# ArchiMDAs: Um Arcabouço de Segurança Baseado em Transformações MDA

**Paulo F. Pires[1], Flávia C. Delicato[1], Milene Fiorio[2], Carlo O. Emmanoel[2]**

[1]Centro de Ciências Exatas, Departamento de Informática e Matemática Aplicada
Universidade Federal do Rio Grande do Norte (UFRN)
Campus Universitário – Lagoa Nova – 59072-970 – Natal – RN – Brasil

[2]Instituto de Matemática – Núcleo de Computação Eletrônica
Universidade Federal do Rio de Janeiro (UFRJ)
Caixa Postal 2324 – 20.001-970 – Rio de Janeiro – RJ – Brasil

{paulo.pires,flavia.delicato}@dimap.ufrn.br,
{milenefc,carlo}@posgrad.nce.ufrj.br

***Resumo.*** *A segurança é um requisito essencial de grande parte dos sistemas de informação atuais. Contudo, frequentemente se tem notícia da ocorrência de falhas de segurança nesse tipo de sistemas. Neste artigo, apresentamos um arcabouço de segurança baseado em MDA (Model Driven Architecture) denominado ArchiMDAs que, em um primeiro momento, simplifica e flexibiliza a modelagem dos requisitos de segurança. A seguir, as funcionalidades de segurança modeladas são aplicadas implicitamente no sistema através de transformação entre modelos e codificação automática, garantindo que o código relativo a segurança não será suscetível a erros ou alterações indesejadas.*

***Abstract.*** *Security is an essential requirement in most of today's information systems. However, the occurrence of security flaws in such systems is not uncommon. In this article, we present ArchiMDAs, a security framework based on Model Driven Architecture (MDA) that simplifies and flexibly the modeling of security requirements, which are then implicitly applied to the whole system through model transformations and automatic code generation. The use of ArchiMDAs provides a high assurance that security related code is not susceptible to coding error.*

## 1 Introdução

Atualmente, a questão de segurança em sistemas de informação tem recebido grande atenção por parte tanto da academia quanto da indústria. Apesar da grande variedade de arcabouços de segurança, como JAAS [JAAS 2007], MISF [MISF 2007], Acegi [ACEGI 2007], e de tecnologias associadas disponíveis atualmente, constantemente há notícias sobre vulnerabilidades em sistemas de informações [FINK, KOCH & PAULS 2004], as quais são exploradas por diversos tipos de ataques. Isto se deve principalmente ao fato da segurança ser raramente considerada nos estágios iniciais do desenvolvimento de software e ser relegada a segundo plano ao longo do mesmo.

Mesmo nos casos onde a segurança é considerada como parte integrante do processo de desenvolvimento, a incorporação dos seus requisitos é problemática devido a três fatores [BASIN, DOSER 2006; FERNANDEZ 2000]: (*i*) os modelos de segurança (que representam os requisitos de segurança) e os modelos de projeto de sistemas (que

representam requisitos de negócio) não são integrados, sendo tipicamente expressados de modos diferentes. Os requisitos de segurança são em geral modelados usando texto estruturado enquanto os requisitos de negócio são modelados através de idiomas como UML [UML, 2007]. Apesar desse fato resultar na necessidade de realizar a integração entre esses modelos, os processos e ferramentas de desenvolvimento de software modernos não possuem suporte adequado para realizar tal integração; (*ii*) a implementação dos requisitos de segurança é, freqüentemente, realizada de forma *post-hoc* nas fases finais do desenvolvimento do sistema (fases de integração e testes). Essa lacuna entre o levantamento dos requisitos de segurança e sua efetiva implementação resulta, por um lado, em sistemas com potenciais falhas de segurança pelo não atendimento de um requisito levantado e, por outro, em dificuldade de manutenção e evolução do sistema, por não haver documentação adequada, em todas as fases do desenvolvimento, das funcionalidades de segurança [FERNANDEZ 2000]; (*iii*) o terceiro fator refere-se ao fato de que a maior parte dos arcabouços de segurança (tais como o JAAS [JAAS 2007]) são complexos de serem integrados com as aplicações e necessitam de customização/extensão de forma a atenderem os requisitos de segurança específicos de uma aplicação [BLAKLEY, HEALTH 2004]. Assim, para desenvolver uma solução adequada de segurança é necessária uma equipe que integre especialistas tanto em políticas de segurança quanto nos arcabouços de software de segurança disponíveis. Essa necessidade onera o desenvolvimento de sistemas fazendo, muitas vezes, com que a segurança seja relegada a segundo plano.

Visando minimizar os atuais problemas inerentes ao desenvolvimento de sistemas de informação seguros, nesse artigo é apresentado o arcabouço de segurança ArchiMDAs, bem como o processo de desenvolvimento a ser adotado pelas aplicações que o utilizem. O processo do ArchiMDAs baseia-se nos conceitos da Arquitetura Orientada a Modelos (MDA, do inglês *Model Driven Architecture*) [OMG 2003], a qual consiste em uma abordagem de desenvolvimento de software centrada na modelagem e no refinamento sucessivo dos modelos produzidos. A MDA propõe três diferentes níveis de abstrações para a modelagem de sistemas: Modelo Independente de Computação (*Computational Independent Model-CIM*), Modelo Independente de Plataforma (*Platform Independent Model-PIM*) e Modelo Específico de Plataforma (*Platform Specific Model-PSM*). Os modelos são mapeados de uma abstração para outra através de processos de transformações sucessivas, durante as quais são incluídos novos elementos no modelo, reduzindo sua abstração até o nível de dependência da plataforma computacional onde o sistema será implementado. Assim, uma possível mudança de plataforma implica a alteração da especificação das transformações, enquanto que a especificação da solução (independente de plataforma) se mantém intacta.

No ArchiMDAs as funcionalidades de segurança são integradas ao sistema através de transformação entre modelos e geração automática de código. Essa abordagem permite o desacoplamento entre a implementação do negócio propriamente dito e a solução de segurança e minimiza a interferência entre o código de negócio e o código de segurança. Desta forma, os 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 das regras de negócio da aplicação. Ainda, o ArchiMDAs favorece a gerência da complexidade inerente a utilização dos arcabouços de segurança, através da criação de uma camada de abstração baseada na modelagem de alto nível dos requisitos de segurança (modelo PIM de segurança) que isola os detalhes de implementação necessários a customização desses arcabouços. A customização em si é encapsulada no processo de transformação MDA. Assim, o ArchiMDAs simplifica o desenvolvimento de sistemas de software seguros ao mesmo

tempo em que garante a qualidade do sistema no que diz respeito aos requisitos de segurança. O encapsulamento da customização e integração de arcabouços de segurança eleva o nível de abstração referente ao reuso da solução de segurança, minimizando, desta forma, a necessidade de especialistas nesses arcabouços de segurança.

