Universidade Federal de Santa Catarina

AMPLIAÇÃO DO SEGURAWEB, UM FRAMEWORK RBAC PARA APLICAÇÕES WEB

Júlio Alexandre de Albuquerque Reis

Daniel Quadros da Silva

UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO DE CIÊNCIAS DA COMPUTAÇÃO

AMPLIAÇÃO DO SEGURAWEB, UM FRAMEWORK RBAC PARA APLICAÇÕES WEB

Autor: Júlio Alexandre de Albuquerque Reis

Daniel Quadros da Silva

Orientador: Dr. Carlos Becker Westphall.

Co-orientadora: Dr.ª Carla Merckle Westphall

Banca Examinadora: Dr. Mário Dantas

Palavras-Chave: RBAC, *framework*, autenticação, auditoria

Florianópolis 28 de fevereiro de 2004

A nossas famílias.

Aos professores Westphall, Carla e Dantas.

Agradecimentos

Agradecemos em primeiro lugar aos nossos pais, por ter nos dado esta oportunidade e pelo apoio no decorrer desta caminhada.

Aos meus orientadores Carlos Becker Westphall e Carla Merkle Westphall pela orientação e sugestão para a continuação deste projeto. Aos membros da banca, pelas críticas e sugestões.

Aos nossos amigos, principalmente os que foram colegas de curso e junto caminharam nessa jornada.

Agradecemos especialmente a Kátyra Kowalski Armanini por ter iniciado a construção deste *framework* e nos ter deixado a oportunidade de ampliá-lo.

LISTA DE ABREVIATURAS 7

LISTA DE FIGURAS 9

RESUMO 10

ABSTRACT 11

CAPÍTULO 1 12

INTRODUÇÃO 12

1.1 MOTIVAÇÃO 12
1.2 OBJETIVOS 15
1.3 ORGANIZAÇÃO DO TRABALHO 17

CAPÍTULO 2 18

2.1 O MODELO DE SEGURANÇA RBAC (ROLE-BASED ACCESS CONTROL) 18
2.2 CARACTERÍSTICAS DO MODELO RBAC 21
2.3 USUÁRIOS, PAPÉIS E PERMISSÕES 22
2.4 PAPÉIS E HIERARQUIA DE PAPÉIS 24
2.5 AUTORIZAÇÃO DE PAPÉIS 24
2.6 ATIVAÇÃO DE PAPÉIS 26
2.7 SEPARAÇÃO OPERACIONAL DE TAREFAS 27
2.8 ACESSO DE OBJETOS 28
2.9 FAMÍLIA DE MODELOS RBAC 29
2.9.1 RBAC Básico 30
2.9.2 RBAC Hierárquico 30
2.9.3 RBAC com Restrições 31
2.9.4 RBAC Simétrico 31
2.10 Conclusões do Capítulo 32

CAPÍTULO 3 33

3.1 FRAMEWORKS ORIENTADOS A OBJETO 33
3.2 Como usar um framework 36
3.3 COMO APRENDER UM FRAMEWORK 37
3.4 COMO AVALIAR UM FRAMEWORK 39
3.5 CLASSIFICAÇÃO DE FRAMEWORKS 40
3.6 CICLO DE VIDA DE FRAMEWORKS 42
3.7 DESENVOLVIMENTO DE FRAMEWORKS 43
3.8 METODOLOGIAS DE DESENVOLVIMENTO DE FRAMEWORKS 45

CAPÍTULO 4 46

4.1 UM FRAMEWORK RBAC PARA APLICAÇÕES WEB 47
4.2 TRABALHOS RELACIONADOS 47
4.2.1 Diferenças entre SeguraWeb e RBAC/Web do NIST 48

4.2.2 Diferenças entre SeguraWeb e propostas como RBAC-JACOWEB e Beznosov/Deng *49*
4.2.3 A Proposta ORBAC *49*
4.2.4 Modelo RBAC Baseado em UML *50*
4.2.5 Controle de Acesso para Servidores Web de um Mesmo Domínio *50*
baseado no modelo RBAC comparado com o SeguraWeb *50*
4.2.6 Um Framework Baseado na Descrição de Papéis *51*
4.3 ETAPAS DE ANÁLISE E PROJETO DO FRAMEWORK SEGURAWEB 52
4.3.2 Projeto de Arquitetura *53*
4.3.3 Projeto e Implementação do Framework *55*
4.4 AUTENTICAÇÃO 65
4.5 CONTROLE DE ACESSO 67
4.6 ADMINISTRAÇÃO 68
4.7 IMPLEMENTAÇÃO DAS REGRAS DO MODELO RBAC 73
4.7.1 Hierarquia de Papéis *73*
4.8 APLICAÇÕES DE TESTE 75
4.8.1 A Aplicação SecServer *76*

CAPÍTULO 5 .... **78**

5.1 AUTENTICAÇÃO POR CHAVE PÚBLICA 78
5.2 AUDITORIA 83
5.3 CLASSES ADICIONADAS AO FRAMEWORK 87
5.3.1 Classes de administração: *87*

CAPÍTULO 6 .... **95**

CONCLUSÕES **95**

6.1 REVISÃO DAS MOTIVAÇÕES E OBJETIVOS 95
6.2 VISÃO GERAL DO TRABALHO 95
6.3 CONTRIBUIÇÕES E ESCOPO DO TRABALHO 96
6.4 PERSPECTIVAS FUTURAS 96

REFERÊNCIAS BIBLIOGRÁFICAS **98**

ANEXOS **105**

CÓDIGO FONTE 105

Lista de Abreviaturas

API : *Application Program Inteface*

ASP : *Active Server Pages*

AWT : *Abstract Window Toolkit*

CDR : *Call Detail Record*

CORBA : *Common Object Request Broker Architecture*

DAC : *Discretionary Access Control*

DCOM : *Distributed Common Object Model*

DSOM : *Distributed System Object Model*

HTTP : *Hypertext Transfer Protocol*

JAAS : *Java Authentication and Authorization Service*

JCE : *Java Cryptography Extension*

JDBC : *Java Database Connectivity*

JFC : *Java Foundation Classes*

JSSE : *Java Secure Socket Extension*

JSP : *Java Server Pages*

JVM : *Java Virtual Machine*

LDAP : *Lightweight Directory Access Protocol*

MAC : *Mandatory Access Control*

MFC : *Microsoft Foundation Classes*

NIST : *National Institute of Standards and Technology*

OCL : *Object Constraints Language*

OID : *Object Identificator*

OLE : *Object Linking and Embedding*

ORB : *Object Request Broker*

PBE : *Password-Based Encryption*

RBAC : *Role-Based Access Control*

RMI : *Remote Method Invocation*

SSL : *Secure Socket Layer*

UML : *Unified Modeling Language*

URL : *Uniform Resource Locator*

Lista de Figuras

Figura 1: Associações do modelo RBAC, muitos para muitos.. 22
Figura 2: Associações do modelo RBAC, muitos para muitos.. 22
Figura 3: Operações estão administrativamente associadas com objetos, assim também os papéis 23
Figura 4: Modelo Geral RBAC ([OBE02]). 29
Figura 5: Esquema Geral do Framework Seguraweb. 55
Figura 6: Diagrama da Classe User 56
Figura 7: Diagrama da Classe Role 57
Figura 8: Diagrama da classe Permission 57
Figura 9: Classes correspondentes aos tipos de Autenticação 58
Figura 10: Diagrama da classe PersistentBroker 61
Figura 11: Diagramas das classes brokers dos elementos do modelo RBAC.................. 62
Figura 12: Diagrama de classe de Virtual Proxy 63
Figura 13: Diagrama da classe UserProxy.. 64
Figura 14: Diagrama da classe RoleProxy.. 64
Figura 15: Diagrama da classe PermissionProxy.. 65
Figura 16: Componentes de Autenticação 67
Figura 17: Diagrama de classes de Autorização 68
Figura 18: Diagrama da Classe DBUserManager... 69
Figura 19: Diagrama da Classe Role Manager 70
Figura 20: Diagrama da Classe DBPermissionManager 71
Figura 21: Diagrama da classe RBACUserRoleAssociationManager............................. 72
Figura 22: Diagrama da classe RBACRolePermissionAssociationManager.................... 73
Figura 23: Arquitetura da camada de auditoria... 85
Figura 24: Diagrama de classe de administração de usuário e sua chaves 88
Figura 25: Procedimento de assinatura digital.. 89
Figura 26: Autenticação através da chave pública.. 92
Figura 27: Classe da chave pública... 94

Resumo

O trabalho apresenta a expansão e melhoria de um framework orientado a objeto baseado no modelo de segurança RBAC, SeguraWeb. A expansão foi focada principalmente na implementação de mecanismos de autenticação baseado em chave pública DSA e auditoria. O framework pode ser utilizado para aplicações Web ou em qualquer aplicação utilizando Java. As modificações no framework visam permitir um maior grau de segurança mantendo os desenvolvedores despreocupados com relação à implementação de mecanismos de segurança.

Abstract

This work presents the expansion and improvement of SeguraWeb an object-oriented framework based on RBAC model. This expansion was focused mainly in the development of mechanism of public key authentication DSA and auditorship.

This framework can be used for Web application or any other application that uses Java.

The modifications proposed in the framework intend to allow more security, so that and developers do not need to concern about security mechanisms.

Capítulo 1

Introdução

1.1 Motivação

Com o aumento da popularidade da Internet os sítios e portais da *Web* têm aumentado a sua abrangência e interatividade adicionando mais capacidade de consultas e atualização em banco de dados, assim como, oferecido um crescente número de aplicações remotas via *Web*. Devido a esta tendência as empresas têm utilizado esta tecnologia da Internet em suas Intranet.

Nestas operações o uso de mecanismos seguros é essencial e indispensável, nem o usuário nem as corporações têm interesse em que os dados envolvidos nestas seja violado ou capturado por terceiros.

As redes locais estão presentes em um grande número de corporações que a utilizam para compartilhar arquivos e dispositivos como impressoras, além de possibilitar a troca de mensagens através de correio eletrônico e programas de mensagens instantâneas.

A Intranet facilita o acesso aos dados de uma empresa de qualquer parte do mundo, porém se exige que estes meios sejam seguros, pois senão os dados da empresa ficam expostos a qualquer pessoa ou empresa. A proposta deste trabalho visa aumentar esta segurança para que as transações possam ser feitas pela internet de uma maneira segura.

Por suas características, esse tipo de rede é uma poderosa ferramenta de gestão empresarial e, ao mesmo tempo, um meio de viabilizar o trabalho em grupo na organização. Além de ser mais barata e fácil de usar e mais flexível quando comparado a outras soluções existentes.

As empresas implantam uma Intranet porque podem aproveitar de toda uma infraestrutura, já disponível da Internet, com baixo custo de software, centralizando aplicações pouca necessidade de fazer treinamento aos usuários. Além disso, reduz o

custo com despesas com divulgação e comercialização de produtos e serviços abrindo ainda um rápido canal de comunicação entre a empresa e os clientes. Uma empresa pode ter seus funcionários trabalhando em outro local, fazendo acesso remotamente à empresa, inclusive clientes podem consultar e atualizar banco de dados de um fornecedor, por exemplo.

