Capítulo 4

Soluções para o desenvolvimento de sistemas seguros

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

[1]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

[2]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

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

Abstract

Modern Information systems are highly concerned with security and it is not rare the occurrence of security failures and vulnerabilities. Most of these problems are due to the dissociation of security requisites from the initial system conception. Security must be aggregated to the software development process to guarantee safety by design as an integral part of system construction. The development of security design patterns embedded into a model driven approach is an answer to help architects and designer to build systems with improved security.

Resumo

Apesar de grande parte dos sistemas de informação possuir a segurança como requisito fundamental, constantemente há noticias sobre vulnerabilidades e falhas de segurança. Isto se deve 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. Portanto, 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. Neste contexto, a utilização de padrões de projeto de segurança e de uma abordagem orientada a modelos pode auxiliar arquitetos e projetistas a construir sistemas seguro.

4.1. Introdução

No desenvolvimento de *software*, a qualidade do produto está diretamente relacionada à qualidade do processo de desenvolvimento sendo comum, portanto, que a busca por um *software* de maior qualidade passe necessariamente por uma melhoria nesse processo. [TSUKOMO 1997].

Nos últimos anos, empresas têm buscado certificações ISO ou CMM como meio de comprovar a qualidade no seu processo de desenvolvimento de *software* e, desta maneira, tornarem-se competitivas em um mercado cada vez mais exigente. Com isso, torna-se de grande importância o desenvolvimento de métodos e técnicas que permitam uma avaliação abrangente da qualidade dos processos e dos produtos de *software*, para garantir que o usuário receba produtos dentro das especificações por ele definidas e esperadas. Isto pode ser alcançado através da definição e especificação de características relevantes de qualidade do produto, com as respectivas avaliações, sempre que possível, usando métricas válidas e aceitas.

Segundo [ISO/IEC 9126-1 (2001)], a qualidade de *software* deve ser medida através da funcionalidade, confiabilidade, usabilidade, eficiência, manutenibilidade e portabilidade do *software*. Cada fator possui uma série de subfatores que também devem ser medidos, entre eles, a segurança e proteção do *software*. Para garantir a qualidade de segurança é comum utilizar um arcabouço de segurança, onde um arcabouço representa uma solução para um conjunto de problemas com características similares o qual disponibiliza componentes com capacidade de capturar funcionalidades comuns a várias aplicações. Ou seja, um arcabouço representa uma solução quase completa para um determinado problema, de forma independente do domínio da aplicação, feito normalmente por desenvolvedores experientes. Um arcabouço deve ser extensível, bem documentado e principalmente reutilizável.

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 noticias sobre vulnerabilidades e falhas de segurança em sistemas de software [FINK et al 2004]. Á medida que os arcabouços aparecem, eles se tornam rapidamente ultrapassados por diferentes motivos. Na maioria das vezes porque eles não oferecem um modelo arquitetural suficientemente flexível para acompanhar as necessidades do negócio, o qual está em desenvolvimento ou porque eles limitam o escopo da arquitetura de que o negócio necessita. Devido à 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.

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. Por outro lado, a necessidade de baixo tempo de resposta no desenvolvimento de soluções faz com que os riscos de sucesso dos projetos se tornem mais críticos.

Uma das abordagens que pode auxiliar arquitetos e projetistas a construir sistemas seguros consiste no uso de padrões de projetos. 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.

Contudo, para obter os benefícios da utilização de padrões de projeto é necessário saber quando e como utilizá-los a partir de suas definições. Neste contexto, uma abordagem orientada a modelo pode auxiliar o processo de desenvolvimento e de criação de arcabouços de segurança, promovendo uma implementação controlada dos padrões de projeto e, desta forma, garantindo aderência às diretrizes arquiteturais da organização e às questões de segurança.

O objetivo desse capítulo é abordar aspectos teóricos e práticos das soluções de segurança existentes para o desenvolvimento de sistemas de software. Para atingir esse objetivo, serão apresentados os modelos básicos de segurança, integração de 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 MDD (*Model Driven Development*).

Na seção 1.2, 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. Na seção 1.3, será feita uma análise de modelagem de sistemas seguros, apresentando as técnicas de integração de soluções de segurança nas fases de análise e projeto de sistemas de software. Na seção 1.4, será apresentada uma conclusão deste trabalho.

4.2. Fundamentos de Segurança em Ambiente Computacional

Com a interligação dos computadores, através de redes físicas ou *wireless* (conexões de equipamentos sem fio), estes computadores se tornam mais suscetíveis a acessos não autorizados e o conceito de segurança, importante em ambientes físicos se torna relevante para ambientes distribuídos. Desta forma é necessário que os usuários de um sistema computacional tenham acesso somente ao que lhe seja pertinente. Pois caso isso

não ocorra, o sistema se torna mais suscetível a ataques podendo causar sérios danos ao sistema.

Nesta seção iremos introduzir uma série de conceitos, modelos e técnicas utilizados com o objetivo de garantir a segurança da informação em sistemas computacionais.

4.2.1. Conceito de segurança

Nos últimos anos os sistemas computacionais de segurança têm sido alvo de muito interesse pela grande maioria das pessoas que se utilizam deles direta ou indiretamente. Na mídia, não são poucos os relatos encontrados sobre notícias de pessoas, instituições ou empresas que tiveram enormes prejuízos causados por ações intrusivas executadas pelos especialistas deste tipo de delito, que são conhecidos como *hackers*. Devido ao conhecimento de fatos como estes e muitas vezes à própria convivência com os mesmos é que chegamos à conclusão de que a principal finalidade da "segurança" consiste em promover a defesa de ataques externos como forma de evitar prejuízos de qualquer natureza que em sua grande parte geram perdas financeiras e até morais ao prejudicado.

A segurança em sistemas computacionais consiste em 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 seja alvo de possíveis ações danosas praticadas por entidades não autorizadas junto às informações e recursos nele existentes. De acordo com [BISHOP 2003], 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.

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

Alguns autores tratam que um sistema será seguro se abordar outros dois itens [BISHOP 2003, LANDWEHR 2001]:

- **Autenticidade:** Garante que a informação ou o usuário da mesma é autêntico; Atesta a origem do dado ou informação
- **Não repúdio:** Não é possível negar (no sentido de dizer que não foi feito) uma operação ou serviço que modificou ou criou uma informação

Outros três itens podem ser desejáveis para se ter um ambiente seguro [BISHOP 2003]:

- **Legalidade:** Garante a legalidade (jurídica) da informação; Aderência de um sistema à legislação; Característica das informações que possuem valor legal dentro de um processo de comunicação, onde todos os ativos estão de acordo com as cláusulas contratuais pactuadas ou a legislação política institucional, nacional ou internacional vigentes
- **Privacidade:** Uma informação pode ser considerada confidencial, mas não privada. Uma informação privada deve ser vista/lida/alterada somente pelo seu proprietário. Consiste em garantir que a informação não será disponibilizada para outras pessoas (neste caso é atribuído o caráter de confidencialidade a informação)
- **Auditoria:** Verificar e mapear os passos que um determinado processo realizou ou que uma informação foi submetida, identificando os participantes, os locais e horários de cada etapa. Auditoria em software significa uma parte da aplicação, ou conjunto de funções do sistema, que viabiliza uma auditoria, consistindo no exame do histórico dos eventos dentro de um sistema para determinar quando e onde ocorreu uma violação de segurança

4.2.2. Política de Segurança

O termo “Política de Segurança” pode ter vários significados dependendo do nível em que é aplicado. Vendo sob o ponto de vista administrativo de uma instituição, podemos definir como sendo um conjunto de leis e práticas utilizadas pela instituição para gerenciar, proteger e distribuir suas informações. E esta mesma ótica administrativa em relação à segurança da informação deve-se refletir nos ambientes computacionais, onde a política de segurança passa a 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.

Quando um administrador de um ambiente computacional necessita tomar decisões para realizar a proteção do ambiente computacional, o mesmo necessita estabelecer regras; definindo as funcionalidades que irá oferecer, e qual será a facilidade de utilizá-la. Às vezes se torna complicado realizar decisões sobre segurança, sem que tenham determinado quais são as suas metas de segurança.

O estabelecimento de um conjunto de regras irá definir as políticas de segurança, o principal objetivo de uma política de segurança é informar aos usuários, equipe e gerentes, as suas obrigações para a proteção da tecnologia e do acesso à informação. Os objetivos determinados devem ser comunicados a todos os usuários, pessoal operacional, e gerentes através da política de segurança adotada.

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 incontornável de regras que expressam algum tipo de organização envolvendo a segurança das informações no sistema como um todo [MACKENZIE 1997], baseando-se em uma administração centralizada de segurança, a qual dita qual serão as regras de acesso à informação.

O objetivo da utilização de políticas de segurança no ambiente computacional é tornar o sistema mais seguro contra intrusos. Com a adoção de políticas de segurança tanto discricionárias quanto obrigatórias tornam o ambiente mais seguro, mas não livre de ações que tentam contornar as políticas de segurança adotadas.

4.2.3. 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. Essas entidades são normalmente identificadas como "principais"**.** Dependendo do nível de aplicação desta política pode-se caracterizar como sendo principal 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. A Tabela 1.1. mostra os tipos de violação alocados em contraposição às propriedades de segurança não verificadas [AMOROSO 1994].

Tabela 1.1. Relação de violações as propriedades de segurança

Tipo de Violação (TV)		Propriedade de Segurança Violada
I	revelação não autorizada	confidencialidade
II	modificação não autorizada	integridade
III	negação de serviço	disponibilidade

Antes de verificarmos as principais violações existentes no ambiente distribuído, devemos fazer algumas definições que serão importantes para o nosso entendimento. Uma ameaça é caracterizada como sendo uma ação possível que, uma vez concretizada, produz efeitos indesejáveis sobre os dados ou recursos de sistema. Uma ameaça, quando posta em ação, é identificada como um ataque à segurança do sistema. Entende-se por vulnerabilidade como sendo uma falha ou característica indevida existente no sistema oriunda de falhas de concepção, implementação ou de configuração, a qual expõe os recursos deste sistema computacional a ataques e que podem ser exploradas para concretizar uma ameaça.

Uma maneira de ilustrarmos o relacionamento entre ameaças, vulnerabilidades e ataques é fazendo uma analogia com uma casa. Uma ameaça associada a uma casa é o roubo de móveis, dinheiro e eletrodomésticos. Vulnerabilidades podem ser comparadas a uma janela aberta ou uma porta que não esteja trancada. O ataque consiste na invasão propriamente dita com o conseqüente roubo dos bens existentes.

A Tabela 1.2 relaciona as ameaças mais comumente presentes em relatos de violações de segurança em sistemas distribuídos [AMOROSO 1994].

Tabela 1.2. Ameaças mais comuns ao sistema de informática

Ameaça	TV	Descrição
Mascaramento (masquerade/spoofing)	II	Técnica utilizada para "se fazer passar"/forjar uma identidade. Utilizada por exemplo em capturadores de senha, IP spoofing, etc.

