RAFAEL RODRIGUES OBELHEIRO

MODELOS DE SEGURANÇA BASEADOS EM PAPÉIS PARA SISTEMAS DE LARGA ESCALA: A PROPOSTA RBAC-JACOWEB

FLORIANÓPOLIS
2001

UNIVERSIDADE FEDERAL DE SANTA CATARINA
CURSO DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

MODELOS DE SEGURANÇA BASEADOS EM PAPÉIS PARA SISTEMAS DE LARGA ESCALA: A PROPOSTA RBAC-JACOWEB

Dissertação submetida à
Universidade Federal de Santa Catarina
como parte dos requisitos para a
obtenção do grau de Mestre em Engenharia Elétrica.

RAFAEL RODRIGUES OBELHEIRO

Florianópolis, fevereiro de 2001.

MODELOS DE SEGURANÇA BASEADOS EM PAPÉIS PARA SISTEMAS DE LARGA ESCALA: A PROPOSTA RBAC-JACOWEB

Rafael Rodrigues Obelheiro

'Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Engenharia Elétrica, Área de Concentração em *Controle, Automação e Informática Industrial*, e aprovada em sua forma final pelo Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Santa Catarina.'

Prof. Joni da Silva Fraga, Dr.
Orientador

Prof. Edson Roberto De Pieri, Dr.
Coordenador do Programa de Pós-Graduação em Engenharia Elétrica

Banca Examinadora:

Prof. Joni da Silva Fraga, Dr.
Presidente

Prof. Jean-Marie Farines, Dr.

Prof. Rômulo Silva de Oliveira, Dr.

Carla Merkle Westphall
Carla Merkle Westphall, Dra.

À Tati e à minha família, os meus maiores incentivadores...

AGRADECIMENTOS

Ao meu orientador, Joni Fraga, pela orientação, pelo conhecimento, pelos constantes questionamentos, pelas críticas (muitas merecidas e outras nem tanto) e, especialmente, pela amizade e confiança depositada.

Aos demais professores do LCMI, especialmente a Jean-Marie Farines, Rômulo Oliveira e Edson de Pieri, que me receberam bem e estabeleceram um vínculo que ultrapassa a mera cordialidade.

Aos amigos da equipe do projeto JACOWEB, que contribuíram significativamente para a evolução do trabalho: Carla Westphall, que forneceu boa parte das referências bibliográficas e participou ativamente da elaboração da proposta RBAC-JACOWEB, discutindo idéias, apresentando críticas e propondo melhorias; Michelle Wangham, que foi fundamental na compreensão do CORBAsec e de aspectos da implementação; Ricardo Padilha, Luciana Souza, Lau Lung e Juliano Tonon, que também auxiliaram na implementação do protótipo. Aos demais membros do grupo de segurança, Altair Santin e Eduardo Machado da Silva, pelas profícuas discussões sobre os mais diversos aspectos da segurança. A Rafael Chaves, que, talvez até mesmo sem saber, deu uma contribuição importante no direcionamento do trabalho.

Aos inúmeros amigos que transformaram o LCMI em uma segunda família: Eder Gonçalves, Augusto Loureiro (Baiano), Luciano Rottava, Fabiano Bachmann, Hélio Loureiro, Carlos Brandão, Alexandre Moraes (Sobral), Cristian Koliver, Isabel Tonin, Karina Barbosa, Cris Paim, Jerusa Marchi, Carlos Ogawa, Renato Oliveira, Terezinha Faria, Fred Paes, Gilberto Wolff, César Motejunas, César Torrico, Karen Farfán, Fernando Passold, Hallthmann dos Reis, Felipe Beck, Frank Siqueira, Antonio Carrilho e Ricardo Martins. Se alguém foi esquecido, fica registrado o *mea culpa*.

Ao Vinícius, pelo ano de agradável convivência.

À minha família, pelo carinho e apoio financeiro e sobretudo afetivo.

À Tati, pela paciência, dedicação, carinho e compreensão.

A Deus, que colocou estas pessoas no meu caminho.

Resumo da Dissertação apresentada à UFSC como parte dos requisitos necessários para obtenção do grau de Mestre em Engenharia Elétrica.

MODELOS DE SEGURANÇA BASEADOS EM PAPÉIS PARA SISTEMAS DE LARGA ESCALA A PROPOSTA RBAC-JACOWEB

Rafael Rodrigues Obelheiro

Fevereiro/2001

Orientador: Prof. Joni da Silva Fraga, Dr.
Área de Concentração: Controle, Automação e Informática Industrial
Palavras-chave: Segurança computacional, modelos de segurança, RBAC
Número de Páginas: xiii + 89

Com a magnitude e a complexidade de aplicações distribuídas interconectadas através da Internet e/ou de grandes redes corporativas, o projeto de esquemas de controle de acesso eficazes para estes sistemas torna-se um grande desafio. A política de segurança utilizada na maioria dos sistemas de informação atuais, baseada em um modelo discricionário de segurança revela-se frágil diante das ameaças que surgem em ambientes distribuídos de larga escala. Nesta dissertação, é feito um estudo de diversos modelos não discricionários de segurança, com ênfase na sua aplicabilidade em sistemas de larga escala. É desenvolvido, então, um esquema de autorização baseado em papéis que utiliza os modelos de segurança CORBA, Java e Web. Nós introduzimos uma nova abordagem que permite a ativação automática de papéis pelos componentes de segurança do *middleware*, trazendo o controle de acesso baseado em papéis para aplicações não cientes da segurança.

Abstract of Dissertation presented to UFSC as a partial fulfillment of the requirements for the degree of Master in Electrical Engineering.

ROLE-BASED SECURITY MODELS FOR LARGE-SCALE SYSTEMS: THE RBAC-JACOWEB PROPOSAL

Rafael Rodrigues Obelheiro

February/2001

Advisor: Prof. Joni da Silva Fraga, Dr.
Area of Concentration: Control, Automation and Industrial Computing
Key words: Computer security, security models, RBAC
Number of Pages: xiii + 89

With the magnitude and complexity of distributed applications interconnected by the Internet and/or large enterprise networks, designing efficient access control schemes for these systems becomes a major challenge. The most used security policy for current information systems, which is based on a discretionary security model, is too fragile to contain the threats that exist in large-scale distributed environments. In this dissertation, we survey several non-discretionary security models focusing on their applicability in large-scale systems. We then develop a role-based authorization scheme using the CORBA, Java and Web security models. We introduce a novel approach that allows automatic role activation by the security components of the middleware, bringing role-based access control to security-unaware applications.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Motivação e Objetivos

Hoje em dia, as organizações dependem cada vez mais de seus sistemas de informação. A disseminação de redes de larga escala como a Internet e as grandes redes corporativas permitem que estes sistemas sejam construídos segundo arquiteturas distribuídas, cuja magnitude e complexidade tornam a manutenção de sua segurança um grande desafio.

A maior acessibilidade da informação proporcionada pelos sistemas distribuídos de larga escala traz consigo grandes benefícios do ponto de vista operacional, ao mesmo tempo em que aumenta as oportunidades de violação da segurança destas informações.

Um aspecto muito importante e que freqüentemente é negligenciado na construção de sistemas de informação é a política de controle de acesso ou autorização, que determina quais os acessos às informaçãoes que são autorizados em um sistema. A definição de políticas de segurança é normalmente orientada por modelos de segurança, que fornecem um conjunto de regras e mecanismos para o funcionamento seguro de uma representação abstrata de sistemas.

O modelo de segurança utilizado em boa parte dos sistemas atuais é o chamado controle de acesso discricionário, que delega aos usuários a tarefa de proteger as informações do sistema. Neste modelo, cada usuário é quem determina quais os direitos de acesso que outros usuários e/ou aplicações do sistema possuem sobre as informações que são de sua responsabilidade. Essa dependência dos usuários é um fator que reduz bastante a confiança que se pode depositar em um sistema que utiliza uma política de segurança discricionária. Modelos discricionários são considerados inadequados para diversas aplicações, uma vez que são relativamente fracos e demasiadamente flexíveis: basta um equívoco por parte de um usuário inocente (ou um ato deliberado de um usuário mal-intencionado) e informações importantes podem ser indevidamente reveladas, alteradas ou destruídas.

Por outro lado, modelos alternativos de segurança mais rígidos e seguros do que o discricionário vêm sendo propostos desde o início da década de 70, quando sistemas *on-line* multiusuários começaram a se tornar populares. Estes modelos baseiam-se nas mais diferentes premissas e possuem os mais diversos objetivos.

Esta dissertação pode ser dividida em duas grandes partes. A primeira parte é um estudo sobre diversos modelos de segurança computacional visando determinar a sua aplicabilidade em sistemas distribuídos de larga escala. A segunda parte utiliza os dados obtidos neste estudo para escolher um modelo específico (baseado em papéis) que é usado como modelo de referência em uma implementação. Esta implementação tem como objetivo comprovar na prática a aplicabilidade de modelos baseados em papéis.

A nossa proposta, a **proposta RBAC-JACOWEB**, introduz o conceito de ativação automática de papéis pelo subsistema de segurança, uma inovação em relação à literatura conhecida sobre controle de acesso baseado em papéis. Essa ativação automática permite que o esquema de autorização seja utilizado por aplicações previamente desenvolvidas e que não possuem nenhuma ciência da segurança do sistema. Outra característica importante da proposta RBAC-JACOWEB é a utilização de padrões: tanto o modelo de segurança quanto a tecnologia empregada (CORBA[1]) são padronizados, o que representa mais um diferencial em relação a outras experiências relatadas na literatura.

Este trabalho foi desenvolvido no contexto do projeto JACOWEB, desenvolvido no Laboratório de Controle e Microinformática da Universidade Federal de Santa Catarina (LCMI-DAS). Este projeto tem como objetivo desenvolver e implementar esquemas de autorização para aplicações distribuídas de larga escala combinando os modelos de segurança Java, CORBA e Web [66, 67].

1.2 Estrutura da Dissertação

Esta dissertação está estruturada em sete capítulos e um apêndice. Este capítulo inicial descreveu o contexto onde o trabalho está inserido, sua motivação e objetivos e um resumo de seus principais resultados.

O capítulo 2 contém os fundamentos sobre segurança computacional necessários ao entendimento deste trabalho. A seguir, é apresentado o estudo realizado sobre modelos de segurança computacional, discutindo suas premissas, objetivos, virtudes e limitações. O capítulo encerra com uma comparação entre os diferentes modelos de segurança.

[1] *Common Object Request Broker Architecture*, um padrão para a construção de sistemas distribuídos abertos recentemente adotado pela ISO (*International Organization for Standardization*), que é uma organização internacional que tem por objetivo a elaboração de padrões adotados mundialmente.

O capítulo 3 discute em detalhes o modelo de controle de acesso baseado em papéis utilizado como referência. O capítulo também contém uma revisão dos modelos de papéis mais importantes na literatura e uma descrição de algumas experiências de implementação.

O capítulo 4 apresenta a arquitetura CORBA e o seu serviço de segurança, o CORBAsec. A discussão concentra-se nos pontos de maior relevância para este trabalho, como o modelo CORBA de segurança e seus aspectos de controle de acesso e autorização.

O capítulo 5 apresenta uma discussão detalhada sobre a proposta RBAC-JACOWEB, mostrando como modelos baseados em papéis podem ser implementados em um contexto CORBAsec. O capítulo contém ainda um exemplo detalhado do funcionamento do esquema de autorização proposto e uma comparação com outros trabalhos similares discutidos na literatura.

O capítulo 6 discute alguns aspectos relacionados à implementação do esquema de autorização proposto no capítulo 5, mostrando a estrutura do protótipo implementado, analisando os resultados obtidos e discutindo as melhorias que podem ser incorporadas ao protótipo.

O capítulo 7 apresenta as nossas conclusões e algumas perspectivas futuras para a continuação deste trabalho.

Finalmente, o apêndice A descreve a *interface* do serviço que encapsula o controle de acesso baseado em papéis no esquema de autorização proposto e implementado.

Capítulo 2

Modelos Clássicos de Segurança Computacional

A pesquisa na área de modelos de segurança computacional começou no início da década de 70. Ao longo desses anos, inúmeros modelos foram propostos, com as mais variadas premissas e os mais diversos objetivos. Este capítulo discute em detalhes quatro modelos de segurança considerados clássicos: matriz de acesso, Bell-LaPadula, Biba e Clark-Wilson. Além disso, as características gerais de modelos baseados em papéis são introduzidas, permitindo uma comparação destes com os modelos clássicos. Primeiramente, porém, faz-se necessário apresentar alguns conceitos fundamentais sobre segurança computacional.

2.1 Conceitos Básicos

2.1.1 Segurança Computacional

Diferentes pessoas possuem diferentes conceitos do que seja segurança computacional. No contexto deste trabalho, a preocupação é com a chamada **segurança de dados** (*data security*)[1], que se refere à manutenção de três propriedades fundamentais [3, 50]:

- **Confidencialidade**: garante que os usuários do sistema só podem ler informações para os quais estejam autorizados; uma violação da confidencialidade é chamada de **revelação não-autorizada** ou **vazamento de informação**.

[1] Em oposição à chamada **segurança de funcionamento** (*safety*) [39].

- **Integridade**: garante que o sistema não corrompa informações nem permita que elas sejam modificadas de forma não-autorizada, independentemente do caráter malicioso ou acidental das modificações.

- **Disponibilidade**: garante que os recursos do sistema estejam disponíveis para os seus usuários legítimos; a violação da disponibilidade é chamada de **negação de serviço.**

2.1.2 Ameaças, Vulnerabilidades e Ataques

Uma **ameaça** a um sistema computacional consiste em uma ação possível que, uma vez concretizada, produziria efeitos indesejáveis sobre os dados ou recursos do sistema. Uma **vulnerabilidade**, por sua vez, é uma falha ou característica indevida que pode ser explorada para concretizar uma ameaça; pode-se perceber que as ameaças a um sistema podem ser minimizadas através da identificação e remoção de vulnerabilidades existentes no sistema. Por fim, um **ataque** a um sistema é uma ação tomada por um intruso malicioso (ou, em alguns casos, um erro de operação por parte um usuário inocente) que envolve a exploração de determinadas vulnerabilidades de modo a concretizar uma ou mais ameaças.

Uma maneira de ilustrar a inter-relação entre ameaças, vulnerabilidades e ataques é tecendo uma analogia com uma casa. Uma ameaça associada a uma casa é o roubo de móveis, dinheiro e eletrodomésticos. Vulnerabilidades podem ser uma janela aberta ou uma porta que não esteja trancada. O ataque consiste na invasão propriamente dita com o conseqüente roubo de bens.

2.1.3 Políticas, Modelos e Mecanismos de Segurança

Políticas, modelos e mecanismos de segurança são conceitos que são freqüentemente confundidos entre si, mesmo porque não usados de maneira absolutamente uniforme na literatura. Assim sendo, faz-se necessário conceituá-los de modo a eliminar possíveis dúvidas quanto ao seu significado dentro do contexto deste trabalho.

A **política de segurança** de um sistema computacional corresponde ao conjunto de regras que estabelecem os limites de operação dos usuários no sistema; uma política sempre se aplica a um sistema específico, e não a uma classe geral de sistemas. Por sua vez, um **modelo de segurança** é uma representação restrita de uma classe de sistemas que abstrai detalhes de modo a realçar uma propriedade específica ou um conjunto de comportamentos. Modelos de segurança são úteis como guias na definição de políticas específicas de segurança.

Finalmente, **mecanismos de segurança** são elementos responsáveis pela implantação de uma determinada política de segurança. Por exemplo, uma política de segurança pode exigir que todos os usuários de um sistema sejam identificados univocamente para fins de contabilidade; mecanismos

para implantar esta política incluem o uso de senhas, de cartões magnéticos e de dispositivos de reconhecimento de impressões digitais.

2.1.4 Controle de Acesso

Para a compreensão do que seja controle de acesso, os conceitos de sujeitos e objetos devem ser conhecidos. Um **sujeito** é uma entidade ativa em um sistema computacional que inicia requisições por recursos; corresponde, via de regra, a um usuário ou a um processo executando em nome de um usuário. Um **objeto** é uma entidade passiva que armazena informações no sistema, como arquivos, diretórios e segmentos de memória.

Virtualmente todos os sistemas computacionais podem ser descritos em termos de sujeitos acessando objetos [3]. 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; 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.[2]

Existem diversas classes de modelos de controle de acesso, dentre as quais [59, 60]:

- **Controle de Acesso Discricionário** (*Discretionary Access Control*—DAC): baseia-se na idéia de que o proprietário da informação deve determinar quem tem acesso a ela. Um 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.
- **Controle de Acesso Obrigatório** (*Mandatory Access Control*—MAC): baseia-se em uma administração centralizada de segurança, a qual dita regras incontornáveis de acesso à informação. A forma mais usual de controle de acesso obrigatório é o controle de acesso baseado em reticulados (*lattice-based access control*), que confina a transferência de informação a uma direção em um reticulado de rótulos de segurança (vide seção 2.1.5).
- **Controle de Acesso Baseado em Papéis** (*Role-Based Access Control*—RBAC): requer que direitos de acesso sejam atribuídos a papéis e não a usuários, como no DAC; os usuários obtêm estes direitos em virtude de terem papéis a si atribuídos.

Modelos correspondentes às diferentes classes de controle de acesso serão discutidos nas seções 2.2 a 2.6.

[2] É importante ter em mente que a política de segurança efetivamente implantada pode não ser a mesma política de segurança oficial do sistema. Isto pode acontecer por razões variadas, tais como erros de quem implantou a política, falhas nos mecanismos de segurança e limitações de natureza prática que impedem que a política expressa seja implementada.

2.1.5 Rótulos de Segurança

As pesquisas que levaram aos modelos pioneiros de segurança computacional, no início da década de 70, foram financiadas pelo Departamento de Defesa (DoD) dos Estados Unidos. Desta forma, estes modelos iniciais foram baseados em práticas de segurança utilizadas em áreas ligadas à segurança nacional. Em que pese esta origem, os modelos e seus conceitos subjacentes são perfeitamente aplicáveis a ambientes não-militares.

Um destes conceitos é o de **níveis de segurança**. Como há custos associados à proteção da informação e nem todas as informações são igualmente importantes (ou **sensíveis**), definem-se diferentes níveis de segurança, ordenados segundo uma hierarquia. Os níveis mais usuais são, em ordem crescente de sensibilidade, NÃO-CLASSIFICADO, CONFIDENCIAL, SECRETO e ULTRA-SECRETO. De maneira similar, uma universidade poderia adotar os níveis ALUNO, FUNCIONÁRIO e PROFESSOR—os níveis devem refletir a necessidade de proteção da informação (dificilmente um ambiente acadêmico classificaria as informações da mesma maneira que um ambiente militar).

Entretanto, a simples associação de níveis de segurança à informação e aos indivíduos que têm acesso a ela não atende a um princípio clássico de segurança conhecido como *need-to-know*. Este princípio diz que o controle da disseminação da informação está diretamente ligado à quantidade de pessoas que têm acesso a essa informação; desta forma, quanto menos pessoas conhecerem um segredo, mais fácil será garantir que o segredo não será revelado. Para que isso seja viabilizado, são definidas **categorias de segurança**, ou compartimentos de segurança, que correspondem a diferentes projetos ou setores. Os indivíduos têm acesso a diferentes categorias na medida em que as suas incumbências demandem este acesso. Assim, por exemplo, professores do Departamento de Física provavelmente não devem ter acesso a informações classificadas com o nível PROFESSOR pertencentes ao Departamento de Geografia.

Transpondo estes conceitos para o contexto computacional, um **rótulo de segurança** é um atributo que denota a sensibilidade de entidades ativas e passivas em um sistema. Ele é composto por um nível de segurança e um conjunto (possivelmente vazio) de categorias de segurança. Em sistemas que fazem uso deste mecanismo, todas as entidades recebem um rótulo de segurança; o rótulo de um objeto é chamado de **classificação** do objeto, e o rótulo de um sujeito é chamado de **habilitação** (*clearance*) do sujeito.

2.1.5.1 Representação Matemática e Reticulados de Segurança

A representação matemática destes conceitos possibilita que eles sejam utilizados na construção de modelos formais de segurança, uma vez que tal representação permite que estes modelos sejam corretamente analisados e tenham sua segurança demonstrada.

Dado o conjunto de níveis de segurança representado por $\mathcal{N}$ e o conjunto de categorias representado por C, o conjunto $\mathcal{R}$ de rótulos de segurança de um sistema é então definido como:

$$\mathcal{R} = \mathcal{N} \times 2^C,$$

onde 2^C é o conjunto potência de C.[3]

Exemplo 2.1

$$\begin{aligned}
\mathcal{N} &= \{\text{CONFIDENCIAL, SECRETO}\} \\
C &= \{E, M\} \\
2^C &= \{\varnothing, \{E\}, \{M\}, \{E, M\}\} \\
\mathcal{R} &= \{(\text{CONFIDENCIAL}, \varnothing), (\text{CONFIDENCIAL}, \{E\}), \\
&\quad (\text{CONFIDENCIAL}, \{M\}), (\text{CONFIDENCIAL}, \{E, M\}), \\
&\quad (\text{SECRETO}, \varnothing), (\text{SECRETO}, \{E\}), \\
&\quad (\text{SECRETO}, \{M\}), (\text{SECRETO}, \{E, M\})\}
\end{aligned}$$

É possível definir uma relação de ordem parcial[4] em $\mathcal{R}$. Esta relação, que é bastante usada neste trabalho, é a relação de dominância, que especifica quando um rótulo de segurança **domina** outro.

Definição 2.1 (Nível de um rótulo) Define-se $nível : \mathcal{R} \rightarrow \mathcal{N}$ como a função que retorna o nível de segurança de um dado rótulo de segurança.

Definição 2.2 (Categorias de um rótulo) Define-se $categ : \mathcal{R} \rightarrow 2^C$ como a função que retorna o conjunto de categorias de segurança de um dado rótulo de segurança.

Definição 2.3 (Dominância) Para todo $R_1, R_2 \in \mathcal{R}$, $R_1 \succeq R_2$ (lê-se R_1 domina R_2) se, e somente se, $nível(R_1) \geq nível(R_2)$ e $categ(R_1) \supseteq categ(R_2)$.

Definição 2.4 (Dominância estrita) Para todo $R_1, R_2 \in \mathcal{R}, R_1 \succ R_2$ (lê-se R_1 domina estritamente R_2) se, e somente se, $R_1 \succeq R_2$ e $R_1 \neq R_2$.

Exemplo 2.2 Exemplos da relação de dominância no conjunto de rótulos de segurança $\mathcal{R}$ do exemplo 2.1:

i. (ULTRA-SECRETO, $\{E\}$) $\succeq$ (ULTRA-SECRETO, $\varnothing$)

[3] O **conjunto potência** de um conjunto $\mathcal{A}$, denotado por $2^{\mathcal{A}}$, é o conjunto de todos os subconjuntos de $\mathcal{A}$.

[4] Uma relação de ordem parcial R é uma relação reflexiva (aRa), transitiva (se aRb e bRc, então aRc) e anti-simétrica (se aRb e bRa, então $a = b$).

ii. (SECRETO, $\{E,M\}$) $\succeq$ (NÃO-CLASSIFICADO, $\{E\}$)

iii. (CONFIDENCIAL, $\{E,M\}$) $\succeq$ (CONFIDENCIAL, $\{E,M\}$)

iv. (SECRETO, $\{E\}$) $\not\succeq$ (ULTRA-SECRETO, $\{E\}$)

v. (ULTRA-SECRETO, $\{E\}$) $\not\succeq$ (NÃO-CLASSIFICADO, $\{E,M\}$)

É importante notar que, em um conjunto ordenado parcialmente, existem elementos a e b tais que $a \not\succeq b$ e $b \not\succeq a$; estes elementos são ditos **não-comparáveis**. Por exemplo, no conjunto $\mathcal{R}$ do exemplo 2.1, (CONFIDENCIAL, $\varnothing$) e (NÃO-CLASSIFICADO, $\{E\}$) são não-comparáveis.

Definição 2.5 (Limite inferior) Seja $\mathcal{B}$ um subconjunto de um conjunto parcialmente ordenado $\mathcal{A}$. Um elemento m em $\mathcal{A}$ é chamado de **limite inferior** de $\mathcal{B}$ se, para qualquer $x \in \mathcal{B}$, $m \preceq x$, isto é, se m preceder (for dominado por) cada elemento em $\mathcal{B}$.

Definição 2.6 (Ínfimo) Seja $\mathcal{B}$ um subconjunto de um conjunto parcialmente ordenado $\mathcal{A}$. Se existe um limite inferior m de $\mathcal{B}$ tal que m domina cada um dos outros limites inferiores de $\mathcal{B}$, m é chamado de limite maior inferior (*lmi*) ou **ínfimo** de $\mathcal{B}$, sendo denotado por $m = \inf(\mathcal{B})$.

Em geral, $\mathcal{B}$ pode ter um, muitos ou nenhum limite inferior, porém existe quando muito um $\inf(\mathcal{B})$.

Definição 2.7 (Limite superior) Seja $\mathcal{B}$ um subconjunto de um conjunto parcialmente ordenado $\mathcal{A}$. Um elemento M em $\mathcal{A}$ é chamado de **limite superior** de $\mathcal{B}$ se, para qualquer $x \in \mathcal{B}$, $M \succeq x$, isto é, se M dominar cada elemento em $\mathcal{B}$.

Definição 2.8 (Supremo) Seja $\mathcal{B}$ um subconjunto de um conjunto parcialmente ordenado $\mathcal{A}$. Se existe um limite superior M de $\mathcal{B}$ tal que M preceda cada um dos outros limites superiores de $\mathcal{B}$, M é chamado de limite menor superior (*lms*) ou **supremo** de $\mathcal{B}$, sendo denotado por $M = \sup(\mathcal{B})$. Pode haver no máximo um $\sup(\mathcal{B})$.

Definição 2.9 (Reticulado) Seja o conjunto $\mathcal{A}$ parcialmente ordenado. Se, para quaisquer elementos $a,b \in \mathcal{A}$ existem $\inf\{a,b\}$ e $\sup\{a,b\}$, então o conjunto $\mathcal{A}$ é chamado um **reticulado**.

No conjunto $\mathcal{R}$ de rótulos de segurança, o ínfimo é representado pelo elemento composto pelo nível mais baixo de segurança (que é o limite inferior do conjunto $\mathcal{N}$) e por um conjunto vazio de categorias de segurança. O supremo de $\mathcal{R}$ é composto pelo nível mais alto de segurança (limite inferior de $\mathcal{N}$) e por todas as categorias de segurança (o próprio conjunto C). Por exemplo, considerando o conjunto $\mathcal{R}$ do exemplo 2.1, tem-se $\inf(\mathcal{R})$ = (CONFIDENCIAL, $\varnothing$) e $\sup(\mathcal{R})$ = (SECRETO, $\{E,M\}$).

Verifica-se que o conjunto de rótulos de segurança **sempre** possui um ínfimo e um supremo; portanto, pode-se dizer que o conjunto de rótulos de segurança ordenado pela relação de dominância

forma um reticulado de rótulos de segurança [30]. Esta conclusão é muito importante, pois é o princípio fundamental de diversos modelos de segurança, como os modelos Bell-LaPadula e Biba (discutidos, respectivamente, nas seções 2.3 e 2.4) e o modelo de fluxo seguro de informação de Denning [13].

