de segurança por níveis, a implementação foi feita seguindo o molde MVC que consiste num padrão organizacional em camadas. Na camada de visão, utilizamos o arcabouço Struts, na camada de serviços, utilizamos EJB e na camada de persistência, utilizamos Hibernate. Como o AndroMDA já dispunha de cartuchos para cada um desses arcabouços, foram necessárias algumas adaptações no conjunto de transformações originais do AndroMDA.

Além disso, utilizamos e estendemos algumas classes do JAAS (*Java Authentication and Authorization Service*) que é uma interface portátil que possibilita que se faça autenticação de autorização de clientes em Java. Neste capítulo, descrevemos a arquitetura do JAAS tanto para autenticação quanto para autorização.

Além disso, descrevemos a lógica de implementação utilizada na autenticação e na autorização do arcabouço, assim como o código gerado. Apresentamos a implementação do módulo administrativo de segurança descrevendo os casos de uso gerados.

No próximo capítulo, iremos apresentar um estudo de caso utilizando a implementação gerada pelo arcabouço ArchiMDAs.

6 ESTUDO DE CASO

No capítulo anterior, apresentamos a implementação do arcabouço ArchiMDAs onde foram descritas a lógica de implementação e a codificação gerada pela ferramenta AndroMDA com o intuito de realizar os processos de autenticação e de autorização. Além disso, apresentamos a implementação do módulo administrativo de segurança descrevendo os casos de uso gerados. Toda codificação de segurança gerada é feita através de extensões das regras de transformação da ferramenta AndroMDA.

Neste capítulo apresentamos um estudo de caso utilizando o arcabouço ArchiMDAs com o intuito de avaliar a precisão da geração do código em relação às regras de transformação definidas na ferramenta AndroMDA. O intuito desse estudo é validar se os requisitos definidos na Seção 4.1 estão sendo atendidos.

Este capítulo está organizado da seguinte forma: na Seção 6.1 definimos o estudo que será realizado. Na Seção 6.2 descrevemos a aplicação exemplo que será utilizada nesse estudo. Na Seção 6.3 apresentamos o código gerado na aplicação. Na 6.4 realizamos a avaliação do que foi gerado. Na Seção 6.5 apresentamos a análise dos resultados de acordo com a avaliação realizada. Por último, na Seção 6.6 encerramos o capítulo, apresentando algumas considerações finais.

6.1 DEFINIÇÃO DO ESTUDO

O objetivo do estudo foi a utilização do arcabouço ArchiMDAs no desenvolvimento de sistemas, onde é adotada uma abordagem orientada a modelos, para geração automática de artefatos de segurança de controle de acesso a partir de modelos UML, visando identificar a viabilidade da utilização do arcabouço no desenvolvimento de sistemas.

O foco de qualidade do estudo é a verificação se os requisitos definidos na Seção 4.1 estão sendo atendidos. Essa validação é feita através da verificação do código gerado para cada requisito de acordo com as regras de transformação definidas na ferramenta AndroMDA.

Neste estudo, não estamos diretamente interessados em mensurar o ganho de produtividade derivado da utilização do arcabouço. Mas sim, consideramos que melhorias podem ser identificadas através de análises e estudos futuros.

Seguindo a notação *Goal-Question-Metric* (GQM) [SOLINGEN, BERGHOUT 1999], a definição do estudo é:

Analisar a utilização do arcabouço ArchiMDAs no desenvolvimento de sistemas

Com o propósito de avaliar a viabilidade de sua utilização

Referente *aos ganhos obtidos por sua utilização e as dificuldades encontradas*

No contexto de desenvolvimento de sistemas

Para isso, iremos utilizar o arcabouço em um sistema real de gerenciamento de projetos que será descrito na próxima Seção.

6.2 EXEMPLO DE APLICAÇÃO

O sistema desenvolvido neste exemplo de aplicação constitui uma parte de um sistema de gerenciamento de projetos denominado Cronus Web. Esse sistema é responsável pelo cadastro de atividades e pelo cadastro de alocações em atividades. Uma atividade pode ser um projeto ou uma tarefa e tarefa por sua vez, pode ser uma tarefa sumária ou uma tarefa padrão. Uma tarefa sumária representa uma tarefa que não pode conter alocações, ela serve para organizar as tarefas padrões por uma determinada atividade, enquanto que uma tarefa padrão representa uma tarefa que poder receber alocações.

De acordo com o processo proposto pelo arcabouço na Seção 4.5, as primeiras atividades a serem realizadas são as do cargo Arquiteto de Negócio. Porém, em nosso estudo, iremos aproveitar as definições de negócio previamente definidas na ferramenta AndroMDA feita por algum arquiteto de negócio. Isto é, neste trabalho, não realizamos as atividades referentes ao cargo Arquiteto de Negócio.

Porém, realizamos as atividades referentes ao cargo Arquiteto de Segurança que correspondem ao arcabouço ArchiMDAs apresentado nesse trabalho.

Além disso, foi necessário desenvolver as atividades do cargo Desenvolvedor de Negócio pois foi necessário criar o modelo de negócio referente ao Cronus Web.

Em relação ao cargo Gerente de Segurança, não foi necessário criar nossa própria aplicação de segurança pois a gerada pelo módulo administrativo de segurança se encaixa no modelo de segurança do Cronus Web. Porém, foi necessário adicionar no modelo de negócio os estereótipos <<User>> e <<Context>> para indicar a classe que representa Usuário e Contexto respectivamente.

Em relação ao cargo gerente de implantação, ativamos a segurança em todos os níveis e geramos o código da aplicação juntamente com o código de segurança.

Após isso, realizamos as atividades referentes ao cargo Administrador de Segurança pois foi necessário realizar o cadastro dos artefatos de segurança para podermos realizar os testes da utilização do ArchiMDAs no desenvolvimento de sistemas.

Nas próximas seções, apresentamos o modelo de negócio criado pelo Desenvolvedor de Negócio acrescido dos estereótipos de segurança, o código de segurança gerado pelo

arcabouço ArchiMDAs e os testes realizados com o intuito de validar se os requisitos definidos na Seção 4.1 estão sendo atendidos.

6.2.1. MODELO

Esse sistema possui dois perfis: desenvolvedor e líder, onde líder possui acesso a todas as funcionalidades do sistema enquanto que desenvolvedor só possui acesso ao cadastro de alocação.

Além disso, o contexto desse sistema é o projeto. Ou seja, um usuário possui um determinado perfil em um determinado projeto. De forma que ao entrar no sistema, o usuário necessita ter um e somente um projeto ativo. A lógica para ativar um projeto é de acordo com a ultima alocação feita pelo usuário. Caso ele não tenha nenhuma alocação, o caso de uso Alternar Projeto é invocado para que ele possa escolher o projeto que deseja estar ativo.

A Figura 31 mostra os casos de uso do sistema que foram considerados nesse estudo:

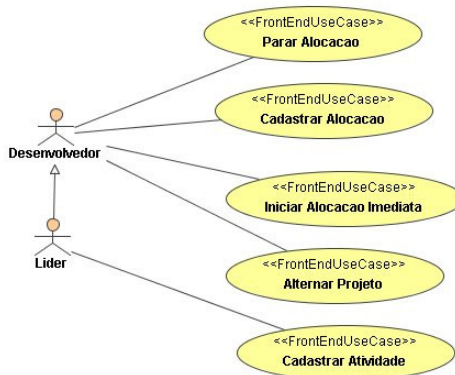


Figura 31. Diagrama de Casos de Uso

Abaixo, iremos descrever os fluxos básicos dos casos de uso que constituem esse sistema.

Alocação

- Cadastrar Alocação

Este caso de uso é composto por cinco telas principais. A primeira tela oferece ao usuário a possibilidade de consultar as alocações finalizadas por ele através do preenchimento dos campos data início, data fim e tarefa. O resultado da consulta é a listagem de alocações finalizadas por ele de acordo com o preenchimento do filtro, o total de horas alocadas durante o período consultado e total de horas alocadas por dia consultado. Esse resultado é apresentado na tela de resultado da consulta.

Nessa primeira tela, também é possível incluir uma nova alocação de forma que as informações preenchidas na tela de inclusão formarão os dados da nova alocação.

Na tela de resultado da consulta, é possível selecionar uma das alocações para edição, remoção ou detalhamento. No caso de edição, as informações sobre a alocação selecionada podem ser alteradas na tela de edição. No caso de remoção e no caso de detalhamento, os detalhes da alocação são exibidos em uma tela de detalhamento sendo que no caso de remoção, o usuário pode remover a alocação selecionada.

- **Iniciar Alocação Imediata**

Este caso de uso é composto por quatro telas e só fica disponível caso não haja nenhuma tarefa sendo alocada no momento. A primeira tela oferece ao usuário a possibilidade de consultar as tarefas disponíveis para o usuário iniciar alocação através do preenchimento de alguns campos relativos à tarefa. Essas tarefas são as que ele é responsável ou as que ele é colaborador e que estão em andamento já que ele não pode iniciar alocação em tarefas que estão aguardando início sendo colaborador. O resultado da consulta é a listagem de tarefas disponíveis para alocação de acordo com o preenchimento do filtro.

Na tela de resultado da consulta, é possível selecionar uma das tarefas para alocação ou detalhamento. No caso de alocação, uma tela é exibida contendo os campos projeto, tarefa e número da tarefa como somente leitura e os campos data início e hora início para serem preenchidos pelo usuário. No caso de detalhamento, os detalhes da tarefa são exibidos em uma tela de detalhamento e partir dessa tela, o usuário pode dar início a uma alocação invocando a tela de alocação.

- **Parar Alocação**

Este caso de uso é composto por uma tela principal e fica disponível quando há alguma alocação aberta pelo usuário, ou seja, há alguma tarefa sendo alocada no momento. Essa tela exibe as seguintes informações da alocação: projeto, nome da tarefa, número da tarefa, data início, hora início, data fim, hora fim, progresso e Observação. Esses campos por padrão, já vem preenchidos, porém, o usuário pode realizar a modificação que desejar e parar a alocação.

Atividade

- **Cadastrar Atividade**

Este caso de uso é composto por cinco telas principais. A primeira tela exibe todas as atividades, excluindo as tarefas padrão, do projeto ativo em forma de árvore.

A partir dessa árvore, o usuário pode incluir uma nova atividade de forma que as informações preenchidas na tela de inclusão formarão os dados da nova atividade.

Ao selecionar uma das atividades da árvore, as atividades filhas da atividade selecionada são consultadas, incluindo as tarefas padrões, e são exibidas abaixo da árvore de atividades.

A partir daí, uma atividade filha pode ser selecionada para edição, remoção ou detalhamento. No caso de edição, as informações sobre a atividade selecionada podem ser alteradas na tela de edição. No caso de remoção e no caso de detalhamento, os detalhes da atividade são exibidos em uma tela de detalhamento contendo além das informações básicas da atividade selecionada, a relação de alocações na atividade e a relação de pessoas que alocaram na atividade com a informação de quanto cada usuário alocou caso a atividade seja uma tarefa padrão. Sendo que no caso de remoção, o usuário pode remover a atividade selecionada.