Com a utilização cada vez maior da Internet e das Intranets dentro das empresas, o crescimento do uso de aplicações *Web* se torna inevitável, surgiu também à necessidade de se utilizar melhores mecanismos para prover a segurança das transações de informações confidenciais. Considerando que em uma empresa existem funcionários com atividades e funções diferentes, onde uns possuem mais privilégios que outros, existe a necessidade de controlar os privilégios dos funcionários na utilização das aplicações *Web* da empresa.

Estes privilégios de acesso podem somente ser garantidos se forem utilizados sob um meio seguro, em que se possa garantir a confidencialidade, onde as informações só são reveladas para pessoas ou organizações envolvidas autorizadas; integridade, onde seja garantindo que a informação só pode ser modificada por funcionários ou organizações envolvidas autorizados das maneiras autorizadas; responsabilidade, na qual funcionários são responsáveis por suas ações relacionadas à segurança; e acessibilidade (disponibilidade e não-repúdio), funcionários autorizados não podem ter seu acesso negado maliciosamente[AKK02].

Além disso, os problemas de segurança cresceram e se tornaram mais difíceis com o crescimento das empresas que levou a um aumento do volume de informação e do número de funcionários [OH00].

É necessária a existência de serviços de segurança como autenticação, auditoria e administração. O controle de acesso por si só não é uma solução completa para obtenção de segurança em um sistema [SAN94].

A autenticação é para garantir que um usuário é quem ele diz ser, portanto garantir também a responsabilidade no acesso aos dados. O controle de acesso serve para controlar o que um usuário pode acessar em um sistema. Comunicações seguras são para proteger informações em trânsito entre componentes. A auditoria de segurança é

realizada para gravar e analisar o que o usuário faz no sistema. A administração de segurança, para gerenciar informações de segurança incluindo as políticas de segurança.

De acordo com um estudo do NIST (*National Institute of Standards and Technology*), essa necessidade de controles de acesso de acordo com os papéis que os usuários e ou funcionários individualmente desempenham na organização define uma política de segurança denominada de RBAC (*Role-Based Access Control*), ou seja, um modelo de segurança baseado em papéis [WES00].

O *Role Based Access Control* (RBAC) é um termo utilizado para descrever políticas de segurança que controlam o acesso de usuários a recursos computacionais, baseado na construção de *roles* (papéis, funções). Esses papéis definem um conjunto de atividades concedidas para usuários autorizados. Para muitos tipos de organização, o modelo RBAC fornece um modo mais intuitivo e eficaz de representar e gerenciar autorizações às informações que outras formas de controle de acesso, pois se pode imaginar um papel como se fosse um cargo ou posição dentro de uma organização, que representa a autoridade necessária para conduzir as tarefas associadas[JAN98] [OH00].

Outra motivação deste trabalho é a reusabilidade de componentes, que atualmente é reconhecida como uma forma importante de aumentar a produtividade no desenvolvimento de software evitando que se implemente várias vezes a mesma base de diferentes aplicações, principalmente à parte de segurança. O potencial de reutilização das partes de análise e projeto de um sistema ainda é pouco explorado, os programadores com mais experiência em sua maioria já reutilizam código e consultam códigos antigos, desperdiçando as vantagens que o reuso de partes de análise e projeto podem proporcionar [LAN95].

Dentro deste contexto existe o conceito de um *framework* que permite a reutilização da parte de análise e projeto não somente a reutilização do código [LAN95].

Um *framework* nada mais é do que uma estrutura de classes inter-relacionadas, que corresponde a uma implementação incompleta para um conjunto de aplicações de um mesmo domínio. Esta estrutura de classes deve serve como base e pode ser adaptada para a geração de aplicações específicas.

Outra motivação decorre de que para que a segurança seja completa, além do controle de acesso e necessário autenticação, auditoria e administração. A

confidencialidade e autenticação e distribuição de chaves é possível se utilizando a criptografia por chave pública ou assimétrica que consiste na utilização de um par de chaves, uma pública e outra privada, diferentemente da criptografia convencional, que utilizava a mesma chave para criptografar e decriptar. A pública, como o próprio nome diz é de conhecimento público e é divulgada em diversas maneiras. A chave privada é de conhecimento somente do próprio usuário. O que for encriptado utilizando uma das chaves somente poderá ser visualizado com a outra.

Com a chave pública é possível prover os serviços de confidencialidade, autenticação e distribuição de chaves.Estes privilégios de acesso podem somente ser garantidos se forem utilizados sob um meio seguro, em que se possa garantir:

- Confidencialidade ou sigilo, ou seja, garantir que somente para as pessoas ou organizações envolvidas na comunicação possam ser reveladas as informações para serem lidas ou utilizadas;
- Integridade, para garantir que o conteúdo de uma mensagem ou resultado de uma consulta não será alterado durante seu tráfego ou a informação armazenada só pode ser modificada por funcionários autorizados das maneiras autorizadas;
- Autenticação para se garantir a identificação das pessoas ou organizações envolvidas na comunicação e apurar suas responsabilidades, funcionários são responsáveis por suas ações relacionadas à segurança;

- Não-repúdio, garantia que o emissor de uma mensagem ou a pessoa que executou determinada transação de forma eletrônica, não poderá, posteriormente negar sua autoria, e também funcionários autorizados não podem ter seu acesso negado maliciosamente.

1.2 Objetivos

Baseado nas condições que as empresas e corporações encontram, como foi descrito acima, o objetivo deste trabalho é ampliar um *framework* baseado no modelo de segurança RBAC que possa ser utilizado por aplicações *Web* e também desenvolver uma aplicação exemplo. O *framework* é composto por várias classes (concretas e abstratas) que implementam mecanismos de autenticação, controle de acesso, auditoria e administração. A grande vantagem da utilização do *framework* é que se diminui muito o tempo de desenvolvimento, principalmente em relação à segurança das aplicações *Web* ou de aplicações Java que utilizem a política RBAC [AKK02].

Este *framework* foi desenvolvido inicialmente com o propósito de ser utilizado principalmente por aplicações com interface *Web*, por este motivo ele foi implementado na linguagem de programação Java, permitindo uma gama de aplicações *Web* multiplataforma.

O item de segurança de comunicação foi omitido, já que no ambiente proposto este item é garantido pelo protocolo SSL (*Secure Socket Layer*), o qual já se encontra implementado na maioria dos servidores *Web* existentes no mercado. Caso outra linguagem de programação fosse utilizada seria necessária uma preocupação com relação à comunicação segura.

Devido às funcionalidades e vantagens que um *framework* RBAC pode proporcionar este foi expandido para permitir autenticação de chave pública. Garantindo então os serviços de confidencialidade, autenticação e distribuição de chaves.

Para alcançar este objetivo geral, os seguintes específicos foram definidos:

- Estudo do modelo de segurança RBAC;
- Estudo do *framework* RBAC SeguraWeb;
- Estudo dos mecanismos de segurança: controle de acesso, criptografia e autenticação;
- Implementação do algoritmo de assinatura digital, DSA utilizando a linguagem Java;

1.3 Organização do Trabalho

Este trabalho está organizado em cinco capítulos. O primeiro capítulo descreveu a motivação e o contexto no qual este trabalho está inserido, também abrangendo o objetivo geral e os objetivos específicos.

O capítulo 2 apresenta uma breve classificação dos modelos de controle de acesso, e alguns conceitos relacionados. Descrevem também os elementos e regras do modelo.

O capítulo 3 apresenta os conceitos de *frameworks* e os aspectos que caracterizam um *framework*. Também são apresentados a classificação dos *frameworks* e seu ciclo de vida, bem como resumidamente as fases e as metodologias de desenvolvimento de *frameworks*.

No capítulo 4 são apresentadas as partes da estrutura do *framework* desenvolvidas, bem como foram implementadas as regras do modelo RBAC dentro do *framework*.Por fim, no último capítulo são apresentadas algumas conclusões e perspectivas da continuidade deste trabalho.

O capítulo 5 conceitua e descreve a criptografia por chave pública e suas aplicações. Também são descritos os diversos modos de utilização da chave pública dependendo da finalidade desejada. Descreve também as diferenças entre criptografia por chave pública e chave privada. Como foi implementado o modelo DSA no *framework* SeguraWeb. O capítulo também descreve os conceitos de auditoria e como eles podem ser usados para aumentar o nível de segurança do *framework.* Finalmente é descrito como foi feita a implementação da autenticação e de auditoria.

Por fim o capítulo 8 descreve as conclusões e algumas indicações para trabalhos futuros.

Capítulo 2

Introdução

Existem diversos tipos de políticas de acesso, algumas destinadas a meios acadêmicos outras muito usadas em meios militares e uma proposta mais flexível e de fácil gerenciamento a política de acesso RBAC. Na política de acesso RBAC existem algumas características fundamentais que permitem o seu fácil gerenciamento, uma destas características é a definição estar mais próxima do real funcionamento de uma empresa

Dentro da política RBAC existem conceitos importantes com papéis, usuários e permissões. Algumas regras têm que ser seguidas para que estes conceitos sejam relacionados como ativação, autorização e separação operacional de tarefas.

E por fim existe uma classificação de acordo com as principais características do modelo RBAC

2.1 O Modelo de Segurança RBAC (Role-Based Access Control)

Usuários e privilégios estão espalhados por diversas plataformas e aplicações, nesta situação fica difícil distinguir quem são os usuários válidos, quais são os privilégios de cada um, como manter os direitos de acesso atualizados e como especificar e forçar políticas de acesso. Para gerenciar todos os aspectos citados é necessário se implantar uma política de acesso e também entender o que significa uma política de acesso.

Para a melhor compreensão do que significa um controle de acesso é necessário se definir dois termos importantes, sujeito e objeto. Um sujeito é uma entidade ativa em um sistema computacional, o qual inicia as 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, todos os sistemas computacionais podem ser descritos como

sujeitos acessando objetos. O controle de acesso é o mediador entre as requisições de acesso de sujeitos a objetos [OBE01].

Uma política de controle de acesso é uma forma de relacionar sujeitos e objetos, ou seja, quais operações serão permitidas ao sujeito e quais serão proibidas. De acordo com [RAM94], uma política de segurança é "um conjunto de regras, aplicadas sobre um domínio, que específica o que é e o que não é permitido no campo da segurança".