2.2 O Modelo de Matriz de Acesso

O modelo de **matriz de acesso** [28] é o modelo conceitual subjacente ao controle de acesso discricionário. 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. Uma célula M_{ij} representa os direitos de acesso do sujeito i sobre o objeto j. É importante ressaltar que sujeitos podem ser também objetos. Por exemplo, um dos acessos representados na matriz pode ser o envio de um sinal a um processo; neste caso, os sujeitos correspondentes a processos deveriam ser incluídos também nas colunas da matriz. A figura 2.1 mostra um exemplo de matriz de acesso. Neste exemplo, há seis objetos, sendo três arquivos (arq 1–3), duas contas (conta 1–2) e um programa (SCont). Há também quatro sujeitos, sendo três usuárias (Ana, Bia e Cris) mais o programa SCont. Os direitos sobre arquivos são os usuais dono, r (leitura) e w (escrita). Os direitos sobre as contas são consulta, crédito e débito. A programas se aplicam os direitos r e w, além de x (execução).

	arq 1	arq 2	arq 3	conta 1	conta 2	SCont
Ana	dono r w		r	consulta crédito	consulta débito	x
Bia	r	dono r w		consulta crédito débito	consulta crédito débito	x
Cris		r	dono r w	consulta	consulta	r w x
SCont	r w	r	w			

Figura 2.1: Exemplo de matriz de acesso

No controle de acesso discricionário, a concessão e a revogação dos direitos de acesso a um objeto são feitas pelo usuário que é dono desse objeto (à sua *discrição*). Isso fornece ao usuário uma grande flexibilidade na proteção de seus objetos, o que é uma vantagem deste modelo de controle de acesso. Entretanto, o controle de acesso discricionário não permite controlar a disseminação da informação. Por exemplo, considerando a matriz de acesso da figura 2.1, Bia permite que Cris leia o seu arquivo arq 2, mas proíbe que estas informações sejam lidas por Ana. Entretanto, não há nada que ela possa fazer para impedir que Cris aja maliciosamente, copiando as informações de arq 2 para arq 3, o

que possibilitaria que Ana lesse tais informações mesmo contra a vontade da usuária que as detém. Cris pode copiar estas informações diretamente ou agir de maneira mais sutil, instalando um cavalo de Tróia[5] no sistema de contabilidade SCont que faria sub-repticiamente a cópia das informações quando fosse executado por Bia.

O modelo de matriz de acesso proposto por Lampson [28] é um modelo relativamente informal. Foram propostos diversos modelos formais de segurança baseados no modelo de matriz de acesso, como o HRU [23] e o Take-Grant [62].

2.2.1 Estratégias de Implementação

Em um sistema de grande porte a matriz de acesso será bastante grande, e a maioria de suas células estará, provavelmente, vazia. Sendo assim, dificilmente a matriz de acesso é implementada como uma matriz propriamente dita. Existem duas abordagens tradicionais para a implementação de matrizes de acesso: as listas de controle de acesso e as *capabilities*.

2.2.1.1 Listas de Controle de Acesso

Uma das formas de se implementar uma matriz de acesso são as **listas de controle de acesso** (ACLs—*access control lists*). Cada objeto é associado a uma ACL, que indica, para cada sujeito no sistema, os direitos que o sujeito tem sobre o objeto. Esta abordagem corresponde a armazenar a matriz pelas suas colunas. A figura 2.2 mostra listas de controle de acesso correspondentes à matriz de acesso da figura 2.1.

arq 1:	(Ana, {dono,r,w}), (Bia, {r}), (SCont, {r,w})
arq 2:	(Bia, {dono,r,w}), (Cris, {r}), (SCont, {r})
arq 3:	(Ana, {r}), (Cris, {dono,r,w}), (SCont, {w})
conta 1:	(Ana, {consulta,crédito}), (Bia, {consulta,crédito,débito}), (Cris, {consulta})
conta 2:	(Ana, {consulta,débito}), (Bia, {consulta,crédito,débito}), (Cris, {consulta})
SCont:	(Ana, {x}), (Bia, {x}), (Cris, {r,w,x})

Figura 2.2: Exemplo de listas de controle de acesso

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

[5] Um cavalo de Tróia é um fragmento de código escondido dentro de um programa e que realiza alguma função indesejável à revelia do usuário que executa o programa [50].

2.2.1.2 *Capabilities*

As ***capabilities*** são uma abordagem dual às listas de controle de acesso. Cada sujeito é associado a uma lista (a **lista de *capabilities***) que indica, para cada objeto no sistema, os acessos aos quais o sujeito está autorizado. Isto corresponde a armazenar a matriz de acesso por linhas. A figura 2.3 mostra as listas de *capabilities* correspondentes à matriz de acesso da figura 2.1.

Ana:	(arq 1, {dono,r,w}), (arq 3, {r}), (conta 1, {consulta,crédito}), (conta 2, {consulta,débito}), (SCont, {x})
Bia:	(arq 1, {r}), (arq 2, {dono,r,w}), (conta 1, {consulta,crédito,débito}), (conta 2, {consulta,crédito,débito}), (SCont, {x})
Cris :	(arq 2, {r}), (arq 3, {dono,r,w}), (conta 1, {consulta}), (conta 2, {consulta}), (SCont, {r,w,x})
SCont:	(arq 1, {r,w}), (arq 2, {r}), (arq 3, {w})

Figura 2.3: Exemplo de listas de *capabilities*

As *capabilities* permitem fácil verificação e revogação dos acessos autorizados para um determinado sujeito. Em contrapartida, a determinação de quais sujeitos podem acessar um objeto requer a inspeção de todas as listas de *capabilities* do sistema. As vantagens e desvantagens de ACLs e *capabilities* são, como as próprias estratégias, ortogonais entre si.

Capabilities são vantajosas em sistemas distribuídos. A posse de uma *capability* é suficiente para que um sujeito obtenha o acesso autorizado por esta *capability*. Em um sistema distribuído, isso possibilita que um sujeito se autentique uma vez, obtenha a sua lista de *capabilities* e apresente estas *capabilities* para obter os acessos aos quais ele está autorizado; os servidores precisam apenas verificar a validade da *capability* para liberar o acesso [59].

2.3 O Modelo Bell-LaPadula

No início da década de 70, o governo dos Estados Unidos financiou diversas pesquisas sobre modelos de segurança e prevenção de violações de confidencialidade. Dois cientistas da MITRE Corporation, David Bell e Leonard LaPadula, desenvolveram um modelo baseado nos procedimentos usuais de manipulação de informação em áreas ligadas à segurança nacional americana. Esse modelo ficou conhecido como **modelo Bell-LaPadula**, ou modelo BLP [5]. Existem diversas outras descrições do modelo Bell-LaPadula disponíveis na literatura, como [3, 30, 52], algumas delas apresentando pequenas variações em relação ao modelo original.

Na apresentação original do modelo [5], um sistema é descrito através de uma máquina de estados finitos. As transições de estados no sistema obedecem a determinadas regras. Bell e LaPadula demonstram indutivamente que a segurança do sistema é mantida se ele parte de um estado seguro e as únicas transições de estado permitidas são as que conduzem o modelo a um outro estado seguro.

A apresentação desta seção abstrai-se da representação através de máquina de estados, concentrando-se em vez disso nas regras de controle de acesso que garantem a confidencialidade da informação segundo o modelo BLP.

2.3.1 Regras do Modelo Bell-LaPadula

O modelo Bell-LaPadula considera os seguintes tipos de direitos de acesso sobre os objetos:

- e: acesso sem observação e sem modificação;
- r: observação sem modificação;
- a: modificação sem observação;
- w: observação e modificação.

Os direitos e, r, a e w podem ser geralmente associados às operações *execute*, *read*, *append* e *write*, respectivamente. É importante ressaltar, porém, que esta associação pode não ser semanticamente significativa. Por exemplo, muitas plataformas não têm como diferenciar entre acessos de leitura e execução, uma vez que os resultados da execução refletem o conteúdo (e, portanto, constituem uma observação) do programa executado [5]. Outro ponto a considerar é que um acesso do tipo w pode ser encarado como uma combinação dos acessos r e a.

A presente descrição do modelo BLP simplifica os direitos de acesso do modelo original da seguinte forma:

- O direito de acesso e será considerado equivalente ao direito de acesso r;[6]
- Uma **leitura** corresponde a um acesso de observação (r);
- Uma **escrita** corresponde a um acesso de modificação (a);
- O direito de acesso w corresponde a acessos simultâneos de leitura e escrita.

Estas simplificações reduzem a complexidade das regras e aproximam o modelo BLP dos ambientes computacionais de uso corrente, sem, no entanto, enfraquecê-lo em aspecto algum.

O sistema é descrito em termos de sujeitos que acessam objetos (vide seção 2.1.5), onde cada sujeito possui uma habilitação e cada objeto possui uma classificação. A cada sujeito está associado

[6]Em [5], um direito de acesso e está sujeito apenas a um controle de acesso discricionário, que não será considerado aqui por ser essencialmente dissociado do caráter obrigatório do modelo BLP.

também um **rótulo corrente de segurança**, que representa a classificação mais alta dentre as informações já consultadas pelo sujeito no sistema, sendo, portanto, uma classificação flutuante (dinâmica). A habilitação de um sujeito sempre domina o seu rótulo corrente de segurança.

A **propriedade de segurança simples**, também conhecida como propriedade-ss ou regra *no read up*[7] (NRU), diz que um sujeito só pode observar informações para as quais esteja habilitado; em outras palavras, *rótulo*(S) deve dominar *rótulo*(O). Por exemplo, uma informação classificada como SECRETO só pode ser lida por sujeitos com habilitação SECRETO ou ULTRA-SECRETO.[8]

Definição 2.10 (Propriedade-ss) Uma leitura de um sujeito S sobre um objeto O é autorizada se, e somente se, *rótulo*$(S) \succeq$ *rótulo*(O).

A propriedade-ss não é suficiente para garantir a segurança desejada do sistema: ela não evita que um sujeito malicioso coloque informações privilegiadas em um recipiente com classificação inferior à das informações—o que constitui claramente um fluxo não-autorizado de informação. Assim, torna-se necessário adicionar outra propriedade a ser satisfeita pelo sistema.

A **propriedade-*** (lê-se propriedade estrela), também chamada de regra *no write down*[9] (NWD), é satisfeita se, quando um sujeito tem simultaneamente um acesso de observação sobre um objeto O_1 e um acesso de modificação sobre um objeto O_2, então *rótulo*(O_1) é dominado por *rótulo*(O_2). Por exemplo, se um sujeito está lendo (observando) um objeto SECRETO, ele só pode escrever (modificar) um objeto SECRETO ou ULTRA-SECRETO. A propriedade-* pode ser refinada em termos do nível corrente de segurança de um sujeito, conforme mostra a definição 2.11.

Definição 2.11 (Propriedade-*) Um acesso de um sujeito S sobre um objeto O é autorizado se

- *rótulo*$(O) \succeq$ *rótulo-corrente*(S) quando o acesso for de escrita;
- *rótulo*$(O) \preceq$ *rótulo-corrente*(S) quando o acesso for de leitura.

Há duas observações importantes a se fazer respeito da propriedade-*. Primeiro, ela não se aplica a sujeitos de confiança: um sujeito de confiança é aquele em quem se confia a não transferir informação de modo a quebrar a segurança, mesmo que esta transferência seja possível.[10] Segundo, é importante lembrar que a propriedade-ss e a propriedade-* devem ser ambas satisfeitas; nenhuma delas garante, por si só, a segurança desejada.

[7] *No read up* vem do fato de um sujeito não poder ler objetos localizados acima dele no reticulado de rótulos de segurança.

[8] Os exemplos apresentados no texto mencionam apenas o nível de segurança em um rótulo de segurança, desconsiderando as categorias (convenciona-se que todos os rótulos possuem um conjunto vazio de categorias); evidentemente, uma implementação do modelo fará uso das categorias se elas forem pertinentes ao domínio de aplicação.

[9] Assim chamada porque impede que um sujeito escreva em objetos localizados abaixo dele no reticulado de rótulos de segurança.

[10] Os sujeitos de confiança serão vistos com mais detalhes na seção 2.3.3.2.

2.3.2 Dinâmica do Modelo Bell-LaPadula

A discussão do modelo Bell-LaPadula na seção anterior conceitua o rótulo corrente de segurança de um sujeito como uma classificação flutuante e define a propriedade-* em termos do rótulo corrente de segurança de um sujeito, sem no entanto explicitar como este rótulo efetivamente flutua dentro do sistema. Esta seção tem o propósito de analisar o comportamento dinâmico do modelo BLP, isto é, como o rótulo corrente de segurança de um sujeito evolui durante a operação do sistema.[11]

Quando um usuário entra no sistema, ele recebe um rótulo corrente de segurança que seja dominado pela sua habilitação. Este rótulo pode ser escolhido pelo usuário ou atribuído automaticamente pelo sistema; a abordagem adotada não interfere no comportamento dinâmico. Os sujeitos criados em nome de um usuário herdam tanto a habilitação como o rótulo corrente de segurança do usuário. Os acessos destes sujeitos aos objetos do sistema devem observar a propriedade-ss e a propriedade-*.

Bell e LaPadula [5] fornecem um conjunto de regras para a operação de um sistema seguro.[12] Uma destas regras dita que o rótulo corrente de segurança de um sujeito só é modificado mediante uma requisição explícita deste sujeito; isto significa que o rótulo corrente de segurança não flutua de maneira automática no sistema, e, também, que esta flutuação ocorre por iniciativa do próprio sujeito. A regra especifica também que a alteração do rótulo corrente de segurança só é autorizada se ela não violar a propriedade-*. Por exemplo, seja a seguinte situação: um sujeito S, com rótulo corrente NÃO-CLASSIFICADO e habilitação (estática) SECRETO, deseja ler um objeto O_1, que é CONFIDENCIAL. A propriedade-ss permite que S leia O_1, pois $\textit{rótulo}(S) \succeq \textit{rótulo}(O_1)$. Entretanto, essa operação não satisfaz a propriedade-*, pois $\textit{rótulo}(O_1) \not\preceq \textit{rótulo-corrente}(S)$. Logo, S precisa solicitar a atualização de seu rótulo corrente de segurança para (pelo menos) CONFIDENCIAL. Entretanto, se S, ao solicitar a atualização de seu rótulo corrente para CONFIDENCIAL possuir um acesso de escrita para O_2, onde O_2 é igualmente NÃO-CLASSIFICADO, ele deve ter esta solicitação negada pelo sistema, uma vez que a sua aceitação violaria a propriedade-*.

A regra que governa a atualização do rótulo corrente de segurança não impõe qualquer restrição além da satisfação da propriedade-* e da condição de que o rótulo corrente seja dominado pela habilitação do sujeito. Portanto, o rótulo corrente pode tanto subir como descer no reticulado de rótulos de segurança, respeitadas as condições acima. Entretanto, algumas complicações de natureza prática podem surgir em implementações do modelo Bell-LaPadula. Seja o seguinte trecho em pseudocódigo [35]:

[11] **O princípio da tranqüilidade** estabelece que nenhuma operação pode alterar a classificação de objetos ativos no sistema [30]. Entretanto, implementações baseadas no modelo BLP tipicamente lançam mão de sujeitos de confiança para a reclassificação de objetos.

[12] Evidentemente, a noção de sistema seguro, no contexto do modelo BLP, corresponde a um sistema a salvo de ameaças de revelação não-autorizada.

```
1  passa(A1: arquivo, A2: arquivo)
2  abre A1 para leitura
3  lê I de A1
4  fecha A1
5  abre A2 para escrita
6  escreve I em A2
7  fecha A2
```

Se uma implementação considerar que os objetos no sistema são representados por arquivos, o código acima pode servir de base para uma revelação não-autorizada. Seja A1 um objeto SECRETO e A2 um objeto CONFIDENCIAL. Entre as linhas 2 e 4, passa tem que ter rótulo corrente SECRETO, para que possa ter acesso de leitura a A1. Entretanto, nada (a princípio) impede que entre as linhas 4 e 5 ele rebaixe seu rótulo corrente para CONFIDENCIAL, o que lhe permite escrever a informação I no arquivo A2 (linhas 5–7). Isto representa um fluxo não-autorizado de informação, mas uma seqüência de operações como esta não está em conformidade com o modelo BLP (uma vez que acaba por violar a propriedade-*). Se o controle de acesso for aplicado somente aos arquivos do sistema (ou seja, os arquivos correspondem aos objetos) não é possível permitir que um sujeito rebaixe seu rótulo corrente de segurança, sob pena de ocorrerem violações da propriedade-*. Embora esta restrição seja bastante rígida, ela não deve ser percebido como uma limitação do modelo Bell-LaPadula, mas, sim, como uma restrição imposta pelas condições de implementação do modelo.

2.3.3 Limitações do Modelo Bell-LaPadula

Quando foi proposto, o modelo Bell-LaPadula representou um avanço significativo ao definir formalmente conceitos de segurança sobre informações que seguem classificações militares de maneira aplicável a sistemas computacionais. Ele serviu de base para diversos esforços de projeto e implementação. Em parte devido a estes esforços, alguma limitações do modelo foram descobertas e discutidas na literatura. Discussões mais completas sobre problemas do modelo BLP podem ser encontradas em [3, 30, 35, 39].

2.3.3.1 Escritas Cegas

A adoção do modelo BLP conforme apresentado na seção 2.3.1 pode acarretar problemas se o sistema tiver que lidar também com potenciais ameaças de integridade. Quando um sujeito escreve em um objeto com uma classificação superior à sua habilitação (o que satisfaz a propriedade-*), ele não pode observar os efeitos desta operação de escrita (o que violaria a propriedade-ss); por esse motivo, tal operação é chamada de **escrita cega** [3, 52].

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 objetos com níveis mais altos de segurança podem ser proibidas; um sujeito só poderia escrever em um objeto que tivesse o mesmo nível de segurança. Entretanto, tal modificação restringe, de certa forma, o modelo BLP e muda o 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 da propriedade-* revisada é bastante comum em implementações de sistemas computacionais que seguem o modelo BLP.

2.3.3.2 Sujeitos de Confiança

Conforme mencionado na seção 2.3.1, Bell e LaPadula incluem no seu modelo a noção de **sujeitos de confiança** (*trusted subjects*) [5, 30]. Um sujeito de confiança é aquele em quem se confia a não quebrar a segurança mesmo que alguns dos seus acessos atuais violem a propriedade-*. Neste caso, a propriedade-* só se aplica aos demais sujeitos do sistema.

Por exemplo, o conceito de sujeitos de confiança pode ser usado para qualificar os processos relacionados com a manutenção do sistema, pois se o administrador do sistema tiver que obedecer estritamente às regras do modelo BLP ele dificilmente conseguirá realizar qualquer tarefa significativa de administração. Outra classe de processos que faz uso da noção de sujeitos de confiança compreende os subsistemas mais críticos do sistema operacional, como gerência de memória e *drivers* de dispositivos [3].

Tarefas como reclassificação e sanitização[13], por sua vez, também só podem ser efetuadas usando o conceito de sujeitos de confiança, e o modelo BLP em si fornece pouca orientação para determinar quais os processos que podem ser considerados de confiança [30]. Bell e LaPadula sustentam que qualquer processo no qual se pretende confiar deve ser provado seguro [5], o que, na grande maioria dos casos, é uma tarefa extremamente difícil.

2.3.3.3 Superclassificação da Informação

Um dos principais problemas do modelo Bell-LaPadula reside no aspecto extremamente restritivo da propriedade-*.[14] Por exemplo, se um sujeito com rótulo corrente de segurança SECRETO

[13] A **sanitização** de um documento corresponde à remoção das informações com classificação superior à do documento sanitizado. Por exemplo, um documento secreto pode passar por um processo de sanitização onde são removidas as informações secretas e classificadas, sendo o documento resultante não-classificado.

[14] Segundo Landwehr [30], a provisão de sujeitos de confiança é um reconhecimento de que a propriedade-* impõe restrições de acesso mais rigorosas do que aquelas usadas extracomputacionalmente em ambientes de segurança militar, uma vez que o seu propósito é evitar que programas malcomportados causem vazamentos de informação.

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. Este fenômeno é conhecido como **superclassificação da informação** [30, 39]. 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.

2.3.3.4 Canais Cobertos

Lampson foi um dos pioneiros na investigação de fluxos ilegítimos de informação em sistemas computacionais [29]. Segundo ele, a informação pode ser transmitida através de três tipos de canais de comunicação:

- **Canais legítimos**: aqueles por onde passam os fluxos autorizados de informação;
- **Canais de memória** (*storage channels*): estruturas de dados mantidas pelo sistema e que podem ser utilizadas como armazenamento temporário entre emissor e receptor da informação;
- **Canais temporais** (*timing channels*): aqueles que não são destinados a transferência de informações mas que são abusados de forma a fazê-lo.

Os canais de memória e os canais temporais compreendem os **canais cobertos** (*covert channels*).[15]

Exemplos de canais de memória incluem arquivos temporários e variáveis compartilhadas. Os canais temporais, por sua vez, correspondem a interferências no funcionamento do sistema e que podem transmitir informações segundo um código previamente estabelecido; por exemplo, um processo pode enviar dados variando a sua taxa de paginação enquanto outro processo monitora o desempenho do sistema, decodificando os dados transmitidos. Normalmente, a taxa de transferência de informação conseguida através de canais temporais é bastante baixa, mas isso nem sempre é verdade. Outro aspecto importante é que o uso de canais temporais implica em uma sincronização entre emissor e receptor; canais de memória, por outro lado, podem ser utilizados assincronamente.

O modelo Bell-LaPadula previne a revelação não-autorizada de informação através de canais legítimos e de canais cobertos de memória, mas não resolve o problema de canais cobertos temporais [5, 30, 52]. A principal razão para isto é que canais temporais constituem um problema intrínseco a sistemas onde recursos são compartilhados entre vários usuários, tornando bastante difícil a sua eliminação.

[15] Em [29], os canais temporais são chamados de canais cobertos. O uso corrente, todavia, define canais cobertos como aqueles que podem ser explorados para transferir informações de modo a violar uma política de segurança; estes canais dividem-se em canais cobertos de memória e canais cobertos temporais [14].

2.4 O Modelo de Integridade Biba

O modelo Bell-LaPadula tem por objetivo conter ameaças de revelação não-autorizada; não obstante, os próprios criadores do modelo BLP discutem como ele poderia ser adaptado para conter ameaças de integridade [5]. Embora as idéias de Bell e LaPadula careçam de maior consistência, elas serviram de base para que Ken Biba, igualmente da MITRE Corporation, desenvolvesse, na segunda metade da década de 70, um modelo de segurança com o propósito de garantir a integridade da informação; este modelo ficou conhecido como **modelo de integridade Biba**, ou, simplesmente, modelo Biba [7]. O modelo Biba é expresso de uma maneira similar ao modelo BLP, com a exceção das regras serem aproximadamente opostas às do modelo BLP. Neste capítulo serão discutidas três instâncias específicas do modelo Biba. Na verdade, a denominação genérica "modelo Biba" pode ser interpretada como todos os modelos propostos em [7] ou qualquer um deles em particular. Descrições do modelo Biba podem ser também encontradas em [3, 30, 33, 34, 52].

2.4.1 O Modelo Obrigatório de Integridade

O modelo obrigatório de integridade de Biba, também chamado modelo de integridade estrita, é por vezes descrito como o inverso do modelo BLP. Esta descrição é razoavelmente precisa, uma vez que as regras básicas deste modelo essencialmente invertem as regras do modelo BLP. A diferença é que, no modelo Biba, um outro rótulo de segurança, chamado de **rótulo de integridade**, é utilizado. Rótulos de integridade, assim como os rótulos de segurança apresentados na seção 2.1.5, também são compostos por um nível de integridade e por categorias (ou compartimentos) de integridade. Do mesmo modo, estes rótulos de integridade são parcialmente ordenados, formando um reticulado de integridade. Tradicionalmente, os níveis de integridade recebem os mesmos nomes dos níveis de segurança; a integridade ULTRA-SECRETO corresponde à informação que possui maior necessidade de proteção contra modificações não-autorizadas.[16]

A **propriedade de integridade simples**, também conhecida como propriedade-is ou regra *no read down* (NRD), diz que um sujeito só pode ler informações de objetos que possuam integridade igual ou superior à sua; em outras palavras, $rótulo(O)$ deve dominar $rótulo(S)$.

Definição 2.12 (Propriedade-is) Uma leitura de um sujeito S sobre um objeto O é autorizada se, e somente se, $rótulo(O) \succeq rótulo(S)$.

A **propriedade-* de integridade**, ou regra *no write up* (NWU), diz que um sujeito só pode escrever em objetos que possuam integridade igual ou inferior à sua. Ou seja, $rótulo(S)$ deve dominar $rótulo(O)$.

[16] Esta escolha de nomes para os rótulos de integridade é freqüentemente criticada, uma vez que informações com integridade ULTRA-SECRETO podem não possuir qualquer necessidade de segredo [30].

Definição 2.13 (Propriedade-*) Uma escrita de um sujeito S em um objeto O é autorizada se, e somente se, $rótulo(S) \succeq rótulo(O)$.

Percebe-se claramente que as regras do modelo obrigatório de integridade Biba, com rótulos de integridade substituindo os rótulos de segurança, são essencialmente o oposto das regras do modelo BLP.[17]

2.4.2 O Modelo de Marca d'Água Baixa de Sujeitos

O modelo obrigatório de integridade é consideravelmente inflexível. Como uma implementação baseada neste modelo seria pouco conveniente para o uso, Biba propôs outros modelos que relaxam as restrições do modelo obrigatório. Um destes modelos é o **modelo de marca d'água baixa de sujeitos** (*subject low-water mark*), onde um sujeito pode ler informações cujos rótulos de integridade estejam abaixo do seu no reticulado de rótulos de integridade. Entretanto, a conseqüência destas leituras é um rebaixamento do nível de integridade do sujeito para o nível de integridade do objeto sendo lido.

No modelo de marca d'água baixa de sujeitos, um sujeito de alta integridade pode ser visto como "puro". Quando este sujeito puro incorpora informações de uma fonte menos pura, ele é "corrompido" e seu nível de integridade deve ser ajustado de maneira correspondente [3].

Uma das características interessantes do modelo de marca d'água baixa de sujeitos é que ele não impõe qualquer restrição sobre o que um sujeito pode ler. Se, por exemplo, um sujeito nunca puder ser corrompido para um nível mais baixo de integridade, este modelo não é o mais indicado porque poderia resultar neste tipo de corrupção. Se o modelo tivesse que ser seguido em uma implementação, poderiam eventualmente ser usados mecanismos adicionais que avisassem o sujeito das conseqüências potenciais de tais operações de leitura antes que elas fossem efetuadas.

Cabe notar também que as mudanças de nível de integridade de sujeitos obedecem a uma monotonicidade, ou seja, o nível de integridade de um sujeito permanece o mesmo ou apenas decresce, uma vez que o modelo não apresenta provisão para o aumento do nível de integridade de sujeitos.

2.4.3 O Modelo de Marca d'Água Baixa de Objetos

O último modelo proposto por Biba a ser discutido aqui é análogo ao modelo de marca d'água baixa de sujeitos, e é conhecido como **modelo de marca d'água baixa de objetos**. Neste modelo, um sujeito podem escrever em objetos com rótulos de integridade acima do seu no reticulado de rótulos de integridade. De maneira análoga ao modelo de marca d'água baixa de sujeitos, esta escrita acarreta

[17] Essa simetria não leva em consideração a existência de rótulos correntes de segurança ou integridade.

o rebaixamento do rótulo de integridade do objeto para o rótulo do sujeito que nele está escrevendo. Esta regra é motivada pelas mesmas razões da regra do modelo de marca d'água baixa de sujeitos.