- Alternar Projeto

Este caso de uso é composto por uma tela principal e tem como finalidade alterar o projeto ativo. O usuário seleciona o projeto que deseja a partir de uma listagem de projetos e o sistema alterna o projeto configurando os perfis do usuário associado ao projeto escolhido.

Na Figura 32 apresentamos a modelagem das classes do domínio da aplicação, com seus respectivos atributos e relacionamentos.

De acordo com o diagrama apresentado na Figura 32, uma atividade pode ser um projeto ou uma tarefa, onde uma tarefa pode ser sumária ou padrão. Essas especializações são representadas pela herança da classe TarefaPadrao e TarefaSumaria com Tarefa e pela herança da classe Projeto e Tarefa com Atividade.

Um Projeto pode possuir vários usuários, e um usuário pode participar de vários projetos. Essa relação é representada pela associação com nome usuarioParticipante.

Um usuário pode possuir varias alocações e somente tarefa padrão pode possui alocações. Um usuário pode ser solicitante e/ou responsável de varias atividades, porém só pode ser colaborador de tarefas padrões.

A classe Usuario representa a entidade Usuario do modelo RBAC estendido, e a classe Projeto a entidade Contexto. Por isso a classe Usuario possui o estereótipo <<User>> e a classe Projeto o estereótipo <<Context>>.

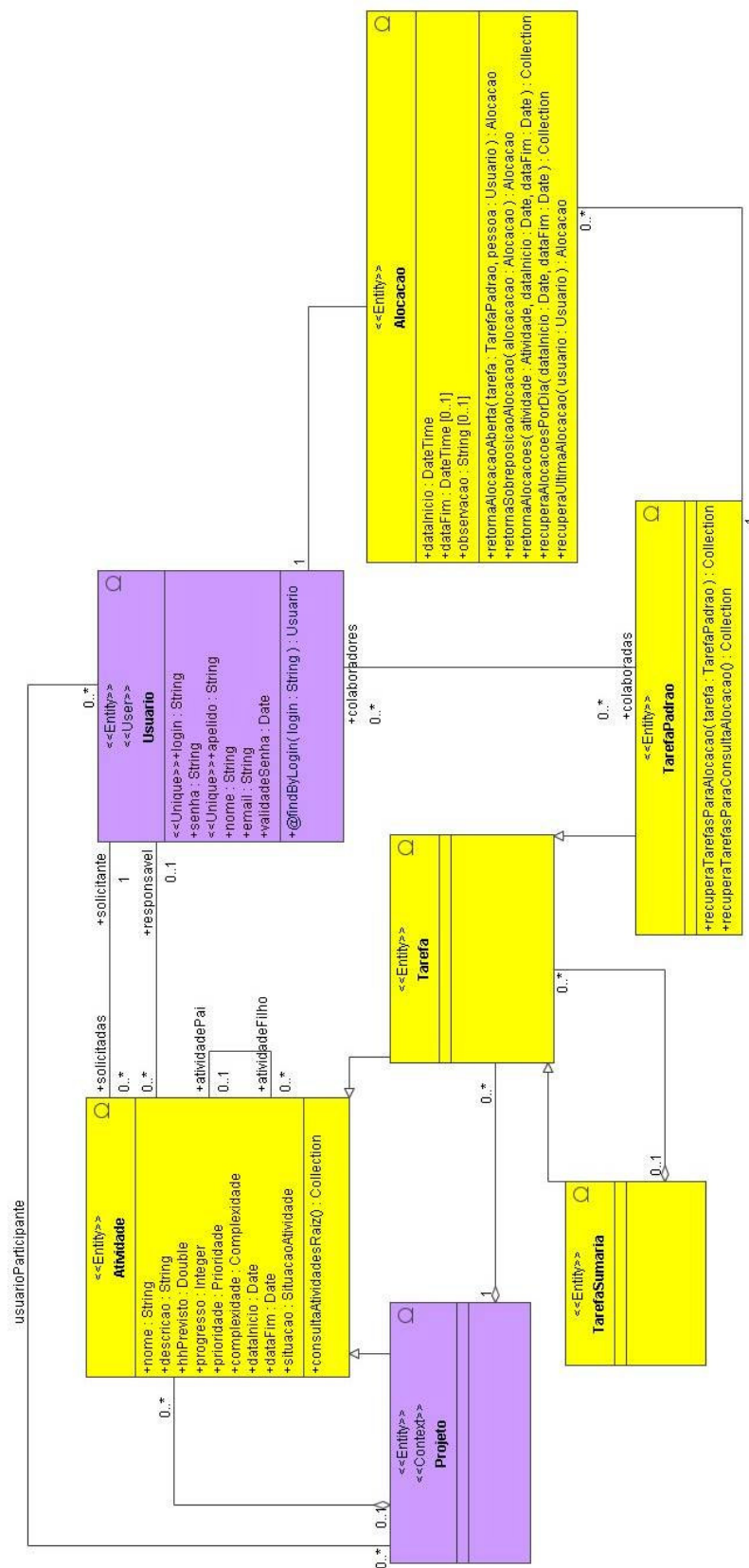


Figura 32. Diagrama de Classes de Domínio

Na Figura 33 apresentamos a modelagem das classes de serviço da aplicação, com seus respectivos atributos e relacionamentos. Nesse sistema, há dois serviços: *AtividadeHandler* e *AlocacaoHandler*. O primeiro é responsável pela manutenção das atividades enquanto que o segundo é responsável pela manutenção das alocações.

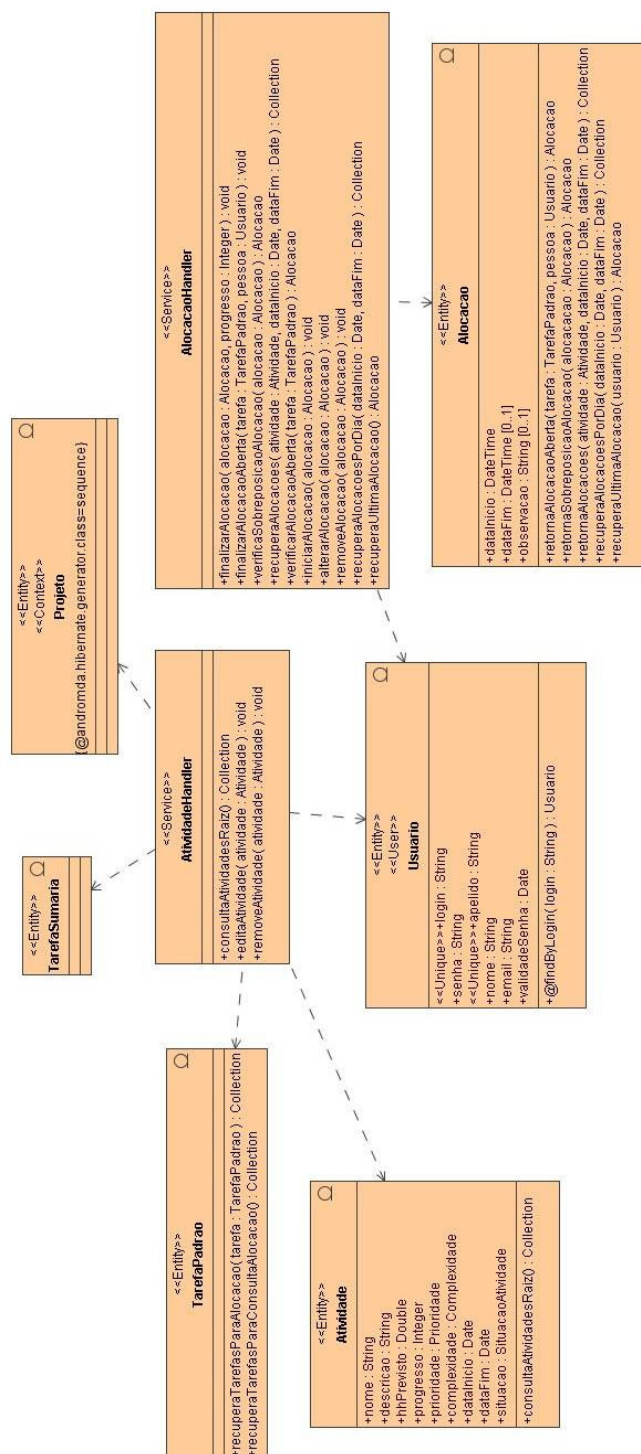


Figura 33. Diagrama de Classes de Serviço

6.2.2. GERAÇÃO DE CÓDIGO

Como dito no capítulo 5, para possibilitar a geração de código, as regras de transformação da ferramenta AndroMDA foram alteradas de forma que todos os mecanismos necessários para realizar os processos de controle de acesso, que incluem autenticação e autorizacao, fossem gerados a partir de modelos UML.

A geração do código do Cronus web foi feito utilizando a implementação do ArchiMDAs apresentada do capítulo 5. Todo o código de autenticação e autorizacao é gerado a partir dos modelos descritos na seção anterior.

Nesta Seção, iremos apresentar o código de segurança gerado considerando que a segurança está ativada em todas as camadas.

6.2.2.1. AUTENTICAÇÃO

Em relação a autenticação, foram geradas pelo arcabouço ArchiMDAs as classes ***EntrarLoginValidarAction*** e ***EntrarLoginValidarForm*** em um pacote denominado controleAcesso. Essas classes são responsáveis por receber o usuário e a senha preenchidos na tela de autenticação e delegar a responsabilidade de realizar a autenticação para a classe responsável. Além disso, foi gerada a classe ***LoginController*** que é o controlador dos casos de uso de autenticação e o jsp entrar-login.jsp que representa a tela de autenticação.

A classe ***ControleAcessoImpl*** também foi gerada. Ela representa a ligação entre a aplicação de gerencia de segurança e o módulo de regras de segurança. Nesse exemplo, iremos utilizar a aplicação de gerencia de segurança padrão gerada pelo módulo administrativo de segurança e com isso, a implementação padrão dos métodos da classe ControleAcessoImpl que pode ser visto no Apêndice 6.

6.2.2.2. AUTORIZAÇÃO

Para realizar a Autorização, vários artefatos são gerados, entre eles, a classe ActionServletImpl que é responsável por listar os serviços associados a suas autorizações ao iniciar o sistema caso o sistema não esteja utilizando autorização por demanda. Essa classe pode ser vista no Apêndice 7.

Abaixo serão descritos a geração obtida por nível de segurança.

- **Vista**

Como dito anteriormente, na camada de vista, todos os componentes de vista podem possuir acesso controlado. Esses componentes incluem botões, links ou campos em uma determinada tela, um caso de uso ou uma terminada tela de um caso de uso.

Na camada de vista, para realizar a autorização a URL, é gerada uma classe chamada `LoginAction` onde todas as Actions do Struts são geradas estendendo essa classe de forma que passos de verificação da autorização são realizados antes da solicitação ser realmente invocada.