Embora a solução provida pelo ArchiMDAs seja capaz de atender a requisitos das diferentes dimensões de segurança de sistemas [BISHOP 2003], a implementação atual do arcabouço trata especificamente do controle de acesso. O restante do artigo está organizado como segue. Na Seção 2 são apresentados os requisitos de segurança que nortearam a construção do ArchiMDAs e os padrões de projeto usados. Na Seção 3 o ArchiMDAs é apresentado, descrevendo-se sua arquitetura e o processo de desenvolvimento a ser adotado por aplicações que o utilizem. Na Seção 4 são descritos sua implementação e um estudo de caso. Na Seção 5 são discutidos trabalhos relacionados e a Seção 6 conclui o artigo.

## 2 Requisitos de Segurança e Padrões de Projeto do ArchiMDAs

A construção do ArchiMDAs foi baseada em um conjunto de requisitos de segurança levantados na literatura e na experiência dos autores no desenvolvimento de sistemas comerciais [FIORIO 2007; YODER, BARCALOW 1997; SCHUMACHER *et al.* 2005]. De posse desses requisitos foram pesquisados padrões de projetos de segurança que os atendessem. Padrões de projeto [GAMMA *et al.* 1995] 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 pioneiros na utilização dessa abordagem como uma tentativa de criar arcabouços de segurança bem projetados. O uso de padrões nesse contexto justifica-se pelo fato de grande parte dos problemas de segurança serem bem conhecidos e possuírem soluções bem estabelecidas. A Tabela 1 descreve os requisitos de segurança considerados no projeto do ArchiMDAs, enquanto que a Tabela 2 descreve os padrões utilizados, suas intenções e os requisitos que eles atendem.

**Tabela 1. Requisitos de Segurança**

| |
|---|
| 1. Deve haver um ponto único de acesso ao sistema (single-sign-on); |
| 2. Cada usuário deve possuir um nome de usuário 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 encaminhando 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. |
| 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. |

**Tabela 2 – Padrões de Projeto de Segurança usados no ArchiMDAs.**

| Padrão de Segurança | Intenção | Requisitos |
|---|---|---|
| I) Ponto de Acesso único | Prover um módulo de segurança e uma forma de autenticação | 1, 2 |
| II) Ponto de verificação | Organizar pontos de verificação de segurança e suas repercussões | 2, 3, 9 |
| III) Papéis | Organizar usuários com privilégios de segurança similares | 4, 5, 6 |
| IV) Sessão | Localizar informação global em um ambiente multi-usuário | 7, 8 |
| V) Visão completa com erros | Prover uma visão completa aos usuários, exibindo exceções quando necessário | 9 |
| VI) Visão limitada | Permitir que os usuários visualizem somente aquilo a que eles têm acesso | 9 |
| VII) 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 |
| VIII) Política | Isolar a lógica da política de segurança em um componente discreto do sistema | 3, 10 |
| IX) Descritor do sujeito | O controle de acesso a diferentes recursos depende dos atributos desse sujeito (entidade humana ou programa). O descritor do sujeito provê acesso aos atributos de um sujeito e facilita gerência e proteção a esses atributos | 7 |
| X) RBAC Estendido | RBAC com contexto, restrições a dados e níveis de segurança | 4, 5, 6, 11, 12, 13 |

Os padrões de projeto I a IX são extraídos da literatura [YODER, BARCALOW 1997; BLAKLEY, HEALTH 2004]. Já o padrão RBAC estendido (X) é uma extensão do modelo RBAC [SANDHU, FERRAIOLO, KUHN 2000] desenvolvida no escopo do arcabouço ArchiMDAs [FIORIO 2007]. O modelo RBAC determina que cada usuário deve possuir o seu papel dentro do sistema, podendo acessar somente as informações que estão acessíveis para esse papel. Um papel representa um conjunto de responsabilidades que um usuário pode assumir, sendo que vários usuários podem possuir o mesmo papel. Contudo, os modelos RBAC existentes não são suficientes para alguns requisitos de segurança levantados, tais como: 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). Portanto, o modelo RBAC estendido utilizado no ArchiMDAs incopora extensões para representar contexto e restrições a dados.

# 3 Descrição do ArchiMDAs

## 3.1 Metamodelo PIM de Segurança

O arcabouço de segurança ArchiMDAs foi desenvolvido para atender os requisitos de segurança levantados na Seção 2. Na abordagem proposta, esses requisitos de segurança são utilizados para construir um modelo PIM de segurança que serve de base para as transformações que darão origem aos artefatos concretos de segurança de uma aplicação. Esse modelo é uma instância do metamodelo apresentado na Figura 1, o qual é uma representação UML da parte estrutural dos padrões de projeto descritos na Seção 2. Os elementos desse metamodelo são descritos a seguir.

***Usuario*** representa uma entidade do sistema, como uma pessoa ou um processo. Cada usuário pode possuir um ou mais perfis (***Perfil)***, os quais representam 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, sendo essa herança identificada através da auto-associação (***Hierarquia de Perfis***). Cada usuário pode pertencer a contextos, representados pela entidade ***Contexto***, onde contextos representam as divisões existentes dentro da organização. A entidade ***PerfilContexto*** representa o usuário em um determinado contexto. Um ***PerfilContexto*** está associado a ***PerfilContextoServico*** que por

sua vez está associado a ***Servico***. Um ***PerfilContextoServico*** representa o serviço para o qual um determinado perfil, em um determinado contexto, possui permissão de acesso. ***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. ***ServicoTela*** representa os serviços relacionadas a camada de vista da aplicação. 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***. ***ServicoNegocio*** representa os serviços relacionados a camada de controle que representam o fluxo de execução da aplicação. Cada ***ServicoNegócio*** está associado a um evento operacional da aplicação, que é representado pela entidade ***Negócio***.