Este modelo, assim como o modelo de marca d'água baixa de sujeitos, não impõe qualquer restrição sobre os objetos em que um sujeito pode escrever. Como resultado, situações onde as conseqüências do rebaixamento da integridade de objetos são inaceitáveis desaconselham a utilização deste modelo. Por exemplo, uma base de dados crítica que deve conter informações cuja integridade é de capital importância não admite implementações que sigam este modelo. Entretanto, se o modelo tivesse que ser usado, os sujeitos poderiam assumir a responsabilidade de não causar degradação em objetos de mais alta integridade. Eventualmente, algum tipo de mediação poderia ser empregada para conseguir isto.

Novamente, observa-se que há uma monotonicidade nas mudanças de nível de integridade de objetos. Como no modelo anterior, não há qualquer provisão para o aumento do nível de integridade de um objeto. Ainda, cabe notar que os modelos de marca d'água baixa de sujeitos e objetos poderiam ser combinados no mesmo sistema.

2.4.4 Limitações do Modelo Biba

Devido à sua similaridade com o modelo BLP, o modelo Biba possui muitas das vantagens daquele modelo. Por exemplo, ambos modelos são relativamente simples e intuitivos, e conseguem demonstrar de maneira indutiva a sua capacidade de conservar a propriedade de segurança considerada (confidencialidade no BLP e integridade no Biba).

Por outro lado, o modelo Biba compartilha algumas das limitações do modelo BLP (discutidos na seção 2.3.3). Por exemplo, foi sugerido que o modelo Biba também depende em demasia de sujeitos de confiança em situações práticas [3]: a necessidade de um processo de confiança para aumentar ou reduzir a integridade de sujeitos ou objetos é especialmente problemática para a integridade. Esta crítica ao uso de sujeitos de confiança segue a discussão da seção 2.3.3.2.

Uma 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 e objetos ou rebaixam a integridade de algum sujeito ou objeto. Isto permite imaginar que, à medida em que o tempo progride, os sistemas sofrem um decaimento de integridade monotonicamente decrescente, com os sujeitos e objetos migrando gradativamente para o nível mais baixo de integridade. Esta degradação da integridade da informação é análoga ao problema de superclassificação da informação no modelo BLP, discutido na seção 2.3.3.3.

Muitos pesquisadores criticam a implicação, presente no modelo Biba, de que integridade é uma medida e que a noção de "maior integridade" tem algum significado [3]. O seu argumento é que a integridade de sujeitos e objetos deve ser encarada como um atributo binário, que está presente ou não.

2.5 O Modelo Clark-Wilson

Em 1987, David Clark, do MIT, e David Wilson, da Ernst and Whinney, apresentaram um modelo de integridade radicalmente diferente dos modelos baseados em reticulados de segurança, tais como os modelos Bell-LaPadula e Biba. Este novo modelo, que ficou conhecido como **modelo Clark-Wilson**, ou modelo CW, foi motivado principalmente pela maneira como organizações comerciais controlam a integridade de seus documentos em papel em um ambiente de trabalho não-automatizado [12]. Em outras palavras, Clark e Wilson consideraram práticas administrativas e (principalmente) contábeis largamente difundidas e tentaram extrapolá-las para uso em aplicações computacionais. O modelo de integridade resultante deste esforço tem sido considerado bastante efetivo como guia para projetistas e desenvolvedores de sistemas computacionais onde a integridade desempenha um papel importante.

O modelo CW é expresso em termos de um conjunto de regras que governam a operação e manutenção de um dado ambiente ou aplicação computacional de modo a garantir a integridade de um subconjunto bem-definido dos seus dados. A noção crítica no modelo CW é que estas regras são expressas através das chamadas transações bem-formadas, onde um sujeito inicia uma seqüência de ações que é completada de maneira controlada e previsível. Descrições do modelo Clark-Wilson podem ser também encontradas em [3, 34].

2.5.1 Conceitos Básicos

A proteção da confidencialidade da informação é importante tanto no âmbito militar como no comercial.[18] Entretanto, um objetivo bastante importante (por vezes o mais importante) no processamento de dados comerciais é garantir a integridade dos dados para evitar fraudes e erros. Nenhum usuário do sistema, mesmo que autorizado, deve poder modificar itens de dados de tal forma que registros contábeis ou de bens da organização sejam perdidos ou corrompidos. Alguns mecanismos, como autenticação de usuários, são cruciais para a implantação tanto de políticas de segurança militar como de políticas de segurança comercial. Entetanto, outros mecanismos são bastante diferentes [12].

Os mecanismos de alto nível utilizados para implantar políticas de segurança comercial relacionadas à integridade de dados foram desenvolvidos muito antes da existência dos sistemas computacionais. Existem dois mecanismos básicos para o controle de erros e fraudes: a transação bem-formada (*well-formed transaction*) e a separação de tarefas (*separation of duty*) entre funcionários.

O conceito de uma **transação bem-formada** diz que um usuário não deve manipular dados arbitrariamente, mas apenas de modo controlado, preservando ou garantindo a sua integridade. Um

[18] Clark e Wilson consideram "segurança militar" o controle de informações classificadas no contexto do Livro Laranja do DoD [14], que se baseia fortemente no modelo Bell-LaPadula.

mecanismo bastante comum em transações bem-formadas é o registro de todas as modificações de dados em um *log* de tal forma que as atividades possam ser posteriormente auditadas. Antes do uso de computadores, a escrituração comercial era feita com tinta, e as correções eram feitas através de ressalvas; a entrada incorreta não poderia ser apagada ou rasurada. Deste modo, os livros em si constituíam o *log*, e qualquer rasura era um indicativo de fraude.

O exemplo mais formalmente estruturado de transações bem-formadas ocorre provavelmente em sistemas contábeis, que utilizam o método de partidas dobradas. O método de partidas dobradas garante a consistência interna dos dados requerendo que qualquer modificação dos registros contábeis envolva duas ou mais contas, que devem compensar-se entre si. Por exemplo, se um cheque é emitido (uma entrada na conta referente ao banco), um registro do mesmo valor deve ser feito na conta referente ao objeto do pagamento, como uma fatura a pagar. Se uma entrada não é feita de maneira apropriada, de modo que as contas não fechem, isto pode ser detectado através de um teste independente (quando é feito um balanço dos livros). Desta forma, é possível detectar fraudes simples como a emissão indevida de cheques (em valor diferente da fatura, por exemplo).

O segundo mecanismo de controle de erros e fraudes, a **separação de tarefas**, tenta garantir a consistência externa dos dados, ou seja, que os dados registrados no sistema reproduzam fielmente a situação de seus correspondentes no mundo real. Como computadores geralmente não possuem sensores para monitorar o mundo real, eles não podem verificar a consistência externa diretamente. Em vez disso, a consistência é garantida indiretamente separando-se todas as operações em várias etapas e requerendo que cada etapa seja executada por uma pessoa diferente. Por exemplo, o processo de aquisição e pagamento de mercadorias pode envolver diversas etapas, como autorização da ordem de compra, registro da entrada da mercadoria, registro da entrada da fatura e pagamento da fatura. O último passo não pode ser executado se os três anteriores não tiverem sido executados corretamente. Se cada subtarefa for desempenhada por uma pessoa diferente, as representações externa e interna devem estar de acordo, a menos que algumas destas pessoas conspirem. Se uma pessoa pode realizar todas as etapas, então pode ocorrer uma fraude bastante simples em que um pedido é emitido e um pagamento é efetuado para uma empresa fictícia sem nenhum envio de mercadorias. Neste caso, as contas aparentemente fecharão (isto é, a saída de dinheiro corresponde a um ingresso de mercadorias), mas o erro só será detectado quando for comparado o inventário do estoque real com os registros contábeis.

2.5.2 Regras do Modelo Clark-Wilson

No modelo Clark-Wilson, os dados do sistema (que correspondem ao conceito de objetos definido na seção 2.1.4) são divididos em dois conjuntos disjuntos. O primeiro contém itens de dados **restritos** (IDRs), que são os dados considerados íntegros, enquanto que o segundo contém os itens de dados **irrestritos** (IDIs), que são aqueles considerados sem integridade.

A política de integridade desejada é definida por duas classes de procedimentos, os **procedimentos de verificação de integridade** (PVIs) e os **procedimentos de transformação** (PTs). O objetivo de um PVI é confirmar que todos os IDRs no sistema estão em conformidade com a especificação de integridade no momento em que o PVI é executado. Na analogia contábil, isto corresponde a uma auditoria, onde é feito o balanço e a conciliação dos livros contábeis com o ambiente externo. Um PT é uma transação bem-formada que tem o propósito de levar o conjunto $\mathcal{R}$ de IDRs de um estado válido a outro, e que é análogo a uma transação pelo método de partidas dobradas em um sistema contábil. A validade de PVIs e PTs só pode ser determinada através da sua certificação em relação a uma política específica de integridade. Por exemplo, no caso contábil, cada PT deveria receber uma certificação de que ele implementa transações que levam a registros contábeis efetuados de acordo com o método de partidas dobradas. A função de certificação é normalmente uma operação manual, muito embora possa ser assistida por procedimentos automatizados; ela envolve técnicas como revisões de código e validação de especificações formais.

A garantia de integridade é um processo composto por duas partes [12]: a certificação, que é feita pelo administrador de segurança (*security officer*) ou administrador do sistema, e a implantação (*enforcement*), que é feita pelo sistema. As regras do modelo CW, portanto, são divididas em regras de (C)ertificação e regras de (I)mplantação.

A consistência interna do sistema pode ser garantida basicamente através das seguintes regras:

C1: Todos os PVIs devem garantir que todos os IDRs estejam em um estado válido quando o PVI for executado.

C2: Todos os PTs devem ser certificados como válidos, ou seja, devem levar um IDR a um estado final válido (desde que ele inicie em um estado válido). Para cada PT e cada conjunto de IDRs que este PT pode manipular, o administrador de segurança deve especificar uma relação de execução. Uma relação, portanto, tem a forma (PT_i, (IDR_a, IDR_b, IDR_c, ...)), onde a lista de IDRs define um conjunto particular de argumentos para os quais o PT foi certificado.

I1: O sistema deve manter a lista de relações especificadas (regra C2), e deve garantir que um IDR só pode ser manipulado por um PT correspondente, onde o PT opera sobre o IDR como especificado em uma relação.

Para garantir a consistência externa através do mecanismo de separação de tarefas, são necessárias regras adicionais para controlar quais pessoas podem executar quais programas em IDRs específicos:

I2: O sistema deve manter uma lista de relações que relacionam um usuário, um PT e os objetos de dados que o PT pode acessar em nome do usuário. Estas listas, portanto, têm a forma (ID-Usuário, PT_i, (IDR_a, IDR_b, IDR_c, ...)). O sistema deve garantir que apenas as execuções descritas nas relações sejam efetuadas.

C3: A lista de relações em I2 deve possuir uma certificação de que preenche o requisito de separação de tarefas.

Formalmente, as relações especificadas na regra I2 são mais completas que as especificadas na regra I1, de modo que I1 é desnecessária. Entretanto, tanto por razões filosóficas quanto por razões práticas, é útil que se tenha ambos os tipos de relações [12].

A relação em I2 faz uso de ID-Usuário, que é um identificador que representa um usuário do sistema. Isto faz com que seja necessária uma regra que defina estes identificadores:

I3: O sistema deve autenticar todos os usuários que tentam executar um PT.

A maior parte dos sistemas que consideram a questão da integridade requerem que todas as atividades sejam registradas em um *log* para propiciar uma trilha de auditoria (*audit trail*). Entretanto, nenhuma regra especial de implantação é necessária para prover este recurso; o *log* pode ser modelado como outro IDR, ao qual todos os PTs adicionam registros. A única regra necessária é

C4: A certificação dos PTs deve garantir que eles escrevam em um IDR *append-only* (o *log*) todas as informações necessárias à reconstrução da natureza das suas operações.

Há mais um componente crítico definido neste modelo. As regras acima consideram somente os IDRs. Os IDIs também devem ser levados em conta, pois constituem a forma pela qual novas informações entram no sistema. Para tanto, é necessário que existam certos PTs que aceitem IDIs como entrada e que modifiquem ou criem IDRs baseados nestas informações, o que implica na seguinte regra de certificação:

C5: Qualquer PT que aceita um IDI como entrada deve ser certificado a fazer apenas transformações válidas, ou nenhuma transformação, para qualquer valor possível do IDI. A transformação deve levar a entrada de um IDI para um IDR, ou o IDI é rejeitado.

Para que este modelo seja eficaz, as várias regras de certificação não podem ser contornadas. Por exemplo, se um usuário puder criar e executar um novo PT sem que este seja certificado, o sistema não poderá atingir seus objetivos. Por esta razão, o sistema deve satisfazer a seguinte regra:

I4: Apenas o agente capaz de certificar PTs e PVIs pode alterar a lista destas entidades associadas com usuários e/ou IDRs. Um agente que pode certificar uma entidade não pode ter direito de execução sobre esta entidade.

Esta última regra define que o mecanismo de implantação de integridade é obrigatório, não discricionário. Para que esta estrutura funcione corretamente, a capacidade de modificar listas de permissões deve estar vinculada à capacidade de certificação e não a alguma outra capacidade (como a de executar um PT). Esta vinculação é a característica fundamental que garante que o comportamento do sistema seja efetivamente governado pelas regras de certificação.

2.5.3 Limitações do Modelo Clark-Wilson

Em conjunto, as nove regras do modelo CW definem um modelo de sistema que implanta uma política de integridade consistente. O principal objetivo deste modelo, na verdade, foi o de estimular a discussão e o desenvolvimento de melhores sistemas e ferramentas de segurança voltados ao ambiente comercial [34]. De fato, a publicação do modelo Clark-Wilson reacendeu o interesse da comunidade acadêmica na área de proteção e modelagem de integridade.

O método de separação de tarefas, de importância capital no modelo, é eficaz como um mecanismo de garantia de integridade, a menos que exista um complô entre vários funcionários. Embora isso possa parecer arriscado, o método possui, comprovadamente, bastante eficácia no controle prático de fraudes. De fato, a separação de tarefas pode ser bastante poderosa se a técnica for aplicada criteriosamente; por exemplo, uma seleção aleatória dos conjuntos de pessoas que desempenharão cada tarefa pode reduzir a possibilidade da existência de complôs a um valor probabilisticamente seguro [34].

Uma das regras do modelo, a regra I2, demonstra a principal diferença entre o modelo CW e os modelos Bell-LaPadula e Biba. Enquanto os modelos baseados em reticulados definem restrições sobre o acesso de sujeitos a objetos, o modelo CW particiona os objetos em programas e dados, e a regra I2 exige triplas de acesso do tipo sujeito/programa/dados (as triplas-CW). Estas triplas-CW previnem a modificação de dados por usuários não-autorizados e implantam o conceito de separação de tarefas.

Em que pesem estas vantagens, o modelo CW enfrenta algumas dificuldades de natureza prática. A principal delas é que os PVIs e as técnicas para garantir que os PTs preservam a integridade não são facilmente implementadas em sistemas computacionais. É perfeitamente possível conceber a sua implementação em aplicações restritas, mas em aplicações menos triviais o uso de PVIs e PTs é muito mais complexo. Outro aspecto problemático do modelo Clark-Wilson é o impacto no desempenho do sistema causado pela implementação das triplas-CW, que pode ser proibitivo [34].

2.6 Modelos Baseados em Papéis

Modelos baseados em papéis são uma tendência que ganhou impulso na última década. Esta seção apresenta brevemente as características gerais destes modelos de modo a permitir a sua comparação destes com os modelos considerados clássicos. O capítulo 3 discute estas e outras características com maior profundidade.

Modelos baseadas em papéis regulam o acesso dos usuários à informação com base nas atividades que os usuários executam no sistema. Estes modelos requerem a identificação de **papéis** no sistema, onde um papel pode ser definido como um conjunto de atividades e responsabilidades associadas a um determinado cargo ou função. Desta forma, em vez de especificar o conjunto de acessos autorizados

para cada usuário do sistema, as permissões são conferidas aos papéis; os usuários, então, são autorizados a exercer um ou mais papéis. Um usuário que exerce um papel pode realizar todos os acessos para os quais o papel está autorizado. Em geral, o usuário ativa (exerce) apenas um subconjunto dos papéis aos quais está autorizado; esta ativação de papéis pode ou não estar sujeita a restrições.

Os modelos baseados em papéis possuem diversas características importantes, tais como [54, 59]:

- **Gerência de autorizações mais simples**: a especificação de autorizações é dividida em duas partes, associação de direitos de acesso a papéis e associação de papéis a usuários. Isso simplifica bastante a gerência da segurança, facilitando tarefas como ajustar os direitos de acesso de um usuário em função de uma promoção ou transferência de setor na organização.
- **Suporte a hierarquias de papéis**: em muitas aplicações existe uma hierarquia natural de papéis baseada nas noções de generalização e especialização. Isto permite que permissões sejam herdadas e compartilhadas através da hierarquia.
- **Suporte a mínimo privilégio**: os papéis permitem que um usuário trabalhe com o mínimo privilégio exigido para uma determinada tarefa. Usuários autorizados a exercer papéis poderosos só precisam exercê-los quando forem absolutamente necessários, minimizando a possibilidade de danos por causa de erros inadvertidos.
- **Suporte a separação de tarefas**: os modelos baseados em papéis suportam separação de tarefas (vide seção 2.5.1). Nestes modelos, a separação de tarefas é obtida através de restrições à autorização e/ou à ativação de papéis considerados mutuamente exclusivos.
- **Delegação da administração de segurança**: modelos baseados em papéis permitem que a administração da segurança seja descentralizada de maneira controlada. Isto significa que o administrador de segurança pode delegar parte de suas atribuições de acordo com a estrutura organizacional ou com a arquitetura do sistema computacional, permitindo, por exemplo, que administradores regionais gerenciem a segurança dos subsistemas locais.

2.7 Comparação entre os Modelos

A diversidade de origem e objetivos dos diferentes modelos de segurança não permite que eles seja 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 2.1 ilustra as principais diferenças entre os modelos de controle de acesso discutidos neste capítulo.

Conforme mostra a tabela, o modelo matriz de acesso e modelos de papéis são considerados flexíveis, pois permitem definir uma diversidade de políticas de segurança. O modelo Clark-Wilson não

Característica	Modelo				
	matriz de acesso	**BLP**	**Biba**	**CW**	**papéis**
Flexibilidade	sim	não	não	não	sim
Política de controle de acesso	descentralizada	centralizada	centralizada	centralizada	centralizada
Administração da política	descentralizada	centralizada	centralizada	centralizada	centralizada ou descentralizada
Separação de tarefas	não	não	não	sim	sim
Suporte a mínimo privilégio	não	sim	sim	não	sim
Suporte à hierarquia organizacional	não	não	não	não	sim
Facilmente implementável	sim	médio	médio	não	sim

Tabela 2.1: Comparação entre os modelos de segurança

pode ser considerado flexível por impor que a política de segurança inclua separação de tarefas (seção 2.5.2). Outro critério de comparação é a forma como a política de controle de acesso é definida. Nos modelos obrigatórios e baseados em papéis, esta política é definida de forma centralizada pela administração de segurança do sistema, através de rótulos de segurança, triplas-CW ou associações usuário-papel e papel-permissão. No modelo matriz de acesso, porém, a política fica descentralizada, uma vez que normalmente as informações de controle de acesso ficam armazenadas junto aos objetos (ACLs) ou junto aos sujeitos (*capabilities*). Como são os próprios sujeitos que determinam quais os acessos que outros sujeitos possuem sobre os seus objetos, diz-se que a administração da política de segurança no modelo matriz de acesso é descentralizada. Os modelos obrigatórios exigem a presença no sistema de um administrador de segurança (*security officer*), que é assumido como único. Modelos de papéis também consideram a figura do administrador de segurança, mas as tarefas deste administrador podem ser delegadas, descentralizando a admnistração de segurança do sistema.

A separação de tarefas é um conceito suportado somente pelos modelos Clark-Wilson ou baseados 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 criteriosos 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 2.1 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 todos as estruturas em um sistema que precisam ser rotuladas para evitar as violações de confidencialidade (BLP) ou integridade (Biba). O modelo Clark-Wilson é o mais complexo em termos de implementação. Como já foi discutido na seção 2.5.3, a construção e certificação de PTs e PVIs é uma tarefa muito difícil em sistemas reais.

2.8 Conclusões do Capítulo

Modelos de segurança são uma importante ferramenta para a definição e especificação 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.

Este capítulo procurou fornecer um apanhado dos trabalhos mais relevantes na área de modelos de segurança computacional, destacando modelos considerados clássicos na área e que ainda hoje são largamente utilizados nos mais diversos ambientes. Foi apresentada a essência destes modelos, analisando-se criticamente a sua validade e adequação, bem como alguns aspectos de sua implementação.

O capítulo encerrou com uma descrição das características gerais de modelos baseados em papéis e uma comparação destes com os modelos clássicos. Nesta comparação pode se perceber claramente que políticas de papéis possuem diversas vantagens sobre os modelos clássicos, sendo uma importante tendência na área de controle de acesso em sistemas computacionais.

Capítulo 3

Controle de Acesso Baseado em Papéis

O conceito de **controle de acesso baseado em papéis** (*role-based access control*—RBAC) surgiu com os primeiros sistemas computacionais multiusuários interativos, no início da década de 70. A idéia central do RBAC é que permissões de acesso são associadas a papéis, e estes papéis são associados a usuários. Papéis são criados de acordo com os diferentes cargos ou funções em uma organização, e os usuários são associados a papéis de acordo com as suas responsabilidades e qualificações. Os usuários podem ser facilmente remanejados de um papel para outro. Mudanças no ambiente computacional, como instalação de novos sistemas e remoção de aplicações antigas, modificam o conjunto de permissões atribuídas aos diferentes papéis.

Apesar desta idéia estar presente sob diversas formas ao longo dos últimos trinta anos, foi só recentemente que a pesquisa na área de controle de acesso baseado em papéis veio a ganhar impulso, com diversos modelos tendo sido propostos na década de 90 [19, 40, 56, 57].

Este capítulo introduz alguns conceitos sobre o RBAC, apresenta em detalhes o modelo unificado RBAC-NIST e discute alguns outros modelos e experiências de implementação de RBAC.

3.1 Introdução

Nos últimos anos, o RBAC vem sendo reconhecido como uma forma de controle de acesso diferenciada dos tradicionais esquemas discricionário e obrigatório. Esta seção discute as razões que levaram a este crescimento recente da importância do controle de acesso baseado em papéis.

Estudos conduzidos nos Estados Unidos pelo NIST (*National Institute of Standards and Technology*) no início da década de 90 [18] mostram que boa parte das organizações deseja que o controle de acesso à informação seja feito segundo uma política centralizada e de forma flexível, possibilitando fácil adaptação a novos requisitos que surgem naturalmente com a evolução organizacional.

Claramente, estes objetivos são difíceis de serem atingidos simultaneamente usando controles discricionários (flexíveis mas descentralizados) ou obrigatórios (centralizados porém inflexíveis). O controle de acesso baseado em papéis, por sua vez, é capaz de satisfazer estes requisitos de maneira efetiva [56]. Outro fator importante é que, em um ambiente organizacional, a informação geralmente pertence à própria organização e não aos seus membros (usuários); o controle de acesso é feito segundo as funções de cada usuário e não segundo uma "propriedade" da informação [17, 19]. Os organismos envolvidos com a definição de padrões também reconhecem a importância do RBAC: o suporte a papéis está previsto no padrão SQL3 [24] para bancos de dados (em desenvolvimento)[1] e também nos Critérios Comuns para avaliação de segurança [10, 48].

O RBAC é **independente de política**, diferente do que acontece, por exemplo, com as formas tradicionais de controle de acesso.[2] Essa independência de política traz consigo uma grande flexibilidade e também facilidade de ajuste do controle de acesso à medida em que ocorram mudanças no ambiente. Ainda que independente de política, o RBAC possibilita garantir três princípios clássicos de segurança: o **princípio de mínimo privilégio** (o usuário só utiliza os direitos de acesso necessários para a realização de uma dada tarefa), a **separação de tarefas** (através de restrições à ativação de papéis), e a **abstração de dados** (pois não há restrições quanto à natureza das permissões, que podem ser abstratas, tais como débito e crédito em um objeto conta).[3]

Este princípio da abstração de dados evidencia a adequação do uso do RBAC em sistemas baseados em objetos, pois é possível estabelecer o controle com uma granularidade apropriada definindo como permissão a invocação de métodos dos objetos [4]. Isto pode ser implementado tanto em sistemas autocontidos como em arquiteturas abertas, como o CORBA no contexto de seu modelo de segurança [6].

Atualmente, a maior parte das implementações com RBAC embute o controle de acesso diretamente nas aplicações. Entretanto, um tópico bastante promissor na área é viabilizar a implementação do RBAC em suportes de propósito geral, como sistemas operacionais e *middlewares*, de forma a oferecer as vantagens de esquemas de controle de acesso baseados em papéis e, ao mesmo tempo, minimizar a necessidade de adaptações nas aplicações [56].

3.2 O Modelo Unificado RBAC-NIST

Nos últimos anos, diversos modelos de controle de acesso baseado em papéis foram propostos e implementados [11, 16, 41, 56]. Estes modelos foram propostos de maneira independente, sem

[1] Uma análise do RBAC em sistemas comerciais de bancos de dados pode ser encontrada em [11].

[2] Tanto o controle obrigatório quanto o discricionário impõem uma política de segurança. No MAC, fluxos de informação contrários a um determinado sentido no reticulado de rótulos de segurança são proibidos; no DAC, a política imposta é que o dono do objeto é quem determina os seus direitos de acesso.

[3] É importante salientar que a obediência a estes princípios depende da configuração adequada do RBAC por parte do administrador de segurança; o RBAC não tem, por si só, como garantir tais princípios [56].

buscar a padronização das características do RBAC. Isso significa que o RBAC é um conceito que ainda carece de uma definição precisa. Uma das formas de disseminar o desenvolvimento e utilização da tecnologia de controle de acesso baseado em papéis é o estabelecimento de padrões; o **modelo unificado RBAC-NIST** é um passo nessa direção [57]. Ele representa uma primeira iniciativa de estabelecer um consenso sobre as características fundamentais do RBAC e um modelo de referência para pesquisadores e implementadores de RBAC.

Ainda que recente, o modelo RBAC-NIST reflete um amadurecimento da compreensão e modelagem do RBAC por parte de dois grupos de pesquisa líderes na área de RBAC, o grupo do NIST e o grupo liderado por Ravi Sandhu, da George Mason University. Este modelo não é, em absoluto, definitivo, mas é um excelente ponto de partida para uma padronização do conhecimento na área de controle de acesso baseado em papéis.

3.2.1 Estrutura do Modelo

O RBAC é um conceito bastante amplo e aberto, que abrange um espectro de complexidade que vai desde o muito simples até o extremamente sofisticado. Existe um consenso de que um único modelo definitivo para o RBAC estaria fora da realidade, uma vez que seria demasiadamente restritivo ou excessivamente complexo, representando apenas um dos pontos possíveis dentro do espectro [56, 57]. Uma abordagem mais realista passa pela definição de uma **família de modelos**, que parte de um componente básico que contempla as características fundamentais do RBAC e passa por componentes adicionais que acrescentam funcionalidade e requisitos ao modelo básico. A família de modelos RBAC96 [54, 56] é, possivelmente, o exemplo mais conhecido desta linha; o modelo RBAC-NIST também compartilha desta abordagem.