Para realizar a autorização dos componentes de vista, criamos uma *tag lib* denominada *security* que é invocada para cada componente. Na Figura 34 é mostrado um exemplo de geração de controle de permissão a um elemento de vista. Em azul, está a chamada a *tag* `containsPermission` da *tag lib security* passando o nome do serviço *alterar.alocacao.uc.edicao.da.alocacao.salvar*. Onde esse serviço representa o botão salvar na tela de edição de alocação. Ou seja, esse botão só será apresentado na tela caso o usuário possua permissão para acessá-lo.

```
<security:containsPermission name="alterar.alocacao.uc.edicao.da.alocacao.salvar">
  <html:button onmouseover="hints.show('Salvar')" onmouseout="hints.hide()" styleId="form_submit"
onclick="setaAcao('/AlterarAlocacaoUC/EdicaoDaAlocacaoSalvar', 'validateEdicaoDaAlocacaoSalvarForm', true ,
'alterarAlocacaoUCEdicaoDaAlocacaoSalvarForm' )" property="alterarAlocacaoUCEdicaoDaAlocacaoSalvarForm" >
  <bean:message key="alterar.alocacao.uc.edicao.da.alocacao.salvar"/>
  </html:button>
</security:containsPermission>
```

Figura 34. Exemplo autorização componentes de vista

Parte da implementação da *tag* `containsPermission` pode ser vista no Apêndice 1.

- **Serviço**

Como dito anteriormente, para cada serviço EJB, é gerado um proxy da interface de negócio referente ao serviço EJB que denominamos PBI. No Apêndice 3 é mostrado parte do PBI do serviço `AlocacaoHandler`. Neste exemplo, a implementação a chamada à operação `finalizarAlocacao(br.gov.cronus.gerencia.cd.Alocacao, java.lang.Integer)`.

Ao passar pelo PBI, o proxy de segurança denominado `SecurityProxyImpl` é invocado. Essa classe é gerada automaticamente e o arquivo de configuração `jboss.xml` é gerado invocando essa classe. A Figura 35 exemplifica a geração do `jboss.xml` para o serviço EJB `AlocacaoHandler`.

```

<session>
  <ejb-name>AlocacaoHandler</ejb-name>
  <jndi-name>br.gov.cronus.gerencia.cs.AlocacaoHandler</jndi-name>
  <local-jndi-name>br.gov.cronus.gerencia.cs.AlocacaoHandlerLocal</local-jndi-name>
  <security-proxy>accessControl.SecurityProxyImpl</security-proxy>
</session>

```

Figura 35. Configuração do jboss.xml

Parte da implementação gerada do SecurityProxyImpl pode ser vista no Apêndice 2 onde é apresentada a geração do código do método invoke(Method, Object[], Object) da classe SecurityProxyImpl que é invocado em toda chamada de operação de serviço EJB.

- **Dados**

Como dito anteriormente, o controle de acesso aos dados é feito adicionando automaticamente restrições às consultas a serem realizadas. Essas restrições são adicionadas de acordo com o perfil do cliente e o tipo do dado que está sendo consultado.

Para cada operação de serviço EJB, é gerada a implementação da adição e remoção de restrições. O Apêndice 4 mostra a geração de código para o método finalizarAlocacao(br.gov.cronus.gerencia.cd.Alocacao, java.lang.Integer) do serviço EJB AlocacaoHandler.

A adição das restrições à consulta é feita ao invocar uma consulta. Para isso, a classe CriteriaImpl do Hibernate foi alterada como mostra o Apêndice 5.

6.2.2.3. CASOS DE USO SAIR DO SISTEMA E TROCAR SENHA

Para realizar a saída do sistema, foi gerada a classe NovaConexaoAction que representa a Action do Struts responsável pelo caso de uso Sair do Sistema.

O mesmo ocorre para o caso de uso Trocar Senha. As classes TrocarSenhaAction e TrocarSenhaForm, que representam a Action e o Form responsáveis pelo caso de uso Trocar Senha, foram geradas além do jsp trocarSenha.jsp que representa a tela de troca de senha.

A geração do jsp que representa o menu foi alterada, de forma que foram acrescentados um link que invoca o caso de uso Sair do Sistema e um que invoca o caso de uso Trocar Senha.

6.2.3. AVALIAÇÃO

Para realizar os testes necessários a avaliação do arcabouço, foi necessário cadastrar os artefatos de segurança de acordo com as atividades realizadas pelo Administrador de Segurança descritas na Seção 4.5.

Por padrão, a ferramenta AndroMDA gera um script de banco contendo a criação das tabelas representadas no modelo com o estereótipo <<Entity>>. Uma das extensões realizadas pelo Arquiteto de Segurança foi acrescentar as tabelas referentes à aplicação de segurança gerada pelo módulo administrativo de segurança no script gerado. Para realizar a autenticação, rodamos o script de banco gerado.

O modelo de dados gerado pode ser visto na Figura 36.

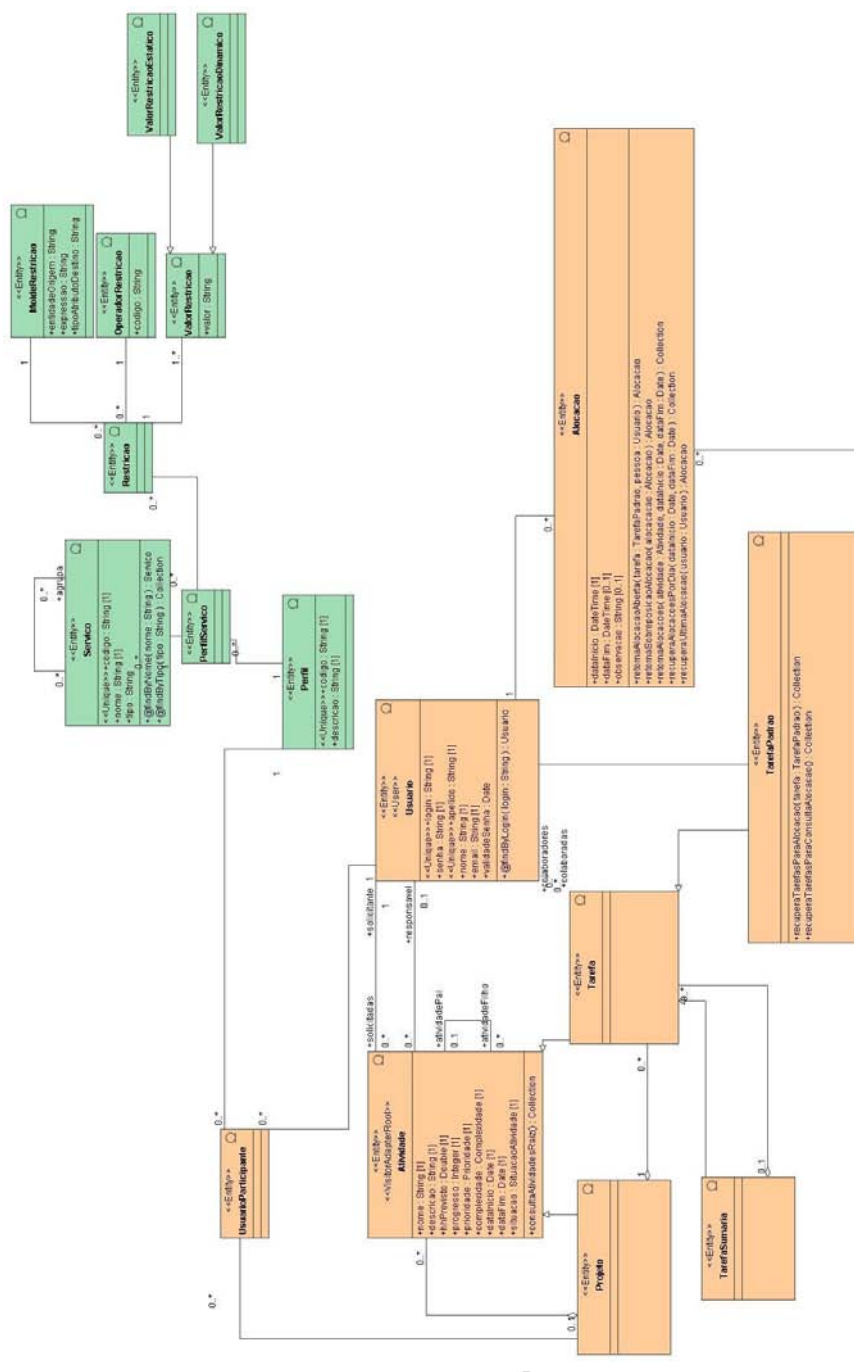


Figura 36. Modelo de Dados após transformação

Podemos perceber que após aplicar as transformações de segurança, o modelo de dados passou a ter classes referentes ao controle de acesso: *Servico*, *Perfil*, *PerfilServico*, *Restricao*, *MoldeRestricao*, *OperadorRestricao*, *ValorRestricao*, *ValorRestricaoEstatico* e *ValorRestricaoDinamico*. Todas essas classes são geradas de acordo com o metamodelo definido na Figura 11.

Para realizar os testes, criamos 5 usuários, dois projetos e dois perfis (Desenvolvedor e Líder). A Tabela 2 representa o perfil que cada usuário possui em cada projeto. Em branco significa que o usuário não participa do projeto em questão, ou seja, apenas o usuário 1 participa de dois projetos, sendo desenvolvedor no projeto 1 e líder no projeto 2.

Tabela 2. Perfil do Usuário no Projeto

	Projeto 1	Projeto 2
Usuário 1	Desenvolvedor	Líder
Usuário 2		Desenvolvedor
Usuário 3		Líder
Usuário 4	Líder	
Usuário 5	Líder	

Outra extensão realizada foi fazer com que a ferramenta AndroMDA gerasse uma listagem contendo todos os serviços do sistema de forma a auxiliar a gerencia de segurança feita pelo administrador de segurança. Inserimos todos os serviços de acordo com a listagem gerada e associamos os serviços aos perfis que possuem acesso.

Criamos três restrições dinâmicas que são descritas na Tabela 3. A semântica dessas restrições podem ser vistas na Seção 5.4.2.2 onde é apresentada a implementação da autorização por camadas, entre elas, a camada de dados.