Bypassing control	I	Técnica utilizada para explorar vulnerabilidades do sistema. É comum a utilização desta técnica em invasões de sistemas com versões (patchs, service packs) desatualizadas.
Código maléfico (malicious code)	I, II, III	Técnica que utiliza software com código contendo partes aparentemente inofensivas ou invisíveis. Estas, quando executadas, comprometem a segurança dos recursos do sistema. Utilizada por exemplo em cavalos de tróia, vírus, bombas lógicas, etc.
Backdoor/Trapdoor	I, II	Técnica utilizada para inserir/criar funções/escutas no sistema, que aceitam entradas específicas e permitem contornar/driblar os mecanismo s de segurança do sistema.
Inspeção de lixeira (midia scavenging)	I	Técnica que consiste em bisbilhotar/revirar lixeiras pro curando papéis, discos, etc, com informações importantes que não foram adequadamente destruídas.
Scanning	I	Técnica utilizada para bisbilhotar recursos do sistema com algum objetivo específico. Um ataque deste tipo utiliza-se de um software scanner para ser executado.
Negação de Serviço	III	Técnica utilizada para impedir o acesso legítimo ao sistema ou a algum recurso deste. Alguns exemplos desta técnica são Distributed Denial of Service, Worms, repetidor de discagem e diversas variações de Denial of Service.
Ataque nas Comunicações	I, II	Técnica utilizada para escuta, interceptação, inserção ou alteração de mensagens durante a transmissão. Geralmente, a escuta da comunicação com um software sniffer, por exemplo, precede o uso de ataques como Man in-the-middle ou roubo da sessão autenticada.

4.2.4. 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 1982]. Na literatura é comum encontrarmos os modelos divididos em três tipos básicos [SANDHU 1996, OSBORN 2000]:

- Baseados em identidade ou discricionários (*discretionary)*;
- Baseados em regras ou obrigatórios (*mandatory)*;
- Baseados em papéis (*roles*);

4.2.4.1. Modelos discricionários (discretionary)

Nos modelos discricionários, 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 94].

Os modelos discricionários se baseiam 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 destes modelos 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 a 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.

Os modelos discricionários, na sua essência, são baseados no modelo matriz de acesso proposto por Lampson [LAMPSON 1971]. Neste modelo, o estado de proteção do sistema é representado através de uma matriz, onde as linhas correspondem aos sujeitos e as colunas correspondem aos objetos do sistema. Esta matriz relaciona-se através dos objetos, que podem ser definidos como sendo os recursos do sistema, dos sujeitos, que são as entidades ativas existentes no sistema e dos atributos de acesso, que são os direitos ou permissões de acesso (*read*, *write*) cabíveis ao sistema. Cada sujeito (S1, S2, S3, etc.) é representado por uma linha da matriz e cada objeto (O1, O2, etc.) por uma coluna. Na célula de intersecção das duas, onde ambas (linha x coluna) se encontram são especificados os atributos de acesso do respectivo sujeito em relação ao objeto considerado. O modelo de matriz de acesso pode ser visto na Figura 1.1.

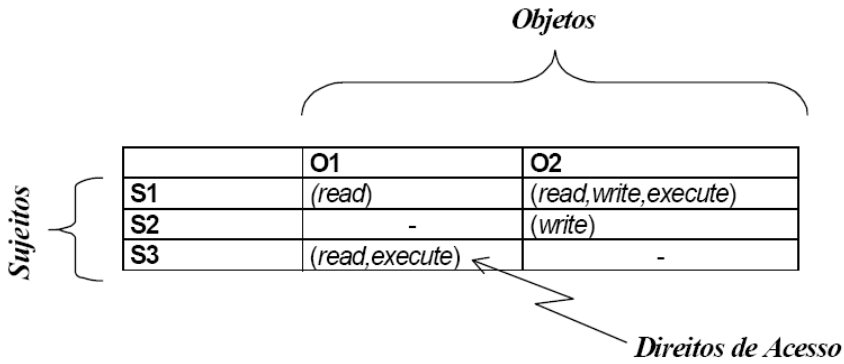

Figura 1.1. Modelo Matriz de Acesso

Se considerarmos uma coluna da matriz de acesso, veremos que a relação de todos os sujeitos existentes, com seus respectivos direitos de acesso sobre o objeto, correspondem à coluna, formando o que é identificado como sendo uma lista de controle de acesso (LCA) do objeto considerado. As LCAs correspondem a uma forma de apresentação do modelo matriz de acesso. Na Tabela 1.3, é apresentado um conjunto de LCAs onde cada lista corresponde a uma entrada do objeto correspondente.

Tabela 1.3. Listas de Controle de Acesso

Objeto	Listas de Controle de Acesso
O1	S1(read), S3(read, execute)
O2	S2(write), S1(read, write, execute)

A LCA de um objeto permite uma fácil revisão dos acessos autorizados a um objeto. Outra operação bastante simples com o uso de LCAs é a revogação de todos os acessos a um objeto, bastando para isso substituir a LCA corrente por outra vazia. Por outro lado, a determinação dos acessos aos quais um sujeito está autorizado é bastante problemática; é necessário percorrer todas as LCAs do sistema para fazer este tipo de revisão de acesso. A revogação de todos os acessos de um sujeito também requer que todas as LCAs sejam inspecionadas e, eventualmente, modificadas.

A matriz de acesso pode ainda apresentar uma outra representação através de Listas de Competência. Neste tipo de representação cada sujeito é associado a uma lista que indica, para cada objeto no sistema, os acessos que o sujeito está autorizado a efetuar. Isso corresponde a armazenar a matriz de acesso por linhas. Na Tabela 1.4, cada lista é representada em uma entrada por sujeito da matriz.

Tabela 1.4. Listas de Competências

Sujeitos	Listas de Competências
S1	O1(read), O2(read, write, execute)
S2	O2(write)
S3	O1(read, execute)

As competências permitem uma fácil verificação e revogação dos acessos autorizados para um determinado sujeito. Porém, em contrapartida, a determinação de quais sujeitos poderão acessar um objeto requer a inspeção de todas as competências do sistema. As vantagens e desvantagens de LCAs e competências são, como as próprias estratégias, ortogonais entre si.

Competências são vantajosas em sistemas distribuídos. A posse de uma competência é suficiente para que um sujeito obtenha o acesso autorizado por esta competência. Em um sistema distribuído, isso possibilita que um sujeito se autentique uma vez, obtenha a sua lista de competência e apresente estas competências para obter os acessos aos quais ele está autorizado; os servidores precisam apenas verificar a validade dessa competência para liberar o acesso.

4.2.4.2. Modelos Obrigatórios (mandatory)

Os Modelos obrigatórios têm em seu esquema de autorização um conjunto de regras incontornáveis 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; os 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.

A seguir serão descritos os principais modelos obrigatórios constantes na literatura.

4.2.4.2.1. Modelo Bell-LaPadula (BLP) de confidencialidade

O modelo Bell e Lapadula (BLP) tem este nome devido aos cientistas David Bell e Leonard Lapadulla que desenvolveram este modelo no inicio da década de 70. O referido modelo é baseado nos procedimentos de manipulação de informação em áreas ligadas à segurança nacional americana [AMOROSO 94, BELL 1776].

A idéia principal deste modelo é acrescentar controles de acesso obrigatório aos controles de acesso discricionário, desta forma, aplicando políticas de segurança que impeçam o fluxo de informações de níveis mais altos aos níveis mais baixos de segurança. As permissões de acesso são definidas através de uma matriz de acesso e de rótulos de segurança. A matriz de acesso armazena os direitos de cada sujeito sobre os objetos do sistema e pode ser modificada pelos sujeitos através de regras específicas.

O modelo Bell-LaPadula priva a confidencialidade e está baseado na classificação dos elementos de segurança, que definem a política de acesso ao sistema. Esta classificação é expressa por níveis de segurança, onde cada nível é definido por

dois componentes: uma classificação e um conjunto de categorias. O modelo BLP classifica as informações em quatro níveis hierárquicos de sensibilidade (não-classificada, confidencial, secreta e ultra-secreta – respectivamente do menor para o maior nível). O conjunto de categorias não possui nenhuma hierarquia e os seus elementos são definidos de acordo com o ambiente ou área de aplicação do modelo.

Ao utilizar este método de classificação reduz-se a complexidade das regras, aproximando o modelo BLP dos ambientes computacionais de uso corrente, sem enfraquecê-lo em nenhum aspecto.

O sistema é descrito em termos de sujeitos que acessam objetos, onde cada sujeito possui uma habilitação e cada objeto possui uma classificação. A cada sujeito está associado também um rótulo corrente de segurança - representando a classificação mais alta dentre as informações já consultadas pelo sujeito no sistema, sendo portanto, uma classificação flutuante (dinâmica). Para evitar a revelação da informação a sujeitos não autorizados, basicamente, o modelo BLP impõe duas propriedades que devem ser satisfeitas para que a segurança do sistema seja garantida.

- **Propriedade de Segurança Simples *ou No Read Up (NRU):*** um sujeito pode somente observar informações para as quais esteja habilitado, evitando assim que um sujeito de nível inferior leia informações de um nível mais elevado que o seu nível de segurança (habilitação e compartimento)*;*
- **Propriedade Estrela ou No Write Down (NWD):** um sujeito só pode escrever em objetos cujos níveis de segurança dominam o nível de segurança do sujeito.

Estas propriedades devem ser satisfeitas para que se possa evitar que a informação de um nível mais alto acabe fluindo para níveis baixos de segurança, caracterizando a revelação não autorizada de informação. Além das duas propriedades descritas o modelo também mantém um controle discricionário por nível de segurança (*discretionary security propriety*) que reflete os princípios de autorização expressos no modelo matriz de acesso.

O modelo de *BLP* apresenta entre as suas principais limitações [MCLEAN 1990, MACKENZIE 1997], a escrita às cegas e a superclassificação da informação.

A Escrita às Cegas é assim denominada porque a propriedade estrela permite que um sujeito escreva num objeto de nível de sensibilidade superior a sua habilitação, o que pode determinar a destruição de informações sem caracterizar uma violação de segurança e das regras do modelo propriamente dito. O cenário de escritas cegas torna-se uma preocupação na medida em que o mesmo sujeito considerado inadequado para ver o conteúdo de um objeto possui permissão para fazer modificações arbitrárias neste mesmo objeto. Isto pode causar problemas de integridade que só podem ser resolvidos através de alterações nas regras do modelo BLP. Por exemplo, escritas em níveis de segurança mais altos que o corrente podem ser proibidas, ou seja, um sujeito somente poderia escrever em um objeto que tivesse o mesmo nível de segurança. Entretanto, tal modificação restringe, de certa forma, o modelo BLP e muda seu enfoque, que deixa de ser exclusivamente a ameaça de revelação não autorizada e passa a ser uma combinação de revelação e integridade. Por outro lado, a adoção de propriedade-* revisada é bastante comum em implementações de sistemas computacionais que seguem o modelo BLP.