A família RBAC-NIST define quatro modelos:

- RBAC Básico (*Flat RBAC*)
- RBAC Hierárquico (*Hierarchical RBAC*)
- RBAC com Restrições (*Constrained RBAC*)
- RBAC Simétrico (*Symmetric RBAC*)

Existem duas interpretações possíveis para a hierarquização do modelo RBAC-NIST. Na primeira, a ordenação dos modelos é total, ou seja, cada modelo possui todas as características do anterior e ainda características adicionais. A segunda interpretação, que é mais poderosa e que será adotada como padrão, reconhece o RBAC Básico e o RBAC Hierárquico como uma seqüência, mas trata o RBAC com Restrições e o RBAC Simétrico como requisitos independentes, não-ordenados.[4]

[4] As justificativas por trás da dupla interpretação e da preferência por aquela que não segue uma ordenação total podem ser encontradas em [57].

As seções seguintes discutem cada um dos modelos que compõem o modelo RBAC-NIST.

3.2.2 RBAC Básico

O RBAC Básico está mostrado na figura 3.1. O modelo possui três conjuntos de entidades: usuários ($\mathcal{U}$), papéis ($\mathcal{R}$, de *roles*) e permissões ($\mathcal{P}$). Um usuário neste modelo é uma pessoa ou um processo agindo em nome dela; este conceito corresponde ao conceito de sujeito apresentado na seção 2.1.4. Um papel é uma função ou cargo dentro da organização e que possui uma semântica que representa a autoridade e a responsabilidade conferidas aos membros desse papel. Uma permissão é um direito específico de acesso a um ou mais objetos no sistema; a natureza exata de uma permissão é altamente dependente da implementação e não é especificada pelo modelo RBAC-NIST [57].

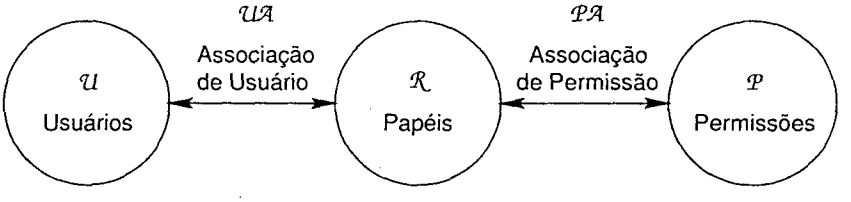


Figura 3.1: RBAC Básico

O RBAC Básico exige que a associação usuário-papel ($\mathcal{UA}$, de *user-role assignment*) e a associação permissão-papel ($\mathcal{PA}$, de *permission-role assignment*) sejam relações do tipo muitos-para-muitos (isto é indicado pelas setas cheias na figura 3.1). Este é um aspecto essencial do RBAC. O conceito de sessão, por sua vez, não faz parte do RBAC Básico. Uma sessão corresponde a um usuário acessando o sistema utilizando um conjunto de papéis [54]. A semântica exata de uma sessão é dependente de implementação. Em alguns sistemas, todos os papéis de um usuário são ativados em uma sessão; em outros, o usuário ativa e desativa os papéis que deseja utilizar. Esta última abordagem permite que o usuário exerça o princípio do mínimo privilégio, ativando apenas os papéis necessários à execução de determinada tarefa. Quando todos os papéis são ativados, por outro lado, o princípio do mínimo privilégio dificilmente pode ser respeitado.

O último requisito do RBAC Básico é o de suporte à revisão usuário-papel, que possibilita determinar, de maneira eficiente, quais os usuários associados a um papel e a quais papéis um usuário está associado. Um mecanismo para revisão permissão-papel também é de grande valia no RBAC; entretanto, considerando a dificuldade intrínseca de se implementar este mecanismo em sistemas distribuídos de larga escala, ele é requerido somente no RBAC Simétrico [57].[5]

[5]Cabe salientar que o requisito de suporte a revisões no modelo RBAC-NIST refere-se a um suporte **eficiente**; ainda que revisões permissão-papel eficientes sejam ditas do âmbito do RBAC Simétrico, deve ser possível identificar associações permissão-papel nos outros modelos, mesmo que este processo seja altamente ineficiente.

3.2.3 RBAC Hierárquico

O RBAC Hierárquico está mostrado na figura 3.2. A única diferença desta figura para a figura 3.1 é a inclusão da relação de hierarquia de papéis ($\mathcal{RH}$, de *role hierarchy*). O suporte a hierarquias de papéis é uma característica bastante conhecida do RBAC, sendo freqüentemente citado como uma de suas principais motivações.

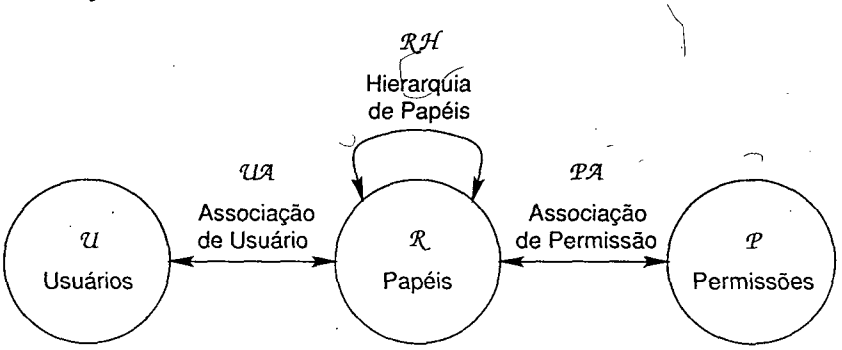


Figura 3.2: RBAC Hierárquico

Hierarquias de papéis constituem uma forma natural de estruturar os papéis de forma a refletir as linhas de autoridade e responsabilidade em uma organização. Estas hierarquias são representadas matematicamente por relações de ordem parcial [56], ou grafos acíclicos dirigidos [16]. Muitas implementações de RBAC restringem a estrutura das hierarquias de papéis. Embora hierarquias arbitrárias estejam mais próximas da realidade, a larga utilização de hierarquias limitadas levou a uma subdivisão do RBAC Hierárquico:

- **RBAC Hierárquico Geral**: uma hierarquia de papéis pode constituir qualquer tipo de ordem parcial.
- **RBAC Hierárquico Limitado**: quando existe qualquer restrição em relação à estrutura da hierarquia de papéis. Geralmente, isto significa que hierarquias são limitadas a estruturas simples como árvores ou árvores invertidas.

O RBAC Hierárquico exige que o mecanismo de revisão usuário-papel do RBAC Básico seja estendido para suportar hierarquias de papéis. Neste modelo, o mecanismo de revisão deve 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).

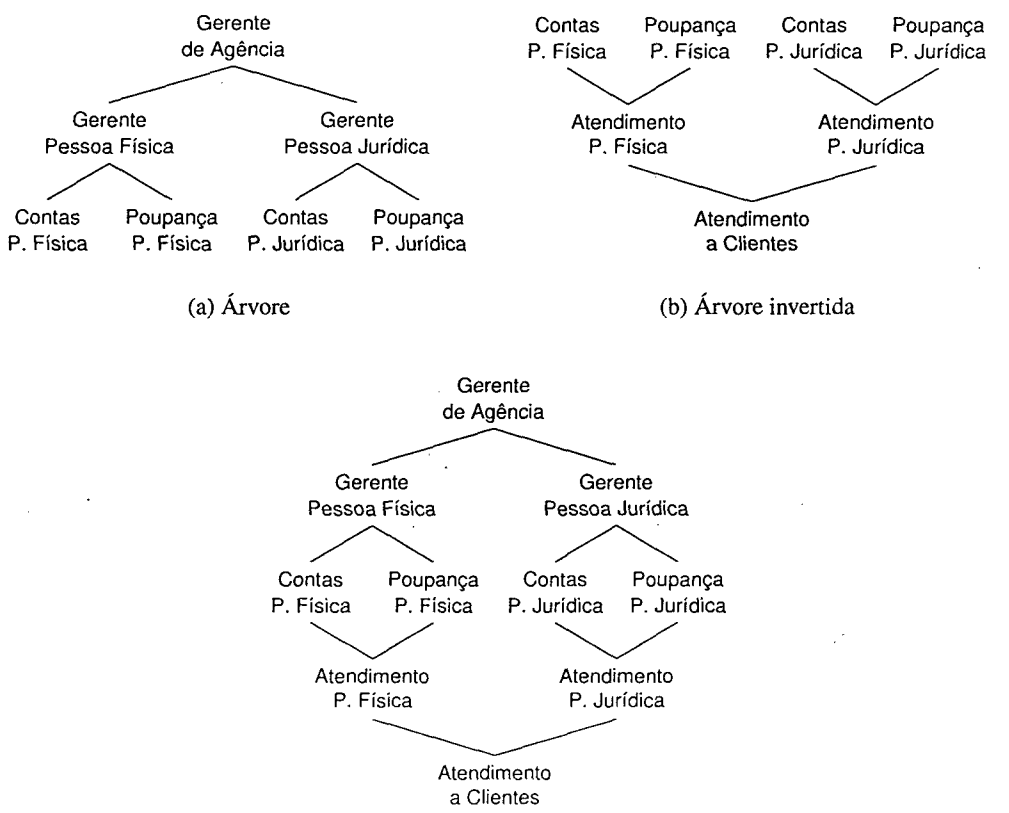
3.2.3.1 Hierarquias Gerais e Limitadas

A figura 3.3(a) mostra um exemplo de hierarquia em árvore que poderia existir em um banco fictício. Nesta hierarquia, os papéis mais privilegiados (isto é, com mais permissões) ocupam posições

mais altas; estes papéis herdam as permissões dos papéis localizados abaixo deles na árvore. No exemplo, o Gerente Pessoa Física tem, além de suas próprias permissões, as permissões herdadas de Contas P. Física e Poupança P. Física. O Gerente de Agência tem as suas permissões e mais as de Gerente Pessoa Física e Gerente Pessoa Jurídica. Este tipo de hierarquia permite a agregação de permissões de diferentes papéis em um outro papel; entretanto, ela não possibilita o compartilhamento de permissões através de um papel.

A figura 3.3(b) mostra um exemplo de hierarquia em árvore invertida. Aqui, Contas P. Física e Poupança P. Física compartilham permissões de Atendimento P. Física, que, por sua vez, compartilha com Atendimento P. Jurídica as permissões de Atendimento a Clientes. Como pode ser visto, árvores invertidas são boas para o compartilhamento de permissões através de papéis, mas não permitem a agregação de permissões de vários papéis.

A figura 3.3(c) mostra um exemplo de hierarquia geral. Neste caso, tanto a agregação quanto o compartilhamento de permissões são possíveis. Cabe salientar que, na prática, as hierarquias de papéis tendem a ser bem mais irregulares do que a hierarquia simétrica da figura 3.3.



(a) Árvore

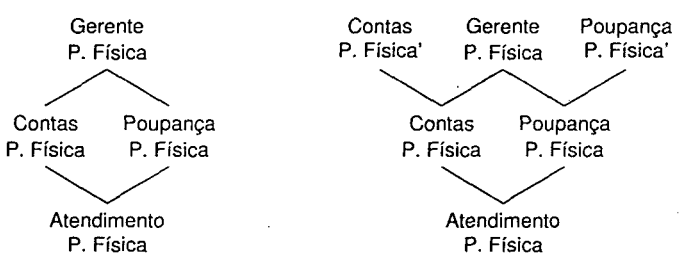
(b) Árvore invertida

(c) Hierarquia geral

Figura 3.3: Exemplos de hierarquias de papéis

3.2.3.2 Herança Limitada

Papéis no topo da hierarquia, tais como Gerente de Agência na figura 3.3(c), são por vezes considerados perigosos pelo fato de concentrarem muito poder [56]. Mesmo que os usuários associados a esses papéis sejam bastante confiáveis, os danos causados por um erro que eles venham a cometer ou pela ação de um *software* malicioso que eles venham a utilizar são, potencialmente, muito grandes. É possível limitar a herança de permissões em hierarquias de papéis; a figura 3.4 mostra como isso pode ser feito. Na figura 3.4(a), Gerente P. Física herda todas as permissões dos papéis inferiores. Por outro lado, na figura 3.4(b) os atendentes de contas de pessoas físicas podem ter permissões no papel Contas P. Física' que não são herdadas por Gerente P. Física. Os usuários são, na verdade, associados ao papel Contas P. Física'; o papel Contas P. Física serve somente para conter as permissões que devem ser compartilhadas. Papéis como Contas P. Física' são chamados **papéis privados**. O papel Poupança P. Física' também é um papel privado, e a discussão acima aplica-se igualmente a ele.



(a) Ilimitada (b) Limitada

Figura 3.4: Exemplo de herança limitada

3.2.3.3 Hierarquias de Herança e de Ativação

Existem duas interpretações distintas para uma hierarquia de papéis discutidas na literatura. Na primeira delas, papéis superiores (*senior roles*) herdam as permissões dos papéis inferiores (*junior roles*); neste caso, tem-se uma **hierarquia de herança**. Se a figura 3.3(c) for vista como uma hierarquia de herança, quando o papel Gerente Pessoa Física é ativado ele herda todas as permissões de Contas P. Física, Poupança P. Física, Atendimento P. Física e Atendimento a Clientes.

Na outra interpretação para a hierarquia de papéis, a ativação de um papel superior não implica na 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 [53]. O modelo RBAC-NIST não define uma interpretação específica para as hierarquias de papéis [57].

3.2.4 RBAC com Restrições

Uma das características mais desejadas de um esquema de controle de acesso é o seu suporte ao princípio da separação de tarefas (ST) [18].[6] Vários modelos de RBAC suportam separação de tarefas sob uma ou outra forma [19, 41, 56]. Na família RBAC-NIST, a separação de tarefas é suportada pelo modelo RBAC com Restrições, mostrado nas figuras 3.5 e 3.6.

É importante salientar que, para que seja possível respeitar a separação de tarefas, é necessário atender o princípio do mínimo privilégio na definição dos papéis, ou seja, para que a separação de tarefas seja atingida através do RBAC, os papéis têm que estar associados ao mínimo de permissões necessárias ao cumprimento de suas tarefas.

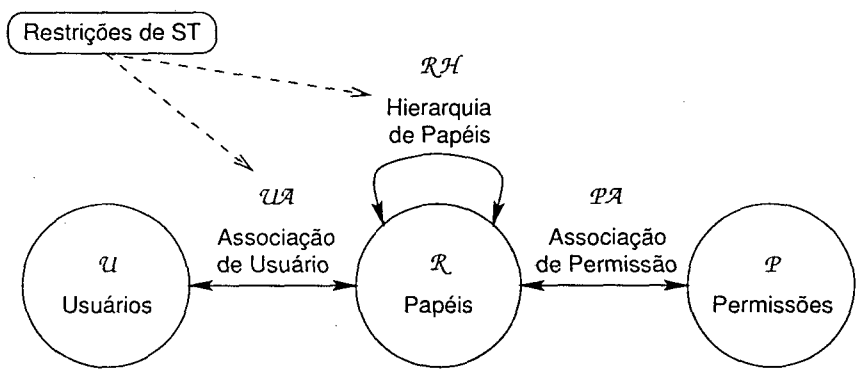


Figura 3.5: RBAC com Restrições—separação estática de tarefas

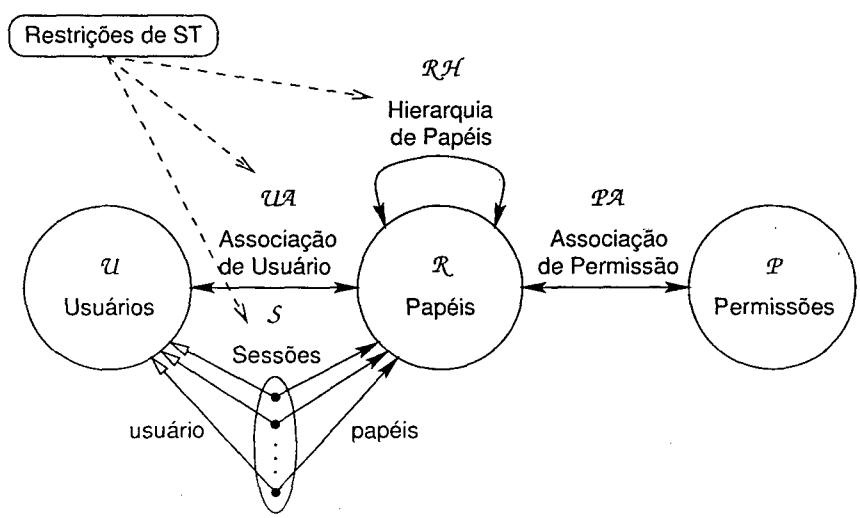


Figura 3.6: RBAC com Restrições—separação dinâmica de tarefas

[6] A separação de tarefas é discutida em detalhes na seção 2.5.1.

3.2.4.1 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 [57]. Uma forma de prevenir esta forma de conflito de interesse é através da separação estática de tarefas (figura 3.5), ou seja, impondo restrições à associação de usuários a papéis. Nesta abordagem, usuários associados a um papel não podem ser associados a um segundo papel, de acordo com restrições definidas pela administrador de segurança. Por exemplo, o papel Compras (associado aos funcionários do setor de compras) pode ser definido como mutuamente exclusivo em relação ao papel Almoxarifado (vide figura 3.7). Esta restrição impede que um funcionário desonesto use o papel Compras para forjar uma requisição de compra de um produto e, a seguir, usando o papel Almoxarifado, faça um registro fraudulento de entrada do produto no estoque do almoxarifado, o que lhe possibilitaria embolsar o valor referente à compra.

Cabe notar que estas restrições se propagam através de uma hierarquia de papéis. Por exemplo, se o papel Supervisor de Compras herda o papel Compras, então ele também é mutuamente exclusivo em relação a Almoxarifado; o mesmo vale para Compras em relação a Chefe de Almoxarifado e também para este em relação a Supervisor de Compras. Como um papel superior é, na verdade, uma instância de um papel inferior, não pode existir ST estática entre eles. No exemplo da figura 3.7, não faz sentido estabelecer uma exclusão entre Supervisor de Compras e Compras, já que, por definição, não pode haver conflito de interesse entre eles. Caso houvesse, a relação de herança não deveria ter sido usada [16, 57].

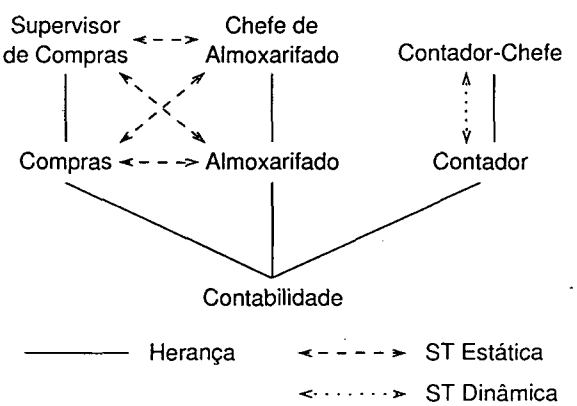


Figura 3.7: RBAC com Restrições—exemplo de ST

3.2.4.2 Separação Dinâmica de Tarefas

O RBAC com Restrições também permite o uso de políticas de separação dinâmica de tarefas (figura 3.6).[7] Nesta abordagem, os usuários podem ser associados a papéis que só constituem um

[7]Para a separação dinâmica de tarefas, é necessário usar o conceito de sessão. Uma sessão corresponde a um usuário acessando o sistema com um determinado conjunto de papéis ativos. O conjunto S de sessões tem um relacionamento

conflito de interesse quando ativados simultaneamente, isto é, é perfeitamente possível e seguro que um usuário ative mais de um papel do conjunto, desde que esta ativação não seja simultânea. No exemplo da figura 3.7, um mesmo usuário pode ser associado aos papéis Contador e Contador-Chefe, onde o Contador lança os registros contábeis e o Contador-Chefe pode efetuar correções em lotes de lançamentos ainda não consolidados. Se um usuário agindo como Contador tentasse ativar o papel Contador-Chefe, ele teria que desativar primeiro o papel Contador—forçando a consolidação dos lançamentos—antes que o indivíduo assumisse o papel Contador-Chefe. Desde que o mesmo usuário não possa ativar ambos papéis ao mesmo tempo, não há conflito de interesse. Embora seja possível obter a exclusão através de uma separação estática de tarefas, a separação dinâmica é, geralmente, mais flexível.

Pode-se observar também que, ao contrário do que acontece na separação estática, a separação dinâmica de tarefas pode ser perfeitamente aplicada a papéis que tenham uma relação de herança [57].

3.2.5 RBAC Simétrico

Um dos requisitos do RBAC Básico é o suporte à revisão de associações usuário-papel. O RBAC Simétrico possui como único requisito o suporte à revisão de associações permissão-papel, tornando possível identificar, de maneira eficiente, as permissões associadas a um papel e, também, quais os papéis que possuem determinadas permissões. É importante frisar que, no RBAC Simétrico, o desempenho de revisões permissão-papel deve ser comparável ao desempenho de revisões usuário-papel.

Revisões permissão-papel são tão importantes quanto revisões usuário-papel para o gerenciamento de segurança de um sistema. Além disso, a possibilidade de identificar com facilidade as permissões atribuídas a um papel (e vice-versa) é um dos principais fatores que diferenciam papéis de grupos. Todavia, as dificuldades inerentes à determinação de associações papel-permissão em ambientes distribuídos de larga escala constituíram um fator determinante para que este requisito tenha sido incluído em um modelo que não o RBAC Básico [57]. Na verdade, se for considerada a interpretação ordenada dos modelos que compõem o RBAC-NIST, revisões permissão-papel são consideradas o requisito mais avançado, o que reflete a potencial complexidade de sua implementação.

3.2.6 Configurações do Modelo RBAC-NIST

Conforme discutido na seção 3.2.1, existem duas interpretações possíveis para os quatro modelos que compõem o modelo RBAC-NIST. Na interpretação original, existe uma ordenação total entre os

muitos-para-muitos—representado pelas setas cheias na figura 3.6—com o conjunto $\mathcal{R}$ de papéis (uma sessão possui vários papéis ativos, e um papel pode estar ativo em várias sessões) e um relacionamento um-para-muitos—representado pelas setas vazadas— com o conjunto $\mathcal{U}$ de usuários (um usuário pode ter várias sessões, mas cada sessão corresponde a apenas um usuário).

modelos; nesta interpretação, existem sete configurações possíveis, mostradas na tabela 3.1.

Nível	Modelo	Hierarquia	Restrições	Revisão Permissão-Papel
1	Básico	—	Não	Não
2a	Hierárquico	Geral	Não	Não
2b	Hierárquico	Limitada	Não	Não
3a	com Restrições	Geral	Sim	Não
3b	com Restrições	Limitada	Sim	Não
4a	Simétrico	Geral	Sim	Sim
4b	Simétrico	Limitada	Sim	Sim

Tabela 3.1: Configurações do modelo RBAC-NIST: ordenação total dos modelos

A interpretação não-ordenada permite, além das configurações da tabela 3.1, cinco configurações adicionais, o que reflete a superioridade desta abordagem. A tabela 3.2 mostra todas as configurações do modelo RBAC-NIST segundo a interpretação não-ordenada.

Modelo Base	Hierarquia	Restrições	Revisão Permissão-Papel
Básico	—	Não	Não
Básico	—	Sim	Não
Básico	—	Não	Sim
Básico	—	Sim	Sim
Hierárquico	Geral	Não	Não
Hierárquico	Limitada	Não	Não
Hierárquico	Geral	Sim	Não
Hierárquico	Limitada	Sim	Não
Hierárquico	Geral	Não	Sim
Hierárquico	Limitada	Não	Sim
Hierárquico	Geral	Sim	Sim
Hierárquico	Limitada	Sim	Sim

Tabela 3.2: Configurações do modelo RBAC-NIST: modelos não-ordenados

3.2.7 Limitações do modelo RBAC-NIST

O modelo unificado RBAC-NIST, como todos os modelos de controle de acesso, possui algumas limitações. Muitas delas derivam do fato deste modelo objetivar a padronização na área de RBAC, o que implica na omissão de características importantes para uma implementação de controle de acesso baseado em papéis. Esta seção discute algumas destas limitações.

3.2.7.1 Ativação Seletiva de Papéis

O modelo RBAC-NIST não exige que os usuários tenham a possibilidade de selecionar quais os papéis que devem ser ativados em uma determinada sessão; o único requisito é que múltiplos papéis possam ser ativados simultaneamente. O entendimento aqui é que o mecanismo através do qual papéis são ativados é dependente de implementação, constituindo um elemento diferenciador entre implementações distintas.

3.2.7.2 Restrições

O modelo RBAC-NIST reconhece apenas restrições de separação de tarefas (tanto estática como dinâmica). Outros modelos de RBAC suportam outros tipos de restrições, como restrições de cardinalidade (determinados papéis podem ter um número máximo de usuários associados) [16] e precondições (um usuário só pode ser membro de um determinado papel se ele satisfizer determinadas condições) [54]. Embora estes tipos de restrições sejam igualmente válidos, não existe ainda consenso sobre a sua necessidade; desta forma, considerou-se prematuro incluí-los em um modelo de referência como o modelo RBAC-NIST [57].

3.2.7.3 Administração do RBAC

O modelo RBAC-NIST não especifica como os seus componentes são administrados. Diversos modelos administrativos para o RBAC foram propostos, com diferentes abordagens para o problema [36, 37, 55, 58]. Devido à ausência de consenso sobre o assunto, o modelo RBAC-NIST não incorpora um componente administrativo.

3.3 Outros Modelos RBAC

Embora o conceito básico de papel venha sendo utilizado há décadas como um mecanismo para a gerência de permissões, modelos RBAC formais começaram a surgir há pouco tempo. Esta seção revisa alguns dos principais modelos RBAC publicados na literatura.

Todos estes modelos compartilham um núcleo comum de conceitos, mas cada um deles possui particularidades que os diferenciam. Entretanto, há visivelmente bem mais semelhanças do que diferenças. Isto levou o NIST e o grupo liderado por Sandhu a propor o modelo unificado RBAC-NIST, que é baseado largamente no modelo do NIST (seção 3.3.1) e na família RBAC96 (seção 3.3.2). Embora alguns aspectos do modelo unificado tenham sido contestados [25], a proposta foi bem recebida pela comunidade de pesquisa na área de papéis, e uma revisão do modelo está atualmente sendo elaborada [15].

3.3.1 O Modelo do NIST

O primeiro modelo RBAC formalizado na literatura foi o de Ferraiolo e Kuhn [19], do NIST. Este modelo passou por revisões, e alguns de seus conceitos foram atualizados ao longo do tempo [16, 17].

O modelo do NIST é um modelo único, e não uma família como o modelo unificado. Em termos de suas características, este modelo é bastante similar ao RBAC com Restrições da família RBAC-NIST; a principal diferença é que o modelo do NIST possui suporte a restrições de cardinalidade.

3.3.2 A Família RBAC96

O trabalho de Sandhu *et. al.* [56] foi o primeiro a reconhecer a impossibilidade de capturar todas as nuances do RBAC em um único modelo, o que levou à definição da família RBAC96 de modelos. Esta família, a exemplo do modelo unificado RBAC-NIST, possui quatro modelos, numerados $RBAC_0$ a $RBAC_3$. Os modelos são basicamente os mesmos do modelo unificado na interpretação não ordenada; na família RBAC96, contudo, o conceito de sessões é introduzido no modelo básico ($RBAC_0$) [54, 56].