Existem três classes quanto às políticas de controle de acesso:

- **Controle de Acesso Discricionário (*Discretionary Access Control* - DAC):** Em uma política discricionária, os direitos de acesso a cada recurso, a cada informação, são manipulados livremente pelo responsável do recurso ou da informação, segundo a sua vontade (à sua discrição). As políticas discricionárias não são coerentes [WES00], ou seja, podem ter estados inseguros, pois o usuário quem atribui livremente os direitos de acesso.
- **Controle de Acesso Obrigatório (*Mandatory Access Control* - MAC):** As políticas ditas obrigatórias ou não-discricionárias resumem em seus esquemas de autorização um conjunto de regras rígidas que expressam um tipo de organização envolvendo a segurança das informações no sistema como um todo. A política obrigatória supõe que os usuários e objetos ou recursos do sistema estão todos etiquetados. As etiquetas dos objetos seguem uma classificação específica enquanto os usuários ou os sujeitos do acesso possuem níveis de habilitação. Os controles que determinam as autorizações de acesso comparam a habilitação do usuário com classificação do objeto. As regras definidas nesses controles que são ditas incontornáveis asseguram que o sistema verifique as propriedades de confidencialidade e de integridade [WES00].
- **Controle de Acesso Baseado em Papéis:** Com o controle de acesso baseado em papéis, as decisões de acesso são baseadas nos papéis que os próprios usuários têm como parte de uma organização. Os usuários assumem determinados papéis (como por exemplo, caixa, gerente, atente, supervisor etc). O processo de definição de papéis requer uma análise

completa de como uma organização funciona e pode abranger uma grande variedade de usuários em uma organização [NIBU98]

O *Role Based Access Control* (RBAC) é uma política de controle de acesso que prove mecanismos de segurança que controlam o acesso de usuários a recursos computacionais, baseado na construção de papéis. Esses papéis definem um conjunto de atividades concedidas para usuários autorizados. Um papel pode ser um cargo ou posição dentro de uma organização, que representa a autoridade necessária para conduzir as tarefas associadas [JAN98]. Os usuários pertencem a um papel de acordo com suas responsabilidades e qualificações e podem ser facilmente transferidos sem modificar a estrutura de acesso básica. Novas permissões e incorporação de novas aplicações e ações podem ser concedidas aos papéis, e as permissões podem ser revogadas quando necessário [FER95]. O modelo RBAC fornece um modo mais intuitivo e eficaz de representar e gerenciar autorizações às informações em comparação a outras formas de controle de acesso para a maioria dos pesquisadores [JAN98].

Na opinião de pesquisadores e especialistas muitos requisitos práticos não são cobertos pelas políticas discricionárias e obrigatórias. As políticas obrigatórias se originam em ambientes rígidos, como o ambiente militar. Já as políticas discricionárias têm origem em ambientes acadêmicos. Nenhum destes dois tipos de política satisfaz as necessidades da maioria dos ambientes empresariais, por este motivo o modelo RBAC tem atraído cada vez mais atenção para o desenvolvimento de aplicações comerciais.

As políticas baseadas em papéis se beneficiam por sua independência lógica, a qual específica as autorizações de um usuário em duas partes: uma que relaciona os usuários aos papéis e outra que relaciona estes papéis aos direitos de acesso a um objeto. Isto simplifica a administração de segurança, pois quando a mudança de um funcionário muda de cargo basta associa-lo a outro papel. Se todas as autorizações fossem entre usuários e objetos diretamente, seria necessário retirar todos os direitos de acesso existentes ao usuário e associar os novos direitos de acesso. Esta é uma tarefa trabalhosa e que consome tempo [SAN94], principalmente em grandes empresas, onde a quantidade de usuários é enorme, podendo chegar a milhares.

No modelo RBAC, a segurança é gerenciada em um nível muito próximo à estrutura real da organização. Cada usuário está associado a um ou mais papéis, similarmente ao seu cargo na organização, e cada papel está associado a um ou mais privilégios que são concedidos aos usuários daquele papel, onde estes privilégios são atribuídos de acordo com sua ocupação na organização. Os papéis podem possuir hierarquia. Alguns papéis podem abranger todos as permissões de um papel e mais alguns outros. A administração de segurança com RBAC consiste em determinar as operações que devem ser executadas por pessoas em tarefas específicas e associar os sujeitos (empregados) aos papéis apropriados[AKK02].

As principais vantagens da utilização do modelo RBAC são:

- A administração central e local é melhorada com políticas de controle de acesso baseado em papéis;
- Altamente configurável, com riqueza de parâmetros;
- Expande o controle de acesso além dos limites físicos da empresa, autorizando tanto empregados como fornecedores e consultores.

- A associação de papéis é baseada em competências, responsabilidade, autoridade, dando ao usuário a autorização necessária para executar privilégios;

- Orientado a papéis, que são globais e persistentes.

2.2 Características do Modelo RBAC

As políticas RBAC são descritas em termos de usuários, sujeitos (*subjects*), papéis, hierarquia de papéis, operações e objetos (que são os alvos de proteção). Para executar uma operação em um objeto controlado por RBAC, o usuário deve estar ativo em algum papel. Para estar ativo ele necessita estar autorizado, como membro do papel, pelo administrador do sistema.Um usuário pode ser membro de mais de um papel, e um papel pode ter múltiplos membros, formando uma associação de muitos-para-muitos.Do

mesmo modo um papel pode ter várias permissões, e uma mesma permissão pode estar associada a muitos papéis, também formando uma associação de muitos-para-muitos.

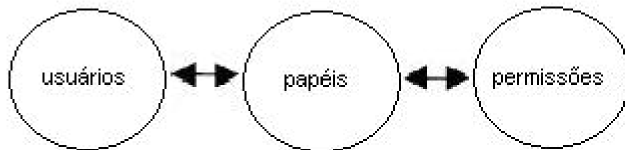

Figura 1: Associações do modelo RBAC, muitos para muitos

2.3 Usuários, Papéis e Permissões

Em um *framework* RBAC existem várias definições que se relacionam umas com as outras, um *usuário* é uma pessoa, um *papel* é um conjunto de funções de trabalho e uma *operação* representa um modo específico de acesso a um conjunto de um ou mais *objetos* RBAC protegidos. Um sujeito representa um processo de usuário ativo com uma seta indicando um relacionamento de um para muitos[AKK02].

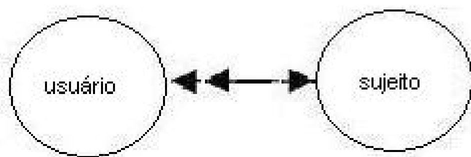

Figura 2: Associações do modelo RBAC, muitos para muitos

Os tipos de operações e objetos que o RBAC controla é dependente do tipo de sistema na qual ele será implementado. Em um sistema operacional, as operações podem incluir leitura, escrita e execução. Em um sistema gerenciador de base de dados, as operações podem incluir inserção, remoção e atualização, etc. O conjunto de objetos incluídos no modelo RBAC inclui todos os objetos acessíveis pelas operações RBAC. Entretanto, nem todos os objetos do sistema e do sistema de arquivos necessitam ser incluídos em um esquema RBAC, como o acesso aos objetos de infra-estrutura, os objetos de sincronização (semáforos, *pipes*, segmentos de mensagens, monitores) e objetos temporários (arquivos e diretórios temporários) não necessitam ser controlados dentro do conjunto de objetos protegidos RBAC [FER95].

Uma operação representa uma unidade de controle que pode ser referenciado por um papel em particular, que é o responsável por regular as restrições dentro do *framework* RBAC. É importante notar a diferença entre um simples modo de acesso e uma operação. Uma operação pode ser utilizada para capturar detalhes de segurança relevantes ou restrições que não podem ser determinadas por um simples modo de acesso. Estes detalhes podem ser em termos de método ou granularidade de acesso.

Para demonstrar a importância de uma operação RBAC, considere a diferença entre o acesso de um caixa e um supervisor de contas em um banco. A empresa define um papel de caixa como sendo capaz de realizar uma operação de recebimento de pagamento. Isto requer acesso de leitura e escrita a campos específicos em um arquivo de contabilidade. A empresa também pode definir um papel de supervisor de conta na qual é permitido realizar operações de correção. Estas operações requerem acesso de leitura e escrita aos mesmos campos de um arquivo de contabilidade, igualmente ao caixa. Entretanto, o supervisor de conta não pode ter permissão para iniciar registro de pagamento, ele pode somente realizar correções depois destas ações terem acontecido. Da mesma forma, não é permitido ao caixa realizar qualquer correção em registro de pagamentos que já foram completadas, já que não é sua função. A diferença entre estes dois papéis está nas operações que são executadas. Para demonstrar a importância da granularidade de controle, considere, em uma oficina com o sistema computadorizado, a necessidade de um mecânico de acessar o histórico de um carro para conferir as revisões que foram feitas numa determinada oficina e quais reparos foram executados. Embora essas operações sejam necessárias, o mecânico pode não ter acesso de leitura ou alteração em outras partes do registro do carro.

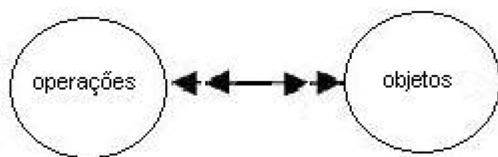

Figura 3: Operações estão administrativamente associadas com objetos, assim também os papéis

Quando um usuário autorizado é associado a um papel, este usuário tem o poder, ou melhor, a capacidade de executar as operações que estão associadas com o papel. Cada operação é referenciada com um único identificador [FER95].

2.4 Papéis e Hierarquia de Papéis

Usuários pertencentes a diferentes papéis podem necessitar realizar operações comuns acarretando em uma sobreposição de responsabilidades e privilégios. Existe também, em muitas empresas, um conjunto de operações gerais que são realizadas por todos os funcionários. Pode-se notar que fica inviável específicar todas as operações para cada papel que é criado devido ao enorme número de específicações que teriam de ser feitas em grandes corporações. Para tornar mais eficiente e proporcionar à estrutura natural de uma empresa, RBAC inclui o conceito de hierarquias de papéis. Uma hierarquia de papel define papéis que têm atributos específicos e que podem englobar outros papéis, um papel pode incluir implicitamente as operações, restrições, e objetos que são associados a outro papel. As hierarquias de papéis são um modo mais próximo a realidade de organizar papéis para refletir autoridade, responsabilidade, e competência [FER95].

Um exemplo de uma hierarquia de papel é mostrado na Figura 4. Um papel pode conter as funcionalidades de outros papéis, ou seja, numa hierarquia, um funcionário de mais alto escalão, terá um papel associado ao seu cargo que englobará as operações, restrições e objetos dos papéis dos cargos subseqüentes. É importante salientar que dois papéis podem não estar relacionados na mesma hierarquia e ambos conterem um terceiro.

A hierarquia de papéis (*Role Hierarchy*), descreve a propriedade de associação implícita de papéis, formando uma regra que permite um sujeito autorizado a acessar um papel e se este papel contém um outro papel, então ao sujeito também é permitido acessar o papel contido.

2.5 Autorização de Papéis

A associação de um usuário com um papel pode estar sujeita a três propriedades.

Primeiramente, pode não ser dado nenhum privilégio a mais ao usuário que seja necessário para executar seu trabalho a assegurar adesão ao princípio de Menor Privilégio. A primeira propriedade então remete ao princípio de Menor Privilégio requer que não seja dado a um usuário nenhum privilégio a mais do que necessário para este executar as funções que para seu cargo foram destinadas.