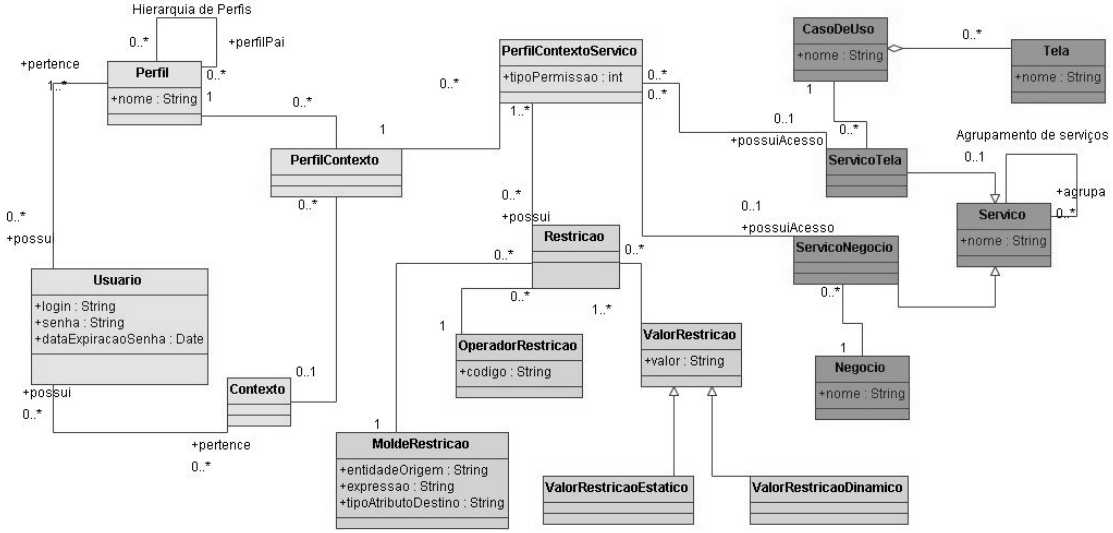


**Figura 1 – Metamodelo PIM de segurança do ArchiMDAs.**

As restrições a dados são representadas pela entidade ***Restricao*** e estão associadas a entidade ***PerfilContextoServico***. Ou seja, um serviço para o qual um determinado perfil em um determinado contexto possui permissão de acesso pode possuir restrições adicionais com relação aos seus dados. ***Restricao***, por sua vez, possui um molde restrição que é representado pela entidade ***MoldeRestricao***, o qual indica o dado (***entidadeOrigem***) e o atributo (***expressao***) deste dado onde a restrição é aplicada. Além de molde restrição, uma restrição possui um operador restrição, representado pela entidade ***OperadorRestricao***, e que 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, juntamente com o operador restrição e um molde restrição, formam uma restrição completa. Uma restrição pode possuir mais de um ***ValorRestricao*** por exemplo, no caso do operador restrição ser 'Em' ou 'Não Em'. Ainda, um valor restrição pode ser estático (***ValorRestricaoEstatico***) ou dinâmico (***ValorRestricaoDinamico***). Quando estático, o valor utilizado na restrição é o próprio valor; 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, considere a hipótese de que um conjunto de usuários pertencentes a um dado perfil só pode alterar dados de itens cuja procedência indique o *Brasil*. Considere que o perfil em questão é *Desenvolvedor*, que não será utilizado contexto, que o dado ao qual será aplicada uma restrição é o atributo *id* da entidade *País* 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 ***entidadeOrigem*** *País* e ***expressão*** *id*, um ***OperadorRestricao*** do tipo '*igual*' e um ***ValorRestricao*** 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 incluída na consulta de forma que o resultado só conterá itens cuja procedência indique o *Brasil*.

### 3.2 Arquitetura e Processo de Desenvolvimento

As funcionalidades do ArchiMDAs estão organizadas em dois módulos: (i) **de regras de segurança** e (ii) **administrativo de segurança**. Ambos os módulos utilizam o metamodelo de segurança previamente descrito.

O **módulo de regras de segurança** é responsável pela gerência das regras de autenticação e autorização dos artefatos da aplicação. Esse módulo é o núcleo do ArchiMDAs, pois contém todas as regras de transformação necessárias para transformar o modelo PIM de segurança de uma aplicação em um modelo PSM que integrará os requisitos de segurança e de negócio. Dessa forma, os regras de autenticação e autorização são incorporadas ao sistema independentemente das regras de negócio, através de transformação entre modelos. Assim, o processo de modelagem das regras de negócio não sofre alterações, devendo o analista de negócio focar somente no negócio, sem se preocupar com a segurança.

O **módulo administrativo de segurança** é responsável pela geração de uma aplicação administrativa de segurança completa, considerando a política de controle de acesso da organização. Visando facilitar o desenvolvimento dos sistemas, no ArchiMDAs uma aplicação administrativa de segurança *default* é gerada considerando uma política de segurança padrão, baseada nos requisitos apresentados na Seção 2. Esta aplicação, também gerada automaticamente através de transformações MDA, contém os casos de uso responsáveis pelo cadastro dos artefatos de segurança, tais como perfis, usuários, permissões e restrições. Só é necessário criar uma aplicação de administração de segurança própria caso a aplicação *default* fornecida não esteja de acordo com a política de segurança da organização.

O processo de desenvolvimento de sistemas utilizando o arcabouço de segurança ArchiMDAs pode ser visto no diagrama de atividades da Figura 2, onde cada raia inclui as atividades executadas por um determinado ator considerado na abordagem proposta. A seguir serão descritas as responsabilidades e atividades desses atores.

**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 orientado a modelos, ou seja, elas não são específicas da proposta apresentada neste trabalho. Todos os artefatos desenvolvidos nesta atividade são reutilizáveis, ou seja, o trabalho realizado pelo Arquiteto de Negócio pode ser reaproveitado no desenvolvimento de vários sistemas para um mesmo domínio de aplicação.