A superclassificação *da informação* é determinada pelas regras do modelo que definem uma espécie de fluxo da informação dos níveis de segurança mais baixos para os mais altos, o que dificulta, com o tempo, a manipulação da informação no sistema. Por exemplo, se um sujeito com rótulo corrente de segurança SECRETO deseja copiar um arquivo CONFIDENCIAL, a propriedade-* impõe que a cópia tenha classificação SECRETO, mesmo que as informações ali contidas possuam classificação CONFIDENCIAL. Ao longo do tempo, isso faz com que as informações subam no reticulado de rótulos de segurança, recebendo classificações sucessivamente maiores. A superclassificação da informação provoca a necessidade de reclassificações periódicas dos objetos (através de sujeitos de confiança) apenas para garantir a usabilidade de sistemas baseados no modelo BLP.

4.2.4.2.2. O Modelo Biba de Integridade

O modelo obrigatório de integridade BIBA [AMOROSO 1994, BIBA 1977] é descrito como o inverso do modelo BLP [AMOROSO 1994]. Esta é uma descrição razoável, pois as regras básicas do modelo BIBA são bem próximas das regras do modelo BLP. No modelo de BIBA são definidas regras onde um sujeito estando em um nível de integridade mais elevado não pode ler um objeto que esteja em um nível de integridade inferior ao seu (*no read down* NRD). Também estabelece que um sujeito estando em um baixo nível de integridade não poderá escrever em um objeto em um nível de integridade superior ao seu (*no write up* NWU) [AMOROSO 1994].

O modelo obrigatório BIBA está baseado na integridade e tanto uma regra quanto outra deste modelo, são opostas ao modelo BLP que em sua proposta visa a confidencialidade. Deste modo, os níveis mais elevados de integridade devem ser vistos como uma associação daqueles sujeitos e objetos que devem ter um nível de integridade mais elevado, e os níveis mais baixos devem ser vistos como uma associação daqueles sujeitos e objetos que podem tolerar um menor nível de integridade.

O modelo BIBA pode ser representado através de um diagrama de níveis de maneira mais objetiva. As linhas horizontais do diagrama representam os níveis de integridade e as ações que são negadas pelo contexto geral do modelo. O diagrama do modelo BIBA pode ser melhor visualizado na Figura 1.2.

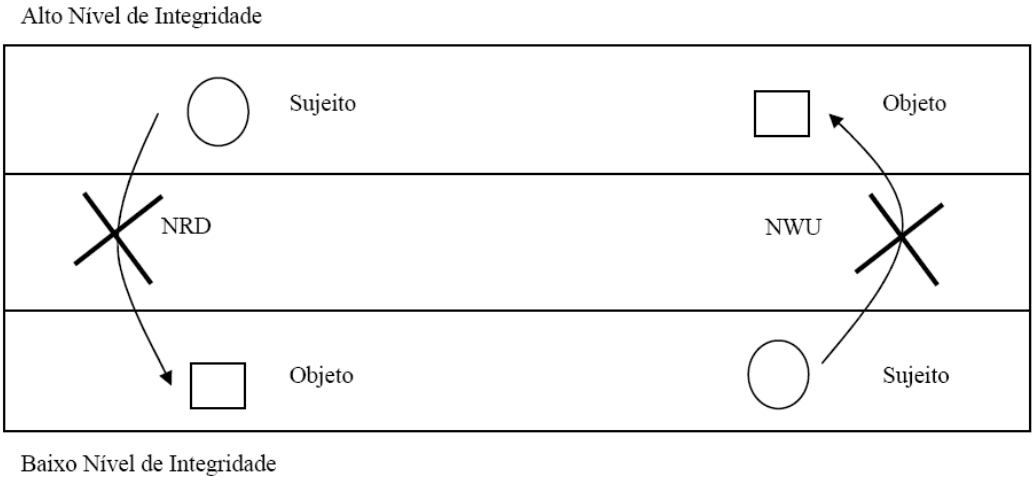

Figura 1.2. Regras do Modelo Obrigatório de Integridade BIBA

Uma das vantagens do modelo BIBA, é a incorporação de algumas características das regras do modelo BLP incluindo a simplicidade e atributos intuitivos. Isto é, os desenvolvedores de sistemas podem facilmente entender as regras de NWD e NRU, podendo incorporá-las em projetos de decisões de sistemas.

O Modelo da marca d´água baixa do sujeito [AMOROSO 1994] introduz um leve relaxamento nas regras de leitura de sujeito mais íntegros em objetos menos íntegros. A revisão do modelo obrigatório BIBA de integridade não permite que os sujeitos de alta integridade leiam os objetos de baixa integridade. Isto pretende assegurar que a informação do sujeito de alta integridade, não seja corrompida pela baixa integridade do objeto. Entretanto no modelo da marca d´água baixa do sujeito é permitida a leitura de objetos de menor integridade por sujeitos mais íntegros, mas o resultado de tal leitura é o rebaixamento do nível de integridade do sujeito ao nível do objeto lido [AMO94]. As características deste modelo são mostradas na Figura 1.3.

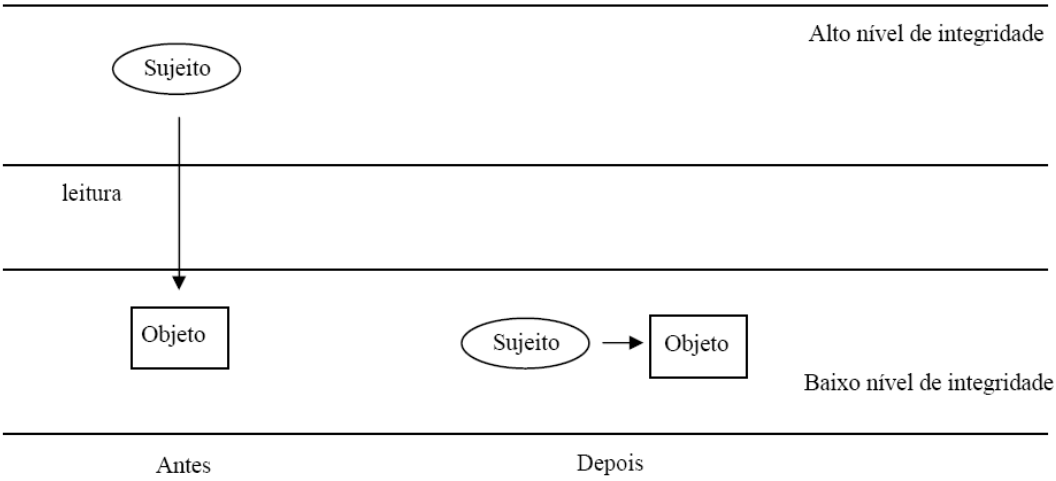

Figura 1.3. Modelo BIBA da Marca D`Água Baixa do Sujeito

Os modelos descritos são relativamente simples e intuitivos, e conseguem demonstrar de uma maneira bastante intuitiva a sua capacidade de conservar a propriedade de segurança considerada (confidencialidade no BLP e integridade no Biba).

Entretanto, o modelo Biba, assim como o BLP, depende em demasia de sujeitos de confiança em situações práticas: a necessidade de um processo de confiança para aumentar ou reduzir a integridade de sujeitos ou objetos é especialmente problemática para a integridade. Outra crítica ao modelo Biba é a ausência de provisão de mecanismos para a promoção da integridade de um sujeito ou objeto. Cabe notar que todas as mudanças possíveis no modelo Biba preservam a integridade de todos os sujeitos ou rebaixam a integridade de algum sujeito ou objeto. Isso permite imaginar que, com o passar do tempo, os sistemas sofrem um rebaixamento do nível de integridade, proporcional a quanto os sujeitos e objetos migram para o nível mais baixo. Esta degradação da integridade da informação é análoga ao problema de superclassificação da informação no modelo BLP.

4.2.4.3. Modelos Baseados em Papéis (Role Based Access Control - 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 (*roles*) em um sistema. Um papel pode ser definido como um conjunto de atividades e responsabilidades associadas a um determinado cargo ou função em uma organização. Assim, no RBAC, as permissões são conferidas aos papéis e os usuários são autorizados a exercer papéis (Figura 1.4). 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 2000]. Esta família de modelos RBAC-NIST se apresenta estratificada a partir do modelo RBAC básico (*Flat RBAC*), evoluindo para outros modelos da família – RBAC Hierárquico (*Hierarchical RBAC*), RBAC com Restrições (*Constrained RBAC*) e RBAC Simétrico (*Symmetric RBAC*) – adicionando funcionalidades ao mesmo.

4.2.4.3.1. RBAC Básico

Os aspectos essenciais e fundamentais do RBAC estão definidos no RBAC Básico. Como visto, o conceito básico do RBAC é a associação de usuários a papéis, a associação de permissões a papéis e a aquisição de permissões do usuário pelo papel que o mesmo desempenha. Além do conceito básico do RBAC, o modelo RBAC Básico ainda incorpora as seguintes características: (i) exige que as associações usuário-papel e permissão-papel possam ser muitos-para-muitos; (ii) provê suporte à revisão usuário-papel; e (iii) permite a ativação múltipla de papéis.

O RBAC Básico mostrado na Figura 1.4. possui três conjuntos de entidades: usuários (U), papéis (R, de *roles*) e permissões (P). Neste modelo um usuário é uma pessoa ou um processo agindo em nome de uma pessoa. Um papel é uma função ou cargo dentro da organização que possui uma semântica representando a autoridade e a responsabilidade conferidas aos membros desse papel. Uma permissão é um direito específico de acesso a um ou mais objetos do sistema; a natureza exata de uma permissão é dependente da implementação e não é especificada pelo modelo RBAC-NIST [SANDHU 2000].

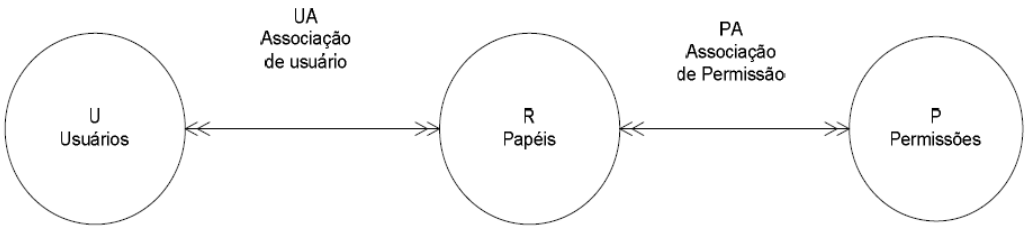

Figura 1.4. RBAC Básico

O RBAC Básico exige que a associação usuário-papel (UA, de *user-role assignment*) e a associação permissão-papel (PA, de *permission-role assignment*) sejam relações do tipo muitos-para-muitos (isto é indicado pelas setas duplas na Figura 1.4). Este é um aspecto essencial do RBAC pois permite que um usuário exerça as permissões de vários papéis. O princípio do mínimo privilégio dificilmente pode ser respeitado quando todos os papéis de um usuário são ativados em uma sessão, mas esse princípio pode ser respeitado mais facilmente permitindo-se o usuário ativar e desativar os papéis que deseja utilizar, pois o usuário pode ativar apenas os papéis necessários à execução de determinada tarefa.