Para garantir o menor privilégio exige identificar as funções do trabalho do usuário, determinando o conjunto de privilégios mínimo exigido para executar aquela função, e restringindo o usuário a um domínio com esses privilégios e nada mais. Nas políticas que não utilizam o modelo RBAC, isto é difícil ou caro de alcançar. Pela inaptidão dos sistemas em definir o acesso baseado em vários atributos ou restrições podem ser permitidos, por exemplo, mais privilégios a alguém associado a uma categoria de trabalho do que a pessoa precisa. Sabendo-se que muitas das responsabilidades se sobrepõem entre categorias de trabalho, o privilégio de máximo acesso para cada categoria de trabalho poderia causar acesso ilegal ou são necessárias mais auditorias e monitoramento sobre um tipo de funcionário ou função acarretando em mais custo. O RBAC pode ser configurado de forma que só essas operações que necessitam ser executadas por membros de um papel sejam concedidas ao papel, e estas operações e papéis podem estar sujeitas a políticas organizacionais ou restrições. Se houverem operações que se sobrepõem, podem ser estabelecidas hierarquias de papéis. Com RBAC, podem ser colocadas restrições ao mesmo tipo de funcionário, somente este podendo ter acesso aos papéis de seu cargo e mais, por exemplo, somente as funções de seu cargo e sua filial podem ser associadas ao seu papel[AKK02].

A segunda propriedade referente a um papel mutuamente exclusivo a outro é destinada a preservar a política de Separação Estática de Tarefas ou conflito de interesses. O papel no qual o usuário está se tornando membro pode ser mutuamente exclusivo com outro papel para o qual o usuário já seja membro. Isto significa que em virtude de um usuário estar autorizado como um membro de um papel, o usuário não está autorizado como membro de um segundo papel. Por exemplo, um usuário que está autorizado para ser um membro de um papel de Caixa em um banco pode não ter permissão para ser membro de um papel de Auditor do mesmo banco. Assim dizendo, os papéis de Caixa e Auditor são mutuamente exclusivos [FER95]. A Separação Estática de Tarefas (*Static*

Separation of Duty) somente permite que um usuário pode se tornar membro deu um novo papel caso não for membro de nenhum outro papel não exclusivo a ele.

A terceira e última propriedade refere-se a limitação numérica, concedida a um usuário membro de papéis é a propriedade da Cardinalidade. Alguns papéis só podem ser ocupados por um certo número de empregados a qualquer determinado período de tempo. Por exemplo, considere o papel de Presidente. Embora outros empregados possam agir naquele papel, só um empregado pode assumir as responsabilidades de um presidente em um certo momento. Um usuário pode se tornar um membro novo de um papel contanto que o número de membros permitidos ao papel não esteja excedido. A Cardinalidade(*Cardinality*) significa a capacidade de um papel não pode ser excedida por um membro adicional ao papel.

2.6 Ativação de Papéis

O RBAC exige que um usuário seja primeiramente autorizado, bem como ser ativado em um papel, antes que possa executar uma ação. No RBAC cada sujeito é um mapeamento de um usuário a um ou possivelmente a vários papéis. Um usuário estabelece uma sessão durante a qual o usuário está associado com um subconjunto de papéis dos quais ele é membro. Uma autorização de papel de um usuário é necessária, mas nem sempre é uma condição suficiente para um usuário ter permissão de executar uma operação é necessária ainda a ativação do papel.A ativação de papéis fornece o contexto para o qual estas políticas organizacionais podem ser aplicadas[AKK02].

Um papel pode ser ativado se (Dependendo da política organizacional) as verificações abaixo são satisfeitas:

- O usuário é autorizado para o papel que está sendo proposto para ativação;
- A ativação do papel proposto não é mutuamente exclusiva a alguma outra regra ativa do usuário;
- A operação proposta é autorizada para o papel que está sendo proposto para ativação;
- A operação que é proposta é consistente dentro de uma seqüência obrigatória de operações.

A Autorização de Papéis (*Role Authorization*) caracteriza proibição de um sujeito ter um papel ativo que não está autorizado para aquele sujeito.Uma vez que é determinado o que um papel faz parte do conjunto de papéis autorizado para o sujeito, a operação pode ser executada contanto que o papel esteja ativo. Embora um papel possa estar no conjunto de papéis, pode haver certas políticas organizacionais (como a Separação Dinâmica de Tarefas) que impedem que o papel seja ativado.

A Execução de Papéis (*Role Execution*): Um sujeito pode executar uma operação somente se o sujeito estiver agindo dentro de um papel ativo:

O modelo RBAC também proporciona aos administradores a capacidade de impor uma política específica de organização, a Separação Dinâmica de Tarefas. A Separação Estática de Tarefas proporciona à empresa a capacidade de tratar potenciais conflitos de interesses em relação ao momento em que um usuário membro de um papel é autorizado para este. Porém, em algumas organizações é permissível a um usuário ser um membro de dois papéis que não estabeleçam conflitos de interesse quando agem independentemente, mas introduzem algumas restrições políticas quando sua ação ocorre simultaneamente [FER95]. Por exemplo, um usuário pode estar autorizado para dois papéis, mas pode assumir dinamicamente um único papel de dois papéis.

A Separação Dinâmica de Tarefas (*Dynamic Separation of Duty*) não permite que um sujeito possa se tornar ativo em um novo papel a menos que o papel proposto não seja mutuamente exclusivo a quaisquer outros papéis no qual o sujeito é atualmente ativo.

A Autorização de Operação (*Operation Authorization*) específica que um sujeito só pode executar uma operação se esta operação é autorizada para o papel ativo proposto do sujeito Um sujeito pode executar uma operação somente se a operação é autorizada para o papel em que o sujeito está atualmente ativo.

2.7 Separação Operacional de Tarefas

O modelo RBAC pode ser utilizado por um administrador de sistema para obrigar a política de Separação Operacional de Tarefas. A Separação Operacional de Tarefas

pode ser um método valioso para intimidar fraudes. Isto está baseado na idéia de que a fraude pode acontecer se existir colaboração entre várias capacidades relacionadas ao trabalho dentro de uma função empresarial crítica. Se cada uma das operações é executada através de papéis diferentes, a probabilidade de fraude pode ser reduzida. Se for permitido a um usuário executar muitas operações, a fraude pode ocorrer mais facilmente, pois basta um funcionário desonesto.

A Separação Operacional de Tarefas determina que para todas as operações associadas com uma determinada função de negócio, nenhum usuário individualmente pode ter permissão para executar todas essas operações. Portanto, pode ser percebida uma falha pela organização se ocorrer uma falha na execução de um papel. Nos termos do modelo RBAC, a Separação Operacional de Tarefas pode ser imposta quando papéis são autorizados para usuários individuais e quando operações são associadas aos papéis [FER95].

A Separação Operacional de Tarefas (*Operational Separation of Duty*) determina que um papel pode ser associado com uma operação de uma função de negócio somente se o papel for autorizado para o sujeito e que este papel não tenha sido associado anteriormente para todas as outras operações.

2.8 Acesso de Objetos

A Autorização de Acesso a Objeto (*Object Access Authorization*) tem que obedecer a uma seqüência de verificações.Um sujeito pode acessar um objeto somente se o papel é parte do conjunto de papéis do sujeito ativos atualmente, bem como se ao papel é permitido executar a operação, e também se a operação de acesso ao objeto é autorizada.

O esquema geral do modelo RBAC, mostrando a relação entre usuários, papéis e permissões, a hierarquia de papéis e as restrições podem ser visto na Figura 4.

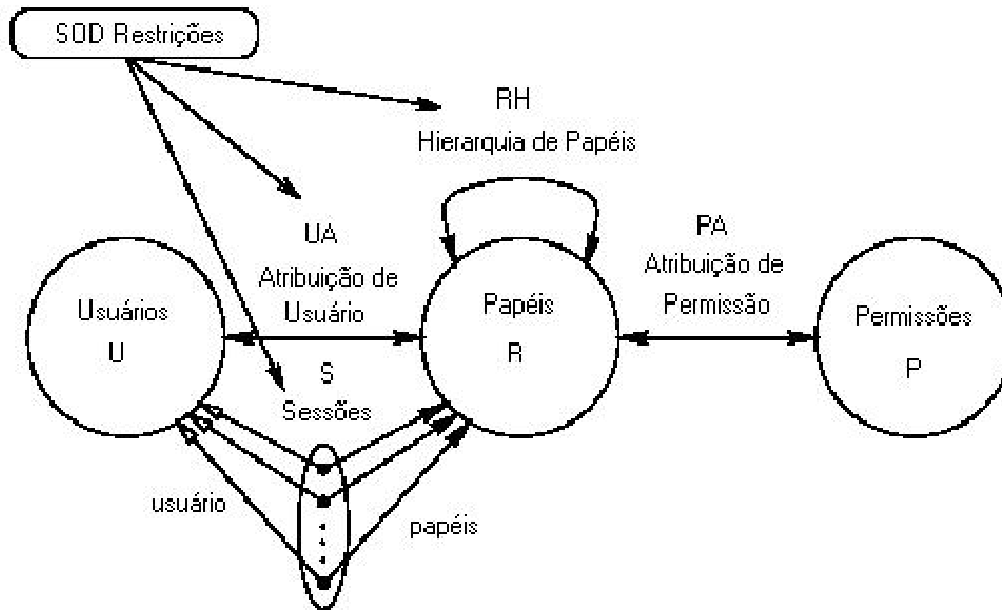

Figura 4: Modelo Geral RBAC ([OBE02]).

2.9 Família de Modelos RBAC

O RBAC é um conceito bastante amplo e aberto, que abrange um conjunto de complexidade que vai desde o muito simples até o extremamente sofisticado. Existe um consenso representando apenas um dos pontos possíveis dentro do conjunto visto que estaria fora da realidade um único modelo definitivo para o RBAC, uma vez que seria demasiadamente restritivo ou excessivamente complexo [OBE01]. 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 componente adicionais que acrescentam funcionalidade e requisitos ao modelo básico. A família de modelos RBAC96 [SAN96] é, possivelmente, o exemplo mais conhecido desta linha; o modelo RBAC-NIST também compartilha desta abordagem[OBE01].

A família RBAC-NIST define quatro modelos:

- RBAC Básico (*Flat RBAC*) ou RBAC0;
- RBAC Hierárquico (*Hierarchical RBAC*) ou RBAC1;
- RBAC com Restrições (*Constrained RBAC*) ou RBAC2;
- RBAC Simétrico (*Symmetric RBAC*) ou RBAC3.

2.9.1 RBAC Básico

O RBAC Básico inclui os aspectos essenciais do modelo RBAC. O conceito básico de RBAC é que usuários e permissões são associados a papéis e que os usuários adquirem permissões sendo membros de papéis. O modelo RBAC NIST exige que a associação de usuários-papéis e permissões-papéis seja de muitos-para-muitos. Deste modo, o mesmo usuário pode estar associado a muitos papéis e um único papel pode possuir muitos usuários. O mesmo modo de pensamento é transferido às permissões [SAN00].