**Arquiteto de Segurança**. De posse dos requisitos de segurança da aplicação levantados pelo Arquiteto de Negócio, o Arquiteto de Segurança verifica se os módulos padrão implementados pelo ArchiMDAs atendem a esses requisitos. Caso atendam, esses módulos serão utilizados sem alterações. Caso contrário, o Arquiteto de Segurança deve desenvolver, a partir desses requisitos, a arquitetura MDA de segurança, criando extensões dos perfis UML, regras de transformação e metamodelo específicos de segurança. A partir desses

artefatos, são desenvolvidos os novos módulo de regras de segurança e módulo administrativo de segurança. Os artefatos dos módulos pré-existentes do ArchiMDAs (perfis UML, regras de transformação e metamodelo de segurança) podem e devem ser reaproveitados na construção de novos módulos. Como visto, a implementação das funcionalidades de segurança é gerada automaticamente através de transformações entre modelos. Essa geração automática provê liberdade ao Arquiteto de Segurança à medida em que ele pode alterar o que é gerado, a qualquer momento, sem influenciar ou prejudicar o desenvolvimento das regras de negócio do sistema. Em caso de necessidade de troca de plataforma de segurança (p.e. JAAS para Acegi), o arquiteto apenas precisa implementar a mesma solução de segurança para a nova plataforma, ou seja, mapear as regras de transformação já definidas para a nova plataforma escolhida. Além disso, o fato do código relativo a segurança ser gerado sem intervenção humana garante a sua qualidade e, conseqüentemente, a do sistema produzido. As atividades realizadas pelo Arquiteto de Segurança não fazem parte de um processo tradicional de desenvolvimento, elas são específicas da proposta apresentada neste trabalho.

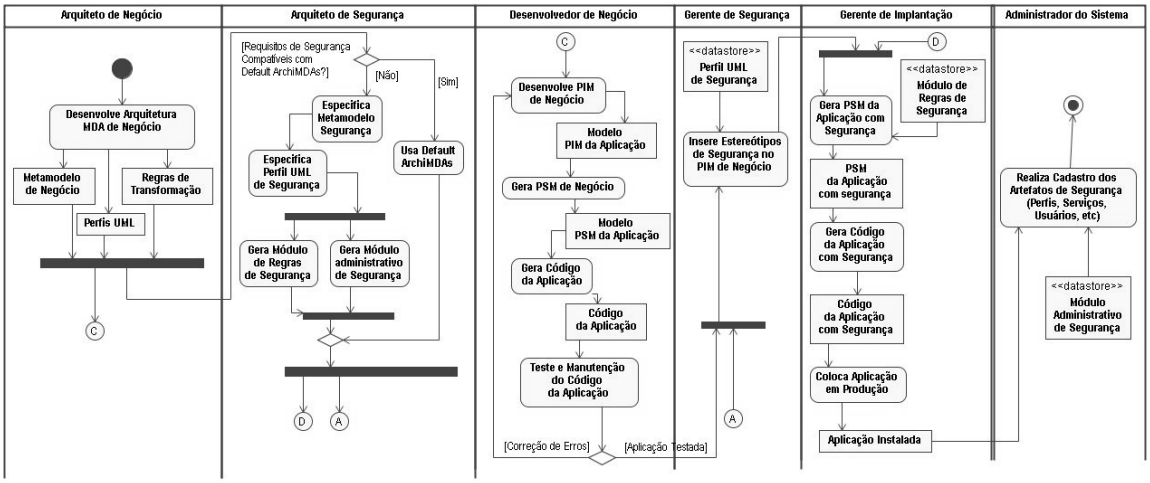


**Figura 2 – Diagrama de Atividades do Processo do ArchiMDAs.**

**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 PIM para PSM e posteriomente de PSM para código. Os desenvolvedores não precisam tomar nenhuma atitude em relação à segurança, pois esta será aplicada de forma transparente para o modelo de negócio. As atividades desses desenvolvedores fazem parte de um processo normal de desenvolvimento, ou seja, não são específicas da presente proposta.

**Gerente de Segurança**. O Gerente de Segurança é responsável pela incorporação das informações referentes a política de segurança no modelo PIM de negócio. Para tanto, ele utiliza o perfil UML do ArchiMDAs para decorar com estereótipos as entidades desse modelo que representam as entidades do RBAC com contextos, ou seja, usuário, papel, autorização e contexto.

**Gerente de Implantação**. O Gerente de Implantação é responsável pela ativação da segurança. Ou seja, é ele que, a partir do PIM modelado pelo Gerente de Segurança, irá aplicar as regras de transformação definidas no módulo de regras de segurança do ArchiMDAs, gerar o PSM da aplicação já acrescido dos artefatos de segurança no nível de modelo e em seguida, usando esse PSM como entrada, gerar o código contendo todos os artefatos de segurança de acordo com a plataforma alvo de segurança (p.e.. JAAS). Esse

código, então, está pronto para ser colocado em produção. O ArchiMDAs permite que a segurança seja excluída ou incluída (*switch on-off*) na geração do sistema de forma configurável, sem interferir no trabalho dos Desenvolvedores de Negócio, dessa forma facilitando o desenvolvimento e a configuração do sistema.

**Administrador do Sistema**. Após a aplicação estar instalada, o Administrador do Sistema realiza o cadastro das entidades 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 conseguirá 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. Outra característica importante é que o modelo de negócio não necessita ser "poluído" com informações de segurança, tais como aquelas relativas a perfis de usuários.

# 4 Implementação do ArchiMDAs e Estudo de Caso

O ArchiMDAs foi implementado como uma extensão da ferramenta AndroMDA [ANDROMDA 2007]. A escolha dessa ferramenta foi feita levando em conta os seguintes critérios: número de usuários, maturidade, suporte eficaz a usuários e desenvolvedores, licença tipo código aberto e extensibilidade. Uma comparação entre algumas ferramentas MDA existentes pode ser encontrada em [MAIA 2006].

Como na versão atual do ArchiMDAs utilizou-se a versão 3.1 da ferramenta AndroMDA, os modelos podem ser feitos em praticamente qualquer ferramenta de modelagem que suporte UML 1.4 [UML 2007] e XMI [XMI 2007]. Para serem processados pelo AndroMDA, os modelos UML devem ser exportados no formato XMI. O AndroMDA utiliza os documentos XMI e os *templates* de transformação para produzir os artefatos de software. As transformações do AndroMDA são baseadas em cartuchos. Um cartucho consiste de um conjunto de *templates* escritos em Velocity [VELOCITY 2007] os quais são responsáveis por transformar elementos de um modelo PIM em elementos de uma plataforma específica. Hibernate [HIBERNATE 2007] e Enterprise Java Beans (EJB) [EJB 2007] são alguns exemplos de cartuchos disponíveis no AndroMDA. O processo de transformação do AndroMDA é aplicado a partir do uso de estereótipos UML e da configuração de cada projeto. Por exemplo, classes estereotipadas como *Entity* podem ser transformadas em classes e descritores de elementos do Hibernate, caso essa seja a plataforma configurada no projeto. Classes estereotipadas como *Service* podem ser transformadas em Session Beans, por exemplo, caso a plataforma escolhida seja EJB. Para implementar o ArchiMDAs, foram necessárias algumas adaptações no conjunto de transformações originais do AndroMDA, descritas a seguir.