Os modelos $RBAC_2$ e $RBAC_3$ suportam restrições sobre papéis, que incluem, além da separação de tarefas presente no RBAC-NIST, restrições de cardinalidade e de precondições (um usuário U só pode ser associado a um papel R1 se já estiver associado também a um papel R2, ou uma permissão P1 só pode ser associada a um papel R se uma outra permissão P2 já estiver também associada a R). A família RBAC96 conta, ainda, com um modelo administrativo associado, o ARBAC97 [54, 55].

3.3.3 O Modelo do Grafo de Papéis

O modelo do grafo de papéis foi proposto por Nyanchama e Osborn [40, 41], e apresenta alguns pontos de interesse que o diferenciam um pouco dos modelos do NIST e da família RBAC96. Em primeiro lugar, este modelo, como o seu próprio nome sugere, baseia-se na teoria de grafos como formalismo subjacente; isto permitiu o desenvolvimento de algoritmos poderosos e conceitualmente simples para a administração do esquema papéis. Hierarquias de papéis são intrínsecas à estrutura do grafo, não requerendo nenhuma modelagem extra. Além disso, o grafo de papéis permite uma representação mais fácil de políticas baseadas em reticulados de rótulos de segurança do que modelos como a família RBAC96 ou o modelo unificado RBAC-NIST [46, 47].

Nyanchama e Osborn mostraram como o grafo de papéis pode resolver diversos tipos de conflito de interesse ou separação de tarefas [41]. Além das tradicionais restrições de exclusão mútua de papéis (estática e dinâmica), eles identificaram e modelaram restrições de exclusão mútua de privilégios, onde determinadas permissões não podem ser associadas a um mesmo usuário.

3.4 Experiências de Implementação de RBAC

Diversas experiências de implementação de RBAC são conhecidas, tanto comerciais quanto acadêmicas. As principais implementações comerciais, até o momento, são em sistemas gerenciadores de bancos de dados (SGBDs) [11]. A seguir, são descritas duas experiências acadêmicas que possuem uma relação mais estreita com o nosso trabalho.[8]

O RBAC/Web [16] é uma aplicação *intranet* em que o RBAC é utilizado como esquema de autorização para controlar o acesso a páginas de um servidor Web. O modelo RBAC utilizado como referência é o modelo do NIST (vide seção 3.3.1). A aplicação em si não utiliza controles criptográficos; em vez disso, assume-se que o servidor Web (que é independente da aplicação) utiliza o protocolo SSL. Os usuários no RBAC/Web correspondem a *logins* no servidor Web, e as transações HTTP que podem ser executadas pelos usuários (através dos seus papéis) nas páginas controladas pelo RBAC representam as permissões. A ativação de papéis é feita pelo próprio usuário, que interage com o RBAC/Web para selecionar, dentre os papéis aos quais ele está associado, aqueles que ele deseja ativar. O RBAC/Web conta, ainda, com uma *interface* de administração do esquema RBAC, que possibilita, entre outros recursos, a visualização gráfica da hierarquia de papéis.

A proposta JRBAC-Web [21] também tem por objetivo a segurança de servidores Web, mas difere do RBAC/Web por centrar-se no modelo de segurança Java. O RBAC é implementado como uma extensão do JAAS (*Java Authorization and Authentication Service*) [26, 27], e baseia-se em um modelo de referência próprio [22]. Assim como no RBAC/Web, as permissões são representadas pelas transações HTTP. Por outro lado, no JRBAC-Web não há uma vinculação direta entre os usuários do modelo RBAC e entidades externas (como um *login*, por exemplo). Na proposta JRBAC-Web, a ativação de papéis fica a cargo das aplicações (*servlets* Java), sem intervenção por parte do usuário.

3.5 Conclusões do Capítulo

Este capítulo apresentou o controle de acesso baseado em papéis como um modelo que permite definir esquemas de autorização ao mesmo tempo flexíveis e centralizados segundo uma política organizacional de proteção da informação. Esquemas com estas características são um desejo de muitas organizações [18] e são difíceis de serem obtidos com os tradicionais modelos discricionário e obrigatório [56].

O modelo unificado RBAC-NIST, apresentado na seção 3.2, representa uma primeira iniciativa de padronização das características básicas do controle de acesso baseado em papéis. Outros modelos de RBAC possuem características que os distinguem do modelo RBAC-NIST, mas esta padronização

[8] A seção 5.4 contém uma comparação destas experiências com a proposta RBAC-JACOWEB.

representa um passo importante para o estabelecimento de um consenso sobre diversas características fundamentais do RBAC e de um ponto de partida para novos desenvolvimentos na área.

O controle de acesso baseado em papéis, por ser uma tecnologia que só recentemente vem sendo desenvolvida ativamente, carece de esforços em algumas áreas [54]. A primeira delas é a de implementações em suportes genéricos, como *middlewares* e sistemas operacionais. A segunda é a engenharia de papéis, que é a metodologia para a configuração do RBAC e definição da hierarquia de papéis em uma organização. Finalmente, tem-se a área de transição de papéis, ou seja, a migração para o RBAC em coexistência com os modos tradicionais de controle de acesso.

Capítulo 4

O Modelo CORBA de Segurança

O OMG (*Object Management Group*) é um consórcio internacional formado por mais de 800 participantes, entre produtores, usuários e vendedores de *software*, com o objetivo de promover a teoria e prática da tecnologia de orientação a objetos no desenvolvimento de *software*. Para atingir este objetivo, o OMG publica especificações sobre diversos aspectos da orientação a objetos, promovendo padrões abertos que aumentam a reusabilidade, portabilidade e interoperabilidade do *software* orientado a objetos em ambientes distribuídos heterogêneos [44].

A arquitetura OMA (*Object Management Architecture*), desenvolvida pelo OMG, pretende disciplinar a construção de um ambiente para aplicações heterogêneas sobre as principais plataformas operacionais disponíveis. A arquitetura CORBA (*Common Object Request Broker Architecture*) é parte fundamental da arquitetura OMA, definido os meios pelos quais objetos distribuídos podem se comunicar [42, 44, 63].

Dentre os diversos serviços especificados pelo OMG para o ambiente CORBA destaca-se o serviço de segurança CORBA, também conhecido como CORBAsec. O serviço CORBA de segurança foi desenvolvido com o propósito de incorporar características e funcionalidades de segurança a sistemas de objetos distribuídos de pequena e larga escala, sem deixar de lado características-chave do padrão CORBA como transparência, interoperabilidade e portabilidade.

Este capítulo introduz os padrões OMG/CORBA através das arquiteturas OMA e CORBA. Em seguida, são discutidos o serviço de segurança CORBA e o modelo CORBA de segurança, com especial atenção para os seus aspectos de controle de acesso.

4.1 A Arquitetura OMA

A arquitetura OMA provê uma infra-estrutura conceitual sobre a qual todas as especificações do OMG são baseadas [42]. Conforme mostra a figura 4.1, esta arquitetura é composta por cinco

componentes:

- ***Object Request Broker* (ORB)**: é o componente central da arquitetura OMA e corresponde à arquitetura CORBA propriamente dita. O ORB é o mecanismo básico através do qual os objetos se comunicam entre si, enviando requisições e recebendo respostas. Isto é feito de maneira transparente, isolando o objeto cliente de detalhes como os mecanismos e protocolos de comunicação, a localização do objeto servidor e como ele foi implementado.

- **Serviços de objeto**: são objetos independentes do domínio da aplicação, que implementam serviços de propósito geral que fornecem um suporte básico ao desenvolvimento de aplicações distribuídas. Os serviços de objeto, ou serviços CORBA, atuam em conjunto com o ORB para prover a infra-estrutura necessária às aplicações. Exemplos destes serviços incluem o serviço de nomes, o serviço de segurança e o serviço de eventos.

- **Facilidades comuns** (*common facilities*): assim como os serviços CORBA, são objetos que podem ser utilizados por várias aplicações; a diferença é que as facilidades implementam funções de mais alto nível, mais próximas das aplicações dos usuários, como suporte a agentes móveis, *workflow* e correio eletrônico.

- ***Interfaces* de domínio**: exercem o mesmo papel dos serviços de objeto e das facilidades comuns, mas são *interfaces* orientadas a domínios de aplicação específicos, como telecomunicações ou finanças.

- **Objetos de aplicação**: objetos que implementam a lógica de negócios de uma determinada aplicação. Como estes objetos são específicos a cada aplicação, suas *interfaces* não são padronizadas pelo OMG.

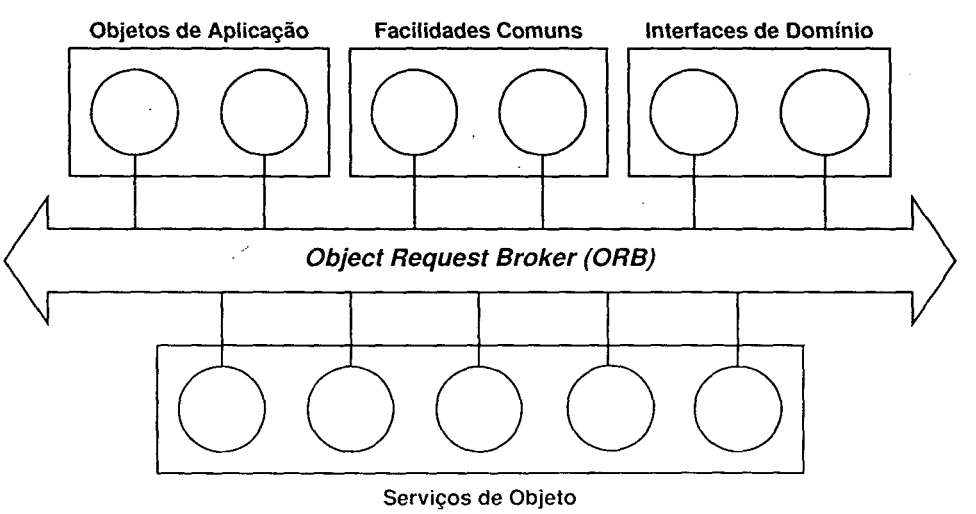


Figura 4.1: Componentes da arquitetura OMA

4.2 A Arquitetura CORBA

O padrão CORBA consiste um uma arquitetura orientada a objetos para sistemas distribuídos heterogêneos que provê uma infra-estrutura que possibilita que objetos se comuniquem independentemente de sua plataforma ou das técnicas usadas na sua implementação. A arquitetura CORBA define os objetos como as unidades básicas de distribuição, sendo que cada objeto pode suportar uma coleção de operações através de sua *interface* [44].

As *interfaces* dos objetos CORBA são definidas de maneira padronizada através da **IDL** (*Interface Definition Language*) OMG. A definição IDL é o contrato entre o implementador de um objeto e o cliente. A IDL é uma linguagem declarativa fortemente tipada que é independente de linguagem de programação. **Mapeamentos de linguagem** permitem que objetos sejam implementados e recebam requisições na linguagem de programação de escolha do programador utilizando-se de recursos nativos da linguagem [42]. O OMG já publicou mapeamentos para diversas linguagens de programação largamente utilizadas, como C, C++, Java, Python e Ada.

Através de uma arquitetura orientada a objetos com características como herança, interoperabilidade entre ORBs e independência de plataforma, o padrão CORBA preenche alguns dos requisitos de *software* mais importantes hoje em dia, como maximização da portabilidade, reusabilidade e interoperabilidade de *software*. Neste contexto, basta recompilar uma aplicação para que ela possa ser usada com diferentes ORBs. Isto abre caminho, também, para que aplicações em ORBs distintos cooperem entre si [44].

Objetos tradicionais são acessíveis somente em um único programa; o mundo exterior nada sabe a seu respeito. Em contrapartida, objetos CORBA podem ser acessados em qualquer ponto da rede, tanto por clientes locais quanto por clientes remotos, através da invocação de suas operações. A linguagem de programação utilizada para implementar os objetos servidores e o sistema operacional onde eles estão sendo executados são totalmente transparentes para os clientes, que precisam conhecer apenas a interface que estes servidores implementam. Os clientes utilizam **referências de objetos** para identificar objetos CORBA. Estas referências são válidas em todo o sistema e podem ser passadas de um nó a outro. Os papéis de cliente e servidor exercidos pelos objetos são transitórios, sendo efetivos apenas pelo tempo de duração de uma requisição; freqüentemente, o servidor em uma requisição passa a ser o cliente em outra e vice-versa [63].

Os componentes da arquitetura CORBA, mostrados na figura 4.2, são os seguintes:

- **Implementações de Objeto:** as implementações de objeto ou servidores definem as operações que implementam as *interfaces* IDL do CORBA em uma linguagem de programação específica.
- **Clientes**: os clientes são as entidades que invocam as operações das implementações de objeto. O acesso aos serviços de um objeto remoto deve ser transparente para os clientes, e deve ser tão simples quanto uma chamada a um método de um objeto local.

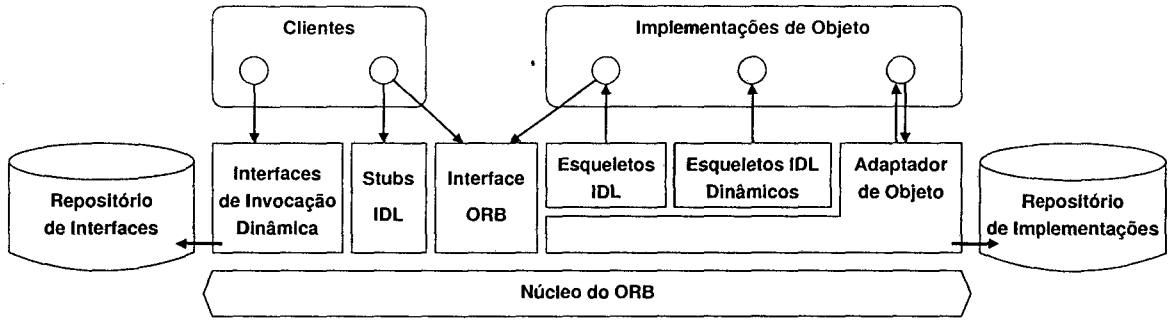


Figura 4.2: Componentes da arquitetura CORBA

- **Núcleo do ORB**: o núcleo do ORB provê mecanismos para transmitir de forma transparente as requisições de um cliente para a implementação de objeto adequada. Quando um cliente invoca uma operação de um servidor, o núcleo do ORB, em conjunto com outras estruturas, é responsável por encontrar a implementação de objeto adequada, ativá-la de forma transparente e retornar a resposta da solicitação ao cliente.

- ***Interface* ORB**: para separar as aplicações dos detalhes de implementação do ORB, a especificação CORBA definiu uma *interface* abstrata para este. Esta *interface* provê várias funções de auxílio, como funções para converter referências de objetos para cadeias de caracteres (e vice-versa) e para criar listas de argumentos para requisições feitas através da DII.

- ***Stubs* IDL** ou ***Interfaces* de Invocação Estática**: o *stub* IDL é o mecanismo que efetivamente cria e envia as requisições em nome de um cliente até o núcleo do ORB, que é o encarregado de localizar a implementação do objeto adequado e ativá-la. Os *stubs* descrevem os serviços fornecidos pelas implementações de objeto.

- **Esqueletos IDL Estáticos**: o esqueleto IDL é um mecanismo que entrega as requisições dos clientes à implementação do objeto CORBA. Em conjunto, os *stubs* e os esqueletos permitem, através do núcleo do ORB, a comunicação entre os clientes e as implementações de objeto.

- ***Interface* de Invocação Dinâmica (DII—*Dynamic Invocation Interface*)**: usando a DII um cliente pode invocar requisições diretamente ao objeto destino sem ter conhecimento, em tempo de compilação, das *interfaces* dos objetos. Aplicações usam uma *interface* de invocação dinâmica para emitir pedidos aos objetos sem requerer um *stub* IDL específico.

- **Esqueletos IDL Dinâmicos (DSI—*Dynamic Skeleton Interface*)**: análogos à DII, os esqueletos IDL dinâmicos permitem ao núcleo ORB entregar requisições a uma implementação de objeto que não tem conhecimento, em tempo de compilação, do tipo de objeto que está implementando. O cliente que faz uma requisição não tem idéia se a implementação está utilizando esqueletos IDL estáticos ou dinâmicos.

- **Adaptador de Objeto**: o adaptador de objeto auxilia o núcleo do ORB na entrega de requisições e na ativação de objetos. Ele associa uma implementação de objeto ao ORB de modo a

permitir que esta possa acessar serviços do ORB.

- **Repositórios de *Interfaces***: as *interfaces* de objetos (IDLs) podem ser adicionadas a um repositório de *interfaces*. Para usar as *interfaces* de invocação dinâmica, os clientes necessitam criar uma requisição que deve incluir a referência do objeto, a operação e a lista de parâmetros. As identificações do objeto e da operação podem ser obtidas em um repositório de *interfaces*, que é um banco de dados que fornece armazenamento persistente de definições de *interfaces* de objetos.

- **Repositório de Implementações**: o repositório de implementações contém informações que são usadas pelo ORB e pelo adaptador de objeto para localizar e ativar implementações de objetos em tempo de execução.

4.3 O Serviço de Segurança CORBA

A primeira versão da especificação do serviço de segurança CORBA (CORBAsec 1.1) foi publicada em maio de 1996. Após algumas revisões, a especificação encontra-se atualmente na versão 1.5 [43].

A arquitetura de segurança CORBA foi projetada com o objetivo de satisfazer os seguintes requisitos gerais [31, 43]:

- **Transparência**: a segurança do sistema deve interferir o mínimo possível nas atividades do usuário.

- **Independência de mecanismos**: o CORBAsec deve ser independente da tecnologia de segurança subjacente.

- **Flexibilidade**: o CORBAsec deve suportar uma variedade de políticas e mecanismos de segurança.

- **Interoperabilidade**: o CORBAsec deve suportar a interoperabilidade entre implementações diferentes, entre ORBs diferentes, entre sistemas com e sem segurança, entre tecnologias de segurança diferentes, e entre domínios de um sistema distribuído (que pode, ainda, suportar diferentes políticas de segurança).

- **Escalabilidade**: o sistema seguro deve se adaptar a sistemas tanto de pequena quanto de larga escala, isto é, deve fornecer suporte ao agrupamento de sujeitos e objetos usando grupos, papéis e domínios de políticas gerenciáveis que podem interoperar.

- **Simplicidade**: o sistema de segurança deve ser fácil de entender, usar e administrar.

- **Desempenho**: a segurança não deve impor um desempenho inaceitável ao sistema.
- **Certificação**: deve ser possível avaliar e certificar a segurança do sistema seguro e dos mecanismos de segurança subjacentes de acordo com critérios de avaliação de segurança padronizados (por exemplo, os Critérios Comuns—ISO 15408 [10]).

A especificação CORBAsec evoluiu de tecnologias de segurança já existentes, como o Serviço de Segurança do DCE (*Distributed Computing Environment*) [49], o protocolo Kerberos para autenticação e criptografia em sistemas distribuídos [38] e o *framework* genérico para acesso a serviços de segurança, a GSS API (*Generic Security Service API*) [32]. O serviço de segurança CORBA cobre os principais aspectos da segurança de sistemas distribuídos através das seguintes funcionalidades:

- Identificação e autenticação de principais;
- Autorização e controle de acesso, para prevenir acessos não-autorizados;
- Auditoria de segurança;
- Segurança da comunicação entre objetos, garantindo autenticação entre as partes, integridade e/ou confidencialidade das mensagens;
- Não-repudiação.

ORBs seguros não precisam suportar todas as funcionalidades descritas acima. A especificação CORBAsec define dois níveis em que os ORBs podem se enquadrar de acordo com a funcionalidade que oferecem: o nível 1, mais elementar, e o nível 2, mais sofisticado.

O **nível 1** de segurança (*Security Level 1*) suporta clientes e servidores que possuem pouca ou nenhuma ciência da presença do serviço de segurança no sistema. As funcionalidades disponíveis neste nível são a autenticação de principais, invocação segura entre cliente e servidor (incluindo estabelecimento da confiança através de autenticação, integridade e/ou confidencialidade dos dados), controle de acesso implantado pelo ORB e auditoria de um conjunto obrigatório de eventos (como autenticação de principais, autenticação de sessão e alterações na política de segurança). A não-repudiação (provisão de evidências irrefutáveis de ações no sistema) é uma funcionalidade opcional no nível 1. O nível 1 possui, ainda, uma *interface* que possibilita a aplicações cientes da segurança implantar suas próprias políticas de segurança em termos de controle de acesso e auditoria.

O **nível 2** de segurança (*Security Level 2*) fornece grande flexibilidade a clientes e servidores cientes da segurança, que interagem com o CORBAsec para controlar diversos aspectos de segurança do sistema. Além das funcionalidades suportadas pelo nível 1, o nível 2 possui outras *interfaces* de aplicação e administração. Facilidades de administração de políticas de segurança são introduzidas neste nível, permitindo que aplicações determinem suas próprias políticas de autorização. Não-repudiação e interoperabilidade segura são funcionalidades opcionais no nivel 2.

4.4 O Modelo CORBA de Segurança

Esta seção apresenta o modelo CORBA de segurança segundo a versão 1.5 da especificação CORBAsec [43]. Como este modelo é bastante complexo, os seus aspectos não essenciais para a compreensão deste trabalho—gerência de domínios de segurança, estabelecimento de associações seguras, auditoria e não-repudiação—são descritos apenas superficialmente ou omitidos. Discussões mais detalhadas destes itens podem ser encontradas em [8, 43, 64, 65].

No modelo CORBA de segurança, um sistema seguro de objetos distribuídos compreende quatro níveis, mostrados na figura 4.3: o nível de aplicação, o nível de *middleware*, o nível de tecnologia de segurança e o nível de proteção básica.

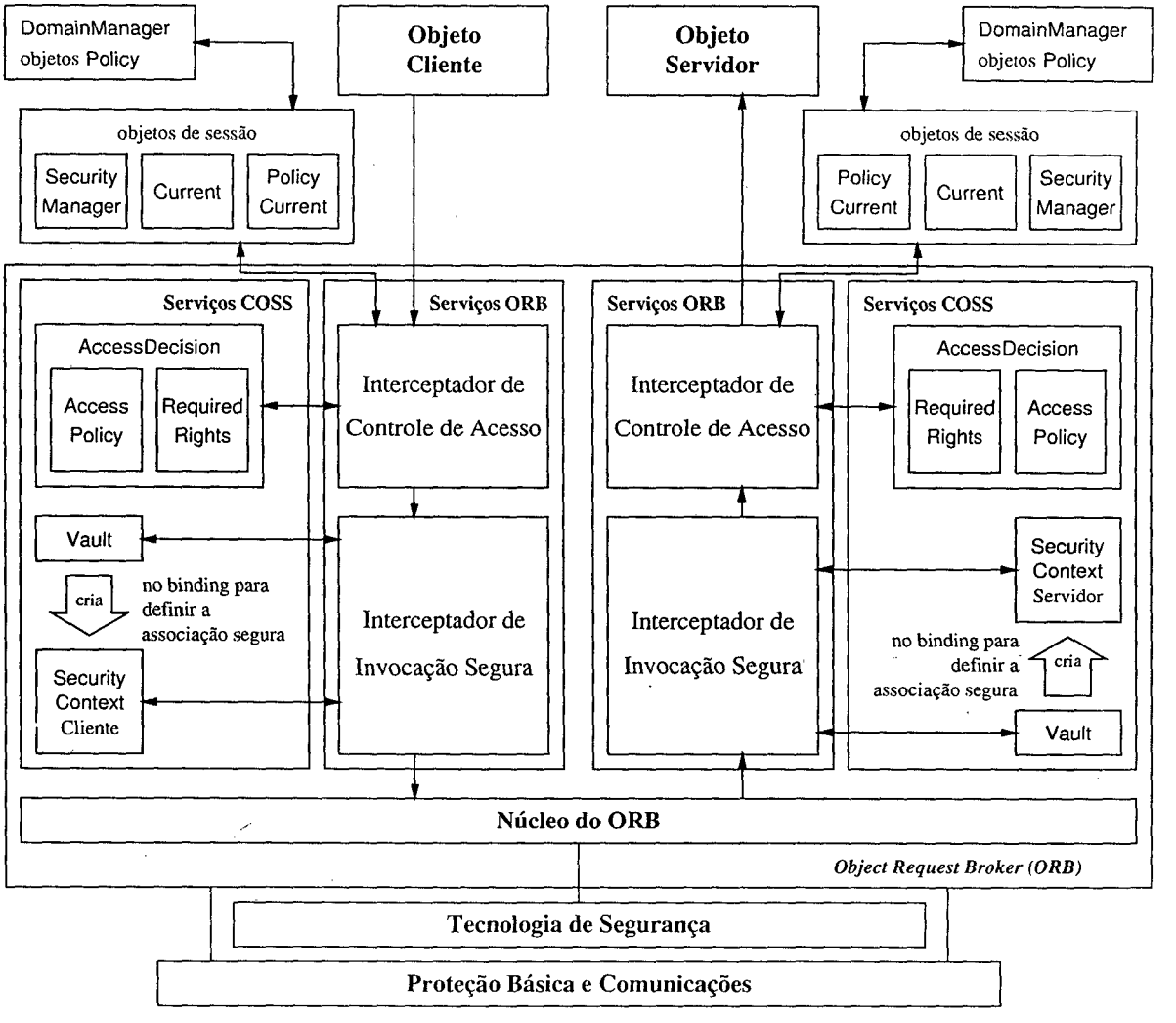


Figura 4.3: Modelo CORBA de segurança

O **nível de aplicação** contém os objetos de aplicação (clientes e servidores). Em alguns casos, as aplicações não são cientes da segurança, delegando ao ORB a tarefa de realizar invocações de forma

segura. Em outros, porém, as aplicações interagem com o serviço de segurança, responsabilizando-se pela administração da sua própria segurança. O modelo CORBA de segurança suporta ambos os tipos de objetos de aplicação.

O **nível de *middleware*** é composto pelos serviços ORB, pelos serviços de objeto (COSS—*Common Object Service Specification*) e pelo núcleo do ORB. Os serviços ORB e os serviços de objeto COSS são construídos sobre o núcleo do ORB e estendem as funções básicas com características adicionais, implementando a segurança de objetos distribuídos no nível de *middleware*. No modelo CORBA de segurança, estes serviços COSS são implementados por **interceptadores**, que são objetos que são logicamente interpostos na seqüência de invocação de um cliente a um servidor. Cada serviço COSS relacionado com a segurança é associado a um interceptador, que tem a finalidade de desviar a chamada de maneira transparente, ativando assim o serviço.

O **nível de tecnologia de segurança** corresponde aos serviços subjacentes de segurança, que definem os protocolos utilizados na implementação de funcionalidades como confidencialidade e integridade na comunicação cliente-servidor. As tecnologias de segurança consideradas, até o momento, para utilização com o CORBAsec são Kerberos, SPKM, SESAME e SSL [2, 43].

O **nível de proteção básica** compreende o sistema operacional e o *hardware* subjacentes, que fornecem meios para proteger os componentes da aplicação uns dos outros e para proteger os componentes que suportam os serviços de segurança no ambiente operacional.

A especificação CORBAsec define um conjunto de serviços de objeto que implementam os controles de segurança em um sistema CORBA seguro (figura 4.3: PrincipalAuthenticator, Credentials, AccessPolicy, RequiredRights, AccessDecision, SecurityManager, Current, PolicyCurrent, Vault e SecurityContext.