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[SAN98]. O significado exato 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 uma determinada tarefa. Quando todos os papéis são ativados, por outro lado, o princípio do mínimo privilégio dificilmente pode ser respeitado [OBE01].

Uma outra exigência do RBAC Básico é o suporte à revisão usuário-papel, o que possibilita determinar quais os usuários associados a um papel e quais os papéis associados a um usuário (uma exigência semelhante é imposta para a revisão permissão papel no RBAC Simétrico) [SAN00].

2.9.2 RBAC Hierárquico

O RBAC Hierárquico exige que exista o suporte a hierarquia de papéis. Estas hierarquias são representadas matematicamente por relações de ordem-parcial [SAN96], ou grafos acíclicos dirigidos [FER99]. O RBAC Hierárquico pode ser subdivido em:

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 [OBE01].

O RBAC Hierárquico exige que o mecanismo de revisão usuário-papel do RBAC Básico seja estendido para suportar hierarquia 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) [OBE01].

2.9.3 RBAC com Restrições

O RBAC com Restrições tem como característica dar suporte ao princípio da separação de tarefas.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.

2.9.4 RBAC Simétrico

O RBAC simétrico tem como requisito único o suporte a revisão de associações permissão-papel, possibilitando identificar as permissões associadas a um papel e também os papéis que possuem determinadas permissões [SAN00].

2.10 Conclusões do Capítulo

Apresentou-se neste capítulo uma visão geral das classes das políticas de controle de acesso, principalmente nas políticas de controle de acesso baseadas em papéis, pois esta política de controle de acesso utilizada neste trabalho. Foram apresentadas as características do modelo de segurança RBAC e as regras definidas neste modelo de forma mais detalhada. Por fim foi apresentado cada modelo pertencente à família de modelos do NIST mostrando algumas características de cada uma das famílias de modelos RBAC[AKK02].

Capítulo 3

Introdução

Os *frameworks* são ferramentas muito úteis por permitirem a reutilização das etapas de análise e projeto além é claro do código. Um *framework* é uma base de uma aplicação e pode ser utilizado por um domínio de aplicações, pois ele sempre é construído com este propósito. Existe a necessidade de aprender e saber como utilizar um *framework* e esta tarefa nunca pode ser mais árdua que a implementação da aplicação sem o uso de um *framework.* Caso isto aconteça, o *framework* não foi bem construído para o domínio proposto.

3.1 *Frameworks* Orientados a Objeto

O poder computacional e a largura de banda vêm crescendo drasticamente na ultima década, porém o custo para se desenvolver software ainda é caro e propenso à falhas. Muito do custo e do esforço do desenvolvimento do software vem da redescoberta e da reinvenção de conceitos já existentes na indústria de software. O crescimento da heterogeneidade de arquiteturas de hardware e a diversidade de Sistemas Operacionais e comunicação de plataformas tornam difícil a construção de correta, eficiente, portável e barata do zero, ou seja, começar um programa do zero e ele ter todas as características citadas acima.

Frameworks de aplicações orientadas a objetos é uma tecnologia que pretende reusar projetos de softwares e suas implementações para reduzir os custos e melhorar a qualidade deste. Um *framework* é uma aplicação semicompleta que pode ser reusada para outras aplicações. No contraste a umas técnicas mais adiantadas reusar de Orientação a Objetos baseadas em bibliotecas da classe, os *frameworks* são usados para unidades de negócio particulares (tais como comunicações de processo, de dados ou celulares) e domínios da aplicação (tais como relações de usuário).

A diferença entre a demanda e a produção de *software* está aumentando continuamente e uma maneira de aumentar a produção de *software* é introduzir o conceito de reusabilidade de *software* [LAN95]. Segundo [LAN95] o conceito de reusabilidade é não desenvolver nada que já existe, mas sim reutilizá-lo. Isto culminará na diminuição do tempo de desenvolvimento.

A reusabilidade provê uma interface estável provida pelo *framework* aumenta a reusabilidade pela definição genérica de componentes que podem ser reaplicados para a criação de novas aplicações.

O projeto de software para reusabilidade tem como objetivo produzir componentes de software genéricos e extensíveis. Os analistas precisam prever possíveis aplicações futuras e incorporar as exigências destas aplicações no projeto atual. Para executar esta tarefa, o enorme número de decisões de projeto que os analistas têm que tomar devem ser limitadas. Tradicionalmente isto tem sido feito provendo bibliotecas de funções específicas de domínio ou bibliotecas de classes reutilizáveis [LAN95].

Os *frameworks* realçam a modularidade, isto ajuda a melhorar a qualidade de um software pela localidade de um impacto de uma mudança no projeto ou na interface reduzindo o esforço para compreender e dar manutenção a um software existente.

De acordo com [LAN95] a reusabilidade de componentes de *software* é reconhecida como um importante meio de aumentar a produtividade no desenvolvimento de *software*. Os programadores experientes sempre reutilizam código utilizando sua experiência ou buscando projetos de códigos antigos, mas a reutilização das partes de análise e projeto possui um potencial significativamente grande. O conceito de *frameworks* torna possível a reutilização não somente de código, mas também das partes de análise e projeto [LAN95]. Para entender esta afirmação é necessário conhecer o conceito de *framework*.

De acordo com [SIL00], a abordagem de *frameworks* orientados a objeto utiliza o paradigma de orientação a objetos para produzir uma descrição de um domínio para ser reutilizada. Essa abordagem utiliza principalmente os conceitos de abstração de dados, polimorfismo e herança [JOH97] [FAY99].

Existem várias definições para *frameworks*, o conceito que utilizado para o desenvolvimento deste *framework* foi o de [SIL00], que define um *framework* como uma

estrutura de classes inter-relacionadas, que corresponde a uma implementação incompleta para um conjunto de aplicações de um domínio. Esta estrutura de classes deve ser adaptada para a geração de aplicações específicas.

Existe uma diferença fundamental entre um *framework* e a reutilização de classes de uma biblioteca, é que neste caso são usados artefatos de *software* isolados, cabendo ao desenvolvedor estabelecer sua interligação, e no caso do *framework*, é procedida a reutilização de um conjunto de classes inter-relacionadas – inter-relacionamento estabelecido no projeto do *framework* [SIL00].

De acordo com [SIL00] e [TAL95] dois aspectos caracterizam um *framework*:

- Os *frameworks* fornecem infra-estrutura e projeto: *frameworks* possuem infra-estrutura de projeto, que é disponibilizada ao desenvolvedor da aplicação e reduz a quantidade de código a ser desenvolvida, testada e depurada. As interconexões pré-estabelecidas definem a arquitetura da aplicação, liberando o desenvolvedor desta responsabilidade. O código escrito pelo desenvolvedor visa estender ou particularizar o comportamento do *framework*, de forma a moldá-lo a uma necessidade específica;
- Os *frameworks* “chamam”, não são “chamados”: um papel do *framework* é fornecer o fluxo de controle da aplicação. Assim, em tempo de execução, as instâncias das classes desenvolvidas esperam ser chamadas pelas instâncias das classes do *framework*.

Um *framework* se destina a gerar diferentes aplicações para um domínio. Precisa, portanto, conter uma descrição dos conceitos deste domínio. As classes abstratas de um *framework* são os repositórios dos conceitos gerais do domínio de aplicação. Apenas atributos a serem utilizados por todas as aplicações de um domínio são incluídos em classes abstratas, em um *framework*, um método de uma classe abstrata pode ser deixado propositalmente incompleto para que sua definição seja acabada na geração de uma aplicação. [SIL00].

Os *frameworks* são estruturas de classes inter-relacionadas, que permitem não apenas a utilização de classes, mas minimizam o esforço para o desenvolvimento de aplicações porque contem o protocolo de controle da aplicação (a definição da

arquitetura), liberando o desenvolvedor de *software* desta preocupação. Os *frameworks* invertem a ótica de reuso de classes, da abordagem *bottom-up2* para a abordagem *topdown*: o desenvolvimento inicia com o entendimento do sistema contido no projeto do *framework*, e segue no detalhamento das particularidades da aplicação específica, o que é definido pelo usuário do *framework*. Assim, a implementação de uma aplicação a partir do *framework* é feita pela adaptação de sua estrutura de classes, fazendo com que esta inclua as particularidades da aplicação [SIL00][TAL95].

Uma outra característica é o nível de granularidade de um *framework*. Um *frameworks* pode agrupar diferentes quantidades de classes, sendo assim, mais ou menos complexos. Além disso, podem conter o projeto genérico completo para um domínio de aplicação, ou construções de alto nível que solucionam situações comuns em projetos.

A utilização de *frameworks* vem crescendo. Alguns exemplos de *frameworks* podem ser citados, como OLE (*Object Linking and Embedding*), DSOM (*Distributed System Object Model*), Java AWT (*Abstract Window Toolkit*) que é parte de JFC (*Java Foundation Classes*), Java RMI (*Remote Method Invocation*), MFC (*Microsoft Foundation Classes*), Microsoft DCOM (*Distributed Common Object Model*) e CORBA (Visibroker, Orbix).

3.2 Como usar um framework

Diversas são as maneiras de utilizar *frameworks*. Algumas requerem mais conhecimento que outras. Mas todas diferem da maneira usual de desenvolver software utilizando tecnologias orientadas a objeto, já que todas elas forçam uma aplicação a se encaixar no *framework*. No entanto, o projeto da aplicação precisa começar com o projeto do framework[AKK02].

Aplicações desenvolvidas usando um *framework* tem três partes: o *framework*, a subclasse concreta da classe de *frameworks*, e todo o restante. “O restante” geralmente inclui rotinas que especificam quais classes concretas vão ser usadas e como elas vão ser interconectadas. Ela pode também incluir objetos que tenham nenhuma relação com o *framework*, ou que usam um ou mais objetos *framework*. Objetos que são chamados por

objetos *framework* terão que ser incluídos no modelo colaborativo do framework, e assim como as partes do *framework*.

A maneira mais fácil de usar um *framework* é conectando componentes existentes. Isto não muda o *framework*, nem cria quaisquer novas subclasses concretas. Isto reusa as interfaces do *framework* e regras de conectar componentes.

Nem todos os *frameworks* podem trabalhar assim. Às vezes cada novo uso de um framework exige novas subclasses do *framework*. Isto nos leva à próxima maneira mais fácil de usar *framework*, que é definir novas subclasses concretas e usá-las para implementar uma aplicação. Se programadores podem usar um *framework*, conectando componentes sem ter que olhar suas implementações então o *framework* é um *framework* "caixa-preta". Frameworks que vêm de heranças geralmente exigem mais conhecimento por parte do programador, e são chamados *frameworks* "caixa-branca". Caixas-pretas são mais fácil de aprender a usar, mas caixas-brancas são mais poderosos em mãos capacitadas. Mas caixas-pretas e brancas não são totalmente opostos.

Todas estas maneiras de usar um *framework* exigem mapeamento da estrutura do problema a ser resolvido na estrutura do *framework*. Um *framework* força a aplicação a reusar seu próprio projeto.