Tabela 3. Restrições

Id Restrição	Entidade	Expressão	Operador	Valor
1	br.gov.cronus.gerencia.cs.AtividadeImpl	projeto.id	eq	getProjeto
2	br.gov.cronus.gerencia.cs.AlocacaoImpl	usuario.id	eq	getUsuario
3	br.gov.cronus.gerencia.cs.AtividadeImpl	atividadePai.id	eq	getProjeto

Essas restrições foram associadas aos serviços como mostra a Tabela 4:

Tabela 4. Restrições associadas aos Serviços por perfil

Id Serviço	Serviço	Perfil	Id Restrição
1	br.gov.cronus.gerencia.cs.AlocacaoHandlerBI .recuperaAlocacoes(br.gov.cronus.gerencia.cd .Atividade, java.util.Date, java.util.Date,	Líder	2

	java.lang.Integer)		
1	br.gov.cronus.gerencia.cs.AlocacaoHandlerBI .recuperaAlocacoes(br.gov.cronus.gerencia.cd .Atividade, java.util.Date, java.util.Date, java.lang.Integer)	Desenvolvedor	2
2	br.gov.cronus.gerencia.cs.AlocacaoHandlerBI .recuperaAlocacoesPorDia(java.util.Date, java.util.Date)	Líder	2
2	br.gov.cronus.gerencia.cs.AlocacaoHandlerBI .recuperaAlocacoesPorDia(java.util.Date, java.util.Date)	Desenvolvedor	2
3	br.gov.cronus.gerencia.cs.AtividadeHandlerBI .consultaAtividadesRaiz()	Líder	1
3	br.gov.cronus.gerencia.cs.AtividadeHandlerBI .consultaAtividadesRaiz()	Líder	3

A Tabela 4 mostra que os serviços com id 1 e 2 só podem retornar as alocações cujo id do usuário seja igual ao valor do método `getUsuario()` implementado na classe `ControleAcessoImpl`. Essa restrição está configurando tanto para o perfil desenvolvedor quando para o perfil Líder.

Já o serviço cujo id é 3 só pode retornar as atividades cujo id do projeto seja igual ao valor do método `getProjeto()` implementado na classe `ControleAcessoImpl`. Esta restrição está configurada somente para o perfil Líder já que desenvolvedor não tem permissão para acessar esse serviço.

Além disso, esse serviço só pode retornar as atividades cujo id da atividade pai seja igual ao valor do método `getProjeto()`. Esta restrição também está configurada somente para o perfil Líder já que desenvolvedor não tem permissão para acessar esse serviço.

Nesta aplicação, utilizamos o banco Postgres e após realizar a administração dos artefatos de segurança, criamos uma política de aplicação no arquivo `login-config.xml` no Jboss da seguinte forma:

```
<application-policy name="cronus">
  <authentication>
    <login-module code="org.jboss.security.ClientLoginModule" flag="required"/>
    <login-module code="accessControl.LoginModuleImpl" flag="required">
      <module-option name="ds.JndiName">java:/postgresDS</module-option>
      <module-option name="unauthenticatedIdentity">guest</module-option>
      <module-option name="principalClass">accessControl.PrincipalImpl</module-option>
      <module-option name="hashEncoding">hex</module-option>
      <module-option name="hashAlgorithm">md5</module-option>
      <module-option name="principalsQuery">
        select SENHA from USUARIO where LOGIN=?</module-option>
      <module-option name="rolesQuery">
        select P.ID, 'Roles'
        from USUARIO U,
        USUARIO_PARTICIPANTE UP,
        PERFIL P
        where U.LOGIN=?
        and P.ID = UP.PERFIL_FK
```

```

                                and U.ID = UP.USUARIO_FK
                                </module-option>
                            </login-module>
                        </authentication>
                    </application-policy>

```

6.2.3.1. CASOS DE TESTE

Para validar que os requisitos levantados estavam sendo atendidos, rodamos vários testes que serão apresentados nesta Seção.

Ponto único de acesso ao sistema (single-sign-on)

A Figura 37 representa a tela de autenticação do sistema gerada pelo arcabouço ArchiMDAs. A implementação gerada pelo arcabouço ArchiMDAs valida se os campos login e senha estão preenchidos e caso ocorra algum erro durante a autenticação, como por exemplo, senha ou login inválido, esse erro é exibido na tela.

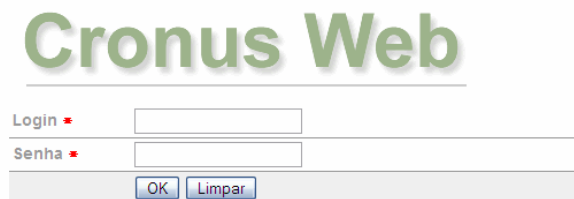


Figura 37. Tela de autenticação

Para qualquer tentativa de acesso a recursos, o sistema deve verificar se o usuário está autenticado. Caso não esteja, o usuário deve ser encaminhado a realizar sua autenticação.

Para qualquer tentativa de acessar qualquer URL do sistema sem estar autenticado o sistema redirecionava para a tela de autenticação e qualquer erro que ocorresse durante a autenticação era exibido na tela.

Usuário com nome de usuário (login) único e uma senha

O campo login e senha são gerados na tabela Usuario onde o campo login é único.

Os usuários devem ser organizados em papéis

O arcabouço gera a tabela perfil associada a tabela serviço e todas as regras de autenticação são feitas em relação a perfil.

Cada usuário pode possuir vários papéis e estes podem estar ativos ao mesmo tempo e não necessariamente deve haver uma hierarquia entre eles

Grande parte dos testes foram feitos com o usuário 1 pois ele possuía tanto o perfil de líder quanto o de desenvolvedor

Cada papel deve possuir um conjunto de permissões que representam acessos a recursos do sistema

O arcabouço gera a tabela perfil associada a tabela serviço que representa o conjunto de permissões do papel.

Deve ser possível identificar o usuário autenticado, assim como suas informações, em qualquer parte do sistema

As informações do usuário autenticado se localizam no Subject que é repassado em todas as camadas.

Permitir que os usuários visualizem somente aquilo a que eles têm acesso

Quando o usuário se autenticava no projeto 2 onde ele possui perfil de líder, a página principal era exibida como mostra a Figura 38.

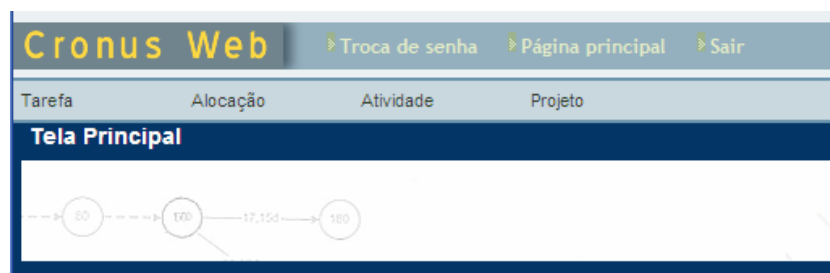


Figura 38. Menu de líder

Ao alterar o projeto para o projeto 1 onde ele possui perfil de desenvolvedor, a página principal era exibida como na Figura 39. Podemos perceber que no caso de desenvolvedor, só aparecem as opções Alocação e Projeto que estão de acordo com a Figura 31.

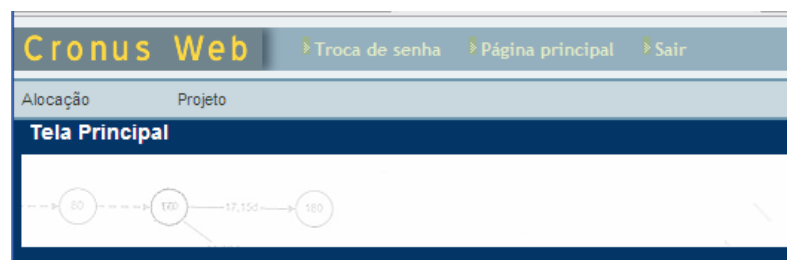


Figura 39. Menu Desenvolvedor

Deve haver um mecanismo que contém a lógica implementada das regras de segurança e que seja acessado a cada requisição de forma a avaliar a política de segurança corretamente

Na camada de vista, esse mecanismo é a tag lib security enquanto que na camada de serviço é a classe SecurityProxyImpl.

Casos de uso Trocar Senha e Sair do Sistema

Podemos perceber que as opções Troca de Senha, Página Principal e Sair são apresentadas no menu automaticamente.

Ao invocar a opção Sair, o usuário automaticamente era desconectado do sistema e a tela de autenticação era exibida.

Ao invocar a opção Página Principal, o caso de uso principal do sistema era exibido. Esse caso de uso é o que possui estereótipo <<FrontEndApplication>>.

Ao invocar a opção Troca de senha, a tela de troca de senha era exibida. Ela pode ser vista na Figura 40. Ao preencher os campos, estes eram validados e caso não houvesse algum erro no preenchimento dos campos, a troca de senha era realizada com perfeição e a nova senha já era criptografada de acordo com o algoritmo definido no login-config.xml. Para realizar a troca de senha, todos os campos devem estar preenchidos, a nova senha deve ser diferente da senha atual e o valores dos campos Repita nova senha e Nova senha devem ser iguais.

Figura 40. Tela de Troca de Senha

Realizamos o teste de troca de senha obrigatória e ao realizar a autenticação, o caso de uso Trocar Senha era invocado e o usuário só conseguia acessar qualquer outro recurso do sistema ao realizar a troca de senha.

Utilização do Protocolo Https

Todas as páginas geradas pelo arcabouço ArchiMDAs utilizavam o protocolo https assim como foi especificado.

Os recursos devem ser protegidos contra acesso não autorizado de forma que toda requisição seja avaliada para saber se ela possui ou não permissão para acessar o recurso.

Tentamos acessar os casos de uso que não estavam aparecendo no caso de desenvolvedor a partir de suas URLs e era exibida um erro de acesso negado. O mesmo

aconteceu quando tentamos acessar algum serviço EJB sem utilizar a camada de vista ou mesmo utilizando a camada de vista, porém não tendo permissão.

Um usuário pode estar associado a um contexto de uso

Utilizamos o contexto de Projeto de forma que o usuário poderia participar de vários projetos com perfis diferentes.

Restrições a Dados

No caso das restrições, ao invocar o caso de uso CadastrarAlocacao, só foram retornadas as alocações do usuário autenticado, de forma que o usuário só conseguisse inserir, remover, alterar e visualizar alocações feitas por ele. E ao invocar o caso de uso CadastrarAtividade, só foram retornadas as atividades do projeto corrente, de que forma que o usuário só conseguisse inserir, remover, alterar e visualizar atividades do próprio projeto.

Um usuário pode estar associado a um contexto de uso

Utilizamos o contexto de Projeto de forma que o usuário poderia participar de vários projetos com perfis diferentes.

Autorização por níveis

Cada camada (vista, serviço e dado) possuía seus próprios mecanismos de autorização

Ativação e Desativação de Segurança (Switch on/off)

Para testar a independência do negocio propriamente dito com a segurança, desativamos a segurança em todos os níveis e o sistema continuou funcionando, só que sem os artefatos de segurança. Ou seja, a inserção de segurança no sistema é transparente para o desenvolvedor e pode ser ativada e desativada sem interferir no negócio.

6.2.4. ANÁLISE DOS RESULTADOS

A análise dos resultados foi feita seguindo os requisitos definidos na Seção 4.1, onde cada requisito definido foi validado. Os resultados obtidos foram bastante satisfatórios e podem ser vistos na Tabela 5.