## 4.1 Extensão do AndroMDA

Com o intuito de garantir que o requisito de segurança por níveis fosse atendido (Requisito 13 da Tabela 1), 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 arquitetural em camadas. O **Modelo** representa os objetos de domínio ou dados de uma aplicação. A **Visão** é a representação deste modelo em formato de apresentação para o cliente, com interface apropriada para a entrada e saída de informações. O **Controlador**, por sua vez, implementa a interatividade, através do processamento das ações tomadas pelo cliente e a atualização do **Modelo**.

Como o AndroMDA já possui cartuchos disponíveis para Struts [STRUTS 2007], EJB e Hibernate, optou-se por utilizar nas camadas de Visão e Controle o arcabouço Struts, que foi desenvolvido com o objetivo de facilitar o desenvolvimento de aplicações para Web. Na camada de Modelo (persistência), foi usado o Hibernate, que é um arcabouço de mapeamento objeto-relacional em Java. Foi ainda utilizada uma outra camada para definir os componentes responsáveis pelo gerenciamento dos dados - a camada de serviços. Nessa camada utilizou-se EJB (*Enterprise Java Beans*), o qual define uma arquitetura de componentes distribuídos, provendo os serviços necessários para o gerenciamento e execução desses componentes. Nesse contexto, esses cartuchos foram estendidos de forma que, quando a segurança estivesse ativada, eles gerassem, além do código especifico de cada tecnologia, o código de segurança. Dessa forma, os padrões de projeto de segurança (Seção 2) são gerados através das regras de transformação adicionadas em cada cartucho. Além disso, para que os padrões de projeto de segurança funcionassem, uma série de outros padrões de projeto foram acoplados na geração do código. Como exemplo, pode-se citar o padrão *Interceptador*, que promove um controle central interceptando chamadas a métodos e incluindo verificação de regras de segurança.

Além de estender a ferramenta AndroMDA, para a implementação do ArchiMDAs também foram utilizadas e estendidas algumas classes do JAAS (*Java Authentication and Authorization Service)*, um arcabouço de segurança portável que possibilita fazer autenticação e autorização de clientes em Java. A escolha desse arcabouço foi devido ao fato dele ser amplamente conhecido e utilizado. Além do JAAS, como o servidor de aplicação utilizado foi o JBoss [JBOSS 2007], também foi necessário estender 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.

### 4.2 Estudo de Caso

Foi realizado um estudo de caso [FIORIO 2007] utilizando o arcabouço ArchiMDAs com o intuito de verificar se os requisitos definidos na Seção 2 são atendidos corretamente pelo arcabouço. Para tal, foi feita a verificação do código gerado para cada requisito de acordo com as regras de transformação definidas no ArchiMDAs. Neste estudo não estamos diretamente interessados em mensurar o ganho de produtividade e/ou qualidade derivado da utilização do arcabouço. 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 de informação para Web*

Para o estudo de caso, utilizou-se o ArchiMDAs na geração da lógica de segurança de um sistema real de gerenciamento de projetos. Esse sistema é responsável pelo cadastro de atividades e 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, servindo para organizar as tarefas padrão relacionadas a uma determinada atividade, enquanto que uma tarefa padrão representa uma tarefa que poder receber alocações.

De acordo com o processo do ArchiMDAs (Seção 3.2), as primeiras atividades a serem realizadas são as de responsabilidade do ator Arquiteto de Negócio. Contudo, essas atividades estão fora do escopo do estudo de caso, já que considerou-se que as

especificações relativas a arquitetura do negócio foram previamente feitas. Assim, o primeiro conjunto de atividades de interesse ao estudo são as atividades referentes ao ator Arquiteto de Segurança. Além dessas atividades, foi necessário desenvolver as atividades do ator Desenvolvedor de Negócio, pois é necessário elaborar o modelo PIM de negócio segundo as regras do ArchiMDAs para que os mecanismos de segurança sejam integrados a esse modelo. Em relação ao ator Gerente de Segurança, como os requisitos de segurança da aplicação estão contidos em um subconjunto dos requisitos de segurança (Seção 2) implementados pelo ArchiMDAs, não foi necessário criar uma aplicação de segurança específica, pois a gerada pelo módulo administrativo de segurança é suficiente para implementar a política de segurança da aplicação. Em relação ao ator Gerente de Implantação, a atividade realizada foi a ativação da segurança em todos os níveis, obtendo-se como resultado a geração da aplicação contendo o código de negócio integrado ao código de segurança. Por fim, foram realizadas as atividades referentes ao ator Administrador de Segurança, pois é necessário efetuar o cadastro dos artefatos de segurança (instância da política de segurança) para que sejam realizados os testes de utilização dos artefatos de segurança gerados pelo ArchiMDAs.