O suporte à revisão usuário-papel possibilita determinar de maneira eficiente quais os usuários associados a um papel e a quais papéis um usuário está associado. A revisão permissão-papel também é muito importante no RBAC para responder quais permissões estão associadas a um papel e a quais papéis uma permissão está associada, mas pela dificuldade de se implementar este mecanismo em sistemas distribuídos de larga escala, ele é requerido somente no RBAC Simétrico [SANDHU 2000].

4.2.4.3.2. RBAC Hierárquico

A inclusão da relação de hierarquia de papéis (RH, de *role hierarchy*), mostrada na Figura 1.5. é a principal diferença do RBAC Hierárquico para o Básico. Uma hierarquia é matematicamente uma ordem parcial definindo uma relação de precedência de papéis, por meio da qual papéis de categoria superior adquirem as permissões dos seus subordinados. Hierarquias de papéis é uma forma natural de estruturar os papéis de modo a refletir as linhas de autoridade e responsabilidade em uma organização.

Existem duas interpretações distintas para uma hierarquia de papéis. Na primeira delas, papéis superiores (*senior roles*) herdam as permissões dos papéis inferiores (*junior roles*); neste caso, tem-se uma hierarquia de herança.

Na outra interpretação para a hierarquia de papéis, a ativação de um papel superior não implica a ativação automática das permissões dos papéis inferiores; neste caso, tem-se uma hierarquia de ativação. Para que as permissões dos papéis inferiores sejam ativadas, estes papéis devem ser explicitamente ativados.

É possível aplicar as duas interpretações simultaneamente. Em tais casos, a hierarquia de ativação pode estender a hierarquia de herança ou ser independente desta [SANDHU 1998]. O modelo RBAC-NIST não define uma interpretação específica para as hierarquias de papéis [SANDHU 2000].

O modelo do NIST divide a RBAC Hierárquico em dois níveis:

- RBAC Hierárquico Geral: neste caso qualquer tipo de ordem parcial pode constituir uma hierarquia de papéis.
- RBAC Hierárquico Limitado: neste caso existem restrições em relação à estrutura da hierarquia de papéis. Geralmente, as hierarquias são limitadas a estruturas simples como árvores ou árvores invertidas.

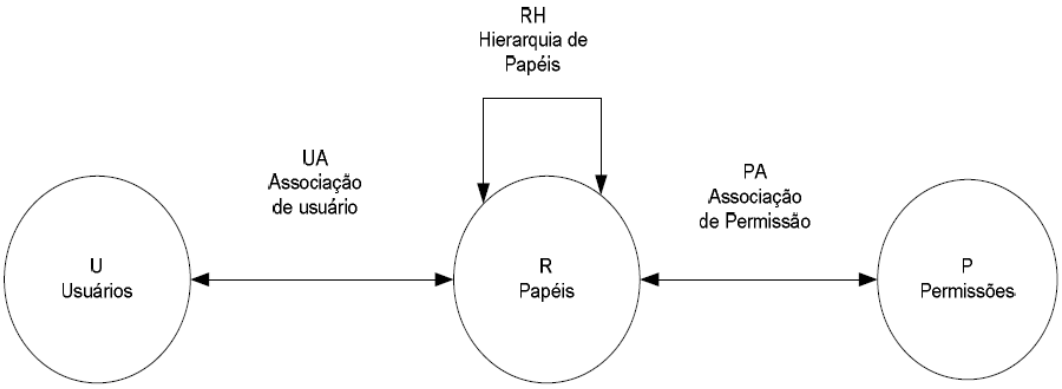

Figura 1.5. RBAC Hierárquico

O mecanismo de revisão usuário-papel do RBAC Básico deve ser estendido para suportar hierarquias de papéis de modo a permitir a identificação tanto dos papéis associados diretamente a um usuário (definidos pelo administrador de segurança) como dos papéis associados indiretamente ao usuário (cuja associação se dá através de herança).

4.2.4.3.3. RBAC com Restrições

Um conceito bastante importante para o modelo de restrições RBAC, no sentido de minimizar a ocorrência de erros e fraudes na manipulação da informação, é a separação de tarefas (*separation of duty*). Este conceito consiste em dividir tarefas que podem gerar conflito de interesses em várias subtarefas menores, executadas por pessoas diferentes, reduzindo, desta maneira, o poder individual de cada usuário. A separação de tarefas teve a sua importância para a segurança da informação reconhecida e discutida em detalhes por Clark e Wilson [CLARK 1987]. O RBAC facilita a implantação de separação de tarefas, utilizando-se, para isso, de relações de exclusão mútua entre papéis. A separação de tarefas é suportada pelo princípio do mínimo privilégio na definição dos papéis, ou seja, os papéis têm que estar associados ao mínimo de permissões necessárias ao cumprimento de suas tarefas.

Existem duas formas de separação de tarefas: estática e dinâmica. Na separação estática de tarefas, dois papéis R1 e R2 que são mutuamente exclusivos não podem ter usuários em comum; em outras palavras, um mesmo usuário não pode ser associado a R1 e a R2. Por outro lado, na separação dinâmica de tarefas uma exclusão mútua entre dois papéis R1 e R2 significa que um usuário pode ser associado a ambos, desde que apenas um deles (R1 ou R2) esteja ativo em um dado momento [SANDHU 2000].

4.2.4.3.3.1. Separação Estática de Tarefas

O conflito de interesses em um sistema baseado em papéis pode surgir como resultado da obtenção de permissões associadas com papéis conflitantes [SANDHU 2000]. Um modo de prevenir esta forma de conflito de interesse é através da separação estática de tarefas (SSD – *Static Separation of Duty*), ou seja, impondo restrições à associação de usuários a papéis (Figura 1.6). Nesta abordagem, usuários associados a um papel não podem ser associados a um segundo papel, de acordo com restrições definidas pelo administrador de segurança.

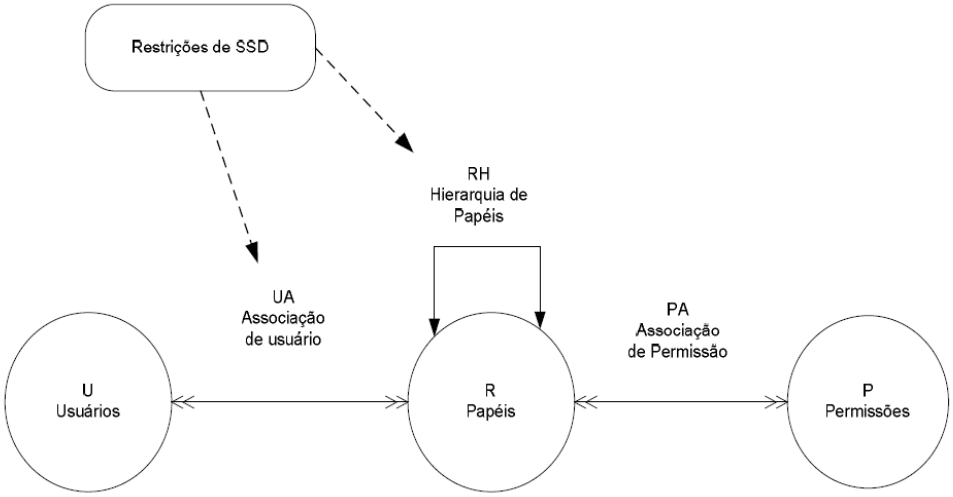

Figura 1.6. RBAC com Restrições – Separação Estática de Tarefas

4.2.4.3.3.2. Separação Dinâmica de Tarefas

O uso de políticas de separação dinâmica de tarefas (DSD – *Dynamic Separation of Duty*) (Figura 1.7) também é permitido no RBAC com Restrições. Nesta abordagem, os usuários podem ser associados a papéis que só constituem um conflito de interesse quando ativados simultaneamente, ou seja, é perfeitamente possível e seguro que um usuário ative mais de um papel do conjunto, desde que estas ativações não sejam simultâneas.

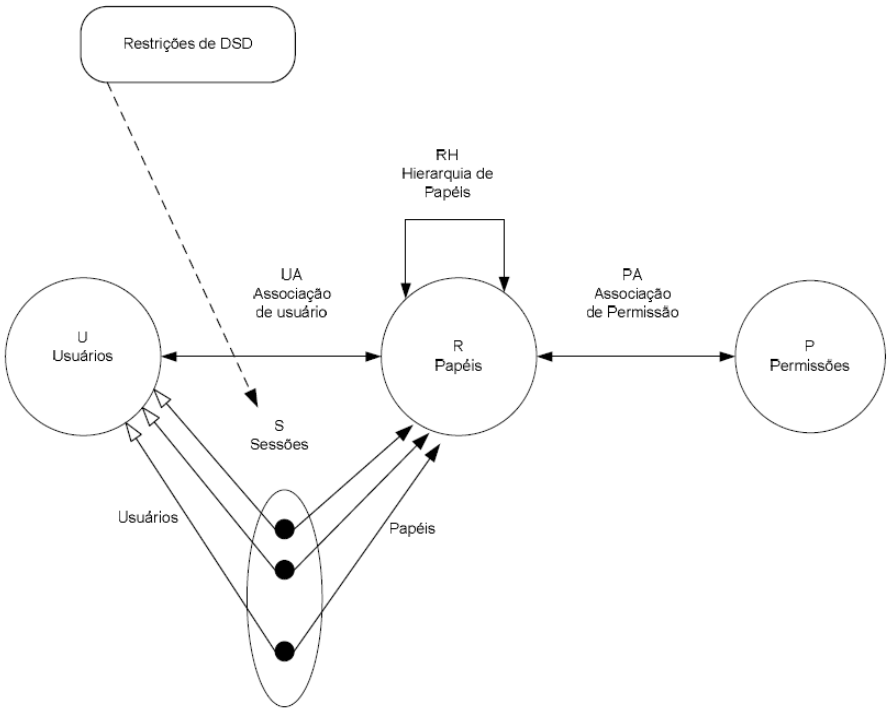

Figura 1.7. RBAC com Restrições – Separação Dinâmcia de Tarefas

A separação dinâmica de tarefas pode ser perfeitamente aplicada a papéis que tenham uma relação de herança, ao contrário do que acontece na separação estática [SANDHU 2000].

4.2.4.3.4. RBAC Simétrico

Um componente essencial para qualquer esquema de gerenciamento de autorização é a manutenção apropriada e precisa das associações permissão-papel. O RBAC Simétrico possui como único requisito o suporte à revisão de associações permissão-papel similar à revisão usuário-papel do RBAC Básico. O desempenho de revisões permissão-papel deve ser comparável ao desempenho de revisões usuário-papel.

A revisão permissão-papel possibilita identificar, de maneira eficiente, as permissões associadas a um papel e, também, quais papéis possuem determinadas permissões. A possibilidade de identificar as associações permissão-papel é um dos principais fatores que diferenciam papéis de grupos.