Tabela 5. Resultados

Requisito	Testado	Observação
Ponto único de acesso ao sistema	Sim	Tela de autenticação como ponto único
Usuário com nome de usuário (<i>login</i>) único e uma senha	Sim	O campo login da tabela Usuario é gerado como único
Para qualquer tentativa de acesso a recursos, o sistema deve verificar se o usuário está autenticado. Caso não esteja, o	Sim	Para qualquer tentativa de acesso ao sistema sem estar autenticado, o sistema redirecionava para o ponto único de acesso o sistema,

usuário deve ser encaminhando a realizar sua autenticação.		ou seja, a tela de autenticação
Os usuários devem ser organizados em papéis.	Sim	A tabela perfil é gerada associada a tabela serviço e todos as regras de autenticação são feitas em cima do perfil.
Cada usuário pode possuir vários papéis e estes podem estar ativos ao mesmo tempo e não necessariamente deve haver uma hierarquia entre eles.	Sim	Foi testado com o usuário 1 que era líder e desenvolvedor
Cada papel deve possuir um conjunto de permissões que representam acessos a recursos do sistema.	Sim	Representado pela associação de Perfil com Serviço
Deve ser possível identificar o usuário autenticado, assim como suas informações, em qualquer parte do sistema	Sim	As informações do usuário autenticado se localizam no Subject que é repassando em todas as camadas
Os recursos devem ser protegidos contra acesso não autorizado de forma que toda requisição seja avaliada para saber se ela possui ou não permissão para acessar o recurso. A princípio, o usuário terá uma visão completa do sistema, podendo visualizar todas as opções. Porém, ao tentar acessar alguma operação a qual ele não possui acesso, o sistema deverá emitir um erro.	Sim	Em cada camada, possuem mecanismos de controle de acesso para validar o acesso. A princípio, todos os serviços da camada de vista são abertos, ou seja, o usuário tem uma visão completa do sistema e vai fechando à medida que o administrador vá dando as permissões. Porém, como na camada de serviços todos os serviços são fechados, ao tentar acessar uma operação a qual não possui acesso, o sistema emite um erro.
Deve haver um mecanismo que contém a lógica implementada das regras de segurança e que seja acessado a cada requisição de forma a avaliar a política de segurança corretamente.	Sim	Na camada de vista, podemos considerar que o mecanismo é a <i>tag lib security</i> enquanto que na camada de serviço é a classe <i>SecurityProxyImpl</i>
Um usuário pode estar associado a um contexto de uso	Sim	Utilizamos o contexto de Projeto de forma que o usuário poderia participar de vários projetos com perfis diferentes
Deve ser possível associar restrições a dados.	Sim	Utilizamos três restrições dinâmicas
A autorização deve ser feita por níveis, ou seja, a aplicação sendo dividida em camadas deve possuir mecanismos de	Sim	Cada camada possuía seus próprios mecanismos de autorização

autorização que controle o acesso a recursos em cada camada		
---	--	--

Podemos perceber que todos os requisitos foram testados e validados de forma que o sistema se encontrava protegido e todos os recursos necessários para realização do controle de autenticação e autorização foram gerados automaticamente.

6.3 CONCLUSÃO

Neste capítulo apresentamos um estudo de caso utilizando a implementação do arcabouço ArchiMDAs apresentada no capítulo anterior.

Para isso, utilizamos uma aplicação exemplo denominada Cronus Web que sistema de gerenciamento de projetos simplificado.

Após a geração do código e a realização dos testes, foi demonstrado que todos os requisitos definidos na Seção 4.1 foram validados e que o resultado de todos os testes foram satisfatórios.

No próximo capítulo tecemos as conclusões acerca do trabalho realizado nesta dissertação bem como a definição de pontos passíveis de serem explorados em trabalhos futuros.

7 CONCLUSÃO

Apesar de grande parte dos sistemas de informação possuir a segurança como requisito fundamental, constantemente há notícias sobre vulnerabilidades e falhas de segurança [FINK, KOCH, PAULS 2004].

Isto se deve por vários motivos, mas principalmente pelo fato deste requisito ser raramente considerado nos estágios iniciais do desenvolvimento de software e ser delegado a segundo plano ao longo do mesmo. Um outro motivo é a lacuna existente entre as soluções teóricas e o que é de fato implementado na área de segurança. Além disso, na maioria das vezes, os projetistas de sistemas não possuem experiência em segurança e acabam provocando problemas de segurança tanto na arquitetura da aplicação quanto na lógica implementada.

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 2000]. 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.

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

Como dito, atualmente, existe uma grande lacuna entre as soluções teóricas e o que é de fato implementado na área de segurança. Portanto, padrões de projeto podem e devem ser aplicados à segurança como forma de preencher essa lacuna promovendo várias vantagens, entre elas: novatos podem atuar como especialistas, especialistas em segurança podem identificar, nomear e discutir problemas e soluções de modo mais eficiente, problemas podem ser resolvidos de uma forma estruturada, dependências de componentes podem ser identificadas e consideradas de forma apropriada.

Contudo, para obter os benefícios da utilização de padrões de projeto é necessário saber quando e como utilizá-los a partir de suas definições, pois os desenvolvedores podem acabar introduzindo falhas de segurança. É necessário que haja analistas e programadores com alto grau de conhecimento de padrões de projeto de segurança e de arcabouços de segurança e

esse conhecimento faz com o que o custo do desenvolvimento de sistemas aumente consideravelmente.

Neste contexto, uma abordagem orientada a modelo pode auxiliar o processo de desenvolvimento de sistemas seguros. Com isso, podemos obter uma implementação controlada dos padrões de projeto e desta forma, garantir aderência às diretrizes arquiteturais da organização e às questões de segurança.

No âmbito de segurança, a utilização de MDA com segurança faz com que a segurança fique integrada na modelagem de sistemas à medida que modelos de negócio 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.

Dessa forma, a utilização de MDA com segurança auxilia a criação de arcabouços de segurança flexíveis 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 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.

Algumas abordagens já utilizam MDA para desenvolver arcabouços de segurança [BASIN, DOSER 2005; FINK, KOCH, PAULS 2004; JIN 2006]. Em todas as abordagens encontradas na literatura, o foco está concentrado na criação de modelos onde a definição de políticas de controle de acesso é feita utilizando-se perfis UML que estendem a UML com elementos específicos que representam modelos de permissões.

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

Além disso, a junção dos dois modelos, o de negócio e o de segurança, pode provocar uma sobrecarga de atribuições ao desenvolvedor do sistema, elevando a complexidade do desenvolvimento e, conseqüentemente, 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.

Neste trabalho, apresentamos o arcabouço ArchiMDAs que é um arcabouço de segurança de controle de acesso orientado a modelo, onde a segurança é introduzida no sistema através de transformação entre modelos. Desta forma, os requisitos de segurança das aplicações geradas pelo arcabouço não são suscetíveis a erros ou alteração do código, garantindo a qualidade do sistema gerado.

O ArchiMDAs é dividido em dois módulos, um responsável pelas regras de segurança, denominado módulo de regras de segurança e outro responsável pela administração de segurança específica de cada aplicação, denominado módulo administrativo de segurança.

O módulo de regras de segurança é responsável pela regras de autenticação e autorização dos artefatos da aplicação onde o processo de modelagem das regras de negócio não sofre alterações, o projetista deve focar somente no negócio, sem se preocupar com a segurança. Os requisitos de segurança são acrescentados ao sistema de forma automatizada e ortogonal ao desenvolvimento das regras de negócio, através dos mecanismos oferecidos pelo ArchiMDAs.

Já o módulo administrativo de segurança representa o modelo de segurança da aplicação, que depende da política de controle de acesso da organização. Visando facilitar o desenvolvimento da aplicação, uma aplicação administrativa 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 caso este não esteja de acordo com a política de segurança da organização.

Neste trabalho apresentamos uma implementação do arcabouço ArchiMDAs e um estudo de caso com o objetivo de analisar a utilização do arcabouço ArchiMDAs no desenvolvimento de sistemas e avaliar a viabilidade de sua utilização no contexto de desenvolvimento de sistemas.

O arcabouço ArchiMDAs já foi utilizado e testado em vários sistemas reais, entre eles, podemos citar um sistema militar brasileiro que possui uma política de segurança de grau elevado, permitindo que pudéssemos perceber inúmeras vantagens e contribuições em sua utilização. Dentre elas, podemos citar:

- O desacoplamento entre a implementação do negócio propriamente dito e a solução de segurança, facilita a evolução da solução fazendo com que ela não se torne obsoleta, principalmente pelo fato do código de segurança ser gerado automaticamente, através de transformações;
- ArchiMDAs foi planejado implementando os principais padrões de segurança e com isso garantindo formalmente ao cliente a segurança da aplicação;

- Provê proteção em diferentes camadas;
- Requisitos de segurança não ficam suscetíveis a erros derivados de evoluções no código da aplicação já que o código responsável pela segurança é gerado de forma automatizada e ortogonalmente ao desenvolvimento da aplicação;
- Contribui na integração de requisitos de segurança durante o desenvolvimento do sistema;
- Neste trabalho, não estamos diretamente interessados em mensurar o ganho de produtividade derivado da utilização do ArchiMDAs. Contudo, devido a sua utilização em sistemas reais, há indícios que ele simplifica o desenvolvimento de sistemas de software, aumentando a produtividade e a qualidade das especificações de segurança e com isso, aumentando consideravelmente a qualidade e a manutenibilidade dos sistemas construídos.

Apesar de termos realizado um estudo de observação, sabemos que este não é suficiente para uma validação efetiva do trabalho, embora ofereça indícios de uma boa aceitação da proposta. Questões que consideramos importantes e que poderão ser tratadas em trabalhos futuros são:

- Um estudo de caso mais detalhado, envolvendo uma aplicação de maior complexidade, com mais participantes e métricas mais objetivas e quantitativas, dos ganhos obtidos quando da utilização do arcabouço;
- Inserir outros aspectos de segurança, como por exemplo, assinaturas digitais e auditoria;
- Possibilitar a definição das restrições com alguma linguagem declarativa;
- A versão da ferramenta AndroMDA utilizada para realizar a implementação das regras de transformação não possui suporte para geração de PSMs ou seja, o código é gerado diretamente a partir do PIM. Como trabalho futuro, podemos considerar a migração para uma versão mais recente do AndroMDA de forma a gerar o PSM contendo o modelo de segurança.