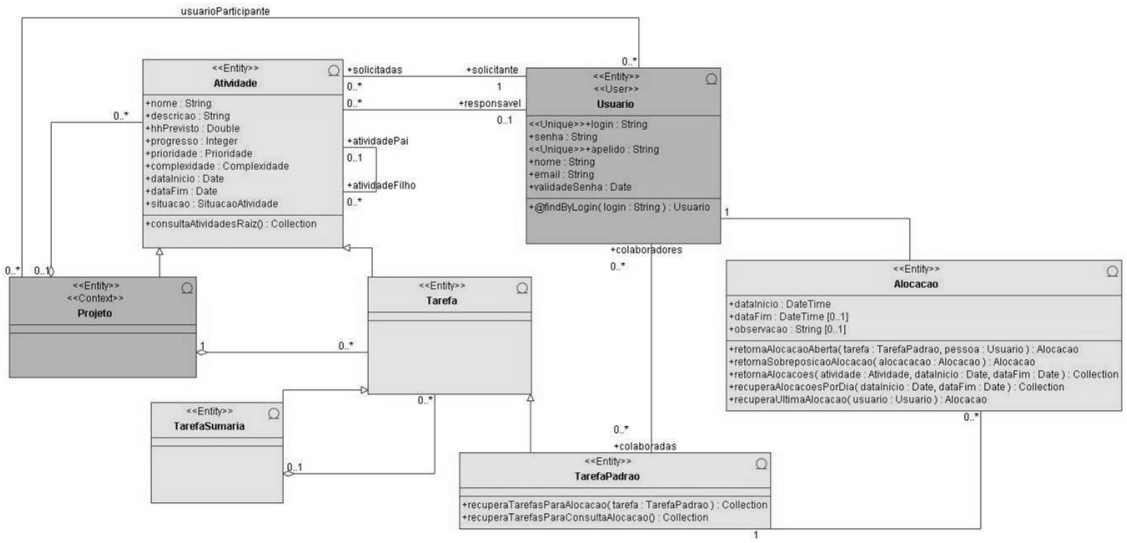


**Figura 3 – Modelo PIM da aplicação.**

A Figura 3 apresenta a modelagem das classes do domínio da aplicação. De acordo com o modelo, uma ***Atividade*** pode ser um projeto ou uma ***Tarefa***, e uma ***Tarefa*** pode ser sumária (***TarefaSumaria*****)** ou padrão (***TarefaPadrao***). Um ***Projeto*** pode possuir vários usuários (***Usuario***) e um ***Usuario*** pode participar de vários projetos. Um ***Usuario*** pode possuir varias alocações (***Alocacao***) e somente ***TarefaPadrao*** pode possuir alocações. Um usuário pode ser solicitante e/ou responsável de/por várias atividades, porém só pode ser colaborador de tarefas padrões. Nesse modelo, a classe ***Usuario*** representa a entidade ***Usuario*** do modelo RBAC estendido e a classe ***Projeto*** a entidade ***Contexto***. Para indicar esse comportamento, a classe ***Usuario*** deve ser modelada com o estereótipo <<User>> e a classe ***Projeto*** com o estereótipo <<Context>>.

Além do modelo de domínio (Figura 3), é necessário especificar os demais modelos de acordo com a arquitetura definida para a aplicação. Neste estudo de caso, foi necessário ainda modelar as classes de serviço: ***AtividadeHandler*** e ***AlocacaoHandler***. A primeira é responsável pela manutenção das Atividades e a segunda é responsável pela manutenção das Alocações. Para indicar que essas classes necessitam de mecanismos de segurança, elas foram modeladas com dependências em relação as classes ***Projeto*** e ***Usuario***. Neste estudo

de caso, as informações contidas nesses dois modelos (domínio e serviço) são suficientes para que o ArchiMDAs gere todo o código de autenticação e autorizacao da aplicação.

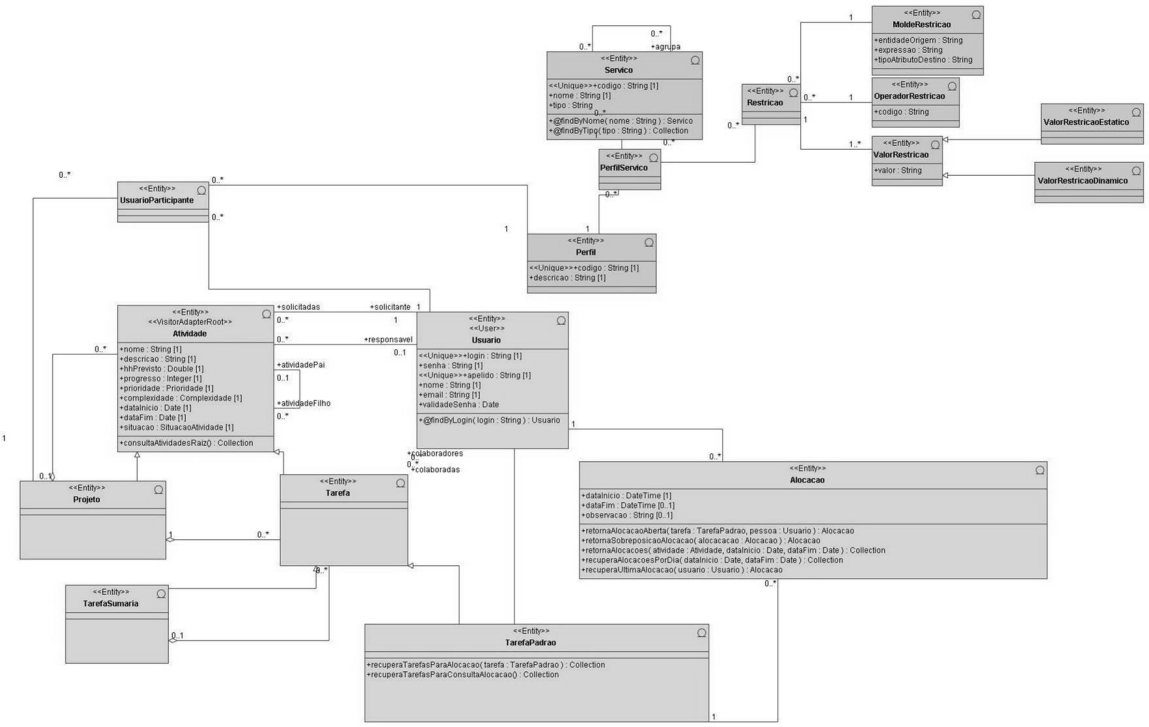


**Figura 4 – Modelo PSM integrando artefatos de segurança e negócio.**

A Figura 4 apresenta o modelo PSM gerado pelo ArchiMDAs. Pode-se perceber que, após aplicar as transformações, o modelo de domínio 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 Seção 3. Cabe ressaltar que, devido a ferramenta AndroMDA 3.1 não possuir facilidades para transformação do tipo modelo-modelo, o PSM de segurança (Figura 4) na implementação atual do ArchiMDAS não é explicitamente gerado, sendo um modelo intermediário manipulado internamente pelo programa de tranformação do engenho do ArchiMDAs.

**Tabela 3 - Pefil do Usuário no Projeto**

| | **Projeto 1** | **Projeto 2** |
|---|---|---|
| Usuário 1 | Desenvolvedor | Líder |
| Usuário 2 | | Desenvolvedor |
| Usuário 3 | | Líder |
| Usuário 4 | Líder | |
| Usuário 5 | Líder | |

Após a geração dos artefatos de segurança, ainda é necessário instanciar o modelo de segurança de acordo com as políticas de segurança específicas para a aplicação. Essa atividade é de responsabilidade do Administrador de Segurança, o qual utiliza, para tanto, a aplicação de administração de segurança gerada automaticamente pelo ArchiMDAs (módulo administrativo de segurança). No estudo de caso, a política de segurança foi instanciada conforme descrito a seguir. Foram criados 5 usuários, dois projetos e dois perfis (Desenvolvedor e Líder). A Tabela 3 representa o perfil que cada usuário possui em cada projeto. Lacunas em “branco” significam que o usuário não participa do projeto em questão.