4.2.4.3.5. Outras características do modelo RBAC-NIST

O modelo unificado RBAC-NIST não abrange todas as características do RBAC. Primeiro, porque não julga apropriado especificar em um modelo detalhes dependentes da aplicação tais como: escalabilidade, permissões negativas, natureza das permissões (granularidade), ativação seletiva de papéis, revogação de papéis. Segundo, porque não existe consenso suficiente da comunidade para justificar a inclusão de algumas características no modelo como, por exemplo, administração e definição de outras restrições além da separação de tarefas.

O modelo RBAC-NIST reconhece apenas restrições de separação de tarefas (tanto estática como dinâmica). O RBAC pode suportar outros tipos de restrições, como pré-condições (determinam as condições que devem ser satisfeitas para que as associações usuário-papel e papel-permissão possam ser efetuadas [SANDHU 1998]), restrições de cardinalidade (determinados papéis podem ter um número máximo de usuários associados [FERRAIOLO 1999]) e obrigações (atributos funcionais que denotam as exigências que um sujeito deve atender antes ou durante um acesso [AHN 1999, SANDHU and PARK 2003]). Embora estes tipos de restrições sejam igualmente importantes, as restrições de cardinalidade e pré-condições não foram incluídas no modelo unificado RBAC-NIST por não existir ainda consenso sobre a sua real utilização.

4.2.4.4. RBAC com contextos

O objetivo do modelo RBAC com contextos, uma extensão do RBAC, 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 foi 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 1.8. 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. Nesse novo modelo, uma sessão é composta apenas por um usuário e um contexto por vez. Para que um usuário ative um novo contexto em uma sessão, é necessário que ele desative o contexto anterior. Dessa forma, o que existe é uma troca de contextos. Ou seja, uma sessão em um determinado momento é formada por apenas um usuário e um contexto.

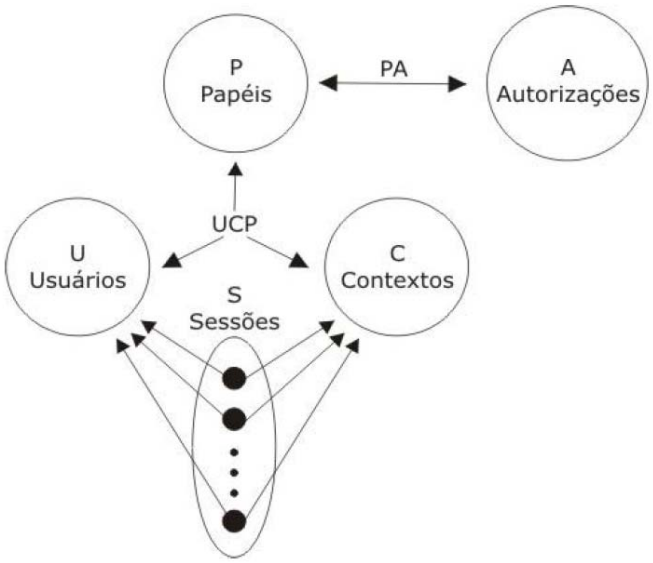

Figura 1.8. Modelo RBAC com Contextos

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. 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. Se, por exemplo, o objeto é o acesso às entradas e saídas na conta, o contexto ajuda na divisão dessas entradas e saídas, de forma que a autorização correspondente fica definida somente para as entradas e saídas relacionadas com aquele contexto, ou seja, daquela filial ou unidade da organização. Isto ajuda na divisão dos objetos não só pelo seu tipo, como no exemplo de entradas e saídas na conta, mas também pelo contexto ao qual está relacionado.

O modelo RBAC com contextos não foi a primeira abordagem na literatura para controle de acesso dependente de contexto. O modelo de autorização contextual para controle de acesso baseado em papéis proposto por Motta e Furuie [MOTTA, FURUIE 2002a] faz com que uma autorização seja positiva ou negativa, de acordo com regras que relacionam informações sobre o contexto em que aquela autorização está sendo solicitada. Esse modelo vem sendo utilizado no MACA, uma ferramenta de autorização e controle de acesso para o prontuário eletrônico de pacientes [MOTTA, FURUIE, 2002b], que se encontra funcionando no Instituto do Coração do Hospital das Clínicas da Faculdade de Medicina da Universidade de São Paulo.

A diferença entre o modelo de autorizações contextuais e o RBAC com contextos está na forma com que o contexto se encaixa no modelo. No sistema de autorizações contextuais o contexto é algo dinâmico e extremamente granular, onde o mesmo é constituído de informações tais como hora, local e qual o paciente. Nesse modelo, o contexto se encaixa na autorização, que será positiva ou negativa, dependendo de regras que utilizam variáveis do contexto.

Já o modelo RBAC com contextos trabalha com contextos fixos e mais amplos, tais como filiais e unidades de uma organização. No modelo proposto, o contexto se encaixa no relacionamento entre usuários e papéis, funcionando como um divisor entre o controle de acesso dos diferentes contextos. Como o contexto é algo bem mais amplo, faz sentido atribuir papéis dentro de cada contexto.

Apesar do objetivo de ambos os modelos ser o de diferenciar autorizações de acesso do mesmo tipo de informação de acordo com o contexto, os ambientes de utilização que tornam os modelos adequados para utilização são diferentes. O modelo de autorizações contextuais trabalha com contextos dinâmicos e granulares, bem como com uma definição dinâmica da autorização, enquanto o modelo RBAC com contextos trabalha com contextos fixos e amplos, bem como com a atribuição de papéis por contexto.

4.2.4.5. Considerações sobre os Modelos

Basicamente os modelos de segurança foram definidos em períodos distintos com tendências e características específicas. No começo dos anos 60, o desenvolvimento desses modelos foi estimulado pelo crescente surgimento dos sistemas de tempo compartilhado. Depois, nos anos 70, seu desenvolvimento foi impulsionado por fins e propósitos militares, e nos anos 80 caracterizou-se pelo enfoque mais comercial. Dos anos 90 em diante, a tônica está sendo em modelos que envolvem os ambientes distribuídos, sem um propósito específico, mas geralmente para suportar políticas de segurança múltiplas. A diversidade de origem e objetivos dos modelos de segurança não permite que eles sejam diretamente comparados entre si. Entretanto, é possível abstrair-se de detalhes específicos de cada modelo e concentrar-se nas suas características gerais, estabelecendo uma base comum a partir da qual é possível tecer uma comparação. A Tabela 1.5. ilustra as principais diferenças entre os modelos de controle de acesso discutidos neste capítulo.

O modelo matriz de acesso é caracterizado pela sua flexibilidade, o que facilita a gerência descentralizada. Como neste modelo são os próprios sujeitos que determinam quais os acessos que outros sujeitos possuem sobre seus objetos, diz-se que a administração da política de segurança do modelo matriz de acesso é descentralizada. Ao contrário, os modelos obrigatórios exigem a presença no sistema de um administrador de segurança, que é assumido como único.

O preço da flexibilidade e da descentralização é a complexidade envolvida no controle à propagação de direitos no sistema. Os modelos obrigatórios geralmente definem um conjunto de regras não contornáveis no sistema, o que diminui as possibilidades de propagação de direitos. Estes modelos são próprios para gerências centralizadas de segurança, representando estruturas pouco flexíveis. Os modelos baseados em papéis são tidos como modelos intermediários entre os discricionários e os obrigatórios. Tais modelos apresentam a estrutura flexível dos primeiros e a

possibilidade de gerência centralizada característica dos obrigatórios. Todos os modelos descritos neste texto podem ser implementados em sistemas distribuídos, com maior ou menor dificuldade [WESTPHALL 2000].

A separação de tarefas é um conceito suportado somente pelo modelo baseado em papéis. O princípio do mínimo privilégio, por sua vez, é suportado tanto pelos modelos baseados em papéis quanto pelos modelos BLP e Biba, através do uso criterioso dos rótulos correntes de segurança. É importante observar, porém, que a granularidade do suporte a mínimo privilégio nos modelos baseados em papéis é bem mais fina do que nos modelos baseados em rótulos. Os modelos baseados em papéis são os únicos que permitem 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. O último critério considerado na comparação da tabela é a facilidade ou viabilidade de implementação.

O modelo matriz de acesso é o mais simples de ser implementado dentre todos. Modelos baseados em papéis vêm em segundo lugar em termos de simplicidade, não trazendo grandes complicações à implementação. Os modelos baseados em rótulos já foram implementados em vários sistemas, mas requerem um razoável esforço para identificar todas as estruturas de um sistema que precisam ser rotuladas para evitar as violações de confiabilidade (BLP) ou integridade (Biba).

Tabela 1.5. Comparação entre os modelos de segurança

Características	Modelo			
	Matriz de Acesso	BLP	Biba	RBAC
Flexibilidade	Sim	Não	Não	Sim
Política de Controle de Acesso Centralizada	Não	Sim	Sim	Sim
Administração da Política Centralizada	Não	Sim	Sim	Sim ou Não
Separação de tarefas	Não	Não	Não	Sim
Suporte a mínimo Privilégio	Não	Sim	Sim	Sim
Suporte à Hierarquia Organizacional	Não	Não	Não	Sim
Facilidade de Implementação	Fácil	Médio	Médio	Fácil

4.2.5. Mecanismos de Segurança

Os mecanismos de segurança são responsáveis pela implementação das políticas de segurança específicas, expressas pelos modelos de segurança. Como exemplo, podemos dizer que uma política de segurança pode exigir que todos os usuários de um sistema sejam identificados univocamente para fins de contabilidade, e os mecanismos para implantar esta política incluem o uso de senhas, de cartões magnéticos e de dispositivos de reconhecimento de impressões digitais. Para viabilizar a implantação de tais políticas, os mecanismos são construídos a partir de controles de acesso e controles criptográficos.

No que se refere a controle de acesso, podemos dizer que virtualmente todos os sistemas computacionais podem ser descritos em termos de sujeitos acessando objetos. O controle de acesso é, portanto, a mediação das requisições de acesso a objetos iniciadas pelos sujeitos. Um monitor de referência é um modelo conceitual do subsistema responsável pelo controle de acesso. Ou seja, é a entidade que recebe todas

as requisições de acesso dos sujeitos e autoriza ou nega o acesso de acordo com a política de segurança implantada.

O monitor de referência, tendo como função intermediar todas as requisições de acesso aos objetos de um sistema, deve ter algumas propriedades, tais como: deve ser inviolável, incontornável (sempre invocado) e pequeno o suficiente para permitir a verificação de sua correção. A noção de núcleo de segurança foi definida em [LANDWEHR 1983] como o conjunto de recursos de hardware e software que permitem a realização de um monitor de referências.

A implementação do monitor de referência é realizada através de mecanismos de controle, que podem ser discricionários (DAC – *Discretionary Access Control*), obrigatórios (MAC – *Mandatory Access Control*) ou baseados em papéis (*RBAC - Role-Based Access* Control).