4.4.1 Autenticação de Principais

O modelo CORBA de segurança utiliza o conceito de **principal** para mediar as invocações de operações dos objetos. Um principal é o usuário ou entidade do sistema que é registrado no sistema de objetos distribuídos e que tem sua autenticidade perante este sistema estabelecida [43]. O objeto PrincipalAuthenticator é o responsável pela autenticação dos principais na arquitetura CORBAsec. Esta autenticação tem como finalidade possibilitar a obtenção das credenciais do principal. Para as aplicações não cientes da segurança, as credenciais são obtidas de forma transparente.

A **credencial** de um principal contém **atributos de identidade** (que identificam o principal no sistema) e **atributos de privilégio** (que qualificam o principal). Estes atributos são usados para que o principal possa invocar objetos servidores no sistema. Existem diversos tipos de atributos de privilégio, como grupos, papéis e *capabilities*, além do AccessId, que representa a identificação do principal para fins de controle de acesso [43].

Em um sistema CORBA seguro, uma credencial é representada por um objeto Credentials. Normalmente, é utilizada uma credencial para cada mecanismo de segurança empregado por um sistema CORBA seguro. Esta abordagem é usada, por exemplo, no ORBAsec SL2, um ORB seguro que segue a especificação CORBAsec [1]. O ORBAsec SL2 emprega, como mecanismos de segurança, o Kerberos e o SSL, usando credenciais distintas para cada um destes mecanismos.

4.4.2 Domínios e Políticas de Segurança

Um **domínio** é um conjunto de objetos que compartilham características comuns ou que estão sujeitos a um mesmo conjunto de regras. **Políticas de segurança**, no contexto CORBAsec, são as regras e critérios que restringem as atividades dos objetos que formam um domínio de segurança. Cada domínio conta com um objeto DomainManager responsável por gerenciar os objetos de política do domínio.

O modelo CORBA de segurança define diversos tipos de domínios, a saber [43]:

- **Domínios de política de segurança**: aqueles onde os objetos estão sujeitos à mesma política de segurança; existem diversos tipos de políticas de segurança, como política de controle de acesso e política de auditoria.
- **Domínios de ambiente de segurança**: aqueles cujos objetos têm os requisitos da política de segurança garantidos de forma local em seu ambiente de execução, e não através do sistema de objetos. Por exemplo, a cifragem de mensagens pode ser desnecessária quando os objetos residem em uma mesma máquina. Domínios de ambiente não são visíveis para as aplicações ou para os serviços ORB.
- **Domínios de tecnologia de segurança**: aqueles onde todos os objetos utilizam os mesmos mecanismos de segurança, como protocolos para autenticação de principais ou para comunicação criptografada.

4.4.3 Estabelecimento do Contexto de Segurança

Antes que um cliente possa usar uma referência de objeto para invocar uma operação do servidor, é necessário estabelecer uma **associação segura** entre o cliente e o objeto destino. Esta associação segura é estabelecida durante o *binding* entre cliente e servidor.

O **interceptador de invocação segura** (figura 4.3) é e responsável pelo estabelecimento do contexto de segurança necessário para a proteção de mensagens entre cliente e servidor, definindo o que é chamada de associação segura. Para aplicações não cientes da segurança, o estabelecimento da associação segura ocorre de forma completamente transparente.

Um **associação segura** é estabelecida entre cliente e servidor quando [43]:

- É estabelecida a confiança mútua nas identidades do cliente e do servidor;
- As credenciais do cliente (incluindo seus atributos de segurança) estão disponíveis ao servidor;
- É estabelecido o contexto de segurança que será usado na proteção das invocações de métodos e das subseqüentes respostas.

O interceptador de invocação segura interage com o objeto Vault, que, por sua vez, instancia o objeto SecurityContext, concretizando uma associação segura.

O objeto Vault faz parte de um módulo substituível que depende da tecnologia de segurança subjacente, uma vez que é o responsável pelo estabelecimento do contexto de segurança.

O objeto SecurityContext armazena o contexto de segurança (tanto do cliente como do servidor) que será usado em uma associação segura. Este contexto de segurança contém as chamadas **opções de associação**, que determinam parâmetros da associação segura, como mecanismos de segurança, credenciais utilizadas na invocação e qualidade de proteção das mensagens (integridade e/ou confidencialidade, entre outras).

4.4.4 Controle de Acesso

A autenticação (com a subseqüente aquisição de credenciais) e o estabelecimento do contexto de segurança representam apenas o primeiro passo para que um principal possa atuar em um sistema CORBA seguro. Cada operação invocada em um objeto CORBA de passar por um processo de **controle de acesso** ou **autorização**. O modelo CORBA de segurança permite definir políticas de autorização usando mecanismos como listas de controle de acesso (ACLs), listas de *capabilities* e papéis.

No modelo de controle de acesso do CORBAsec, as políticas de autorização podem ser verificadas em dois níveis, o nível de *middleware* e o nível de aplicação. No **nível de *middleware***, o controle é feito pelos serviços de objeto durante uma invocação, de maneira transparente para a aplicação. No **nível de aplicação**, por sua vez, o controle é feito pelas próprias aplicações que, portanto, precisam ser cientes da segurança. Como o controle de acesso no nível de aplicação não é padronizado na especificação CORBAsec (uma vez que cada aplicação possui requisitos próprios de controle de acesso), abordamos aqui apenas o controle de acesso no nível de *middleware*.

No modelo CORBA de segurança, políticas de segurança são expressas em termos dos atributos de segurança de recursos do sistema (atributos de controle) e de principais (atributos de privilégio). As políticas de autorização são representadas na especificação CORBAsec pelo objeto AccessPolicy [43];

um exemplo é mostrado na figura 4.4. Este objeto contém os direitos concedidos aos principais para a invocação de operações no sistema CORBA seguro, com base nos seus atributos de privilégio. Apenas os direitos `g` (*get*), `s` (*set*), `m` (*manage*) e `u` (*use*)—que pertencem à família predefinida `corba`—são previstos na especificação CORBAsec, embora seja possível definir livremente outras famílias e tipos de direitos.

Atributo de Privilégio	Direitos Concedidos
`role: cliente`	`corba: gs--`
`role: cliente`	`corba: g---`
`role: gerente`	`corba: g-mu`
`role: gerente`	`corba: g--u`

Figura 4.4: Exemplo de AccessPolicy

Os direitos requeridos (atributos de controle) para a execução de operações em objetos servidores são armazenados no objeto RequiredRights. Conforme pode ser visto no exemplo da figura 4.5, os direitos requeridos são especificados para *interfaces* (classes no modelo CORBA), e não para instância (objetos individuais). Além disso, existe também um combinador, que indica se um principal precisa possuir todos os direitos requeridos (*All*) para executar uma operação ou se apenas um deles é suficiente (*Any*).

Direitos Requeridos	Combinador	Operação	Interface
`corba: g---`	All	`ver_saldo`	`ContaPFis`
`corba: g---`	All	`ver_saldo`	`ContaPJur`
`corba: -sm-`	Any	`abrir`	`ContaPFis`
`corba: g-m-`	All	`abrir`	`ContaPJur`

Figura 4.5: Exemplo de RequiredRights

O objeto AccessDecision (figura 4.3) é responsável por decidir se a invocação de uma operação de um determinado servidor deve ser autorizada ou não. Esta decisão de acesso depende dos atributos de privilégio (representados pelo objeto AccessPolicy) e dos atributos de controle (representados pelo objeto RequiredRights). A lógica dessa decisão de acesso é deixada em aberto pela especificação CORBAsec, e é dependente do contexto do sistema CORBA seguro utilizado [6, 43].

A decisão de acesso pode ocorrer tanto no lado do cliente quanto no lado do servidor. O normal é que ela seja feita no lado do servidor, embora a verificação no lado do cliente possibilite uma redução no tráfego de rede em caso de negação do acesso.

Os **objetos de sessão**—SecurityManager, PolicyCurrent e Current—armazenam informações sobre o contexto corrente de segurança, como referências aos objetos RequiredRights e AccessDecision (SecurityManager), referências aos objetos de política usados no estabelecimento de associações seguras (PolicyCurrent) e as credenciais do principal obtidas pelo servidor (Current).

No momento da ligação entre cliente e servidor, o **interceptador de controle de acesso** é respon-

sável por instanciar o objeto AccessDecision e atualizar a sua referência no objeto SecurityManager. O objeto AccessDecision, por sua vez, deve disponibilizar o objeto de política AccessPolicy do domínio, inserindo sua referência no objeto PolicyCurrent, e também localizar o objeto RequiredRights, atualizando sua referência no objeto SecurityManager. Em tempo de decisão de acesso, o interceptador de controle de acesso invoca a operação `access_allowed` do objeto AccessDecision, que é responsável por autorizar ou não a execução da operação, obtendo os direitos concedidos (contidos em AccessPolicy) e comparando-os com os direitos requeridos (contidos em RequiredRights).

4.5 Limitações do CORBAsec

O objetivo principal do OMG com o serviço CORBA de segurança foi a definição de um *framework* para a segurança em sistemas de objetos distribuídos CORBA que fosse genérico e interoperável. Este objetivo foi alcançado com sucesso, tendo sido produzida uma especificação bastante completa e, como conseqüência, inevitavelmente complexa, pois trata-se de uma especificação que não define mecanismos específicos de segurança, mas, sim, estruturas genéricas que podem ser adaptadas aos mais variados mecanismos existentes.

O CORBAsec foi o primeiro sistema de segurança para *middlewares* baseados em objetos. Um reflexo deste pioneirismo é uma especificação que ainda necessita de amadurecimento em alguns aspectos. Outros problemas encontrados no modelo CORBA de segurança não são deficiências da especificação, mas dificuldades encontradas na segurança de sistemas de objetos distribuídos em geral [2].

Alguns dos problemas do CORBAsec já estão sendo atacados pelos membros do OMG responsáveis pela especificação, como integração do SSL (as primeiras versões da especificação não descreviam como o SSL deveria ser integrado ao CORBAsec), gerência de políticas de segurança (as *interfaces* atuais para gerência de políticas e domínios de segurança são seriamente limitadas), dependências externas (nem todas as *interfaces* para dependências externas podem ser consideradas estáveis) e questões de portabilidade e interoperabilidade entre ORBs seguros de fornecedores distintos [2, 31].

4.6 Conclusões do Capítulo

As características-chave da tecnologia CORBA (reusabilidade, portabilidade e interoperabilidade) a colocam em uma posição privilegiada como um *framework* para a construção de sistemas distribuídos de larga escala. A disseminação de sistemas CORBA baseados na Internet [45, 61] consolida essa posição e evidencia a adequação da arquitetura CORBA para esta classe cada vez mais popular de sistemas.

A especificação do serviço de segurança CORBA é a maior e mais complexa especificação produzida pelo OMG, o que reflete a importância e a complexidade da segurança em sistemas distribuídos. A despeito de algumas limitações [2, 31], o CORBAsec endereça os requisitos mais importantes da construção de sistemas distribuídos seguros.

Este capítulo teve por objetivo introduzir os principais aspectos dos sistemas distribuídos baseados na tecnologia CORBA, buscando ainda evidenciar as características fundamentais do modelo CORBA de segurança, especialmente no que diz respeito ao seu modelo de autorização e controle de acesso.

Capítulo 5

RBAC para o Modelo CORBA de Segurança

O projeto JACOWEB[1], desenvolvido no LCMI–DAS–UFSC, visa investigar a problemática da autorização em sistemas distribuídos de larga escala. O foco do projeto é a definição e implementação de esquemas de autorização para aplicações distribuídas, utilizando como suporte o modelo CORBA de segurança integrado aos modelos de segurança Java e Web [66]. No contexto do projeto JACOWEB foi desenvolvido o serviço de políticas PoliCap, que gerencia políticas de segurança em um ambiente CORBAsec.

Este capítulo apresenta o projeto JACOWEB e o serviço de políticas PoliCap e introduz a proposta RBAC-JACOWEB, cujo objetivo é incorporar um controle de acesso baseado em papéis baseado no modelo unificado RBAC-NIST ao esquema de autorização JACOWEB.

5.1 O Projeto JACOWEB

O projeto JACOWEB tem como objetivo o uso do modelo CORBA de segurança integrado com os modelos de segurança Java e Web de modo a compor um esquema de autorização para aplicações distribuídas em redes de larga escala. Esse esquema está sendo desenvolvido com o objetivo de concretizar a implantação de políticas globais, o que representa um grande desafio, principalmente em redes de larga escala como a Internet [66, 67, 68]. No JACOWEB, aplicações distribuídas são expressas na forma de clientes representados por *applets* Java e de objetos servidores CORBA. O protocolo SSL é usado para garantir a confidencialidade e a integridade das invocações de métodos através da rede.

[1] A página do projeto está disponível em `http://www.lcmi.ufsc.br/jacoweb`.

Na arquitetura JACOWEB, o controle de acesso é realizado pelo *middleware*, de maneira transparente, tanto no lado do cliente quanto no lado do servidor. Serviços de objeto localizados do lado do cliente verificam se o acesso solicitado está em conformidade com a política de segurança, e se ele deve ou não ser autorizado. Caso o acesso seja autorizado, é gerada uma *capability* que é agregada à requisição (*request*) CORBA e enviada ao servidor; objetos de serviço no lado do servidor extraem e validam a *capability* antes de liberar o acesso. Uma descrição completa do JACOWEB pode ser encontrada em [65].

5.2 O Serviço de Políticas PoliCap

O PoliCap é um serviço de políticas para objetos distribuídos cujas invocações são reguladas segundo o modelo CORBAsec [65, 68]. O PoliCap foi desenvolvido no contexto do projeto JACOWEB e corresponde a um primeiro nível de verificação do controle de acesso no esquema de autorização.

O PoliCap possibilita o gerenciamento centralizado de objetos de política em um domínio de segurança de objetos distribuídos, suprindo a carência na especificação CORBAsec quanto ao gerenciamento de objetos de política. Segundo a especificação, as políticas de segurança são disponibilizadas através dos objetos AccessPolicy e RequiredRights. Aplicações administrativas interagem com o PoliCap para gerenciar estes objetos. Por sua vez, aplicações operacionais ou serviços de objeto interagem com o serviço de política PoliCap para obter, no *binding*, as políticas e os direitos necessários aos controles em tempo de execução sobre as invocações de um método. A idéia é que, no *binding*, o serviço de política forneça versões locais dos objetos AccessPolicy e RequiredRights que contenham os atributos de privilégio e de controle que se aplicam à invocação. A decisão de acesso, então, é efetuada com base nestas instâncias locais dos objetos AccessPolicy e RequiredRights. Maiores detalhes sobre o PoliCap podem ser encontrados em [65, 68].

5.3 A Proposta RBAC-JACOWEB

5.3.1 Integrando Modelos RBAC ao CORBAsec

Na proposta RBAC-JACOWEB, cada principal tem apenas uma credencial (uma vez que o JACOWEB suporta apenas o SSL como tecnologia de segurança). Os papéis associados a um principal são representados, na sua credencial, por atributos de privilégio do tipo Role. Após a autenticação do principal, a credencial contém apenas o seu AccessId, que é um atributo de privilégio que representa a identidade do principal para fins de controle de acesso, e que pode ser extraído do certificado X.509 do cliente (usado no estabelecimento da associação segura SSL). Os papéis são agregados à credencial à medida em que forem ativados dentro do sistema. Portanto, a credencial de um principal

representa, na proposta RBAC-JACOWEB, a sua identificação de acesso e os papéis ativos que ele possui no sistema.

A funcionalidade dos interceptadores de controle de acesso descrita na seção 4.4.4 permanece a mesma. As modificações para incorporar o controle de acesso baseado em papéis são feitas no objeto AccessDecision do cliente, que verifica se os direitos concedidos ao principal permitem acesso à operação solicitada. Com a autorização do acesso, o interceptador de controle de acesso do cliente monta uma *capability* que será agregada à requisição CORBA.

O serviço de políticas PoliCap é estendido, passando a conter todos os dados relativos às políticas de segurança do domínio, incluindo usuários, papéis, associações usuário-papel e papel-permissão,[2] relações de hierarquia de papéis e restrições de separação de tarefas.[3] Conforme será visto em detalhes na seção 5.3.2, o PoliCap é acionado pelo objeto AccessDecision quando a autorização através dos objetos de política locais falha; a figura 5.1 ilustra a interação entre o serviço de políticas e o AccessDecision.

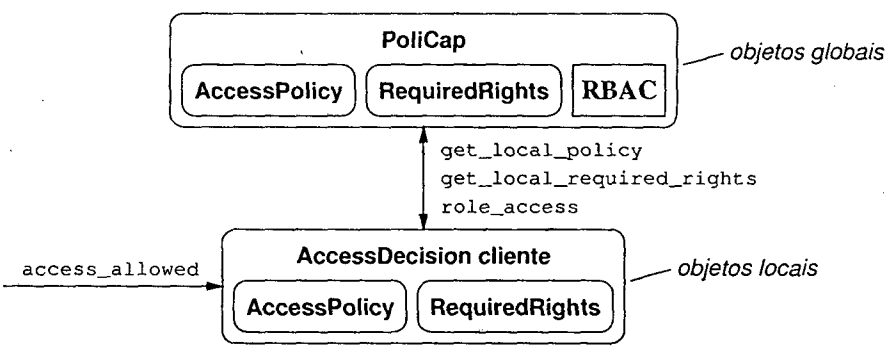


Figura 5.1: Interação entre o AccessDecision e o PoliCap

A operação crucial do PoliCap para o controle de acesso baseado em papéis é `role_access`, cuja *interface* IDL é reproduzida na figura 5.2.

```
boolean role_access(inout SecurityLevel2::CredentialsList cred_list,
                    in CORBA::Identifier operation_name,
                    in CORBA::Identifier target_interface_name,
                    inout SecurityAdmin::AccessPolicy local_ap);
```

Figura 5.2: *Interface* IDL da operação `role_access` do PoliCap

Com base nas credenciais do cliente e na operação invocada, o PoliCap decide se é possível autorizar o acesso ou não. Se o usuário possui um ou mais papéis que lhe conferem as permissões

[2] O conceito de direitos utilizado na especificação CORBAsec é equivalente ao conceito de permissões usado no modelo unificado RBAC-NIST. Desta forma, os dois termos serão utilizados indistintamente neste capítulo e nos subseqüentes.

[3] O apêndice A contém a especificação completa do PoliCap, com a sua *interface* IDL e uma descrição das suas operações.

necessárias à invocação da operação e estes papéis podem ser ativados (isto é, esta ativação não viola nenhuma restrição), o acesso é autorizado, o que é indicado por um valor de retorno `true`. Caso não seja possível ativar papéis que confiram as permissões necessárias, a operação retorna `false`.

Quando o acesso é autorizado, as credenciais do principal são modificadas de forma a incorporar os novos papéis ativados, e o argumento `local_ap` (que representa o objeto AccessPolicy local) é modificado de forma a incorporar os novos direitos concedidos pela ativação dos papéis ao principal.

As restrições de separação estática de tarefas são verificadas pela operação `assign_user_role` do PoliCap, que associa usuários a papéis: um usuário não pode ser associado a um papel que tenha uma restrição de separação estática em relação a outro com o qual ele já esteja associado. Por sua vez, as restrições de separação dinâmica são tratadas pela operação `role_access`. Um papel só pode ser ativado se não tiver restrições de separação dinâmica em relação a qualquer um dos papéis ativos (contidos na credencial). Dessa forma, garante-se que uma ativação automática de papéis não irá violar a política de segurança definida.

5.3.2 Dinâmica do Controle de Acesso usando Papéis

5.3.2.1 Tempo de Ligação (*Binding*)

A ligação ou *binding* ocorre na primeira invocação que o cliente faz para o servidor. Neste instante, conforme mostrado na seção 4.4.4, o interceptador de controle de acesso instancia o objeto AccessDecision, atualizando sua referência no objeto SecurityManager. O AccessDecision deve, então, disponibilizar os objetos de política de autorização que serão utilizados [43].

A figura 5.3 mostra a seqüência de ações executada pelo AccessDecision durante a ligação entre cliente e servidor. Em primeiro lugar, é obtida a referência do serviço de políticas PoliCap através do serviço de nomes CORBA, o CosNaming (*1*). A seguir, é criado o objeto AccessPolicy local (*2*). Em *3*, o AccessDecision utiliza a operação `role_access` do PoliCap para fazer uma verificação global da política, isto é, o PoliCap verifica se o principal possui direitos suficientes para executar a operação solicitada. Em caso de insuficiência de direitos, o acesso é negado, a invocação é interrompida com o retorno de uma exceção para o cliente e o evento é registrado pelo serviço de auditoria do CORBAsec. Se os direitos forem suficientes, a ligação prossegue, com a atualização das credenciais do cliente e do seu objeto AccessPolicy local; neste caso, a referência do AccessPolicy local é atualizada no PolicyCurrent (*4*). No prosseguimento, é criado o objeto RequiredRights local e são obtidos os direitos requeridos para a *interface* alvo, com a subseqüente atualização do RequiredRights local (*6*). Além disso, a referência deste objeto é atualizada no objeto SecurityManager (*7*).

É importante ressaltar que os objetos AccessPolicy e RequiredRights locais só são criados na primeira ligação entre este cliente e um servidor. *Bindings* subseqüentes deste mesmo cliente utiliza-

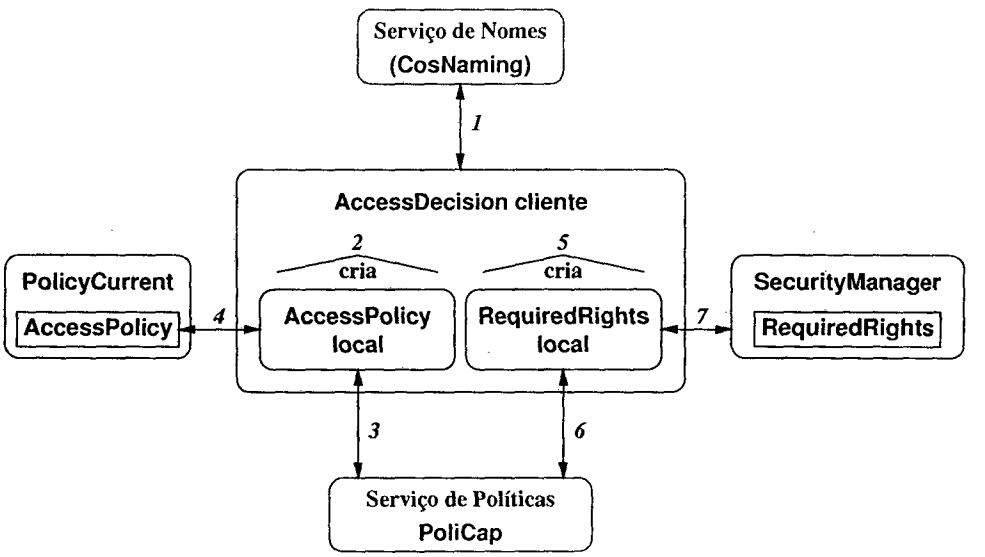


Figura 5.3: Controle de acesso: tempo de ligação (*binding*)

rão os objetos locais previamente instanciados durante a primeira ligação, que permanecem válidos enquanto o cliente estiver ativo no sistema.

5.3.2.2 Decisão de Acesso

Em tempo de decisão de acesso, o interceptador de controle de acesso do lado do cliente invoca a operação `AccessDecision::access_allowed`. A figura 5.4 mostra a seqüência de operações utilizada pelo AccessDecision para efetuar a decisão de acesso. Em primeiro lugar, ele obtém os direitos concedidos ao principal contidos no objeto AccessPolicy local através da operação `get_effective_rights` (*1*). A seguir, são obtidos os direitos requeridos para a invocação através da operação `get_required_rights` do objeto RequiredRights local (*2*). Os direitos concedidos são comparados aos direitos requeridos para verificar se os direitos concedidos ao principal lhe permitem executar a operação desejada. Caso o acesso seja autorizado, a seqüência de invocação prossegue normalmente, com a geração e envio da *capability*.

Por outro lado, em caso de insuficiência dos direitos concedidos, o objeto AccessDecision invoca a operação `role_access` do PoliCap (*3*) para que esta verifique a possibilidade de ativação de novos papéis que confiram os direitos necessários à execução da operação solicitada. Caso o acesso seja autorizado com a ativação de um ou mais papéis, o interceptador de controle de acesso deve, então, gerar a *capability*, dando seqüência à invocação. Se a configuração do esquema RBAC não autoriza o acesso, a seqüência de invocação é interrompida com a negação do acesso e o evento é registrado pelo serviço de auditoria.

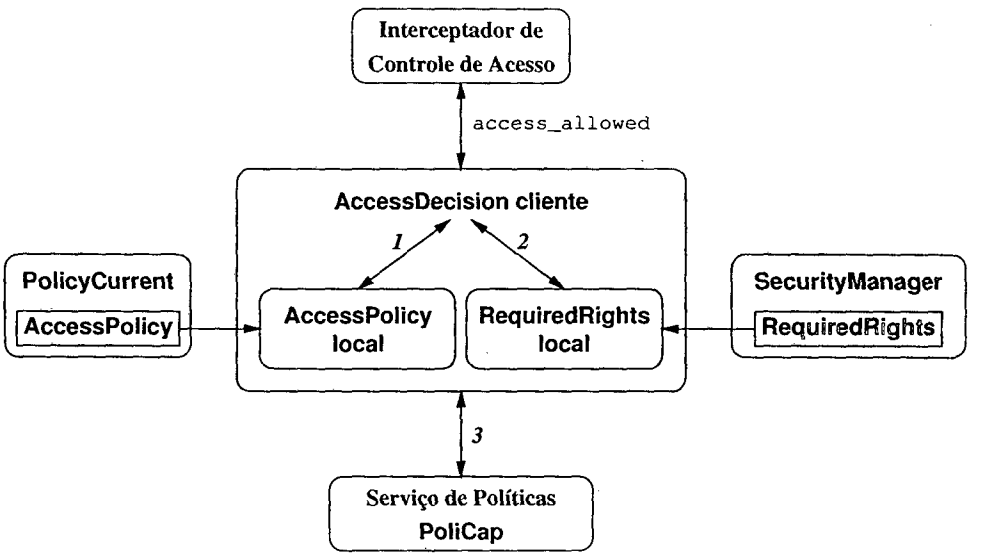


Figura 5.4: Controle de acesso: tempo de decisão de acesso

5.3.3 Exemplo

Consideremos um exemplo de funcionamento do controle de acesso baseado em papéis em uma aplicação bancária. O conjunto de papéis do sistema é $\mathcal{R} = \{\mathsf{cli}, \mathsf{cxpf}, \mathsf{cxpj}, \mathsf{ger}\}$, representando, respectivamente, clientes, caixas para pessoas físicas, caixas para pessoas jurídicas e gerentes. A configuração do esquema RBAC, mantida pelo PoliCap, é mostrada na figura 5.5. Não há restrições de separação estática de papéis. As restrições de separação dinâmica de papéis são dadas pela figura 5.5(a), onde ($\surd$) indica que a linha e a coluna representam papéis mutuamente exclusivos. A figura 5.5(b) representa o AccessPolicy. A figura 5.5(c) mostra a associação entre principais e papéis (conjunto $\mathcal{UA}$). O RequiredRights é representado pela figura 5.5(d).

As figuras 5.6 a 5.9 mostram cenários de funcionamento do controle baseado em papéis proposto, concentrando-se na evolução das ativações de papéis no sistema. Nos exemplos, os principais executam as operações da primeira coluna na seqüência mostrada em cada figura. As colunas $\mathcal{AR}$ *Antes* e $\mathcal{AR}$ *Depois* representam o conjunto de papéis ativos (*active roles*)—armazenado na credencial do cliente—antes e depois da decisão de acesso para a operação correspondente.