Como exemplo, apenas o usuário 1 participa de dois projetos, sendo desenvolvedor no projeto 1 e líder no projeto 2. Foram instânciadas três restrições dinâmicas (Tabela 4). A semântica dessas restrições pode ser vistas na Seção 3.

**Tabela 4 - 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 |

A Tabela 5 mostra a associação dessas restrições com os serviços da aplicação (Seção 4.2). De acordo com as informações dessa tabela, os serviços cujo valor de ***id*** sejam 1 ou 2 só podem retornar as alocações cujo ***id*** do usuário seja igual ao valor retornado pelo método ***getUsuario***() implementado na classe ***AlocacaoImpl***. Essa restrição está associada tanto ao perfil "Desenvolvedor" quando ao perfil "Líder". Já o serviço cujo ***id*** é 3 só deve retornar as atividades cujo ***id*** do projeto seja igual ao valor do método ***getProjeto***() implementado na classe ***AtividadeImpl***. Essa restrição está configurada somente para o perfil "Líder" (o desenvolvedor não tem permissão para acessar esse serviço). Além dessa restrição, esse serviço só deve retornar as atividades cujo ***id*** da atividade pai (***atividadePai.id***) seja igual ao valor do método ***getProjeto***() da classe ***AtividadeImpl***. Esta restrição também está configurada somente para o perfil Líder.

**Tabela 5 - 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, java.lang.Integer) | Líder | 2 |
| 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 |

## 4.3 Casos de Testes

Para verificar se os requisitos de segurança foram atendidos, um conjunto de testes foram especificados e executados usando a aplicação gerada pelo ArchiMDAs [FIORIO 2007]. A análise dos resultados foi feita seguindo os requisitos definidos na Seção 2. Os resultados obtidos podem ser vistos na Tabela 6.

**Tabela 6 – Resultados dos testes.**

| **Requisito** | **Testado** | **Observação** |
|---|---|---|
| Ponto único de acesso ao sistema | Sim, OK | Tela de autenticação como ponto único |
| Usuário com nome de usuário (login) único e uma senha | Sim, OK | 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 usuário deve ser encaminhando a realizar sua autenticação. | Sim, OK | Para qualquer tentativa de acesso ao sistema sem estar autenticado, o sistema redirecionava para o ponto único de acesso o sistema, ou seja, a tela de autenticação |
| Os usuários devem ser organizados em papéis. | Sim, OK | 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 | Sim, OK | Foi testado com o usuário 1 que era líder e desenvolvedor |

| | | |
|---|---|---|
| necessariamente deve haver uma hierarquia entre eles. | | |
| Cada papel deve possuir um conjunto de permissões que representam acessos a recursos do sistema. | Sim, OK | Representado pela associação de Perfil com Servico |
| Deve ser possível identificar o usuário autenticado, assim como suas informações, em qualquer parte do sistema | Sim, OK | 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 emitirá um erro. | Sim, OK | Em cada camada há mecanismos de controle de acesso. A princípio, todos os serviços da camada visão são abertos, ou seja, o usuário tem uma visão completa do sistema e vai fechando à medida que o administrador concede permissões. Porém, como na camada de serviços todos os serviços são fechados, ao tentar acessar uma operação a qual não possui acesso, o sistema gera 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, OK | Na camada de vista, podemos considerar que o mecanismo é a tag lib security enquanto que na camada de serviço é a classe SecurityProxyImpl |
| Um usuário pode estar associado a um contexto de uso | Sim, OK | 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, OK | Utilizamos três restrições dinâmicas |
| 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 | Sim, OK | Cada camada possuía seus próprios mecanismos de autorização |

Pelos resultados da Tabela 6 pode-se perceber que todos os requisitos de segurança foram testados e validados de forma que o sistema se encontrava protegido de acordo com a política de segurança especificada e todos os recursos necessários para realização do controle de autenticação e autorização automaticamente gerados estão completos (tendo em vista o conjunto de requisitos de segurança levantados) e corretos.

## 5 Trabalhos Relacionados

Alguns trabalhos já usam MDA para desenvolver arcabouços de segurança. Em [BASIN, DOSER 2006] é apresentada uma abordagem MDA para engenharia de segurança, denominada segurança dirigida a modelo (*Model Driven Security*). Essa abordagem utiliza uma linguagem de modelagem baseada na UML, denominada SecureUML [BASIN, DOSER 2002], que integra controle de acesso baseado em papéis (*role-based access control - RBAC*) ao processo de desenvolvimento de software dirigido a modelo. Nesse trabalho os modelos UML são estendidos com informações de segurança, as quais são usadas para gerar infra-estruturas de controle de acesso. O modelo de negócio é combinado com o modelo de segurança, criando um novo tipo de modelo, denominado *modelo de projeto de segurança*, no qual as políticas de segurança se referem aos elementos do modelo de negócio, como componentes, objetos de negócio, métodos, atributos, etc.

Em [FINK, KOCH, PAULS 2004] é proposta uma abordagem baseada em MDA para desenvolver políticas de controle de acesso em sistemas distribuídos. Os modelos são expressos como modelos MOF [MOF 2002] enriquecidos por perfis UML. Nesse trabalho, um modelo de controle de acesso é especificado por um meta-modelo independente de domínio e de plataforma, o qual 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.

Em [JIN 2006] é apresentado um arcabouço que utiliza MDA juntamente com perfis UML para construir aplicações baseadas em RBAC para sistemas distribuídos. Além disso, é mostrado um estudo de caso exemplificando como os perfis UML apresentados podem ser utilizados nas fases iniciais do desenvolvimento do sistema, gerando automaticamente as especificações de segurança 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 permite 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. 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.

Em todos esses trabalhos, a definição de políticas de controle de acesso é feita usando-se perfis UML que estendem a UML com elementos específicos para representar modelos de permissões. Nessas abordagens, o modelo do negócio 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 essa abordagem é que o fator humano pode interferir na robustez de segurança em qualquer ponto onde intervenção manual é requerida. 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, dificultando sua compreensão e manutenção. Visando minimizar esses problemas, o ArchiMDAs segue uma abordagem distinta, na qual o foco principal não é na modelagem explícita das políticas de controle de acesso, mas sim na incorporação dessas informações através de regras de transformação entre modelos. No processo proposto pelo ArchiMDAs o desenvolvimento do modelo de negócio é ortogonal ao do modelo de segurança minimizando, assim, o acoplamento entre esses modelos e mantendo o modelo de negócio com o mínimo possível de detalhes relacionados a requisitos de segurança.