REFERÊNCIAS

- ACEGI (2007) “ACEGI Security”, disponível em <http://acegisecurity.sourceforge.net>, acesso em dezembro de 2007.
- ALEXANDER, C. (1978) “A Pattern Language”, Oxford Press, Oxford, R. Unido.
- ALEXANDER, C. (1979) “A Timeless Way of Building”, Oxford Press, Oxford, R. Unido.
- ALUR, D *et al.* (2001) “Core J2EE Patterns: Best Practices and Design Strategies”, Prentice Hall.
- AMOROSO, E. G. (1994) “Fundamentals of Computer Security Technology”. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1994, ISBN: 0-13-108929-3.
- ANDROMDA (2007) “AndroMDA”, disponível em <http://www.andromda.org>, acesso em Setembro de 2007.
- BASIN, D. AND DOSER, J. (2002) “SecureUML: A UML-Based Modeling Language for Model-Driven Security”. 5th International Conference on the Unified Modeling Language, Lecture Notes in Computer Science 2460.
- BASIN, D.; DOSER, J. (2005) “Model Driven Security: from UML Models to Access Control Infrastructures. In 5th International School on Foundations of Security Analysis and Design”, FOSAD.
- BISHOP, M. (2003) “Computer Security: Art and Science”, Addison Wesley.
- BLAKLEY, B.; HEALTH, C. (2004) “Security Design Patterns”, Technical Guide, 2004, Doc. No. G031, ISBN: 1-931624-27-5.
- BROSE, G. (2001) “Access Control Management in Distributed Object Systems”, PhD thesis, Freie Universität Berlin.
- CARUSO, C.; STEFFEN, F. (1999) “Segurança em Informática e de Informações”, São Paulo: Editora Senac São Paulo.
- DENNING, D. (1982) “Cryptography and data security”, Addison-Wesley.
- ECLIPSE (2007) “Eclipse”, disponível em <http://www.eclipse.org/>, acesso em julho de 2007.
- EJB (2007) “Enterprise Java Beans”, disponível em <http://java.sun.com/products/ejb>, acesso em setembro de 2007.
- EMF (2007) “Eclipse Modeling Framework”, disponível em <http://www.eclipse.org/emf/>, acesso em julho de 2007.
- GEF (2007) “Graphical Editing Framework”, disponível em <http://www.eclipse.org/gef/>, acesso em julho de 2007.
- GOGUEN, J.A.; MESAJUER, J. (1982) “Security Policies And Security Models”, Proceedings of IEEE symposium on Research in Security and Privacy.
- FERNANDEZ, E. (2000) “Metadata and authorization Patterns”, Report TR-CSE-00-16, Dept. of Computer Science and Eng., Florida Atlantic University, May.
- FERNANDEZ, E.; PAM, R. (2001) “A Pattern Language for Security Models”, Dept. of Computer Science & Engineering, Florida Atlantic University, 2001.
- FERRAILOLO, D.F.; BARKLEY, J.F.; KUHN R. (1999) “A Role Based Access Control Model and Reference Implementation within a Corporate Intranet”, Proc. of. National Institute of Standards and Technology, Vol. 2, nº. 1, February, p. 34-64.

- FINK, T., KOCH, M., PAULS, K. (2004) "An MDA approach to Access Control Specifications Using MOF and UML Profiles", In Proceedings 1st International Workshop on Views On Designing Complex Architectures.
- GAMMA, E. *et al.* (1995) "Design Patterns: Elements of Reusable Object-Oriented Software", ISBN: 0-201-63361-2.
- GODIK, S.; MOSES T. (2003) "eXtensible Access Control Markup Language (XACML) Version 1.0", OASIS Standard.
- GOGUEN, J.A.; MESAJUER, J. (1982) "Security Policies And Security Models", Proceedings of IEEE symposium on Research in Security and Privacy.
- GUERRA, E. M.; PAICA, R. C.; FERNANDES, C. T. (2004) "RBAC com contextos: modelo de controle de acesso baseado em papéis para sistemas web utilizados por várias divisões de uma organização". 6º Simpósio Segurança em Informática, São José dos Campos, São Paulo, Brasil.
- HDIV (2007) "HTTP Data Integrity Validator", disponível em <http://www.hdiv.org/>, acesso em dezembro de 2007.
- HIBERNATE. (2007) "Hibernate", disponível em <http://www.hibernate.org>, acesso em setembro de 2007.
- JAAS. (2007) "JAAS", disponível em <http://java.sun.com/products/jaas/>, acesso em outubro de 2007.
- JAVA. (2007) "Java Technology". Sun Microsystems, disponível em <http://java.sun.com>, acesso em setembro 2007.
- JBOSS. (2007) "JBoss", disponível em <http://docs.jboss.org/jbossas/jboss4guide/r2/html/index.html>, acesso em outubro de 2007.
- JBOSSSX. (2007) "JBossSX", disponível em <http://docs.jboss.org/jbossas/jboss4guide/r2/html/ch8.chapter.html>, acesso em outubro de 2007.
- JIN, X. (2006) "Applying Model Driven Architecture approach to Model Role Based Access Control System", Master of Science in System Science, University of Ottawa, Ottawa, Ontario, Canada.
- KLEPPE, A.; WARMER, J.; BAST, W. (2002) "MDA Explained: The Model Driven Architecture: Practice and Promise". Addison-Wesley, 2002. ISBN: 4-8443-1869-1
- KIENZLE, D. M.; ELDER, M. C. (2002) "Final Technical Report: Security Patterns for Web Application Development", DARPA Contract F30602-01-C-0164, disponível em http://www.modsecurity.org/archive/securitypatterns/dmdj_final_report.pdf, acesso em junho de 2007.
- KIENZLE, D. M. *et al.* (2002) "Security Patterns Repository Version 1.0", disponível em http://www.modsecurity.org/archive/securitypatterns/dmdj_repository.pdf, acesso em junho de 2007.
- LANDWEHR, C. E. (1981) "Formal models for computer security", Computing surveys.
- LARMAN, C. (2004) "Applying UML and Patterns : An Introduction to Object-Oriented Analysis and Design and the Unified Process", ISBN: 0131489062.
- MAIA, N. (2006) "ODYSSEY-MDA: uma abordagem para a transformação de modelos", mestre em ciências em engenharia de sistemas e computação, Coordenação dos Programas

- de Pós-graduação de Engenharia da Universidade Federal do Rio de Janeiro, UFRJ, Rio de Janeiro, Brasil.
- MACKENZIE, D.E.; POTTINGER, G. (1997) "Mathematics, Technology, and Trust: Formal Verification", Computer Security, and the U.S. Military, IEEE Annals of the History of Computing, Vol. 19, No 3.
- MARINESCU, F. (2002) "EJB Design Patterns Advanced Patterns, Processes, and Idioms", Wiley Computer Publishing, John Wiley & Sons, Inc., ISBN: 0-471-208310
- MELLOR, S.J., SCOTT, K., UHL, A., *et al.* (2004) "MDA Distilled Principles of Model-Driven Architecture", Addison-Wesley.
- MISF (2007) "Microsoft® Internet Security Framework", disponível em <http://www.microsoft.com/security/default.mspx>, acesso em dezembro de 2007.
- MOF (2002) "Meta-Object Facility", disponível em <http://www.omg.org/cgi-bin/doc?formal/2002-04-03>, acesso em julho de 2007.
- MVC (2007) "MVC", disponível em <http://java.sun.com/blueprints/patterns/MVC-detailed.html>, acesso em novembro de 2007.
- NAKAMURA, E.; GEUS, P. (2003) "Segurança de Redes". São Paulo, Futura.
- NET (2007) "Microsoft .NET", disponível em <http://www.microsoft.com/net>, acesso em setembro de 2007.
- OASIS (2007) "OASIS: Advancing E-Business Standards Since 1993", disponível em: <http://www.oasis-open.org>, acesso em junho de 2007.
- OMG (2003) "MDA Guide V1.0.1", disponível em <http://www.omg.org/cgi-bin/doc?omg/03-06-01>, acesso em julho de 2007.
- OPENGROUP (2007) "OPENGROUP", disponível em <http://www.opengroup.org/>, acesso em junho de 2007.
- OSBORN, S.; SANDHU, R.; MUNAWER, Q. (2000) "Configuring Role-Based Access Control to Enforce Mandatory and Discretionary Access Control Policies", ACM Transactions on Information and System Security, Vol. 3, no. 2.
- RABAC XACML (2004). "XACML Profile for Role Based Access Control (RBAC)", Organization for the Advancement of Structured Information Standards, disponível em <http://docs.oasis-open.org/xacml/cd-xacml-rbac-profile-01.pdf>, acesso em junho de 2007.
- SANDHU Ravi S., SAMARATI Pirangela. (1994) "Access Control: Principles and Practice", IEEE Communications, v.32, n.9, p.40-48.
- SANDHU, R. *et al.* (1996) "Role-Based Access Control Models", IEEE Computer, Vol. 29, Nº. 2, Feb, p. 38-47.
- SANDHU, R.; FERRAILOLO, D.; KUHN, D.R. (2000) "The NIST Model for Role-Based Access Control: Towards A Unified Standard", Proc. of 5th ACM Workshop on Role-Based Access Control, Berlin, Germany.
- SCHUMACHER, M. (2003) "Security Engineering With Patterns: Origins, Theoretical Models, and New Applications", Springer Berlin, Heidelber.
- SCHUMACHER, M. *et al.* (2005) "Security patterns. Integrating security and systems engineering", John Wiley & Sons.

- SILVA, E. (2005) “Extensão do modelo de restrições do RBAC para suportar obrigações do modelo ABC”, Dissertação (Mestrado em Informática Aplicada), Programa de Pós-Graduação em Informática Aplicada, Pontifícia Universidade Católica do Paraná, Curitiba, Paraná, Brasil.
- SOLINGEN, R. V., BERGHOUT, E. (1999) “The Goal / Question / Metric Method: A Practical Guide for Quality Improvement of Software Development”, McGraw Hill, ISBN 0- 0770-9553-7.
- STEEL, C.; NAGAPPAN, R.; LAI, R. (2005) “Core Security Patterns: Best Practices and Strategies for J2EE, Web Services, and Identity Management,” Prentice Hall.
- STRUTS. (2007) “Struts”, disponível em <http://struts.apache.org/>, acesso em dezembro de 2007.
- SUN XACML (2007) “Sun’s XACML Implementation Programmer’s Guide”, disponível em: <http://sunxacml.sourceforge.net/guide.html>, acesso em junho de 2007.
- SUN (2007) “Sun Microsystems”, disponível em <http://www.sun.com/>, acesso em julho de 2007.
- UML (2007) “Unified Modeling Language”, disponível em <http://www.uml.org/>, acesso em julho de 2007.
- VELOCITY (2007) “Velocity”, disponível em <http://velocity.apache.org/>, acesso em setembro de 2007.
- XMI (2007) “XML Metadata Interchange Specification version 2.1”, disponível em <http://www.omg.org/technology/documents/formal/xmi.htm>, acesso em dezembro de 2007.
- YODER, J.; BARCALOW, J. (1997) “Architectural Patterns for Enabling Application Security”, Pattern Languages of Programs, Monticello, IL.