A figura 5.6 mostra um cenário normal. O principal `Ana` ativa o papel ger para executar a operação `ContaPJur::abrir` e segue usando os direitos conferidos por esse papel até que tenta invocar `ContaPJur::depositar`, uma operação para a qual nem ger nem qualquer outro papel de `Ana` possuem direitos suficientes.

As figuras 5.7 e 5.8 mostram dois cenários de funcionamento para o principal `Bia`. O cenário da figura 5.7 é bastante similar ao cenário para o principal `Ana` (figura 5.6). É importante atentar para o que acontece com o mesmo principal (`Bia`) no cenário da figura 5.8. Aqui, o PoliCap é forçado a

	cli	cxpf	cxpj	ger
cli	—	√	√	√
cxpf	√	—		√
cxpj	√		—	√
ger	√	√	√	—

(a) ST dinâmica

Atributo de Privilégio	Direitos Concedidos
role: cli	corba: g---
role: cxpf	corba: gs--
role: cxpj	corba: g--u
role: ger	corba: g-m-

(b) AccessPolicy

Principal	Papéis
Ana	cli, cxpj, ger
Bia	cli, cxpf
Cris	cli, cxpf, cxpj

(c) Conjunto $\mathcal{UA}$

Direitos Requeridos	Combinador	Operação	Interface
corba: g---	All	ver_saldo	ContaPFis
corba: g---	All	ver_saldo	ContaPJur
corba: -s--	All	depositar	ContaPFis
corba: ---u	All	depositar	ContaPJur
corba: -sm-	Any	abrir	ContaPFis
corba: g-m-	All	abrir	ContaPJur

(d) RequiredRights

Figura 5.5: Configuração RBAC gerenciada pelo PoliCap

Operação	$\mathcal{AR}$ Antes	$\mathcal{AR}$ Depois	Comentários
ContaPJur::abrir	∅	{ger}	A operação requer os direitos g-m-, conferidos pelo papel ger. Acesso permitido, com a ativação do papel ger.
ContaPJur::ver_saldo	{ger}	{ger}	A operação requer o direito g---, já conferido pelo papel ger. Acesso permitido.
ContaPJur::depositar	{ger}	{ger}	A operação requer o direito ---u, não conferido por nenhum papel do principal Ana. **Acesso negado.**

Figura 5.6: Cenário de funcionamento para o principal Ana

Operação	$\mathcal{AR}$ Antes	$\mathcal{AR}$ Depois	Comentários
ContaPFis::abrir	∅	{cxpf}	A operação requer um dos direitos -sm-, sendo que o papel cxpf confere os direitos gs--. Acesso permitido, com a ativação do papel cxpf.
ContaPFis::depositar	{cxpf}	{cxpf}	A operação requer o direito -s--, já conferido pelo papel cxpf. Acesso permitido.
ContaPFis::ver_saldo	{cxpf}	{cxpf}	A operação requer o direito g---, já conferido pelo papel cxpf. Acesso permitido.
ContaPJur::abrir	{cxpf}	{cxpf}	A operação requer os direitos g-m-, mas nenhum papel do principal Bia lhe confere tais direitos. **Acesso negado.**

Figura 5.7: Primeiro cenário de funcionamento para o principal Bia

Operação	$\mathcal{AR}$ Antes	$\mathcal{AR}$ Depois	Comentários
`ContaPFis::ver_saldo`	∅	{cli}	A operação requer o direito `g---`, conferido pelos papéis cli e cxpf; respeitando o mínimo privilégio, o papel a ativar seria cli, e não cxpf. Acesso permitido, com a ativação do papel cli.
`ContaPFis::depositar`	{cli}	{cli}	A operação requer o direito `-s--`, conferido pelo papel cxpf. Entretanto, este papel é mutuamente exclusivo em relação ao papel (já ativado) cli e, portanto, não pode ser usado. **Acesso negado.**

Figura 5.8: Segundo cenário de funcionamento para o principal `Bia`

fazer uma escolha entre os papéis cli e cxpf, uma vez que ambos conferem o direito `g---` requerido pela operação `ContaPFis::ver_saldo`. Exercendo o princípio do mínimo privilégio, o PoliCap ativa o papel cli. Esta ativação impede que o principal invoque a operação `ContaPFis::depositar`, pois, embora o papel cxpf lhe confira o direito necessário, este papel é mutuamente exclusivo em relação ao papel cli. Dessa forma, o acesso acaba por ser negado. Este aparente paradoxo não deve ser visto como uma deficiência do esquema proposto para implementação de RBAC no JACOWEB, mas sim como uma violação do princípio de mínimo privilégio presente na definição da política (mais especificamente na associação de permissões a papéis).

Operação	$\mathcal{AR}$ Antes	$\mathcal{AR}$ Depois	Comentários
`ContaPFis::abrir`	∅	{cxpf}	A operação requer um dos direitos `-sm-`, sendo que o papel cxpf confere os direitos `gs--`. Acesso permitido, com a ativação do papel cxpf.
`ContaPFis::depositar`	{cxpf}	{cxpf}	A operação requer o direito `-s--`, já conferido pelo papel cxpf. Acesso permitido.
`ContaPJur::depositar`	{cxpf}	{cxpf, cxpj}	A operação requer o direito `--u`, conferido pelo papel cxpj. Acesso permitido, com a ativação do papel cxpj.
`ContaPJur::abrir`	{cxpf, cxpj}	{cxpf, cxpj}	A operação requer os direitos `g-m-`, não conferido por nenhum papel do principal `Cris`. **Acesso negado.**

Figura 5.9: Cenário de funcionamento para o principal `Cris`

O último cenário de funcionamento é o do principal `Cris` (figura 5.9). Neste cenário, o principal liga-se a dois objetos servidores distintos (ContaPFis e ContaPJur) e ativa dois papéis diferentes (cxpf e cxpj).

A seguir, é mostrado em detalhes como os objetos do modelo CORBA de segurança e o PoliCap interagem para realizar o controle de acesso, baseando-se neste último cenário de funcionamento.

5.3.3.1 Execução do cenário da figura 5.9

Quando o sistema CORBA seguro é inicializado, o principal é autenticado através do objeto PrincipalAuthenticator. A sua credencial (objeto Credentials) contém apenas o seu AccessId, extraído de seu certificado SSL:

Tipo de Atributo	Valor do Atributo
AccessId	`Cris`

▷ **Invocação da operação `ContaPFis::abrir`:**

Para que esta operação seja invocada, é necessário fazer o *binding* entre o cliente e o objeto servidor ContaPFis. De acordo com a seção 5.3.2.1, neste momento são obtidos, através do PoliCap, os direitos requeridos para a *interface* ContaPFis e os direitos concedidos para o principal que autorizam a invocação da operação `abrir`. Neste instante, também é instanciado o objeto AccessDecision, atualizando-se sua referência no SecurityManager.

As permissões necessárias para a invocação da operação `abrir` de ContaPFis são `corba: -s--` ou `corba: --m-`, já que o combinador é *Any*. Neste caso, o papel cxpf, por conceder as permissões `corba: gs--`, é então ativado no próprio *binding*, e o acesso é autorizado. O objeto Credentials do principal passa a ter o seguinte conteúdo:

Tipo de Atributo	Valor do Atributo
AccessId	`Cris`
Role	cxpf

O objeto AccessPolicy local passa a ter o conteúdo abaixo:

Atributo de Privilégio	Direitos Concedidos
`role: cxpf`	`corba: gs--`

O objeto RequiredRights local, após o *binding*, é constituído por:

Direitos Requeridos	Combinador	Operação	Interface
`corba: g---`	All	`ver_saldo`	`ContaPFis`
`corba: -s--`	All	`depositar`	`ContaPFis`
`corba: -sm-`	Any	`abrir`	`ContaPFis`

▷ **Invocação da operação `ContaPFis::depositar`:**

Como o *binding* entre o cliente e o objeto servidor ContaPFis já está estabelecido, as verificações em relação à operação `ContaPFis::depositar` se resumem a efetuar o procedimento de decisão de

acesso mostrado na seção 5.3.2.2. Para isso, o interceptador de controle de acesso invoca a operação `AccessDecision::access_allowed`. O direito requerido para a operação é `corba: -s--`, contido no objeto AccessPolicy local; desta forma, o acesso é autorizado, sem modificações nos objetos Credentials e AccessPolicy local.

▷ **Invocação da operação `ContaPJur::depositar`:**

Para a invocação desta operação é necessário estabelecer outro *binding*, desta vez entre o cliente e o objeto servidor ContaPJur. Novamente são obtidos os direitos requeridos (desta vez para a *interface* ContaPJur) e os direitos concedidos que autorizam a invocação da operação `depositar`.

O direito necessário à invocação de `ContaPJur::depositar` é `corba: ---u`, conferido pelo papel cxpj, que é ativado. O acesso é autorizado, e o objeto Credentials é modificado, passando a ter o seguinte conteúdo:

Tipo de Atributo	Valor do Atributo
AccessId	`Cris`
Role	cxpf
Role	cxpj

O objeto AccessPolicy local também é atualizado:

Atributo de Privilégio	Direitos Concedidos
`role: cxpf`	`corba: gs--`
`role: cxpj`	`corba: ---u`

O objeto RequiredRights local passa a ser constituído por:

Direitos Requeridos	Combinador	Operação	Interface
`corba: g---`	All	`ver_saldo`	`ContaPFis`
`corba: -s--`	All	`depositar`	`ContaPFis`
`corba: -sm-`	Any	`abrir`	`ContaPFis`
`corba: g---`	All	`ver_saldo`	`ContaPJur`
`corba: ---u`	All	`depositar`	`ContaPJur`
`corba: g-m-`	All	`abrir`	`ContaPJur`

▷ **Invocação da operação `ContaPJur::abrir`:**

O *binding* entre o cliente e o objeto servidor ContaPJur já está estabelecido, passando-se direto para o procedimento de decisão de acesso. A operação `ContaPJur::abrir` requer os direitos `corba: g-m-`, que não estão presentes no objeto AccessPolicy local. Desta forma, a operação de decisão de acesso `AccessDecision::access_allowed` invoca a operação `role_access` para verificar se os direitos requeridos podem ser conferidos através da ativação de um papel. Todavia, o principal

`Cris` não possui nenhum papel que confira os direitos necessários, de modo que o acesso é negado, sem modificação das credenciais do cliente nem do objeto AccessPolicy local.

Cabe lembrar que, conforme descrito na seção 5.3.2.1, os objetos AccessPolicy e RequiredRights locais são acessados a partir de referências armazenadas nos objetos de sessão PolicyCurrent e SecurityManager, respectivamente; estas referências são válidas para todas as ligações estabelecidas entre o cliente e os servidores durante a sessão, o que explica o fato dos objetos AccessPolicy e RequiredRights serem atualizados (e não instanciados novamente) durante a evolução do sistema.

5.4 Trabalhos Relacionados

Na seção 3.4 foram apresentadas duas experiências de implementação de RBAC. Estas experiências, RBAC/Web [16] e JRBAC-Web [21], podem agora ser comparadas com a proposta RBAC-JACOWEB. O primeiro aspecto que diferencia o RBAC-JACOWEB do RBAC/Web e do JRBAC-Web é o modelo RBAC utilizado como referência. Enquanto o RBAC-JACOWEB utiliza o modelo unificado RBAC-NIST, que é um modelo que está sendo padronizado, as outras implementações utilizam modelos RBAC próprios. Além do padrão RBAC, o RBAC-JACOWEB também utiliza o CORBA—um padrão da ISO (*International Organization for Standardization*) para construção de sistemas distribuídos—juntamente com o seu serviço de segurança (que também faz parte do padrão).

Em termos estruturais, a proposta RBAC-JACOWEB atua no nível do *middleware*, permitindo a sua utilização com qualquer aplicação baseada em CORBA. O RBAC/Web e o RBAC-JACOWEB, por sua vez, são voltados para aplicações específicas, o que significa menor flexibilidade. A transparência na ativação de papéis para aplicações e usuários está presente no RBAC-JACOWEB e ausente das outras implementações, que requerem intervenção dos usuários (RBAC/Web) ou das aplicações (JRBAC-Web) para selecionar os papéis a serem ativados.

O trabalho disponível na literatura que mais se aproxima da nossa proposta é o de Beznosov e Deng [6], que define um *framework* para implementar RBAC usando o serviço de segurança CORBA. As principais diferenças entre a nossa proposta e este *framework* são a transparência para as aplicações e usuários e o modelo RBAC utilizado como referência. Na proposta de Beznosov e Deng, o usuário interage, através de um UserSponsor, com o objeto PrincipalAuthenticator para selecionar os papéis ativados em uma sessão, e o modelo RBAC utilizado é a família RBAC96 [56] (apresentada na seção 3.3.2). Na proposta RBAC-JACOWEB, por outro lado, os papéis são ativados automaticamente pelo PoliCap quando necessários, de maneira transparente para o usuário, e o modelo RBAC de referência é o modelo unificado RBAC-NIST.

5.5 Conclusões do Capítulo

Este capítulo apresentou a proposta RBAC-JACOWEB, que incorpora um controle baseado em papéis ao projeto JACOWEB, que combina os modelos de segurança Java, CORBA e Web para fornecer soluções para autorização em redes e sistemas distribuídos de larga escala.

Uma das principais metas no desenvolvimento do RBAC-JACOWEB foi o de maximizar a transparência da segurança para as aplicações, possibilitando que o esquema de autorização seja implantado com um mínimo de modificação nas aplicações já existentes. Neste sentido, introduzimos a ativação automática de papéis pelo *middleware*, uma abordagem inovadora na literatura conhecida sobre RBAC.

Capítulo 6

Aspectos de Implementação e Resultados

O *home banking* é uma das aplicações baseadas na Internet mais utilizadas no Brasil. De acordo com a 6ª Pesquisa Nacional sobre Segurança da Informação [51], 43% dos usuários Internet corporativos utilizam regularmente este serviço.

Sendo uma aplicação crítica amplamente utilizada em sistemas de larga escala baseados na Internet, um sistema bancário foi escolhido como base do protótipo para avaliar o esquema de autorização RBAC-JACOWEB apresentado no capítulo 5.

Este capítulo apresenta aspectos de implementação do protótipo RBAC-JACOWEB juntamente com os resultados obtidos com o protótipo e tece considerações a respeito de melhorias que podem ser incorporadas à versão atual do protótipo.

6.1 Protótipo de Implementação

A estrutura geral do sistema bancário escolhido como base do protótipo está mostrada na figura 6.1. Os usuários podem utilizar um navegador (*browser*) para acessar o servidor Web do sistema bancário a partir de qualquer ponto da Internet. A página Web carrega um *applet* Java que atua como um cliente do sistema bancário, consultando o servidor de nomes CORBA para obter as referências dos servidores de aplicação (implementados como objetos CORBA) e fazendo requisições a estes servidores através de conexões seguras. Nesta implementação, o servidor Web (que armazena o código do *applet* cliente), o servidor de nomes CORBA e os servidores bancários não necessitam estar residentes na mesma máquina.

Os objetos servidores CORBA utilizam o JacORB [9], um ORB livre escrito em Java, e o *applet* foi desenvolvido com o Java 2 SDK, versão 1.2.1. Os testes foram realizados usando como navegador o Netscape Communicator 4.76. A assinatura digital do modelo de segurança Java foi utilizada

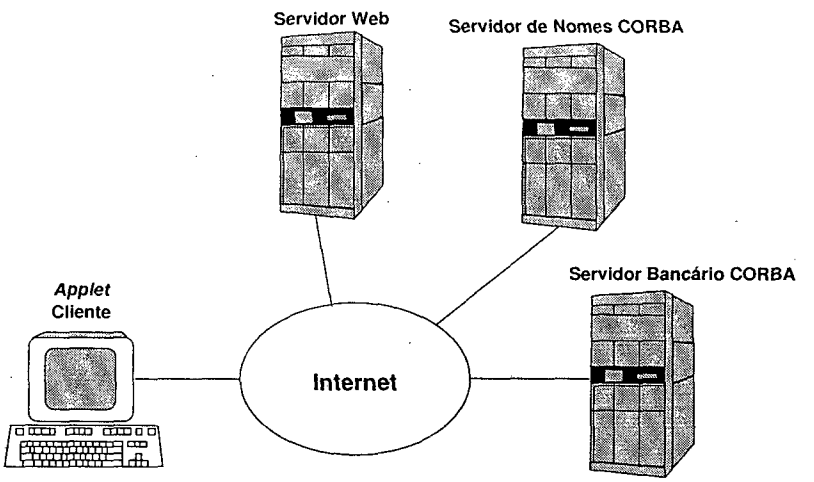


Figura 6.1: Estrutura da aplicação bancária

para permitir ao *applet* cliente acesso ao disco local e conexões com máquinas que não o servidor Web a partir do qual ele é carregado. Isto livra o *applet* das restrições impostas pelo *sandbox* do Java, permitindo-lhe livre acesso aos servidores CORBA independentemente de onde eles estejam localizados na Internet.

O estabelecimento da associação segura emprega o protocolo SSL[1] como tecnologia de segurança subjacente. O SSL foi escolhido por ser o padrão *de facto* para criptografia na Web e também por ser um dos protocolos suportados pela especificação CORBAsec [43] (mesmo que alguns aspectos do uso de SSL no CORBAsec ainda estejam sendo discutidos [2]). No protótipo foram utilizadas as funções criptográficas IAIK-JCE e o pacote iSaSiLk v.2.5, que implementa a versão atual do protocolo SSL (SSLv3) em Java e que foi escolhido por disponibilizar a funcionalidade requerida pelo protótipo.[2]

A figura 6.2 mostra a arquitetura do protótipo implementado. Este protótipo é uma evolução do protótipo desenvolvido para o esquema de autorização discricionário previamente implementado no projeto JACOWEB [64, 66]. A construção do protótipo RBAC-JACOWEB deu-se em duas etapas. Na primeira etapa, o protótipo discricionário foi modificado, implementando-se o controle de acesso do lado do cliente com a geração de *capabilities* e também uma primeira versão do serviço de políticas PoliCap com suporte a políticas discricionárias; em um segundo momento, passou-se à implementação do suporte ao RBAC dentro do PoliCap e à subseqüente integração do controle de acesso baseado em papéis segundo a proposta RBAC-JACOWEB.

[1] O SSL (*Secure Sockets Layer*) [20] é um protocolo que fornece confidencialidade, integridade e autenticação entre as partes em conexões TCP/IP.

[2] As funções IAIK-JCE e o pacote iSaSiLk podem ser obtidos em `http://jcewww.iaik.at`.

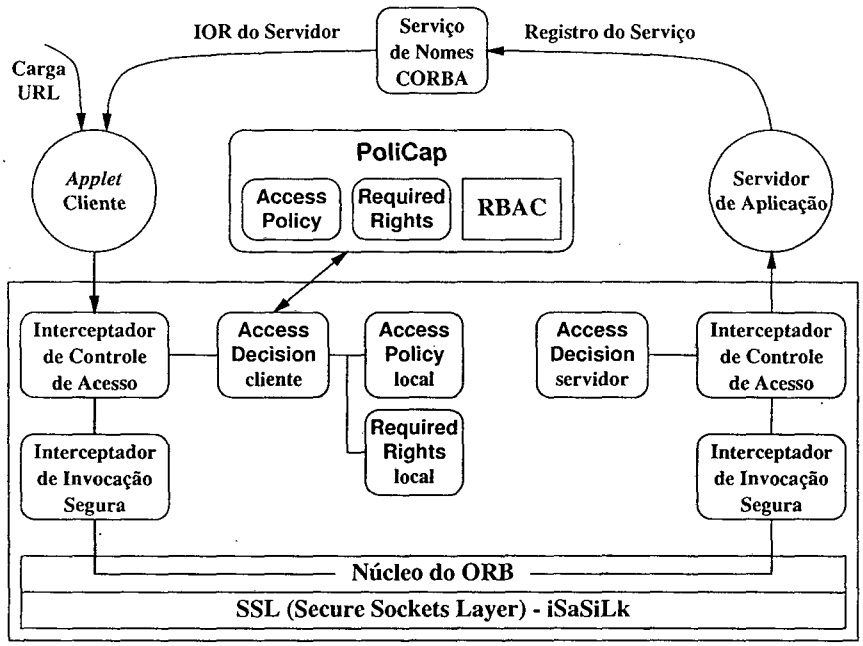


Figura 6.2: Arquitetura do protótipo implementado

6.1.1 Alterações no Protótipo Discricionário

O primeiro passo da implementação do protótipo RBAC-JACOWEB foi modificar a decisão de acesso, ainda dentro do esquema discricionário, passando-a do lado do servidor para o lado do cliente. O subsistema de estabelecimento de associações seguras (formado pelos interceptadores de invocação segura, objetos Vault e SecurityContext e pela integração ORB+SSL) do protótipo discricionário foi mantido.

Os interceptadores de controle de acesso tiveram que ser alterados. O interceptador de controle de acesso do cliente passou a ser responsável por gerar a *capability* em caso de acesso autorizado, e o interceptador de controle de acesso do servidor assumiu a tarefa de extrair a *capability* da requisição CORBA. Neste momento, o objeto AccessDecision do servidor foi modificado para fazer uma simples verificação da integridade da *capability* antes de autorizar um acesso.

Uma primeira versão do serviço de políticas PoliCap com suporte a políticas discricionárias[3] foi implementada a seguir. Com a introdução do PoliCap, o objeto AccessDecision do cliente teve que ser reescrito. A nova versão passou a obter, na ligação entre cliente e servidor, versões locais dos objetos DomainAccessPolicy[4] e RequiredRights. Em tempo de decisão de acesso, o AccessDecision passou a utilizar estas versões locais dos objetos para realizar a verificação da política. Observa-se que aqui a única interação entre o AccessDecision do cliente e o PoliCap ocorre durante o *binding*; se

[3] *Interfaces* AccessPolicyAdmin e RequiredRightsAdmin, cuja definição IDL pode ser encontrada no apêndice A.

[4] No modelo CORBA de segurança, os direitos concedidos em políticas de autorização discricionárias são armazenados em um objeto DomainAccessPolicy e não em um objeto AccessPolicy (utilizado com políticas não discricionárias) [43].

os objetos DomainAccessPolicy e RequiredRights locais não autorizam um acesso, este é negado e a seqüência de invocação é interrompida imediatamente.

6.1.2 Protótipo RBAC-JACOWEB

Após o desenvolvimento da primeira versão do PoliCap e sua integração ao esquema de autorização discricionário, passou-se à segunda etapa de implementação.

O primeiro passo da segunda etapa foi incluir o suporte ao RBAC no serviço de políticas PoliCap[5], que passou a armazenar informações sobre usuários, papéis, associações usuário-papel e papel-permissão, hierarquias de papéis e restrições de separação de tarefas. As operações adicionadas ao PoliCap incluem ainda mecanismos de revisão de associações usuário-papel e papel-permissão e a operação `role_access` discutida na seção 5.3.1.

Com a adição da *interface* RoleAdmin ao serviço PoliCap, foi necessário reescrever o objeto AccessDecision do cliente para integrar o controle de acesso baseado em papéis ao esquema de autorização. O novo AccessDecision interage com o PoliCap tanto no *binding* quanto (em caso de necessidade) em tempo de decisão de acesso, conforme descrito na seção 5.3.2.

Para melhor administrar a configuração RBAC do PoliCap, foi desenvolvida uma *interface* gráfica, mostrada na figura 6.3. Ela permite a gerência de usuários, papéis e permissões, além de hierarquias de papéis e restrições de separação de tarefas. Esta *interface* de administração foi igualmente implementada em Java.

O protótipo RBAC-JACOWEB manteve o subsistema de estabelecimento de associações seguras desenvolvido para o protótipo discricionário. Outro aspecto que não foi modificado foi a obtenção das credenciais dos principais: assim como no protótipo discricionário, as credenciais do cliente—contendo apenas o seu atributo de privilégio AccessId—são geradas na inicialização do sistema seguro com base no seu certificado SSL.

6.1.3 Resultados Obtidos

O funcionamento do protótipo do esquema de autorização foi verificado utilizando-se a configuração do exemplo discutido na seção 5.3.3; os cenários de funcionamento apresentados nas figuras 5.6 a 5.9 serviram de casos de teste. Para possibilitar a realização dos testes, foi necessário implementar os objetos servidores de aplicação ContaPFis e ContaPJur. Estes servidores foram adaptados do servidor bancário usado nos testes do protótipo JACOWEB discricionário. O *applet* cliente também teve

[5] *Interface* RoleAdmin na IDL do apêndice A.

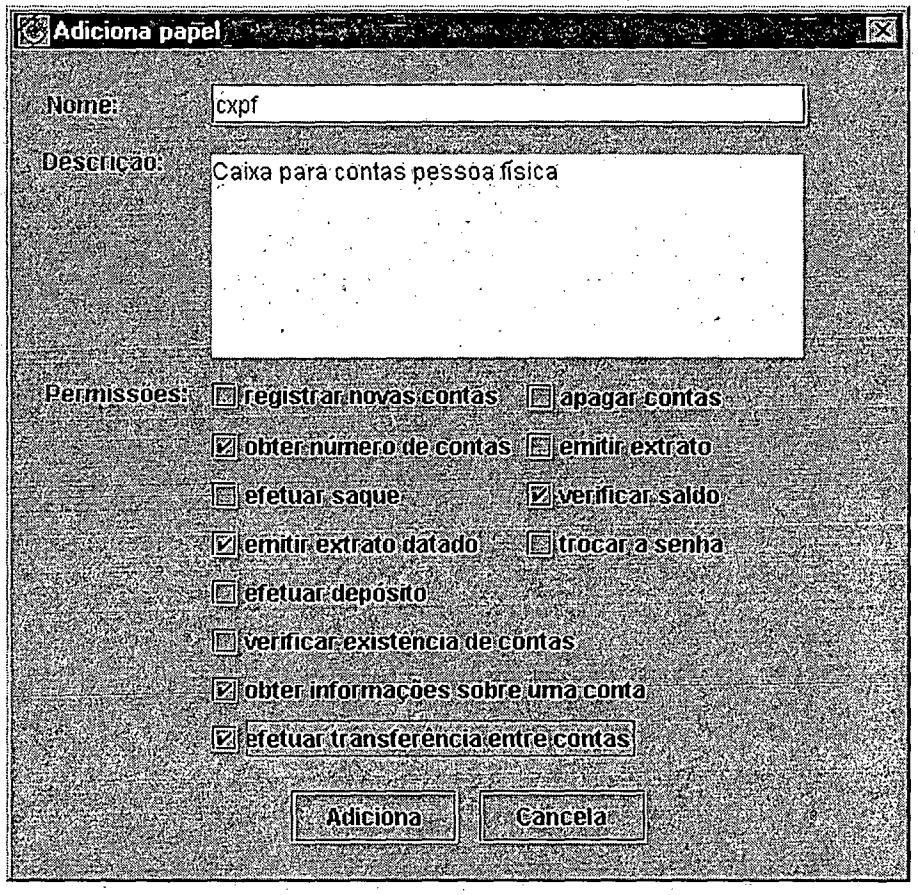


Figura 6.3: *Interface* gráfica de administração do PoliCap

que ser reescrito de modo a refletir a existência de dois servidores CORBA (o protótipo discricionário tinha apenas um objeto servidor).

Com a execução manual das seqüências de operações mostradas nos casos de teste, verificou-se que os resultados do esquema de autorização baseado em papéis implementado concretizaram os cenários das figuras 5.6 a 5.9, demonstrando a viabilidade e a adequação da proposta RBAC-JACOWEB.