3.3 Como aprender um framework

Aprender um *framework* é mais difícil do que aprender uma biblioteca de classes normais, porque você não pode aprender uma classe por vez. As classes do *framework* são desenhadas para funcionar juntas, então você deve aprendê-las todas de uma vez só. As classes mais importantes são abstratas, o que torna elas ainda mais difícil de aprender. Essas classes não implementam todo o comportamento dos componentes do *framework*, mas deixam algo para as subclasses concretas, então você aprender o que nunca muda no *framework* e o que é deixado para os componentes [AKK02].

Frameworks só são fáceis de aprender se eles têm boa documentação. Mesmo os *frameworks* mais simples são mais fáceis de aprender se há um bom treinamento, e *frameworks* complexos exigem treinamento.

O melhor jeito de aprender sobre um *framework* é trabalhando com ele. A melhor maneira de aprender *framework* é por exemplos. A maioria dos frameworks vem com alguns exemplos, alguns mais fáceis, outros mais difíceis. Cada um resolve um determinado problema e pode ser estudado todo os seu comportamento dentro do *framework*. Idealmente, um framework deveria vir com exemplos que cobrissem do trivial ao avançado, e esses exemplos iriam demonstrar todo o alcance das funções do *framework*.

Embora alguns frameworks têm pouca documentação, além do código fonte e alguns exemplos, idealmente um framework vai ter uma completa documentação. Esta documentação deve explicar:

- Propósito do framework
- Como usar o framework
- Como o framework funciona

Parece difícil explicar o propósito do *framework*. Geralmente a maneira mais fácil de aprender a capacidade de um *framework* é através de exemplos, por esta razão é que ajuda na sua compreensão se um framework vem com uma rica série de exemplos. Também é difícil explica como usar o *framework*. Entender o funcionamento do framework não diz ao programador qual subclasse fazer. A melhor documentação parecer ser aquela no estilo livro de receitas. Programadores iniciantes podem usar um livro de receita para fazer seus primeiros aplicativos, e programadores mais avançados podem usá-lo para olhar solução para problemas particulares. Muita documentação de *frameworks* apenas lista as classes do *framework* e os métodos de cada classe. Isto não é útil para os novatos. Programadores precisam entender o framework por inteiro. Documentação que descreve o funcionamento do *framework* deve focar na interação dos objetos e como as responsabilidades são divididas entre eles.

Frameworks são complexos e um dos grandes problemas com a maioria dos *frameworks* é apenas aprender a usá-los. Desenvolvedores de frameworks precisam ter certeza que eles documentam seus *frameworks* bem e desenvolver bons materiais de treinamento para isso. Usuários de *framework* devem planejar o gasto de tempo e dinheiro para aprender o framework.

3.4 Como avaliar um framework

Às vezes é fácil escolher um *framework*. A companhia que quiser desenvolver aplicativos distribuídos pela internet que executam em navegadores vai querer usar Java, e vai preferir frameworks padrões. A empresa que é "Padrão Microsoft" vai preferir MFC a OWL ou zApp. A maioria dos campos de aplicação não tem concorrente, sendo que não é necessário a escolha entre um e outro, apenas escolher se irá utilizá-lo ou não. Mas, muitas vezes existem *framework* concorrentes, e eles precisam ser avaliados[AKK02].

Muitos dos aspectos de um framework são fáceis de avaliar. Um framework deve executar nas plataformas certas, usar a linguagem de programação correta e ser compatível com os padrões certos. Claro que cada organização tem sua própria definição de "certo", mas essas questões são fáceis de responder. Se elas não estão na documentação, o vendedor pode respondê-las. As questões mais difíceis de serem respondidas nesta avaliação são: se o framework é adequado para o seu problema; se ele tem a melhor relação entre poder e simplicidade. *Frameworks* que resolvem problemas técnicos como distribuição e design de interface são relativamente fáceis de avaliar. Nenhum framework é bom para tudo, e pode ser difícil dizer se um *framework* em particular encaixa-se para um determinado problema.

A abordagem padrão para avaliar um programa é fazer uma lista de checagem das ferramentas que o programa deve ter. Já que frameworks são extensíveis, e já que eles provavelmente são supostamente apenas para cuidar de parte da aplicação, é mais importante que um framework seja de fácil extensão do que tenha todas as ferramentas. Porém é mais fácil dizer se o *framework* tem todas as ferramentas do que se ele é extensível. O principal valor de um *framework* é se ele melhora a maneira que você desenvolve programas e o programa que você desenvolve. Deve-se analisar as vantagens da utilização do *framework*, como a manutenção. O framework deve encaixar-se na cultura da empresa. Se a empresa tem alto giro de pessoas e pouco capital para treinamento, então o framework precisa ser simples e fácil de utilizar. O valor do *framework* depende mais da situação em que ele vai ser utilizado do que nele próprio.

3.5 Classificação de Frameworks

Os *frameworks* podem ser classificados de maneira geral como dirigido à arquitetura ou dirigido a dados. No primeiro caso a aplicação deve ser gerada a partir da criação de subclasses das classes do *framework*. No segundo caso, diferentes aplicações são produzidas a partir de diferentes combinações de objetos, instâncias das classes presentes no *framework*. Os *frameworks* dirigidos à arquitetura são mais difíceis de usar, pois a geração de subclasses exige um profundo conhecimento do projeto de um *framework*, bem como em certo esforço no desenvolvimento de código. Os *frameworks* dirigidos a dados são mais fáceis de usar, porém são menos flexíveis [SIL00]. Uma outra abordagem possível seria uma combinação dos dois casos, onde o *framework* apresentaria uma base dirigida à arquitetura e uma camada dirigida a dados. Com isto, possibilita a geração de aplicações a partir da combinação de objetos, mas permite a geração de subclasses [TAL94] . Alguns autores definem os *frameworks dirigidos* à arquitetura como caixa-branca (*white-box frameworks*) e os *frameworks* dirigidos a dados como "caixa-preta" (*black-box frameworks*). Alguns autores denominam de "caixa-cinza" (*gray-box*) a combinação dos dois casos. De acordo com [FAY99] os *frameworks* caixa-cinza são projetados para evitar as desvantagens apresentadas pelos *frameworks* caixa-branca e os caixa-preta. Em outras palavras, um *framework* caixa-cinza bem projetado possui bastante flexibilidade, estensibilidade e a capacidade de esconder informação desnecessária dos desenvolvedores de aplicação.

Os *frameworks* também podem ser classificados, de acordo com o seu escopo, em três tipos [FAY99]:

- ***Frameworks* de Infra-Estrutura de Sistema (*System Infrastructure Frameworks*):** São *frameworks* que simplificam o desenvolvimento de sistemas de infra-estrutura eficientes e portáveis, como por exemplo, sistemas operacionais. Outros exemplos são os *frameworks* de comunicação, *frameworks* para interface com o usuário e ferramentas de processamento de linguagem. Os *frameworks* de infra-estrutura de sistema

são principalmente utilizados internamente dentro da organização do software e não são vendidos a clientes diretamente.

- ***Frameworks* de Integração *Middleware* (*Middleware Integration Frameworks*):** Estes *frameworks* são geralmente utilizados para integrar aplicações e componentes distribuídos. Os *frameworks* de integração *middleware* são projetados para melhorar a capacidade dos desenvolvedores de *software* de modularizar, reutilizar e estender sua infra-estrutura de *software* para trabalhar em um ambiente distribuído sem muitos problemas.

  Alguns exemplos destes tipos de *frameworks* incluem os 35 *frameworks* de ORBs, de *middlewares* orientados a mensagem e as bases de dados transacionais.

- ***Frameworks* de Aplicação de Empresa (*Enterprise Application Frameworks*):** Estes *frameworks* se tratam de amplos domínios de aplicação, como por exemplo, telecomunicações, indústria, engenharia financeira e são base para as atividades de negócio de uma empresa.

  Em comparação com os *frameworks* de infra-estrutura de sistema e com os *frameworks* de integração *middleware*, os *frameworks* de aplicação de empresa possuem um custo de desenvolvimento mais alto. Porém estes últimos podem fornecer um significativo retorno de investimento desde que suportem o desenvolvimento de aplicações *end-user* e de produtos de forma direta. Em contrapartida, os *frameworks* de infra-estrutura de sistemas e de integração *middleware* focalizam em grande parte nos interesses internos ao desenvolvimento de *software*. Embora esses *frameworks* sejam essenciais para a criação de *softwares* de alta qualidade rapidamente, eles geralmente não geram uma renda significativa para grandes empresas [FAY99] .

3.6 Ciclo de Vida de Frameworks

O ciclo de vida de um *framework* difere do ciclo de vida de uma aplicação convencional porque um *framework* nunca é um artefato de *software* isolado, mas sua existência está sempre relacionada à existência de outros artefatos, originadores do *framework*, originados a partir dele ou que exercem alguma influência na definição de estrutura de classes do *framework*. Várias fontes de informação influem na definição da estrutura de um *framework*: artefatos de *softwares* existentes, os quais são produzidos a partir de um *framework* e o conhecimento do desenvolvedor do *framework* (ou da equipe de desenvolvimento) [AKK02].

Abaixo são descritas as principais fontes de informação que influenciam na definição da estrutura de um *framework*:

A atuação do desenvolvedor: Nenhuma abordagem de desenvolvimento de *frameworks* existente dispensa a figura do desenvolvedor. Ele é quem decide que classes comporão a estrutura do *framework*, suas responsabilidades e a flexibilidade provida aos usuários do *framework*. O desenvolvedor não também na sua manutenção do framework.

Aplicações do domínio tratado: Um *framework* constitui um modelo de um domínio de aplicações. Assim, pode ser desenvolvido a partir de um conjunto de aplicações do domínio, que atuam como fontes de informação deste domínio. Esta influência de aplicações do domínio pode ocorrer no processo do desenvolvimento do *framework*, em que um *framework* é constituído como uma generalização de diferentes estruturas, ou na fase de manutenção. No caso da fase de manutenção a alteração seria motivada pela obtenção de conhecimento do domínio tratado, não considerado ou indisponível durante o desenvolvimento do *framework*.

Aplicações geradas sob o *framework*: A finalidade básica de um *framework* é ser reutilizado na produção de diferentes aplicações, minimizando o tempo e esforço requeridos para isto. A construção de um *framework* é sempre precedida por um procedimento de análise de domínio em que são observadas informações do domínio tratado. É compreensível o fato de o *framework* ser incapaz de conter todas as

informações do domínio, pois se trata de uma abstração de uma realidade, é inevitável que um *framework* seja uma descrição aproximada do domínio, construída a partir das informações até então disponíveis.

Idealmente a construção de aplicações sob *frameworks* consiste em completar ou alterar procedimentos e estruturas de dados presentes no *framework*. Sob esta ótica, uma aplicação gerada sob um *framework* não deveria incluir classes que não fossem subclasses das classes do *framework*. Existem casos em que devido à obtenção de novos conhecimentos não tratados ou indisponíveis na fase de construção do *framework* seja necessária a implementação de classes que não sejam subclasse, assim levando a necessidade de se alterar o *framework*.