4.2.5.1. Controles de Acesso Discricionário

Os mecanismos de controle de acesso discricionário (DAC) implementam políticas discricionárias, permitido ao usuário atribuir direitos de acesso sobre seus recursos computacionais de acordo com a sua necessidade. Tais mecanismos baseiam-se na idéia de que o proprietário da informação deve determinar quem terá acesso a essa informação. O controle discricionário permite que os dados sejam livremente copiados de objeto para objeto, de modo que, mesmo que o acesso aos dados originais seja negado, pode-se obter acesso a uma cópia. Porém, se o usuário não atribuir corretamente estes direitos, ou mesmo se o fizer permitindo acesso de cópia a outros sujeitos, a disseminação de suas informações no sistema não pode ser controlada. O controle de acesso discricionário não impõe nenhuma restrição à disseminação de direitos e à própria evolução da matriz de acesso. Apesar dessa limintação, na prática, devido talvez à facilidade de implementação, o controle de acesso discricionário é largamente utilizado nos sistemas atuais.

4.2.5.2. Controle de Acesso Obrigatório

Os mecanismos de controle de acesso obrigatório (MAC) implementam políticas obrigatórias, as regras de controle de acesso são impostas por uma autoridade central. Baseiam-se em uma administração centralizada de segurança, a qual dita regras incontornáveis de acesso à informação. Estes mecanismos, como descrito anteriormente, implementam políticas multinível. Dos vários relatos de implementação de mecanismos MAC, observa-se que os mesmos são bem mais difíceis de viabilizar que os de DAC, devido à rigidez de suas regras e às limitações de seus modelos, além de outras dificuldades de caráter computacional [LANDWEHR 1984] que geram limitações à implementação deste modelo. Deste modo os modelos conceituais necessitam de relaxamento de suas regras para poderem ser viabilizados em mecanismos comerciais.

4.2.5.3. Controle de Acesso Baseado em Papéis

Para os mecanismos que implementam RBAC a identidade no sistema é o papel, uma vez que estes encapsulam as políticas na forma de permissões. Tais mecanismos têm como base que os direitos de acesso sejam atribuídos a papéis e não a usuários, assim como acontece no DAC, já que os usuários obtêm estes direitos em virtude de terem papéis a si atribuídos. Por ser independente das políticas, o RBAC é facilmente

ajustável a mudanças no ambiente e é largamente utilizado, porque não é tão flexível quando o DAC e nem tão rígido quanto o MAC. Por mais que os modelos RBAC ainda estejam em desenvolvimento, existem vários relatos de implementações do mesmo, um deste pode ser encontrado em [GLENN 1999].

4.2.5.4. Controles Adicionais de Segurança

Os mecanismos de segurança têm como seu principal objetivo implantar políticas de segurança, utilizando para isso ações, técnicas ou dispositivos. Mas indo um pouco além, existem ainda outros controles (internos) que não atuam diretamente nas requisições de acesso, mas que devem necessariamente estar presentes nos sistemas. Dentre eles podemos citar:

1. A auditoria de vestígio, ligada à geração periódica de registros de eventos associados à segurança, coletados para uso potencial em detecção de intrusão e/ou auditoria de segurança.
2. A auditoria de segurança, inspeção independente (por terceiros) dos procedimentos e registros do sistema com intuito de verificar a adequação da política de segurança e as possíveis violações do sistema.
3. A detecção de intrusão, que usa os mesmos registros das auditorias em métodos automatizados de análise em tempo real, envolvendo muitas vezes uma seqüência de eventos relacionados ou não, com o intuito de identificar atividades anormais no sistema.

Mecanismos que façam uso de técnicas de *backups*, replicações e que permitam recuperar o sistema em situações onde as violações não puderam ser evitadas completam os controles adicionais.

Outros controles (externos) necessitam ser adicionados aos internos já comentados. Por exemplo, é necessário que se leve ao conhecimento de cada usuário suas atribuições e responsabilidades, para que esse saiba o que está autorizado a fazer. Se este não estiver treinado e convencido da importância da segurança no ambiente computacional as demais medidas podem se tornar sem eficácia [AMOROSO 94].

4.3. 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 noticias sobre vulnerabilidades e falhas de segurança em sistemas de software [FINK et al 2004]. Á medida que os arcabouços aparecem, eles se tornam rapidamente ultrapassados por diferentes motivos. Na maioria das vezes porque eles não oferecem um modelo arquitetural suficientemente flexível para acompanhar as necessidades do negócio, o qual está em desenvolvimento ou porque eles limitam o escopo da arquitetura de que o negócio necessita. Devido à 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.

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 e integrar soluções de segurança nas fases de análise e projeto de sistemas de software 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. Neste contexto, uma abordagem orientada a modelo pode auxiliar o processo de desenvolvimento e de criação de arcabouços de segurança, promovendo uma implementação controlada dos padrões de projeto e desta forma, garantindo aderência às diretrizes arquiteturais da organização e às questões de segurança.

Além disso, uma abordagem orientada a modelo como MDA (*Model Driven Architecture*) 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 serão levados em consideração durante todas as fases do processo de desenvolvimento do sistema.

4.3.1. 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.

O uso de padrões é uma boa ferramenta para ajudar arquitetos e projetistas a construir sistemas seguros, pois enquanto muitos problemas de segurança são novos ou complicados, uma grande parte de outros problemas são bem conhecidos e possuem soluções bem estabelecidas.

4.3.1.1. 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, freqüentemente 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

4.3.1.1.1. 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, consequentemente, 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

4.3.1.2. 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. Os nomes dos participantes nos padrões de projeto são geralmente muito abstratos para aparecer diretamente na aplicação. Contudo, deve-se incorporar o nome do participante no nome que aparece na aplicação. Isto faz com que o padrão se torne mais explicito na implementação. Resumindo, escolha nomes para os participantes que sejam significativos no contexto da aplicação.
5. 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.
6. Defina nomes para as operações de forma específica à aplicação. Utilize as responsabilidades e colaborações associadas a cada operação como um guia.
7. 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.

A maioria das pessoas utiliza padrões de projeto quando percebe um problema com seu projeto - alguma coisa que deveria ser fácil, mas não é - ou com sua implementação - como desempenho. Ao examinar um código, é necessário verificar quais são seus problemas e quais são seus comprometimentos, o que gostaria de realizar que agora esta sendo difícil. A partir daí, é necessário procurar uma referência de padrão de que corresponda aos problemas que se deseja resolver.

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.

4.3.1.3. 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 et al 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 and 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, etc.

Darrel M. Kienzle and Matthew C. Elder construíram um repositório de padrões de segurança com vinte e seis padrões e três mini-padroes. 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 et al 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.

4.3.2. 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 auxiliando a criação de arcabouços de segurança flexíveis e garantindo que os requisitos de segurança sejam levados em consideração durante todas as fases do processo de desenvolvimento do sistema. Com isso, as falhas de segurança podem ser identificadas 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.

4.3.2.1. 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. A MDA (*Model Driven Archictecture*) é um exemplo. Ela é uma abordagem de desenvolvimento de software definido pela OMG (*Object Management Group*) que se baseia na idéia da separação da especificação das funcionalidades de um sistema dos detalhes de implementação.

Os três objetivos principais da MDA são prover 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.

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 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.

As transformações de modelos consistem em converter um modelo em outro através de mapeamentos. A Figura 1.9. exemplifica essas transformações e os conceitos centrais da MDA. Um PIM é transformado em um ou mais PSM através de regras de transformação e o PSM por sua vez é transformado em código de uma determinada linguagem de programação.

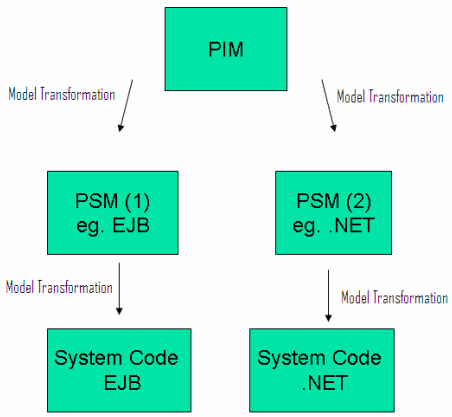

Figura 1.9. Transformações em MDA

Para que o processo da MDA funcione, é necessário que os seguintes requisitos sejam preenchidos:

- Construção de modelos em alto nível que contenham informações detalhadas sobre o domínio em questão.
- Linguagens bem definidas e padronizadas para escrever modelos em alto nível.
- Definições de como o PIM é transformado para um PSM específico, de modo a tal transformação poder ser automaticamente executada. Essas definições estarão em domínio público, padronizadas, mas poderão ser *home-made*, ou seja, feitas pela própria aplicação.
- Uma linguagem para escrever essas definições. Esta linguagem deve ser usada pelas ferramentas de transformação, além disso, deve ser uma linguagem com algum grau de formalismo.
- Ferramentas que executam as definições das transformações. Preferencialmente estas ferramentas devem oferecer a flexibilidade para mudar os passos da transformação de acordo com as necessidades específicas de cada projeto.
- Ferramentas que executam a transformação de PSM para código.

Os principais benefícios da MDA são:

1. Produtividade

Na MDA, o foco maior é dado à construção do modelo PIM. Com esse enfoque a produtividade torna-se maior, pois os desenvolvedores não precisam mais gastar tempo com o projeto específico de acordo com a plataforma escolhida. Esses detalhes técnicos específicos de plataforma são acrescentados ao PIM pela transformação PIM $\Rightarrow$ PSM. Além disso, a maior parte do código também é gerado automaticamente pela transformação PSM $\Rightarrow$ Código. Claro que o projeto e a especificação dessas transformações específicas são tarefas custosas e complexas, portanto influenciam no cronograma do Projeto. Mas uma vez essa transformação definida, ela poderá ser aplicada diretamente aos demais projetos, resultando assim num ganho de produtividade.

2. Portabilidade

A portabilidade se encontra no modelo PIM. O modelo PIM pode ser transformado em vários PSMs de diversas plataformas, ou seja, toda a informação especificada no PIM é completamente portável.

3. Interoperabilidade

A interoperabilidade a que a MDA se refere é a comunicação entre os PSMs.

4. Manutenção e Documentação

Na MDA, com a ajuda das ferramentas MDA, alterações feitas no PSM podem ser refletidas no seu PIM de origem, mantendo PIM, PSM e documentação consistentes até o término do projeto, o que não ocorre entre os modelos durante um processo tradicional de desenvolvimento de software.

4.3.2.2. 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 1.10. 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.