APÊNDICES

Apêndice 1

Implementação da *tag* containsPermission da *tag lib* security

```
private boolean condition() throws JspException {
    HashMap servicos = ServicosSingleton.instance().getServicos();
    Subject subject = (Subject)this.pageContext.getSession().getAttribute(USER_SESSION);
    PrincipalImpl principal = ControleAcesso.getCallerPrincipal(subject);
    HashMap mapServicos;
    if(servicos == null){
        mapServicos = new HashMap();
    }else{
        mapServicos = (HashMap)servicos.get(principal.getNomeProjeto());
    }
    Servico servico = new Servico(name);
    try {
        return accessControl.ControleAcesso.verificaPermissao(subject, mapServicos, servico, true);
    }catch(Exception e){
        throw new JspException(e);
    }
}
```

Apêndice 2

Geração do método invoke da classe SecurityProxyImpl

```
public void invoke(Method arg0, Object[] arg1, Object arg2)
throws Exception {
    Subject subject = Subject.getSubject(AccessController.getContext());
    Servico servico = new Servico(toString(arg0, arg1));
    boolean possuiPermissao = false;
    if (subject == null) {
        throw new SubjectNaoIdentificadoException("subject.null");
    } else {
        PrincipalImpl principal = ControleAcesso.getCallerPrincipal(subject);
        if(principal.getNomeProjeto() == null){
            throw new NomeProjetoNaoEncontradoException("nome.projeto.null");
        }
        HashMap<String, HashMap<String, Collection<Perfil>>> servicos = ServicosSingleton.instance().getServicos();
        HashMap<String, Collection<accessControl.Perfil>> mapServicos;
        if(servicos == null){
            mapServicos = new HashMap<String, Collection<accessControl.Perfil>>();
        }else{
            mapServicos = servicos.get(principal.getNomeProjeto());
        }

        if(principal.isValidado()){
            possuiPermissao = true;
        }else{
            possuiPermissao = ControleAcesso.verificaPermissao(subject,
                mapServicos, servico, false);
        }
        if (!possuiPermissao) {
            throw new AcessoNegadoException("acesso.negado - " + servico.getNome());
        }
    }
}
```

Apêndice 3

Exemplo de geração de PBI

```
public class AlocacaoHandlerPBI
{
    AlocacaoHandlerBI alocacaohandlerBI;
    Subject subject;
    public AlocacaoHandlerPBI(Subject subject)
    {
        alocacaohandlerBI = ServiceLocator.instance( ).getAlocacaoHandlerBI( );
        this.subject = subject;
    }
    br.gov.cronus.gerencia.cd.Alocacao param11 = null;
    java.lang.Integer param12 = null;
    public void finalizarAlocacao (br.gov.cronus.gerencia.cd.Alocacao alocacao, java.lang.Integer progresso) throws Exception ,
    br.gov.cronus.gerencia.cs.AlocacaoException {
        param11 = alocacao;
    }
}
```

```

param12 = progresso;
try{
    Subject.doAsPrivileged(subject, new java.security.PrivilegedExceptionAction() {
        public Object run() throws Exception{
            alocacaoHandlerBI.finalizarAlocacao(param11,param12);
        }
    }, AccessController.getContext());
} catch (PrivilegedActionException e) {
    throw new Exception(e);
}
}

```

Apêndice 4

Adição e Remoção de restrições

```

public abstract void handleFinalizarAlocacao (br.gov.cronus.gerencia.cd.Alocacao alocacao, java.lang.Integer progresso) throws
br.gov.cronus.gerencia.cs.AlocacaoException;
br.gov.cronus.gerencia.cd.Alocacao param11 = null;
java.lang.Integer param12 = null;
boolean responsibleSession1= false;
public void finalizarAlocacao (br.gov.cronus.gerencia.cd.Alocacao alocacao, java.lang.Integer progresso) throws
br.gov.cronus.gerencia.cs.AlocacaoException{
    param11 = alocacao;
    param12 = progresso;
    java.util.HashMap<String,java.util.Collection<accessControl.Restricao>> listaRestricaoServico =
    accessControl.ControleAcessoUtil.getRestricoesServico("br.gov.cronus.gerencia.cs.AlocacaoHandlerBI.finalizarAlocacao(br.gov
.cronus.gerencia.cd.Alocacao, java.lang.Integer)");
    javax.security.auth.Subject subject = javax.security.auth.Subject.getSubject(java.security.AccessController.getContext());
    accessControl.PrincipalImpl principalImpl = accessControl.ControleAcessoImpl.getCallerPrincipal(subject);
    boolean gerente = false;
    if(!principalImpl.isValido()){
        gerente = true;
        principalImpl.setValido(true);
    }
    accessControl.ControleAcessoUtil.addRestricoes(principalImpl, listaRestricaoServico);
    try{
        javax.security.auth.Subject.doAsPrivileged(subject, new java.security.PrivilegedExceptionAction() {
            public Object run() throws Exception{
                handleFinalizarAlocacao (param11,param12);
                return null;
            }
        }, java.security.AccessController.getContext());
    } catch (java.security.PrivilegedActionException e) {
        br.gov.cronus.CronusLogger.error(e.getMessage(), java.security.PrivilegedActionException.class);
    } finally {
        accessControl.ControleAcessoUtil.removeRestricoes(principalImpl,listaRestricaoServico);
        if(gerente)
            principalImpl.setValido(false);
    }
}
}

```

Apêndice 5

Alteração do método list() da classe CriteriaImpl do Hibernate

```

public List list() throws HibernateException {
    javax.security.auth.Subject subject =
    javax.security.auth.Subject.getSubject(java.security.AccessController.getContext());
    if(subject != null){
        //recupera o usuario logado
        PrincipalImpl principalImpl = ControleAcesso.getCallerPrincipal(subject);
        //recupera as restrições onde a chave eh o perfil
        java.util.Map restricoes = principalImpl.getRestricoes();
        Criteria criteria = null;
        if(restricoes != null){
            for(Iterator it = restricoes.keySet().iterator(); it.hasNext();){
                String perfil = (String)it.next();
                //recupera as restrições do perfil
                Collection restricoesPerfil = (Collection)restricoes.get(perfil);
                criteria = this.add(Restrictions.disjunction());
                for(Iterator it2 = restricoesPerfil.iterator(); it2.hasNext();){
                    Restricao restricao = (Restricao) it2.next();
                    if(restricao.getMoldeRestricao().getEntidadeOrigem().equals(this.getEntityOrClassName())){
                        try {
                            String expressao = restricao.getMoldeRestricao().getExpressao();
                            String expressaoFinal = restricao.getMoldeRestricao().getExpressao();

```

```

String valor = restricao.getMoldeRestricao().getExpressao();

if(expressao.contains(".")){
    Random r = new Random();
    int nextInt = r.nextInt();
    nextInt = nextInt < 0? nextInt * -1:nextInt;
    int ultimoPonto = expressao.lastIndexOf(".");
    expressaoFinal = expressao.substring(0, ultimoPonto);
    valor = expressao.substring(ultimoPonto + 1,

criteria = criteria.createAlias(expressaoFinal, "alias" +

valor = "alias" + nextInt + "." + valor;

}

Criterion criterion = getCriterion(restricao, valor);

criteria.add(criterion);
} catch (Exception e) {
    throw new HibernateException(e.getMessage());
}

}

}

}

}

before();
try {
    return session.list( this );
}
finally {
    after();
}
}

private Criterion getCriterion(Restricao restricao, String valor) throws Exception {
    String operadorRestricao = restricao.getOperadorRestricao().getCodigo();
    String tipoAtributo = restricao.getMoldeRestricao().getTipoAtributoDestino();
    if(operadorRestricao.equals(OperadorRestricao.IN)){
        return Restrictions.in(valor, getValoresRestricaoIN(restricao.getValoresRestricao(), tipoAtributo));

    }else if(operadorRestricao.equals(OperadorRestricao.NOT_IN)){
        return Restrictions.not(Restrictions.in(valor, getValoresRestricaoIN(restricao.getValoresRestricao(),

tipoAtributo)));
    }else{
        Method method = Restrictions.class.getMethod(operadorRestricao.toLowerCase(), new
Class[] {String.class, Object.class});
        Criterion criterion = (Criterion)method.invoke(null, new Object[] { valor,
getValoresRestricao(restricao.getValoresRestricao(), tipoAtributo) });
        return criterion;
    }
}

private Collection getValoresRestricaoIN(Collection valoresRestricao, String tipoAtributo) throws Exception {
    try {
        Collection valores = new ArrayList();
        for(java.util.Iterator it = valoresRestricao.iterator();it.hasNext()){
            ValorRestricao valor = (ValorRestricao) it.next();
            valores.add(getObject(valor.getValor(), tipoAtributo));
        }
        return valores;
    } catch (ClassNotFoundException e) {
        throw new Exception(e.getMessage());
    }
}

private Object getValoresRestricao(Collection valoresRestricao, String tipoAtributo) throws Exception{
    try {
        ValorRestricao valor = (ValorRestricao)valoresRestricao.iterator().next();
        return getObject(valor.getValor(), tipoAtributo);
    } catch (ClassNotFoundException e) {
        throw new Exception(e.getMessage());
    }
}

private Object getObject(String valor, String tipoAtributo) throws Exception{

```

```

    try {
        Class classe = Class.forName(tipoAtributo);
        Constructor construtor = classe.getConstructor(new Class[]{String.class});
        Object objeto = construtor.newInstance(new Object[]{valor});
        return objeto;
    } catch (ClassNotFoundException e) {
        throw new Exception(e.getMessage());
    }
}

```

Apêndice 6

Implementação da classe ControleAcessoImp

```

public class ControleAcessoImp extends accessControl.ControleAcesso {

    static final String SUPER_USUARIO = "1";
    static final int PROXIMA_TROCA_SENHA_DIAS = 180;

    public boolean precisaTrocarSenha(accessControl.Operador operador) throws ControleAcessoException {
        java.util.Date data = new java.util.Date();
        UsuarioImpl usuario = (UsuarioImpl)operador;
        boolean precisaTrocarSenha = false;
        if(usuario.getValidadeSenha() != null){
            if(usuario.getValidadeSenha().before(data)){
                precisaTrocarSenha = true;
            }
        }
        return precisaTrocarSenha;
    }

    public accessControl.Operador getOperador(String usuario) throws ControleAcessoException {
        try {
            UsuarioDAOImpl usuarioDAOImpl = new UsuarioDAOImpl();
            return (UsuarioAbstract)usuarioDAOImpl.findByLogin(usuario);
        } catch (DAOException e) {
            throw new AccessControlException(e.getMessage());
        }
    }

    private void inserePerfilServico(){
        try {
            ServicoDAOImpl servicoDAOImpl = new ServicoDAOImpl();
            PerfilDAOImpl perfilDAOImpl = new PerfilDAOImpl();
            PerfilImpl perfil = (PerfilImpl)(perfilDAOImpl.select(new Long(2)).get(0));

            Collection servicos = servicoDAOImpl.list();
            for(Iterator it = servicos.iterator(); it.hasNext();){

                ServicoImpl servico = (ServicoImpl)it.next();
                if(!servico.getServicos().isEmpty() || servico.getAgrupamento().isEmpty()){
                    PerfilServicoImpl perfilServicoImpl = new PerfilServicoImpl();
                    perfilServicoImpl.setPerfil(perfil);
                    perfilServicoImpl.setServico(servico);

                    PerfilServicoDAOImpl perfilServicoDAOImpl = new PerfilServicoDAOImpl();
                    perfilServicoDAOImpl.insert(perfilServicoImpl);
                }
            }
        } catch (DAOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    public java.util.HashMap<String, java.util.Collection<accessControl.Perfil>> listaServicos() throws ControleAcessoException {
        ArrayList al;
        Perfil pf;
        HashMap hm = new HashMap();
        HashMap hmPerfis = new HashMap();

        hmPerfis.put(SUPER_USUARIO, new Perfil(SUPER_USUARIO));

        ServicoDAOImpl servDAO = new ServicoDAOImpl();
    }