6.2 Considerações sobre a Implementação

A versão atual do protótipo não conta, ainda, com autenticação de principais através do objeto PrincipalAuthenticator. A credencial do cliente é inicializada com o AccessId extraído do seu certificado SSL, sem que este cliente passe por um processo de autenticação. Uma aplicação real em um ambiente de larga escala deveria implementar um processo de autenticação a fim de evitar que a simples posse de um certificado SSL (que é armazenado em um arquivo no sistema do cliente, podendo, portanto, ser roubado) permita que um usuário efetivamente não autorizado ganhe acesso ao sistema.

O mecanismo de seleção de papéis embutido na operação `role_access` do PoliCap não é, ainda, o mais sofisticado possível; este algoritmo está sendo refinado para escolher melhor os papéis a serem ativados para uma dada operação. Estes refinamentos pretendem minimizar o conjunto de permissões adquiridas pelo usuário através da ativação de papéis, respeitando tanto quanto possível o princípio do mínimo privilégio.

A *interface* de gerência das políticas de segurança também pode ser melhorada, incorporando elementos como um módulo de visualização gráfica da hierarquia de papéis do sistema (atualmente, é necessário manipular a lista de papéis inferiores a um determinado papel para modificar a hierarquia).

6.3 Conclusões do Capítulo

O sistema de *home banking* utilizado como exemplo de aplicação para o desenvolvimento do protótipo RBAC-JACOWEB forneceu um ambiente mais realista para a implantação do nosso esquema de autorização, auxiliando a identificar de maneira mais precisa as vulnerabilidades que precisam ser contidas pelo esquema de controle de acesso.

O protótipo implementado demonstrou garantir a segurança de aplicações em ambientes de larga escala. Cenários de funcionamento da aplicação bancária geraram casos de teste verossímeis para o protótipo, o que nos permitiu comprovar a viabilidade e a aplicabilidade do esquema de autorização proposto.

As ferramentas e tecnologias utilizadas no desenvolvimento do protótipo (protocolo SSL, linguagem Java e arquitetura CORBA) baseiam-se em padrões abertos, o que é um fator importante para alcançar a interoperabilidade e a independência de plataforma no *software* desenvolvido.

Capítulo 7

Conclusões e Perspectivas Futuras

Modelos de segurança desempenham um papel preponderante na definição de políticas de segurança, que são uma peça-chave em sistemas de informação modernos. O uso crescente de arquiteturas distribuídas baseadas em redes de larga escala, como a Internet e as grandes *intranets* corporativas, aumenta a necessidade de políticas de controle de acesso mais rígidas do que as tradicionais políticas baseadas em modelos discricionários.

Esta dissertação faz uma análise de quatro modelos de controle de acesso não discricionários, sendo que três deles—Bell-LaPadula, Biba e Clark-Wilson—implementam diferentes formas de controle de acesso obrigatório enquanto que o quarto, o modelo unificado RBAC-NIST, representa a classe de modelos baseados em papéis. Embora estes modelos tenham naturezas diversas, uma comparação de suas características fundamentais demonstra que eles são efetivamente capazes de fornecer uma proteção mais rigorosa do que aquela oferecida pelo modelo discricionário. Entretanto, os modelos obrigatórios impõem políticas de segurança pouco flexíveis, freqüentemente restringindo a usabilidade dos sistemas que os empregam. Os modelos baseados em papéis surgem como uma alternativa ao mesmo tempo flexível (na definição da política de segurança adotada) e rigorosa (na implantação da política definida).

A proposta RBAC-JACOWEB mostra como o controle de acesso basedo em papéis pode ser incorporado a sistemas de larga escala que utilizam a tecnologia CORBA, valendo-se de padrões como o modelo de segurança CORBA e o modelo unificado RBAC-NIST. A nossa principal contribuição é a introdução da ativação automática de papéis pelo subsistema de segurança, uma abordagem inovadora na literatura conhecida sobre RBAC. O protótipo implementado, baseado em uma aplicação bancária e desenvolvido no contexto do projeto JACOWEB, enfatiza a viabilidade da proposta e a adequação do RBAC como um modelo de controle de acesso para sistemas distribuídos de larga escala.

Existem diversas possibilidades de continuidade deste trabalho. A mais visível e imediata delas é o aprimoramento do protótipo, com a implementação de aspectos como a autenticação de principais

e um subsistema de auditoria, o refinamento do mecanismo de seleção de papéis e a incorporação de melhorias à ferramenta de administração da configuração RBAC.

Novas versões do modelo unificado RBAC-NIST serão apresentadas; é nossa intenção acompanhar a evolução deste modelo, incorporando as alterações que surgirem à proposta RBAC-JACOWEB, bem como, se possível, participar ativamente da elaboração das revisões do modelo.

Uma outra perspectiva que pode ser abordada dentro da proposta RBAC-JACOWEB é o uso do RBAC para a administração de segurança através da utilização de modelos RBAC administrativos como os descritos em [37, 58].

Apêndice A

Definição IDL do PoliCap

A.1 Definição IDL

Esta é a definição IDL do serviço de políticas PoliCap. A seção A.2 descreve o funcionamento das operações.

```
module PoliCap {
#include "SecurityLevel2.idl"
#include "SecurityAdmin.idl"
#pragma prefix "edu.lcmi.jacoweb"

module PoliCap {

  interface AccessPolicyAdmin: CORBA::DomainManager {
    const CORBA::PolicyType SecClientInvocationAccessDiscretionary = 100;
    const CORBA::PolicyType SecClientInvocationAccessRBAC = 101;

    void set_access_policy(in CORBA::PolicyType           policy_type,
                           in SecurityAdmin::AccessPolicy policy);

    void delete_access_policy(in CORBA::PolicyType policy_type);

    SecurityAdmin::AccessPolicy
         get_local_access_policy(in CORBA::PolicyType       policy_type,
                                 in Security::AttributeList attr_list);
```

```
};

interface RequiredRightsAdmin: SecurityLevel2::RequiredRights {
  SecurityLevel2::RequiredRights get_local_required_rights(
    in CORBA::Identifier target_interface_name);
};

interface RoleAdmin {

  typedef string         Role;
  typedef sequence<Role> RoleList;

  typedef string         User;
  typedef sequence<User> UserList;

  struct RoleInfo {
    string full_name;
    string description;
  };

  struct UserInfo {
    string full_name;
  };

  typedef unsigned short RoleConstraint;
  const RoleConstraint SSD = 1;  // static separation of duty
  const RoleConstraint DSD = 2;  // dynamic separation of duty

  exception invRole       { string reason; };
  exception invUser       { string reason; };
  exception invInfo       { string reason; };
  exception invPerms      { string reason; };
  exception invConstraint { string reason; };

  // role management
  void     add_role(in Role r, in RoleInfo i)    raises (invRole, invInfo);
  void     del_role(in Role r)                   raises (invRole);
  void     update_role(in Role r, in RoleInfo i) raises (invRole, invInfo);
  RoleInfo get_role_info(in Role r)              raises (invRole);
```

```
// user management
void     add_user(in User u, in UserInfo i)    raises (invUser, invInfo);
void     del_user(in User u)                   raises (invUser);
UserInfo get_user_info(in User u)              raises (invUser);
void     update_user(in User u, in UserInfo i) raises (invUser, invInfo);

// user-role assignments
void assign_user_role  (in User u, in Role r) raises (invUser, invRole);
void deassign_user_role(in User u, in Role r) raises (invUser, invRole);

// role-permission assignments
void assign_role_perms  (in Role r, in Security::RightsList p)
       raises (invRole, invPerms);
void deassign_role_perms(in Role r, in Security::RightsList p)
       raises (invRole, invPerms);

// user-role review mechanisms
RoleList get_roles(in User u) raises (invUser);
UserList get_users(in Role r) raises (invRole);

// permission review mechanisms
Security::RightsList get_perms      (in Role r)     raises (invRole);
Security::RightsList get_all_perms  (in Role r)     raises (invRole);
RoleList get_roles_perms(in Security::RightsList p) raises (invPerms);

// role hierarchies
void add_junior_role(in Role sr, in Role jr) raises (invRole);
void del_junior_role(in Role sr, in Role jr) raises (invRole);

RoleList get_junior_roles    (in Role r) raises (invRole);
RoleList get_all_junior_roles(in Role r) raises (invRole);

// role constraints
void     set_exclusive_roles   (in Role r1, in Role r2,
                                in RoleConstraint rc)
          raises (invRole, invConstraint);
void     unset_exclusive_roles (in Role r1, in Role r2,
                                in RoleConstraint rc)
```

```
            raises (invRole, invConstraint);
    RoleList get_exclusive_roles   (in Role r, in RoleConstraint rc)
            raises (invRole, invConstraint);

    // role-based access decision
    boolean role_access(inout SecurityLevel2::CredentialsList cred_list,
                        in CORBA::Identifier operation_name,
                        in CORBA::Identifier target_interface_name,
                        inout SecurityAdmin::DomainAccessPolicy local_dap);
  };
};
```

A.2 Descrição das Operações

A.2.1 *Interface* AccessPolicyAdmin

Esta *interface* gerencia objetos AccessPolicy. A operação `set_access_policy` define o objeto AccessPolicy para um determinado tipo de política (`policy_type`); `delete_access_policy` cancela uma definição anterior. A operação `get_local_access_policy` retorna uma versão local do objeto AccessPolicy com os direitos concedidos pelos atributos de segurança do principal (`attr_list`).

A.2.2 *Interface* RequiredRightsAdmin

Esta *interface* gerencia objetos RequiredRights. A operação `get_local_required_rights` retorna uma versão local do objeto RequiredRights contendo os direitos requeridos para uma determinada *interface*. Operações para definir quais são estes direitos requeridos são fornecidas pela *interface* RequiredRights (herdada por RequiredRightsAdmin).

A.2.3 *Interface* RoleAdmin

Esta *interface* gerencia a configuração RBAC no PoliCap. As suas operações estão descritas a seguir.

▷ `add_role, del_role`

Estas operações incluem e excluem papéis (gerenciamento do conjunto $\mathcal{R}$). Um papel só poderá ser excluído quando não fizer parte de nenhuma hierarquia nem de nenhuma relação de exclusão

mútua; a remoção de um papel implica na exclusão das relações usuário-papel e papel-permissão das quais ele faz parte.

▷ `add_user, del_user`

Estas operações incluem e excluem usuários (gerenciamento do conjunto $\mathcal{U}$). Um usuário somente fará parte do esquema de autorização baseado em papéis se for incluído através de `add_user`. A remoção de um usuário implica na exclusão das relações usuário-papel das quais ele faz parte.

▷ `assign_user_role, deassign_user_role`

Estas operações definem e cancelam uma associação entre um usuário e um papel (gerenciamento da relação $\mathcal{UA}$). O papel e o usuário devem ter sido previamente incluídos (com `add_role`/`add_user`).

▷ `assign_role_perms, deassign_role_perms`

Estas operações adicionam e removem permissões de um papel (gerenciamento da relação $\mathcal{PA}$). O papel deve ter sido previamente incluído com `add_role`.

▷ `get_roles, get_users`

Estas operações fornecem os mecanismos de revisão usuário-papel: `get_roles` permite determinar quais os papéis estão associados a um usuário, enquanto que `get_users` retorna os usuários associados a um papel. Estas operações atuam somente em associações diretas, isto é, não levam em conta hierarquias de papéis.

▷ `get_perms, get_all_perms, get_roles_perms`

Estas operações fornecem os mecanismos de revisão papel-permissão. `get_perms(r)` retorna as permissões associadas diretamente ao papel `r` via `assign_role_perms`. `get_all_perms(r)` retorna as permissões associadas a `r` e também as permissões herdadas através da hierarquia de papéis. `get_roles_perms(p)` retorna o conjunto de papéis que conferem as permissões especificadas por `p`; papéis que apenas herdam as permissões `p` não são incluídos no resultado da operação.

▷ `add_junior_role, del_junior_role`

Estas operações definem a hierarquia de papéis utilizada (gerenciamento da relação $\mathcal{RH}$). A operação `add_junior_role(r1,r2)` relaciona um papel inferior `r2` a um papel superior `r1`, sendo que `r1` passa a herdar as permissões de `r2`. `del_junior_role` cancela uma relação hierárquica previamente estabelecida. A implementação da operação `add_junior_role` deve garantir que a nova relação hierárquica não introduza um ciclo no grafo da hierarquia de papéis.

▷ `get_junior_roles, get_all_junior_roles`

Estas operações permitem obter informações sobre a hierarquia de papéis. O conjunto de papéis imediatamente inferiores de um papel `r` é obtido através de `get_junior_roles(r)`, enquanto que `get_all_junior_roles(r)` retorna todos os papéis herdados por `r`, direta ou indiretamente.

▷ `set_exclusive_roles`, `unset_exclusive_roles`, `get_exclusive_roles`

Estas operações gerenciam as restrições de separação de tarefas, tanto estáticas quanto dinâmicas. As operações `set_exclusive_roles(r1,r2,rc)` e `unset_exclusive_roles(r1,r2,rc)` definem e cancelam relações de exclusão mútua entre os papéis `r1` e `r2`. `get_exclusive_roles(r,rc)` retorna a lista de papéis mutuamente exclusivos em relação a `r`. O argumento `rc` indica se a exclusão desejada é estática (`rc = SSD`) ou dinâmica (`rc = DSD`).

É importante notar que a exclusão de papéis é reflexiva; a implementação deve garantir que `set_exclusive_roles(r1,r2,rc)` ≡ `set_exclusive_roles(r2,r1,rc)`.

▷ `role_access`

Esta operação decide se o acesso solicitado pelo usuário pode ou não ser autorizado de acordo com a configuração do esquema RBAC. Se o usuário possui um ou mais papéis que lhe conferem as permissões necessárias à invocação da operação e estes papéis podem ser ativados (i.e., esta ativação não viola nenhuma restrição), o acesso é autorizado, o que é indicado por um valor de retorno true. Caso não seja possível ativar papéis que confiram as permissões necessárias, a operação retorna false.

Quando o acesso é autorizado, as credenciais do principal são modificadas de forma a incorporar os novos papéis ativados,[1] e o argumento `local_ap` (que representa o objeto AccessPolicy local) é modificado de forma a incorporar os novos direitos concedidos pela ativação dos papéis ao principal.

[1] Apesar da proposta RBAC-JACOWEB especificar que cada principal tenha somente uma credencial (seção 5.3.1), a operação `role_access` suporta o uso de listas de credenciais, sendo que os novos papéis são incorporados a todas as credenciais da lista `cred_list`.

Referências Bibliográficas

[1] Adiron. *ORBAsec SL2 User Guide, version 2.1.4.* Syracuse, NY, July 2000.

[2] Ameneh Alireza, Ulrich Lang, Marios Padelis, Rudolf Schreiner, and Markus Schumacher. The Challenges of CORBA Security. In *Proceedings of the Workshop "Sicherheit in Mediendaten"*. Gesellschaft für Informatik (GI), June 2000.

[3] Edward G. Amoroso. *Fundamentals of Computer Security Technology.* Prentice Hall PTR, Upper Saddle River, NJ, 1994.

[4] John F. Barkley. Implementing Role-Based Access Control Using Object Technology. In *Proceedings of the 1st ACM Workshop on Role-Based Access Control (RBAC'95)*, pages 293–298, Gaithersburg, MD, 1995.

[5] D. Elliott Bell and Leonard J. LaPadula. Secure Computer Systems: Unified Exposition and Multics Interpretation. MITRE Technical Report MTR-2997 Rev. 1, MITRE Corporation, Bedford, MA, March 1976.

[6] Konstantin Beznosov and Yi Deng. A Framework for Implementing Role-Based Access Control Using CORBA Security Service. In *Proceedings of the 4th ACM Workshop on Role-Based Access Control (RBAC'99)*, pages 19–30, Fairfax, VA, 1999.

[7] Kenneth J. Biba. Integrity Considerations for Secure Computer Systems. MITRE Technical Report MTR-3153, MITRE Corporation, Bedford, MA, April 1977.

[8] Bob Blakley. *CORBA Security: An Introduction to Safe Computing with Objects.* Addison-Wesley, Reading, MA, 1999.

[9] Gerald Brose. JacORB—Design and Implementation of a Java ORB. In *Proceedings of the IFIP WG 6.1 International Working Conference on Distributed Aplications and Interoperable Systems (DAIS'97)*, pages 143–154, Cottbus, Germany, September 1997.

[10] CC. Common Criteria Editorial Board. Common Criteria for Information Technology Security Evaluation, Part 2: Security Functional Requirements. ISO/IEC 15408-2, December 1999.

[11] Ramaswamy Chandramouli and Ravi S. Sandhu. Role-Based Access Control Features in Commercial Database Management Systems. In *Proceedings of the 21st NIST-NCSC National Information Systems Security Conference*, pages 503–511, Arlington, VA, October 1998.

[12] David D. Clark and David R. Wilson. A Comparison of Commercial and Military Computer Security Policies. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 184–194, Oakland, CA, 1987.

[13] Dorothy E. Denning. A Lattice Model of Secure Information Flow. *Communications of the ACM*, 19(5):236–243, May 1976.

[14] DoD. Department of Defense Trusted Computer Systems Evaluation Criteria. DoD 5200.28-STD, December 1985.

[15] David F. Ferraiolo. Re: Status of the NIST RBAC model. Comunicação privada, Message-ID: <4.3.2.7.2.20000830103056.00ce77a0@email.nist.gov>, 30 de agosto de 2000.

[16] David F. Ferraiolo, John F. Barkley, and D. Richard Kuhn. A Role-Based Access Control Model and Reference Implementation Within a Corporate Intranet. *ACM Transactions on Information and Systems Security*, 2(1):34–64, February 1999.

[17] David F. Ferraiolo, Janet Cugini, and D. Richard Kuhn. Role-Based Access Control (RBAC): Features and Motivations. In *Proceedings of the 11th Annual Computer Security Conference*, pages 241–248, New Orleans, LA, December 1995.

[18] David F. Ferraiolo, Dennis M. Gilbert, and Nickilyn Lynch. Assessing Federal and Commercial Information Security Needs. NISTIR 4976, National Institute of Standards and Technology, Gaithersburg, MD, November 1992.

[19] David F. Ferraiolo and D. Richard Kuhn. Role-Based Access Controls. In *Proceedings of the 15th NIST-NCSC National Computer Security Conference*, pages 554–563, Baltimore, MD, 1992.

[20] Alan O. Freier, Philip Karlton, and Paul C. Kocher. The SSL Protocol, Version 3.0. Internet Draft, March 1996. Disponível em `http://home.netscape.com/eng/ssl3/ssl-toc.html`.

[21] Luigi Giuri. Role-Based Access Control on the Web Using Java. In *Proceedings of the 4th ACM Workshop on Role-Based Access Control (RBAC'99)*, pages 11–18, Fairfax, VA, 1999.

[22] Luigi Giuri and P. Iglio. A Formal Model for Role-Based Access Control with Constraints. In *Proceedings of the 9th IEEE Computer Security Foundations Workshop*, County Kerry, Ireland, June 1996.

[23] Michael A. Harrison, Walter L. Ruzzo, and Jeffrey D. Ullman. Protection in Operating Systems. *Communications of the ACM*, 19(8):461–471, August 1976.

[24] ISO. ISO/IEC 9075. Database Language SQL – Part 2: Foundation (Working Draft). ISO/IEC JTC1/SC21 N10489, July 1996.

[25] Trent Jaeger. Rebuttal to the NIST RBAC Model Proposal. In *Proceedings of the 5th ACM Workshop on Role-Based Access Control (RBAC'2000)*, pages 65–66, Berlin, Germany, 2000.

[26] Günther Karjoth. An Operational Semantics of Java 2 Access Control. In *Proceedings of the 13th IEEE Computer Security Foundations Workshop*, pages 224–232, Cambridge, England, July 2000.

[27] Charlie Lai, Li Gong, Larry Koved, Anthony Nadalin, and Roland Schemers. User Authentication and Authorization in the Java Platform. In *Proceedings of the 15th Annual Computer Security Applications Conference*, Phoenix, AZ, December 1999.

[28] Butler W. Lampson. Protection. In *Proceedings of the 5th Princeton Symposium on Information Sciences and Systems*, pages 437–443. Princeton University, March 1971.

[29] Butler W. Lampson. A Note on the Confinement Problem. *Communications of the ACM*, 16 (10):613–615, October 1973.

[30] Carl E. Landwehr. Formal Models for Computer Security. *ACM Computing Surveys*, 13(3): 247–278, September 1981.

[31] Ulrich Lang and Rudolf Schreiner. Flexibility and Interoperability in CORBA Security. In Steve Schneider and Peter Ryan, editors, *Electronic Notes in Theoretical Computer Science*, volume 32. Elsevier Science Publishers, 2000.

[32] John Linn. Generic Security Service Application Program Interface, Version 2. Request for Comments (RFC) 2078, January 1997.

[33] Steven B. Lipner. Non-Discretionary Controls for Commercial Applications. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 2–10, Oakland, CA, 1982.

[34] Terry Mayfield, J. Eric Roskos, Stephen R. Welke, and John M. Boone. Integrity in Automated Information Systems. C Technical Report 79-91, Institute for Defense Analysis, Alexandria, VA, September 1991.

[35] John McLean. The Specification and Modeling of Computer Security. *IEEE Computer*, 23(1): 9–16, January 1990.

[36] Jonathan D. Moffett and Morris S. Sloman. Delegation of Authority. In I. Krishnan and W. Zimmer, editors, *Integrated Network Management II*, pages 595–606. Elsevier Science Publishers B.V. (North-Holland), 1991.

[37] Qamar Munawer. *Administrative Models for Role-Based Access Control.* PhD thesis, George Mason University, Fairfax, VA, 2000.

[38] B. Clifford Neuman and Theodore Ts'o. Kerberos: An Authentication Service for Computer Networks. *IEEE Communications*, 32(9):33–38, September 1994.

[39] Vincent Nicomette. *La Protection dans les Systèmes à Objets Répartis*. Thèse de doctorat, Institut National Polytechnique de Toulouse, France, 1996.

[40] Matunda Nyanchama and Sylvia Osborn. Access Rights Administration in Role-Based Security Systems. In *Proceedings of the IFIP Working Group 11.3 Working Conference on Database Security*, Amsterdam, The Netherlands, 1994. Elsevier North-Holland, Inc.

[41] Matunda Nyanchama and Sylvia Osborn. The Role Graph Model and Conflict of Interest. *ACM Transactions on Information and Systems Security*, 2(1):3–33, February 1999.

[42] OMG. Object Management Group. A Discussion of the Object Management Architecture. OMG Document 00-06-41, January 1997.

[43] OMG. Object Management Group. Security Service Specification, version 1.5. OMG Document 00-06-25, June 2000.

[44] OMG. Object Management Group. The Common Object Request Broker: Architecture and Specification. OMG Document 00-11-07, November 2000.

[45] Robert Orfali and Dan Harkey. *Client/Server Programming with Java and CORBA*. John Wiley & Sons, New York, NY, 2nd edition, 1998.

[46] Sylvia Osborn. Mandatory Access Control and Role-Based Access Control Revisited. In *Proceedings of the 2nd ACM Workshop on Role-Based Access Control (RBAC'97)*, pages 31–40, Fairfax, VA, 1997.

[47] Sylvia Osborn, Ravi S. Sandhu, and Qamar Munawer. Configuring Role-Based Access Control to Enforce Mandatory and Discretionary Access Control Policies. *ACM Transactions on Information and Systems Security*, 3(2), 2000.

[48] Jim Reynolds and Ramaswamy Chandramouli. Role-Based Access Control Protection Profile v1.0 (Final), July 1998. Disponível em `http://csrc.nist.gov/cc/pp/pplist.htm`.

[49] Ward Rosenberry, David Kenney, and Gerry Fisher. *Understanding DCE*. O'Reilly & Associates, Sebastopol, CA, 1992.

[50] Deborah Russell and G. T. Gangemi, Sr. *Computer Security Basics*. O'Reilly & Associates, Sebastopol, CA, 1991.

[51] Módulo Security Solutions S.A. 6ª Pesquisa Nacional sobre Segurança da Informação. Disponível em `http://www.modulo.com.br`, junho de 2000.

[52] Ravi S. Sandhu. Lattice-Based Access Control Models. *IEEE Computer*, 26(11):9–19, November 1993.

[53] Ravi S. Sandhu. Role Activation Hierarchies. In *Proceedings of the 3rd ACM Workshop on Role-Based Access Control (RBAC'98)*, pages 33–40, Fairfax, VA, October 1998.

[54] Ravi S. Sandhu. Role-Based Access Control. In *Advances in Computers*, volume 46. Academic Press, 1998.

[55] Ravi S. Sandhu, Venkata Bhamidipati, and Qamar Munawer. The ARBAC97 Model for Role-Based Administration of Roles. *ACM Transactions on Information and Systems Security*, 2(1): 105–135, February 1999.

[56] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-Based Access Control Models. *IEEE Computer*, 29(2):38–47, February 1996.

[57] Ravi S. Sandhu, David F. Ferraiolo, and D. Richard Kuhn. The NIST Model for Role-Based Access Control: Towards a Unified Standard. In *Proceedings of the 5th ACM Workshop on Role-Based Access Control (RBAC'2000)*, Berlin, Germany, 2000.

[58] Ravi S. Sandhu and Qamar Munawer. The ARBAC99 Model for Administration of Roles. In *Proceedings of the 15th Annual Computer Security Application Conference*, Scottsdale, AZ, December 1999.

[59] Ravi S. Sandhu and Pierangela S. Samarati. Access Control: Principles and Practice. *IEEE Communications*, 32(9):40–48, September 1994.

[60] Ravi S. Sandhu and Pierangela S. Samarati. Authentication, Access Control, and Audit. *ACM Computing Surveys*, 28(1):241–243, March 1996.

[61] Dirk Slama, Jason Garbis, and Perry Russell. *Enterprise CORBA*. Prentice-Hall PTR, Upper Saddle River, NJ, 1999.

[62] Lawrence Snyder. Theft and Conspiracy in the Take-Grant Protection Model. *Journal of Computer and System Sciences*, 23(3):333–347, December 1981.

[63] Steve Vinoski. CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments. *IEEE Communications*, 35(2):46–55, February 1997.

[64] Michelle S. Wangham. Estudo e Implementação de um Esquema de Autorização Discricionária Baseado na Especificação CORBAsec. Dissertação de mestrado, PGEEL–UFSC, Florianópolis, SC, março de 2000.

[65] Carla M. Westphall. *Um Esquema de Autorização para a Segurança de Sistemas Distribuídos de Larga Escala*. Tese de doutorado, PGEEL–UFSC, Florianópolis, SC, dezembro de 2000.

[66] Carla M. Westphall and Joni S. Fraga. A Large-Scale System Authorization Scheme Proposal Integrating Java, CORBA and Web Security Models and a Discretionary Prototype. In *Proceedings of the IEEE LANOMS'99*, pages 14–25, Rio de Janeiro, RJ, dezembro de 1999.

[67] Carla M. Westphall and Joni S. Fraga. Authorization Schemes for Large-Scale Systems Based on Java, CORBA and Web Security Models. In *Proceedings of the IEEE ICON'99*, pages 327–334, Brisbane, Australia, September 1999.

[68] Carla M. Westphall, Joni S. Fraga, and Michelle S. Wangham. PoliCap—Um Serviço de Política para o Modelo CORBA de Segurança. In *Anais do 18º Simpósio Brasileiro de Redes de Computadores (SBRC 2000)*, pages 355–370, Belo Horizonte, MG, maio de 2000.