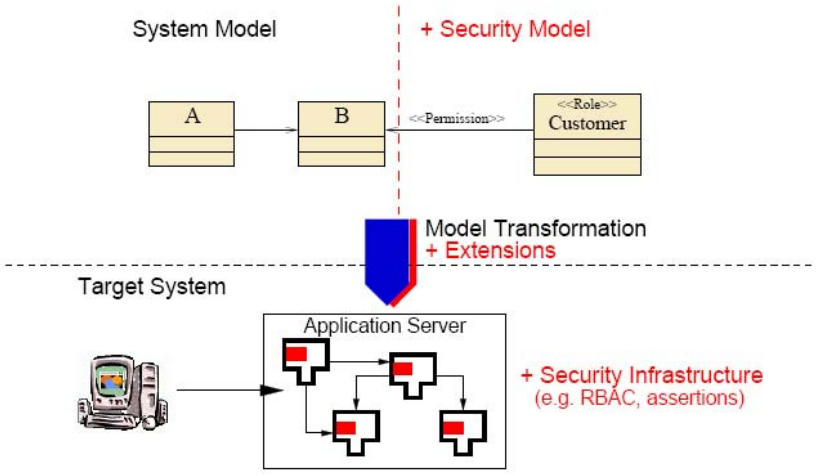

Figura 1.10. Segurança dirigida a modelo

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 1.11. 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 deletar contas

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

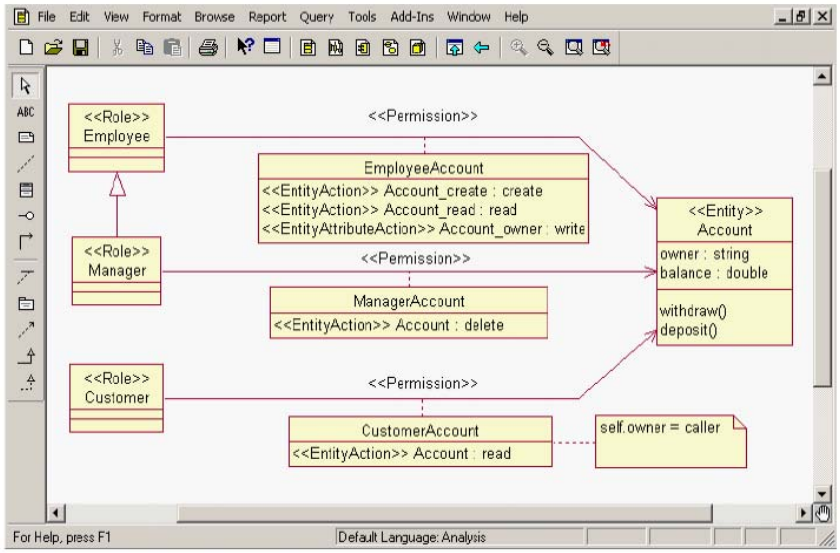

Figura 1.11. Modelando um cenário de um banco

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 especifico 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 especifica de alguma plataforma.

O modelo de controle de acesso utilizado é o VBAC (*View-Based Acces 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 principals, 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 1.12.

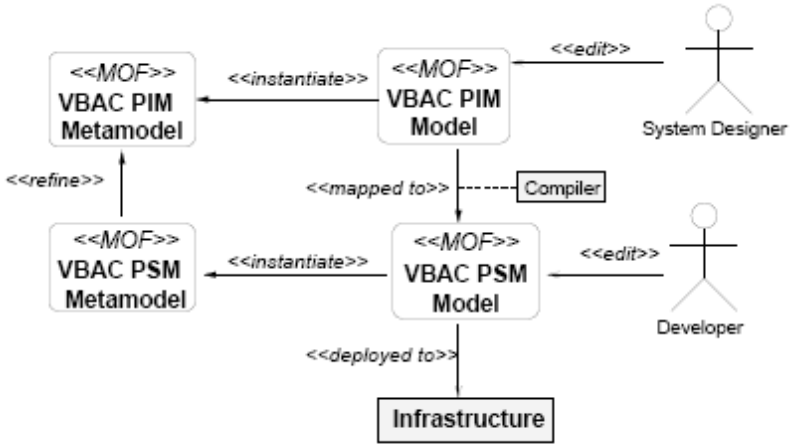

Figura 1.12. Modelos MDA-VBAC

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 independe 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.

Todas as abordagens descritas até o momento, permitem a definição de políticas de controle de acesso utilizando perfis UML [UML 2007] (estereótipos, valores etiquetados) e OCL [OCL 2003] para especificar restrições, onde esses modelos são modelos de permissões. Ou seja, o PIM é modelado com requisitos de segurança. Porém, o fator humano pode interferir na robustez de segurança em qualquer ponto onde intervenção manual é requerida. Isto é, a partir do momento onde a segurança do sistema é ativada somente através da modelagem feita manualmente, ela se torna suscetível a erros humanos, prejudicando a qualidade da implementação dos requisitos de segurança. Ou seja, o PIM sendo marcado com requisitos de segurança pode provocar uma sobrecarga de atribuições ao desenvolvedor, expondo à falhas de segurança e poluindo o modelo de negócio e consequentemente, dificultando sua compreensão.

Nesse contexto, outra abordagem que utiliza MDA para desenvolver arcabouços de segurança é [FIORIO 2007]. Nessa abordagem foi criado um arcabouço de segurança que é dividido em duas partes, uma responsável pelas regras de segurança, chamada de arcabouço de regras de segurança, e outro responsável pela administração de segurança específica de cada aplicação, chamada de arcabouço administrativo de segurança.

O arcabouço de regras de segurança é responsável pela autenticação e autorização dos artefatos da aplicação. Ele simplifica e flexibiliza a modelagem dos requisitos de segurança seguindo o padrão arquitetural MVC [MVC 2007], provendo proteção por níveis (camadas) de vista, controle e dados. A segurança é aplicada implicitamente no sistema através de transformação entre modelos e codificação automática. O desenvolvedor continua modelando seu sistema normalmente, sem se preocupar com a segurança e mesmo assim, a segurança será gerada e o sistema estará protegido.

Já o arcabouço administrativo de segurança representa o modelo específico de segurança da aplicação, que depende da política de controle de acesso da organização. O arcabouço criado estende o modelo de controle de acesso RBAC com contextos, porém o padrão deixa em aberto a interpretação de usuário, papel, autorização e contexto para ser especificado em modelos mais detalhados. Porém, visando facilitar o desenvolvimento da aplicação, um arcabouço administrativo de segurança genérico é gerado automaticamente, de forma que só é necessário que a aplicação tenha que criar seu próprio modelo caso este não esteja de acordo com a política de segurança da organização.

O processo de desenvolvimento do arcabouço de segurança criado pode ser visto na Figura 1.13. O arquiteto de negócio desenvolve a arquitetura MDA de negócio através de extensões dos perfis UML e criando dessa forma, um engenho de transformação convencional. O arcabouço criado é transparente ao desenvolvedor de negócio, pois este continua modelando o PIM de negócio como de costume e utilizando o engenho de transformação convencional criado 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. Eles apenas precisam indicar no PIM as entidades do RBAC com contextos, ou seja, usuário, papel, autorização e contexto.

Por sua vez, o arquiteto de segurança, desenvolve arquitetura de segurança criando um engenho de transformação de segurança seguro e um gerenciável. O arcabouço de segurança é gerado através do engenho de transformação de segurança seguro. Já o gerenciável é responsável pela geração padrão da aplicação de segurança do sistema. Esta aplicação é uma aplicação administrativa de segurança. Ela é responsável pelo cadastro de perfis, usuários, permissões, etc. Porém, cada sistema possui sua própria política de segurança, de forma que a aplicação padrão possa não se encaixar no modelo de segurança do sistema. Nesse caso, o engenheiro de segurança deve desenvolver a aplicação de segurança específica do sistema e fazer a ligação dessa aplicação com o arcabouço de segurança gerado.

A implementação de segurança é gerada automaticamente através do modelo. O fato de ser gerado automaticamente provê uma liberdade ao arquiteto de segurança à medida que ele pode alterar o que é gerado, a todo 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, criar novas regras de transformação específicas da plataforma escolhida. Além disso, o fato de ser gerado automaticamente e sem intervenção humana garante a qualidade do código e conseqüentemente do sistema.

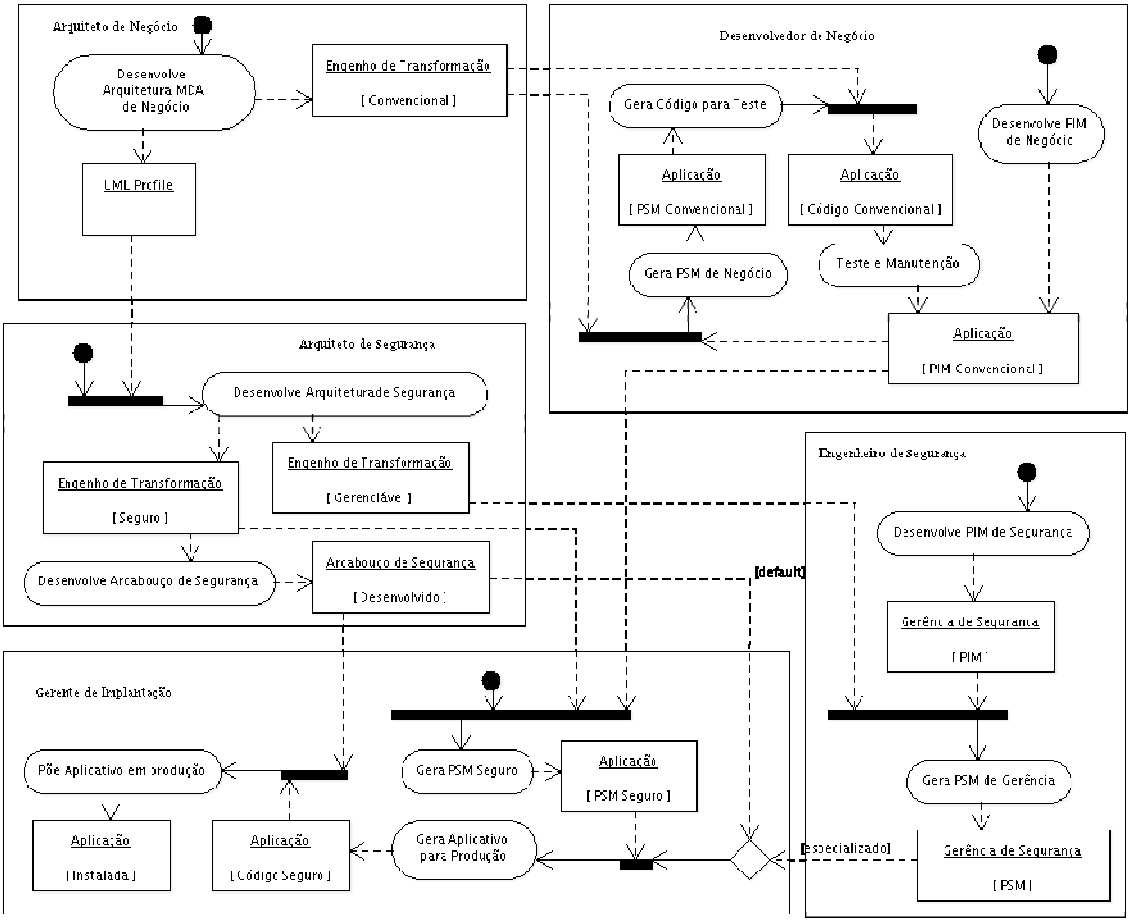

Figura 1.13. Projeto do arcabouço

Após a implementação da ligação entre o arcabouço de segurança e a aplicação de segurança é necessário que haja um gerente de segurança que faça a manipulação 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 artefato. 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.