```

```

        try {

            Collection col = servDAO.list();

            for (Iterator it = col.iterator(); it.hasNext(); ) {
                ServicoImpl srv = (ServicoImpl) it.next();

                al = new ArrayList<accessControl.Perfil>();
                al.add(hmPerfis.get(SUPER_USUARIO));

                for (Iterator itps = srv.getPerfilServicos().iterator(); itps.hasNext(); ) {
                    PerfilServicoImpl spo = (PerfilServicoImpl) itps.next();
                    pf = (Perfil) hmPerfis.get(spo.getPerfil().getId().toString());
                    if (pf == null) {
                        pf = new Perfil(spo.getPerfil().getId().toString());
                        hmPerfis.put(spo.getPerfil().getId().toString(), pf);
                    }
                    al.add(pf);
                }
                adicionaPerfisListaServicos(srv, hm, al);
            }
        } catch (DAOException e) {
            throw new ControleAcessoException(e.getMessage());
        }

        return hm;
    }

    private void adicionaPerfisListaServicos(ServicoImpl srv, java.util.HashMap<String, java.util.Collection<accessControl.Perfil>> hm,
        ArrayList al) {
        ArrayList alAtual;
        ArrayList alNovo = new ArrayList<accessControl.Perfil>();

        alNovo.addAll(al);
        alAtual = (ArrayList<accessControl.Perfil>) hm.get(srv.getCodigo());

        if (alAtual != null) {
            alAtual.removeAll(alNovo);
            alNovo.addAll(alAtual);
        }

        hm.put(srv.getCodigo(), alNovo);

        if (!srv.getAgrupado().isEmpty()) {
            for (Iterator it = srv.getAgrupado().iterator(); it.hasNext(); ) {
                ServicoImpl srvAgrupado = (ServicoImpl) it.next();
                adicionaPerfisListaServicos(srvAgrupado, hm, al);
            }
        }
    }

    public boolean trocaSenha(accessControl.Operador operador, String senhaAtual, String novaSenha) throws
        ControleAcessoException {
        try {
            UsuarioImpl usuario = (UsuarioImpl) operador;
            UsuarioDAOImpl usuarioDAOImpl = new UsuarioDAOImpl();

            if (!senhaAtual.equals(usuario.getSenha())) return false;

            usuario.setSenha(novaSenha);
            GregorianCalendar data = new GregorianCalendar();
            data.add(GregorianCalendar.DATE, PROXIMA_TROCA_SENHA_DIAS);
            usuario.setValidadeSenha(data.getTime());
            usuarioDAOImpl.update(usuario);

            return true;
        } catch (DAOException e) {
            throw new AccessControlException(e.getMessage());
        }
    }
}

```

```

    public java.util.HashMap<String,java.util.Collection<accessControl.Restricao>> getRestricoes(accessControl.Operador operador,
    accessControl.Servico servico) throws ControleAcessoException {

```

```

        HashMap hmRestricoes = new HashMap();

        ServicoImpl srvImpl;
        ServicoDAOImpl srvDAO = new ServicoDAOImpl();

        HashMap hm = ServicosSingleton.instance().getServicos();

        ArrayList<accessControl.Perfil> perfisAcessoServico = (ArrayList) hm.get(servico.getNome());
        ArrayList<accessControl.Perfil> perfisUsuarioLogado = (ArrayList) ControleAcessoUtil.getPerfis();
        ArrayList<accessControl.Perfil> perfisUsuarioAcessoServico = new ArrayList();

        if (perfisAcessoServico == null) return hmRestricoes;

        perfisUsuarioAcessoServico.addAll(perfisUsuarioLogado);
        perfisUsuarioAcessoServico.retainAll(perfisAcessoServico);

        try {
            srvImpl = (ServicoImpl)srvDAO.findByNome(servico.getNome());
        }
        catch (DAOException e){
            throw new ControleAcessoException(e.getMessage());
        }

        Set perfisServicos = new HashSet();
        if(!srvImpl.getServicos().isEmpty()){
            for(Iterator itServicos = srvImpl.getServicos().iterator(); itServicos.hasNext();){
                ServicoImpl servicoAgrupa = (ServicoImpl)itServicos.next();
                if(!servicoAgrupa.getPerfilServicos().isEmpty()){
                    perfisServicos.addAll(servicoAgrupa.getPerfilServicos());
                }
            }
        }
        for (Iterator itspo = perfisServicos.iterator(); itspo.hasNext();) {
            PerfilServicoImpl ps = (PerfilServicoImpl) itspo.next();
            ArrayList colRestricao = new ArrayList();
            for (Iterator itpuas = perfisUsuarioAcessoServico.iterator(); itpuas.hasNext();) {
                Perfil pf = (Perfil) itpuas.next();
                if (pf.getNome().equals(ps.getPerfil().getId().toString()) ) {
                    Collection restricoes = ps.getRestricoes();
                    for (Iterator itrest = restricoes.iterator(); itrest.hasNext();) {
                        RestricaoImpl restImpl = (RestricaoImpl) itrest.next();

                        Restricao restricao = criaRestricaoControleAcesso(restImpl);
                        colRestricao.add(restricao);
                    }
                }
                if (!colRestricao.isEmpty()) {
                    hmRestricoes.put(pf.getNome(),colRestricao);
                }
            }
        }

        return hmRestricoes;
    }

    public Restricao criaRestricaoControleAcesso(RestricaoImpl restImpl) {
        MoldeRestricao molde = new MoldeRestricao();
        molde.setEntidadeOrigem(restImpl.getMoldeRestricao().getEntidadeOrigem());
        molde.setExpressao(restImpl.getMoldeRestricao().getExpressao());
        molde.setTipoAtributoDestino(restImpl.getMoldeRestricao().getTipoAtributoDestino());

        OperadorRestricao opRestricao = new OperadorRestricao();
        opRestricao.setCodigo(restImpl.getOperadorRestricao().getCodigo());

        ArrayList colValorRestricao = new ArrayList();

        for (Iterator itvalrest = restImpl.getValorRestricoes().iterator(); itvalrest.hasNext();) {
            ValorRestricaoImpl valorRestricaoImpl = (ValorRestricaoImpl)itvalrest.next();
            ValorRestricao valRestricao;
            if(valorRestricaoImpl instanceof ValorRestricaoDinamicoImpl){

```

```

        valRestricao = new ValorRestricaoDinamico();
    }else{
        valRestricao = new ValorRestricaoEstatico();
    }

    valRestricao.setValor(valorRestricaoImpl.getValor());
    colValorRestricao.add(valRestricao);
}

Restricao restricao = new Restricao();
restricao.setMoldeRestricao(molde);
restricao.setOperadorRestricao(opRestricao);
restricao.setValoresRestricao(colValorRestricao);

return restricao;
}

public static class RestricaoDinamicaImpl {

    public java.lang.Long getUsuario () {
        PrincipalImpl principal = ControleAcesso.getPrincipal();
        return ((UsuarioImpl)principal.getOperador()).getId();
    }

    public java.lang.Long getProjeto () {
        PrincipalImpl principal = ControleAcesso.getPrincipal();
        HashMap parametros = principal.getParametrosRestricoesDinamicas();
        if(parametros != null && parametros.get("idProjeto") != null){
            Long idProjeto = (Long)parametros.get("idProjeto");
            return idProjeto;
        }
        return null;
    }

    public java.util.Date getHora () {
        return null;
    }

}

public Collection<accessControl.Perfil> buscaPerfisServico(Servico servico) throws ControleAcessoException {
    try{
        boolean possuiPermissao = false;
        ServicoDAOImpl servicoDAOImpl = new ServicoDAOImpl();
        ServicoImpl servicoImpl = (ServicoImpl)servicoDAOImpl.findByNome(servico.getNome());
        Collection<Perfil> perfis = new HashSet<Perfil>();
        if(servicoImpl != null)
            perfis = recuperaPerfis(servicoImpl, perfis);
        return perfis;
    }catch (DAOException e){
        throw new ControleAcessoException(e.getMessage());
    }
}

private static Collection recuperaPerfis(ServicoImpl servicoImpl, Collection<Perfil> perfis){
    Collection perfilServico = servicoImpl.getPerfilServicos();
    if(perfilServico == null || perfilServico.isEmpty()){
        Collection servicos = servicoImpl.getServicos();
        for(Iterator itServico = servicos.iterator(); itServico.hasNext();){
            perfis.addAll(recuperaPerfis((ServicoImpl)itServico.next(), perfis));
        }
    }else{
        for(Iterator it = perfilServico.iterator(); it.hasNext();){
            PerfilServicoImpl perfilServicoImpl = (PerfilServicoImpl)it.next();
            PerfilImpl perfilImpl = (PerfilImpl)perfilServicoImpl.getPerfil();
            Perfil perfil = new Perfil(perfilImpl.getId().toString());
            perfis.add(perfil);
        }
    }
    return perfis;
}

```

Apêndice 7

Implementação da classe ActionServletImpl sem demanda

```

public final class ActionServletImpl extends ActionServlet
{
    private static Log s_log = LogFactory.getLog(ActionServletImpl.class);
    public void init( javax.servlet.ServletConfig config ) throws javax.servlet.ServletException
    {
        super.init( config );
        try
        {
            ControleAcessoImpl controleAcesso = new ControleAcessoImpl();
            HashMap<String, ControleAcesso> controles = ServicosSingleton.instance().getControles();
            if(controles == null){
                controles = new HashMap<String, ControleAcesso>();
            }
            controles.put("cronus", controleAcesso);
            ServicosSingleton.instance().setControles(controles);
            HashMap listaServicosProjeto = controleAcesso.listaServicos();
            HashMap<String, HashMap<String, java.util.Collection<Perfil>>> listaServicos = ServicosSingleton.instance().getServicos();
            if(listaServicos == null){
                listaServicos = new HashMap<String, HashMap<String, java.util.Collection<Perfil>>>());
            }
            listaServicos.put("cronus", listaServicosProjeto);
            ServicosSingleton.instance().setServicos(listaServicos);

            HashMap<String, Boolean> demanda = ServicosSingleton.instance().getDemanda();
            if(demanda == null){
                demanda = new HashMap<String, Boolean>();
            }
            demanda.put("cronus", false);
            ServicosSingleton.instance().setDemanda(demanda);
        }
        catch( Exception e )
        {
            s_log.error("Nao foi possivel listar os servicos", new ServletException());
        }
    }
}

```