3.7 Desenvolvimento de *Frameworks*

O desenvolvimento de um *framework* é um pouco diferente do desenvolvimento de uma aplicação padrão. A grande diferença está no fato de que o *framework* tem que cobrir todos os conceitos relevantes no domínio e a aplicação só abrange aqueles conceitos mencionados em seus requisitos [BOS97] Devido à necessidade de considerar os requisitos de um conjunto significativo de aplicações, de modo a dotar a estrutura de classes do *framework* de generalidade, em relação ao domínio tratado; à necessidade de ciclos de evolução para prover a estrutura de classes do *framework* de alterabilidade e estensibilidade o desenvolvimento de um *framework* é mais complexo que o desenvolvimento de aplicações específicas do mesmo domínio[AKK02].

Para estabelecer o contexto de problemas experimentados e identificados no desenvolvimento de *frameworks* [BOS97] define-se as seguintes atividades como parte de um simples modelo de desenvolvimento de *frameworks*:

- **Análise de Domínio (*Domain Analysis*):** Esta etapa visa descrever o domínio que será abrangido pelo *framework*. Uma forma de capturar os requisitos e identificar conceitos relacionados com o domínio é procurar aplicações desenvolvidas anteriormente, obter informações com especialistas no domínio e buscar os padrões existentes. O resultado

desta atividade é um *modelo de análise de domínio*, contendo os requisitos e os conceitos do domínio e suas relações;

- **Projeto da Arquitetura (*Architectural Design*):** Esta atividade utiliza o modelo de análise de domínio como ponto de partida. O analista tem que escolher um *estilo de arquitetura3* satisfatório ao desenvolvimento do *framework.* A partir deste ponto é que o projeto mais alto nível do *framework* é elaborado;
- **Projeto do *Framework* (*Framework Design)*:** Durante esta fase o projeto de nível mais alto do *framework* é refinado e classes adicionais são projetadas. Os resultados desta atividade são a definição do escopo de funcionalidades dada pelo projeto do *framework*, a *interface* de reutilização do *framework*, as regras de projeto que devem ser obedecidas e que são baseadas nas decisões de arquitetura e um documento com o histórico do projeto, descrevendo os problemas encontrados e as soluções escolhidas juntamente com as argumentação de suas escolhas;
- **Implementação do *Framework (Framework Implementation)*:** É a etapa de codificação das classes abstratas e concretas que foram definidas para o *framework*;
- **Testes do *Framework (Framework Testing*):** É a fase que se determina se o *framework* fornece realmente as funcionalidades que foram planejadas, e também avaliar a usabilidade deste. Porém o único modo de descobrir se algo é reutilizável ou não é realmente reutilizando. No caso de *frameworks*, isto se resume a desenvolver aplicações que utilizem o *framework*;

- **Geração de Aplicação de Teste (*Test Application Generation*):** Para avaliar a usabilidade de um *framework*, verificar se o *framework* necessita ser reprojetado ou está suficientemente bom para ser utilizado.Esta atividade de geração de uma aplicação de teste é interessante com o desenvolvimento de aplicações de teste baseadas no *framework*.

Dependendo do tipo de aplicação, cada aplicação pode testar diferentes aspectos do *framework*.

A documentação é uma das mais importantes atividades no desenvolvimento de *frameworks*, apesar de que sua importância nem sempre ser reconhecida. Ela é fundamental, visto que, sem uma documentação clara, completa e correta que descreve como utilizar o *framework*, um manual do usuário, e um documento de projeto descrevendo como o *framework* funciona, será quase que impossível à utilização do *framework* por pessoas que não se envolveram no projeto deste.

3.8 Metodologias de Desenvolvimento de *Frameworks*

Em [SIL00] são descritas três propostas de desenvolvimento de *frameworks*: Projeto Dirigido por Exemplo (*Example-Driven Design*), Projeto Dirigido por *Hot Spot* (*Hot Spot Driven Design*) e a metodologia de projeto da empresa Taligent ([TAL95]). Estas metodologias se caracterizam por estabelecer o processo de desenvolvimento de *frameworks* em linhas gerais, sem se ater à definição de técnicas de modelagem ou detalhar o processo. No Projeto Dirigido por Exemplo é estabelecido que o desenvolvimento de um *framework* para um domínio de aplicação é decorrente de um processo de aprendizado a respeito deste domínio, que se processa concretamente a partir do desenvolvimento de aplicações ou do estudo de aplicações desenvolvidas. Como as pessoas pensam de forma concreta e não abstrata, a abstração do domínio, que é o próprio *framework*, é obtida através da generalização de casos concretos, as aplicações [SIL00].

Uma aplicação orientada a objetos é completamente definida. Um *framework*, ao contrário, possui partes propositalmente indefinidas, o que lhe dá a capacidade de ser flexível e se moldar a diferentes aplicações. Os *hot spots* são as partes do *framework* mantidas flexíveis. A essência da metodologia de Projeto Dirigido por *Hot-Spot* é identificar os *hot spots* na estrutura de classes de um domínio, e então o *framework* é construído [SIL00].

De acordo com [SIL00] a metodologia proposta pela empresa Taligent (empresa já extinta) difere das anteriores pelo conjunto de princípios que norteia o desenvolvimento de *frameworks*. Primeiramente, a visão de desenvolver um *framework* que englobe as características e as necessidades de um domínio é substituída pela visão de produzir um conjunto de *frameworks* estruturalmente menores e mais simples, que usados em conjunto, originam as aplicações. A justificativa para isto é que "pequenos *frameworks* são mais flexíveis e podem ser reutilizados mais freqüentemente". Assim, a ênfase passa a ser o desenvolvimento de *frameworks* pequenos e direcionados a aspectos específicos do domínio.

3.9 Conclusões do Capítulo

Este capítulo apresentou o conceito de *framework*, suas características principais, os tipos de classificação e o ciclo de vida. O capítulo ainda revela a importância da reusabilidade de *software* e como esta pode ser obtida com a utilização de *frameworks,* além de apresentar alguns aspectos do estudo, compreensão e utilização de um *framework*. Após foram apresentadas as principais atividades que devem ser realizadas no desenvolvimento de *frameworks*, bem como as metodologias de desenvolvimento existentes[AKK02].

Capítulo 4

4.1 Um *Framework* RBAC para Aplicações *Web*

Dentro de uma empresa ou corporação existem várias pessoas com responsabilidades ou papéis diferentes, que por sua vez possuem privilégios diferentes, como já foi citado anteriormente. É neste contexto que o trabalho foi organizado, ou seja, em auxiliar no controle de privilégios dos usuários no desenvolvimento de aplicações Web. O trabalho se concentrou na ampliação de um *framework*, cuja idéia permite a reusabilidade não somente de código, mas também das etapas de análise e projeto que fazem parte da específicação de um sistema. O *framework* foi utilizada e expandido para a utilização e autenticação por chaves públicas. Como o princípio da reusabilidade foi utilizado em sua construção é possível afirmar que a específicação deste *framework* poderá ser utilizada em implementações em outras linguagens de programação e que seja utilizada por qualquer tipo de aplicação, não somente em aplicações com *interface* Web. A escolha do modelo de segurança RBAC se deveu ao fato de este ser o que adequado ao modelo de funcionamento das empresas, pois a maioria destas baseiam-se em suas decisões de controle de acesso "nos papéis que os usuários individuais desempenham na organização". Este é o principal motivo para a cada vez maior utilização em aplicações comerciais e de banco de dados[AKK02].

4.2 Trabalhos Relacionados

Apesar de existirem vários trabalhos de implementação do modelo RBAC, tanto comerciais quanto acadêmicos, nenhuma das implementações comerciais segue à risca um conjunto apropriado de características do modelo RBAC [FER99]. As principais implementações comerciais são os SGBDs, ou sistemas gerenciadores de bancos de dados. Também existem implementações do modelo RBAC nas áreas de gerenciamento de sistemas e de sistemas operacionais. No artigo de [KEL00] pode ser vista uma breve introdução de como o modelo RBAC é implementado no sistema operacional Solaris 8 da Sun Microsystems, bem como uma implementação de RBAC utilizando o protocolo LDAP (*Lightweight Directory Access Protocol*). Os trabalhos de âmbito acadêmico seguem com mais fidelidade as características do modelo RBAC, nas implementações

comerciais os princípios não são seguidos à risca. Alguns destes trabalhos serão apresentados nas seções a seguir.

4.2.1 Diferenças entre SeguraWeb e RBAC/Web do NIST

Existem que possuem a mesma idéia de implementação de RBAC para aplicações Web, como a do *framework* aqui utilizado para a expansão, o SeguraWeb. Um exemplo de trabalho acadêmico é a proposta RBAC/Web, do NIST, que é uma implementação de RBAC para ser utilizadas pelos servidores Web [FER99]. O *framework* Seguraweb não trabalha em nível de operações HTTP, e não foi proposto para ser utilizado pelos servidores Web. A vantagem do *framework* Seguraweb está no controle de operações que podem ser realizadas através de uma página. Como os componentes deste *framework* são implementados em Java, eles podem ser tanto utilizados em *applets* como em páginas JSP dinâmicas. Desta maneira, as chamadas ao componente de autorização do *framework* podem ser incluídas dentro do código JSP ou do código do *applet*. A chamada ao componente de autorização irá obter o papel do usuário, e deste modo pode-se montar a página dinamicamente ou apresentar no *applet* apenas as operações que são permitidas para o papel deste usuário. O *framework* Seguraweb também pode permitir controle de acesso a páginas dentro de uma aplicação Web implementada em Java. A desvantagem do *framework* Seguraweb, é que ele não pode ser utilizado por quaisquer aplicações de interface Web, mas apenas aquelas implementadas em Java; a não ser que este seja implementado em outra linguagem, como por exemplo, Visual Basic, para ser utilizado em páginas dinâmicas feitas em ASP (*Active Server Pages)*. Mas mesmo assim, o *framework* não pode ser utilizado em aplicações Web simples, baseadas apenas em páginas estáticas e JavaScript [BEZ99].

A diferença do SeguraWeb em relação ao RBAC/Web, bem como da maioria das propostas para implementação de RBAC para Web, é que o RBAC/Web utiliza o RBAC como um esquema de autorização para controlar o acesso a páginas de um servidor Web. Ou seja, as operações permitidas implementadas pelo RBAC/Web são os “métodos” definidos no protocolo HTTP (*HyperText Transfer Protocol*). Estes métodos são GET,

HEAD, PUT, POST, etc. O RBAC/Web controla a habilidade de um usuário ativo em um papel de executar um método HTTP em um URL (*Uniform Resource Locator*).