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 arcabouço de segurança, gera o PSM 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.

Nesse contexto, podemos perceber que a grande mudança ocorre quando a segurança está ativada, pois nesse momento, na transformação do PIM para o PSM, os novos requisitos de segurança requerem que um novo PSM seja gerado incluindo os artefatos de segurança. Consequentemente, a transformação de PSM para código também é alterada devido ao fato da criação de toda infra-estrutura de segurança do sistema. A Figura 1.14. exemplifica as transformações em alto nível de MDA com as extensões de segurança.

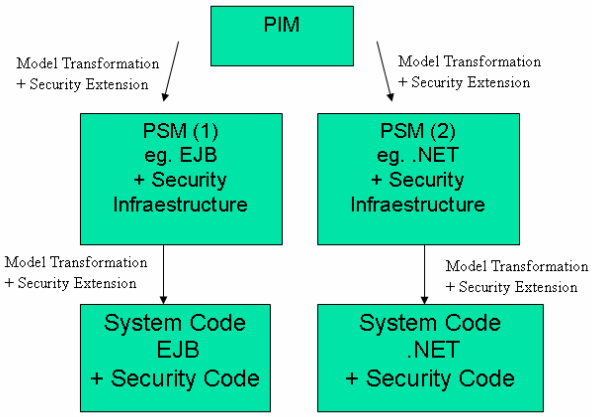

Figura 1.14. Transformações em MDA com extensões de segurança

O arcabouço foi utilizado e testado em um sistema militar brasileiro que possui uma política de segurança elevada, permitindo inúmeras vantagens em sua utilização pudessem ser percebidas.

Ele contribui na integração de segurança durante o desenvolvimento do sistema simplificando o desenvolvimento de sistema, 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 da segurança poder ser gerada automaticamente.

4.4. Conclusão

Os Modelos de Segurança constituem uma importante ferramenta para a definição de políticas de segurança para qualquer ambiente computacional. Ao longo dos anos, diversos modelos de segurança foram propostos, baseando-se nas mais variadas premissas e visando atender a diferentes aspectos de segurança.

Neste capítulo, fizemos uma revisão dos trabalhos mais recentes na área de modelos de segurança computacional, destacando modelos considerados clássicos na área e que ainda hoje são amplamente utilizados nos mais variados ambientes. Vimos a essência destes modelos, analisando criticamente a sua validade e adequação, bem como alguns aspectos de sua implementação.

Podemos perceber que os modelos baseados em papéis possuem algumas vantagens sobre os outros modelos, dentre as quais podemos destacar 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. Devido às suas características os sistemas baseados em papéis representam uma importante tendência na área de controle de acesso em sistemas computacionais.

Apesar de grande parte dos sistemas de informação depender de requisitos de segurança de alta qualidade, constantemente há noticias sobre vulnerabilidades e falhas de segurança. Isto se deve a 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. Outro fato que contribui para esse problema é que os arcabouços de segurança existentes não oferecem um modelo arquitetural suficientemente flexível para acompanhar as necessidades do negócio em desenvolvimento ou 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.

Vimos que são necessárias novas formas de desenvolver software seguro, baseadas não somente na aplicação das teorias existentes mas também na adoção de um processo de desenvolvimento que considere os requisitos de segurança como parte integral do projeto de construção de software. 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 melhor 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 seguro.

Presentemente, existe uma grande lacuna entre as soluções teóricas e o que é de fato implementado na área de segurança. Portanto, 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.

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 et al 2004] publicado pelo Opengroup [OpenGroup 2007]. Já em [Fernandez and Pan 2001] são discutidos três padrões que correspondem a modelos de segurança: Autorização, Controle de Acesso Baseado em Papéis e Segurança Multi-nível; em [KIENZLE, ELDER 2002] encontramos 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]. Além desses trabalhos, vários livros foram publicados, dentre eles podemos citar Security Engineering With Patterns: Origins, Theoretical Models, and New Applications [Schumacher 2003], Security Patterns: Integrating Security and

Systems Engineering [Schumacher et al 2005] e Core Security Patterns: Best Practices and Strategies for J2EE, Web Services, and Identity Management [Steel et al 2005].

Vimos que para obter os benefícios da utilização de padrões de projeto, é necessário saber quando e como utilizá-los a partir de suas definições e que uma abordagem orientada a modelo (MDA) pode auxiliar o processo de desenvolvimento e criação de arcabouços de segurança. 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.

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, dessa forma auxiliando a criação de arcabouços de segurança flexíveis e garantindo que os requisitos de segurança sejam levados em consideração durante todas as fases do processo de desenvolvimento do sistema. Com isso, as falhas de segurança podem ser identificadas 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.

Além disso, a utilização de MDA contribui na integração de segurança durante a construção do sistema simplificando o seu desenvolvimento, aumentando a produtividade e a qualidade das especificações de segurança e com isso, aumentando consideravelmente a qualidade e a manutenibilidade dos sistemas construídos além de facilitar a evolução da solução de segurança.

Vimos que algumas abordagens já integram a segurança a MDA. Grande parte dessas abordagens permitem a definição de políticas de controle de acesso através da marcação do modelo utilizando perfis UML [UML 2007] (estereótipos, valores etiquetados) e OCL [OCL 2003] para especificar restrições. Ou seja, o PIM de negócio é marcado com requisitos de segurança.

Referências

AHN, G.-J.; SANDHU, R. (1999) “The RSL99 Language for Role-Based Separation of Duty Constraints”, In Proceedings of 4th ACM Workshop on Role-Based Access Control, Fairfax, VA, Oct, p. 43-54.

ALEXANDER, C. (1978) “A Pattern Language”, Oxford Press, Oxford, R. Unido.

ALEXANDER, C. (1979) “A Timeless Way of Building”, Oxford Press, Oxford, R. Unido.

AMOROSO, E. G. (1994) “Fundamentals of Computer Security Technology”. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1994, ISBN: 0-13-108929-3.

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.

BELL, D. E.; LA PADULA, L. J. (1973) “Secure Computer Systems : Mathematical Foundations”, MTR-2547 Vol. I, The MITRE Corporation, Bedford, Massachusetts, Mar.

BIBA, K. J. (1977) “Integrity Considerations for Secure Computer Systems”, Technical Report ESD-TR-76-372, USAF Electronic Systems Division, Hanscom Air Force Base, Bedford, Massachusetts, Apr.

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.

CLARK, D.; WILSON, D. (1987) “A comparasion of commercial and military computer security policies”, Proceedings of the IEEE Computer Society Simposium of Research in Security and Privacy, Los Alamitos, Calif., p. 184-194.

CHRISTOPHER STEEL, RAMESH NAGAPPAN, RAY LAI. (2005) “Core Security Patterns: Best Practices and Strategies for J2EE, Web Services, and Identity Management,” Prentice Hall.

DENNING, D. (1976) “A Lattice Model of Secure Information Flow”, Communication of ACM, Vol. 19, Nº. 5, May.

DENNING, D. (1982) “Criptography and data security”, Addison-Wesley.

ECLIPSE (2007) “Eclipse”, disponível em http://www.eclipse.org/, acesso em julho 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.

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.

FERRAIOLO, 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 Standars 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.

FIORIO, M.; EMMANOEL, C. O; PIRES, P. F. (2007) “Um arcabouço de segurança baseado em transformações de modelos em MDA”, Relatório técnico, Núcleo de Computação Eletrônica, UFRJ.

GAMMA, E.; HELM, R.; JOHNSON, H.; VLISSIDES, J.; WESLEY, A. (1995) "Design Patterns: Elements of Reusable Object-Oriented Software", ISBN: 0-201-63361-2.

GLENN, F. (1999) "RBAC in UNIX Administration", Proceedings of 4th ACM Workshop on Role-Based Access Control, Fairfax, VA, Oct., p. 95-101.

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 Reseach in Security and Privacy.

ISO/IEC 9126-1. (2001) "Software Engineering — Product Quality — Part 1: Quality Model", International Organization for Standardization, ISO, Geneva, Switzerland.

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.

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.; ELDER, M. C.; TYREE, D. S.; EDWARDS-HEWITT, J. (2002) "Security Patterns Repository Version 1.0", disponível em http://www.modsecurity.org/archive/securitypatterns/dmdj_repository.pdf, acesso em junho de 2007.

LAMPSON, B.W. (1971) "Protection", Proc. 5th Princeton Conf. on Information Sciences and Systems, Princeton, p. 437.

LANDWEHR, C. E. (1981) "Formal models for computer security", Computing surveys.

LANDWEHR, C. E. (1983) "Best available technologies for computer security", IEEE Computer Vol. 16, No. 7, Jul, p. 86-100.

LANDWEHR, C.E.; HEITMEYER, C.L.; MCLEAN, J. (1984) "A security Model for Military Message Systems", ACM Transactions on Computer Systems, Vol. 2, No 3, Aug, p. 198-222.

LANDWEHR, C. E. (2001) "*Computer security*", IJIS.

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

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.

MCLEAN, J. (1990) "The Specification and Modeling of Computer Security", IEEE Computer, Vol. 23, No 1.

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

MOTTA G. H. M. B.; FURUIE S. S. (2002a) “Um Modelo de Autorização Contextual para Controle de Acesso Baseado em Papéis”, WSeg.

MOTTA G. H. M. B.; FURUIE S. S. (2002b) “MACA: Uma Ferramenta de Autorização e Controle de Acesso para o Prontuário Eletrônico de Pacientes”, VIII CBIS.

OASIS (2007) “OASIS: Avancing 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) 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, R.; COYNE, E.J.; FEINSTEIN, H.L.; YOUMAN, C.E. (1996) “Role-Based Access Control Models”, IEEE Computer, Vol. 29, Nº. 2, Feb, p. 38-47.

SANDHU, R. (1998) “Role-Based Access Control”, In Advances in Computers, Volume 46. Academic Press.

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 Acess Control, Berlin, Germany.

SANDHU, R.; PARK, J. (2003) “Usage Control: A vision for Next Generation Access Control”, In The Second International Workshop Mathematical Methods, Models and Architectures for Computer Networks Security (MMM-ACNS), St. Petersburg, Russia, Sep.

SCHUMACHER, M. (2003) “Security Engineering With Patterns: Origins, Theoretical Models, and New Applications”, Springer Berlin, Heidelber.

SCHUMACHER, M.; FERNANDEZ-BUGLIONI, E.; HYBERTSON, D.; BUSCHMANN, F.; SOMMERLAD, P. (2005) “Security patterns. Integrating security and systems engineering”, John Wiley & Sons.

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.

TSUKOMO, A et al. (1997) “Qualidade de Software: Visões de Produto e Processo de Software”, II ERI – SBC, Piracicaba, São Paulo, Brasil.

UML (2007) “Unified Modeling Language”, disponível em http://www.uml.org/, acesso em julho de 2007.

WESTPHALL, C.M. (2000) “Um esquema de autorização para a segurança em sistemas distribuídos de larga escala”, Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Santa Catarina, Tese de doutorado, Florianópolis – Santa Catarina.

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