## 6 Conclusão

Neste trabalho foi apresentado o ArchiMDAs, um arcabouço de segurança orientado a modelo, onde a segurança é introduzida no sistema através de transformação entre modelos. O arcabouço contribui para especificação e implementação dos requisitos de segurança durante o processo de desenvolvimento de sistemas, simplificando o desenvolvimento, 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 do sistema gerado. Além disso, como o módulo administrativo do ArchiMDAs permite a definição de permissões de forma dinâmica, aumenta-se a flexibilidade do sistema, já que informações tipicamente relacionadas a implantação como, por exemplo, perfis (papéis) de usuários, não precisam ser modeladas em tempo de projeto (de negócio). A implementação do ArchiMDAs incorpora os principais padrões de projeto de segurança disponíveis na literatura, aumentando, com isso, a qualidade do sistema gerado com relação a segurança. Cabe ressaltar que os módulos do ArchiMDAs foram

projetados de forma a facilitar a sua extensibilidade. Assim, em domínios de aplicações que necessitem de requisitos diferentes daqueles atendidos pelo arcabouço, o Arquiteto de Segurança pode estender esses módulos de acordo com a política de segurança específica da organização. A única restrição é que essa extensão deve ser feita seguindo o processo proposto pelo ArchiMDAs.

É importante acrescentar que o ArchiMDAs já foi utilizado e testado em vários sistemas reais, permitindo perceber os benefícios de sua utilização. Dentre esses sistemas pode-se citar um sistema militar brasileiro[1] que possui uma política de segurança bastante restritiva. Os principais benefícios observados com o emprego prático do ArchiMDAs são:

- O desacoplamento entre a implementação do negócio e a solução de segurança facilita a evolução da solução fazendo com que ela não se torne obsoleta a médio prazo;
- Efetiva proteção de artefatos de diferentes camadas de uma aplicação (modelo MVC);
- Os requisitos de segurança não ficam suscetíveis a erros decorrentes de evoluções no código de negócio da aplicação e
- Facilidade de integração de requisitos de segurança durante o desenvolvimento do sistema.

## 7 Referências


ACEGI (2007) "ACEGI Security", http://acegisecurity.sourceforge.net, acessado em 12/2007.

ANDROMDA (2007) "AndroMDA", http://www.andromda.org, acessado em 09/2007.

BASIN, D. AND DOSER, J. (2002) "SecureUML: A UML-Based Modeling Language for Model-Driven Security". Conference UML 2002, LNCS, n. 2460, ISBN 3-540-44254-5.

BASIN, D.; DOSER, J. (2006) "Model Driven Security: from UML Models to Access Control Infrastructures. ACM TOSEM, Volume 15 , Issue 1, pp. 39 – 91, ISSN 1049-331X.

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 Universit¨at Berlin.

EJB (2007) "Enterprise Java Beans", http://java.sun.com/products/ejb, acessado em 09/2007.

FERNANDEZ, E. (2000) "Metadata and authorization Patterns", Report TR-CSE-00-16, Dept. of Computer Science and Eng., Florida Atlantic University, May.

FINK, T., KOCH, M., PAULS, K. (2004) "An MDA approach to Access Control Specifications Using MOF and UML Profiles", In Proc. Of VODCA 2004, Bertinoro, Italy.

FIORIO, M. (2007) "ArchiMDAs: Um Arcabouço De Segurança Baseado Em Transformações De Modelos Em MDA", Dissertação de mestrado, PPGI/UFRJ, Rio de Janeiro, Brasil.


[1] Devido a restrições contratuais, referências explicitas a esses sistemas não puderam ser incluídas no corpo do artigo. Para maires informações, solicitamos que os autores sejam contactados.

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.

HIBERNATE. (2007) “Hibernate”, http://www.hibernate.org, acessado em 09/2007.

JAAS. (2007) “JAAS”, http://java.sun.com/products/jaas/, acessado em 10/2007.

JBOSS. (2007) “JBoss”, http://docs.jboss.org/jbossas/jboss4guide/r2/html/index.html, acessado em 10/2007.

JBOSSSX. (2007) “JBossSX”, http://docs.jboss.org/jbossas/jboss4guide/r2 /html/ch8.chapter.html, acessado em 10/2007.

JIN, X. (2006) “Applying Model Driven Architecture approach to Model Role Based Access Control System”, MSc in System Science, University of Ottawa, Ottawa, Ontario, Canada.

MAIA, N. (2006) “ODYSSEY-MDA: uma abordagem para a transformação de modelos”, Dissertação de mestrado, COPPE/UFRJ, Rio de Janeiro, Brasil.

MISF (2007) “Microsoft® Internet Security Framework”, http://www.microsoft.com/security/default.mspx, acessado em dezembro de 2007.

MOF (2002) “Meta-Object Facility“, http://www.omg.org/cgi-bin/doc?formal/2002-04-03, acessado em julho de 2007.

OASIS (2007) “OASIS: Avancing E-Business Standards Since 1993”, disponível em: http://www.oasis-open.org, último acesso em 06/2007.

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.

RABAC XACML (2004). “XACML Profile for Role Based Access Control (RBAC)”, Organization for the Advancement of Structured Information Standards, http://docs.oasis-open.org/xacml/cd-xacml-rbac-profile-01.pdf, acessado em 06/2007.

SANDHU, R.; FERRAIOLO, 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. *et al.* (2005) “Security patterns. Integrating security and systems engineering”, John Wiley & Sons.

SOLINGEN, R. V., BERGHOUT, E. (1999) “The Goal/Question/Metric Method: A Practical Guide for Quality Improvement of Software Development”, McGrawn Hill, ISBN 0- 0770-9553-7.

STRUTS. (2007) “Struts”, http://struts.apache.org/, acessado em 12/2007.

UML (2007) “Unified Modeling Language”, http://www.uml.org/, acessado em 07/2007.

VELOCITY (2007) “Velocity”, http://velocity.apache.org/, acessado em 09/2007.

XMI (2007) “XML Metadata Interchange Specification version 2.1”, http://www.omg.org/technology/documents/formal/xmi.htm, acessado em 12/2007.

YODER, J.; BARCALOW, J. (1997) “Architectural Patterns for Enabling Application Security”, Pattern Languages of Programs, Monticello, IL.