4.2.2 Diferenças entre SeguraWeb e propostas como RBAC-JACOWEB e Beznosov/Deng

Existem muitos trabalhos que implementam RBAC utilizando o Serviço de Segurança CORBA, como por exemplo, a proposta de Beznosov e Deng [BEZ99] apresenta como os modelos de controle de acesso baseados em papéis podem ser implementados em sistemas distribuídos orientados a objeto que seguem os padrões OMG/CORBA, bem como descreve o que é necessário para o Serviço de Segurança CORBA para dar suporte aos modelos RBAC0 a RBAC3. A proposta de RBAC-JACOWEB [OBE01] [OBE02] introduz o conceito de ativação automática de papéis pelo subsistema de segurança, que permite que o esquema de autorização seja utilizado por aplicações previamente desenvolvidas e que não possuem nenhuma ciência da segurança do sistema. Este trabalho atua no nível de *middleware*, permitindo sua utilização com qualquer aplicação baseada em CORBA (*Common Object Request Broker Architecture*). A primeira diferença entre o *framework* Seguraweb e estes trabalhos é que o primeiro não é baseado no Serviço de Segurança CORBA. A segunda diferença é que o Seguraweb é um *framework* e as propostas citadas não são. A princípio, o objetivo do *framework* Seguraweb não é utilizar CORBA, o enfoque está no desenvolvimento de um *framework*, que possui interfaces, classes abstratas e concretas que auxiliem no desenvolvimento dos sistemas de segurança das aplicações com interface Web.

4.2.3 A Proposta ORBAC

Na proposta ORBAC é apresentado um modelo RBAC orientado a objetos, com o objetivo de simplificar a administração das políticas de segurança. Este também propõe uma arquitetura de gerência de segurança descentralizada, baseada em controle de acesso de múltiplos domínios e mostra como o modelo ORBAC funciona nesta arquitetura [ZHA01].

O modelo ORBAC é uma alternativa de modelagem do modelo RBAC utilizando o paradigma de orientação a objeto. O modelo ORBAC possui algumas semelhanças de modelagem com o *framework* Seguraweb, principalmente no que diz respeito aos elementos do modelo RBAC (usuário, papel e permissão). Este modelo também implementa as regras de separação estática e dinâmica de tarefas, bem como hierarquia de papéis.

A proposta do *framework* Seguraweb apresenta preocupações em modelar outros requisitos que são necessários na implementação de aplicações que necessitem de segurança, como por exemplo, autenticação, auditoria e persistência dos dados. O *framework* SeguraWeb apresenta uma alternativa semelhante ao modelo ORBAC de como implementar o modelo RBAC utilizando orientação a objetos.

4.2.4 Modelo RBAC Baseado em UML

UML (*Unified Modeling Language*) é uma linguagem padrão utilizada pela comunidade de desenvolvedores de aplicações [SHI00]. Esta proposta apresenta uma representação genérica do modelo RBAC utilizando UML. Este trabalho é útil aos desenvolvedores de aplicações no momento da análise de aplicações de segurança, pois apresenta as visões estática, funcional e dinâmica do modelo RBAC, e foram utilizadas na definição da parte de controle de acesso do *framework* Seguraweb, principalmente à parte de casos de uso (visão funcional) e dos diagramas de colaboração (visão dinâmica).

Um outro trabalho que utiliza UML é apresentado em [AHN01], que utiliza uma linguagem declarativa, a OCL (*Object Constraints Language*), que faz parte da UML, para expressar formalmente as restrições existentes no modelo RBAC.

4.2.5 Controle de Acesso para Servidores Web de um Mesmo Domínio baseado no modelo RBAC comparado com o SeguraWeb

Este trabalho apresenta um método para fazer controle de acesso de servidores Web distribuídos para controlar a visualização de documentos dependendo do papel do usuário utilizando as informações de um servidor RBAC. Esta proposta utiliza um

mecanismo de *cookies* na memória quando o usuário acessa os servidores *Web* em um mesmo domínio. Deste modo, o usuário pode acessar transparentemente todos os recursos em múltiplos servidores *Web*, sem ter que realizar uma nova autenticação em cada servidor *Web*, dando a impressão que o usuário está acessando apenas um servidor *Web* [SHI01].

Este trabalho se aproxima mais da proposta do *framework* Seguraweb do que a proposta RBAC/Web apresentada anteriormente, pois este também controla a apresentação de informações no momento da criação das páginas dinâmicas. Mas a proposta Seguraweb ainda possui a vantagem da característica de reutilização dos *frameworks*, sendo que a implementação deste trabalho pode ser reutilizada tanto na criação de aplicações *Web* ou em aplicações Java locais. Já a proposta de [SHI01] apresenta uma aplicação de demonstração utilizando PHP versão 4 e o banco de dados MySQL. Uma característica interessante deste trabalho é a definição de *cookies* em memória, que podem ser passados para múltiplos servidores *Web* de forma transparente evitando que o usuário tenha que realizar várias sessões de autenticação. Esta característica pode ser incluída no *framework* Seguraweb em futuras expansões.

4.2.6 Um *Framework* Baseado na Descrição de Papéis

Este trabalho apresenta uma proposta de *framework* baseado na descrição de papéis e que pode ser utilizado na modelagem de aplicações baseadas em RBAC e de Gerência de *Workflows* [GUS96]. A diferença deste trabalho é que a definição do *framework* é baseada em papéis, ou seja, o elemento principal do *framework* é o elemento papel, enquanto que no *framework* Seguraweb os elementos principais são usuário, papel e permissão. Deste modo, a definição das classes dos elementos do *framework* de [GUS96] foi baseada em um ponto de vista diferente do *framework* Seguraweb, uma abstração diferente foi utilizada. Um exemplo que ilustra bem esta diferença é a modelagem do relacionamento entre usuários e papéis. No *framework* de [GUS96] apenas o elemento RDO (*Role Descriptor Object*) contém a lista de usuários que podem assumir o papel, enquanto no *framework* Seguraweb, isso também é realizado na classe, mas a classe *User* também contém a lista de papéis que o usuário pode assumir. O

framework Seguraweb possui esta característica para dar suporte à revisão usuário-papel do RBAC Básico, que possibilita determinar quais os usuários estão associados a um papel e também determinar quais papéis estão associados a um usuário.

O artigo [GUS96] também apresenta como podem ser implementadas as regras do modelo RBAC (principalmente no que diz respeito à hierarquia e restrições) utilizando o *framework* proposto por eles. A vantagem do *framework* Seguraweb está no fato de que este está praticamente pronto para o uso, pois além de ter implementado a parte de controle de acesso, também dá suporte à parte de autenticação, administração das informações de segurança, auditoria e persistência. Além disso, a implementação do *framework* Seguraweb utilizou técnicas de padrões de projeto que permitem maior extensibilidade.

4.3 Etapas de Análise e Projeto do *Framework* Seguraweb

A seguir serão descritas cada uma destas etapas no que se refere ao desenvolvimento deste trabalho.

4.3.1 Análise de Domínio

A análise de domínio visa descrever o domínio que será abrangido pelo *framework*. É necessário obter os requisitos e identificar conceitos relacionados com o domínio e isto pode ser feito buscando aplicações desenvolvidas anteriormente, além da busca de padrões existentes. Foram utilizadas as aplicações de segurança desenvolvidas na empresa Thermus Com. Serv. Repr. Ltda (especializada no desenvolvimento de softwares para a área de telecomunicações), bem como o estudo do modelo de segurança RBAC.

Na construção do framework foram primeiramente analisadas as aplicações e foram definidos os seguintes requisitos para a implementação do *framework* e assim ele foi implementado.

- Autenticação: para garantir que um usuário é quem ele diz ser;
- Confidencialidade: garantindo proteção de informações em

trânsito;

- Controle de Acesso: para controlar o que um usuário pode acessar em um sistema;
- Auditoria: para registrar e analisar o que o usuário faz no sistema;
- Administração das informações de segurança: para gerenciar informações de segurança incluindo as políticas de segurança (gerência de perfis de usuários, etc);
- Persistência dos dados (em arquivos ou base de dados): para armazenamento dos dados de autenticação (senhas, etc), de controle de acesso, de auditoria e de administração.

Na construção do framework o modelo de segurança RBAC foi utilizado principalmente na implementação dos requisitos de controle de acesso e administração das informações de segurança.

4.3.2 Projeto de Arquitetura

O SeguraWeb foi desenvolvido para ser *framework* para aplicações com *interface* Web. A partir deste objetivo procurou-se obter informações sobre as tecnologias existentes atualmente para desenvolvimento de aplicações Web. As mais utilizadas atualmente são ASP (*Active Server Pages*), PHP (*Personal Home Page* ou um acrônimo recursivo para "PHP: *Hypertext Preprocessor*"), *Servlets*, JSP (*Java Server Pages*) e *Applets*, sendo que estas três últimas são baseadas na utilização da linguagem Java.

A tecnologia ASP possui um ambiente para programação por *scripts* baseados em VisualBasic (VBScript) que funcionam como uma extensão da linguagem HTML. Os *scripts* que são executados no servidor e permitem a criação de páginas dinâmicas. O servidor ASP é quem transforma os *scripts* em páginas HTML padrão, que podem ser acessados de qualquer *browser*. O ASP surgiu juntamente com o lançamento do servidor HTTP *Internet Information Server* 3.0. Por ser uma solução Microsoft, o servidor só pode ser executado nos sistemas operacionais Windows (95, 98, 2000 ou NT). Existem tentativas de implementar o ASP rodando no servidor Apache, existe um projeto chamado Apache-ASP que disponibiliza algumas bibliotecas para se colocar o servidor

Apache com suporte a páginas ASP, porém o recomenda-se utilizar ASP com os servidores da Microsoft, devido à falta de suporte a esta solução.

A tecnologia PHP tem a mesma ideologia do ASP, ou seja, também é um ambiente para programação por *scripts* que são executados no servidor para a geração de páginas dinâmicas. A sintaxe do PHP tem um formato semelhante às linguagens C, Java e Perl, sendo que possui algumas características específicas adicionadas.

As outras tecnologias (*Servlets*, JSP e *Applets*), por serem baseadas na linguagem Java possuem a vantagem de serem independente de plataforma, ou seja, podem ser executadas tanto em plataforma Unix, Linux ou Windows. Este é o principal motivo pela escolha de Java como a linguagem de implementação do *framework*, bem como também por Java ser uma linguagem com suporte a orientação a objetos.

Deste modo o *framework* se torna mais abrangente, pois pode ser utilizado por aplicações baseadas em *Servlets*, JSP ou em *Applets*. Estas características fazem com que Java seja a linguagem mais escolhida para implementação de aplicações na Internet [PAP00]. Além disso, a linguagem Java possui um modelo de segurança, dispondo de algumas APIs que implementam funções de segurança, como por exemplo, JCE (*Java Cryptography Extension*), JSSE (*Java Secure Socket Extension*) e JAAS (*Java Authentication and Authorization Service*). Outra vantagem da implementação em Java é a facilidade de portar a aplicação para utilização de CORBA, já que existem vários ORBs implementados em Java, bem como já existe na versão do JDK 1.4 uma API com suporte à CORBA.

Por ser um *framework* destinado a aplicações Web, o elemento de segurança de confidencialidade não será levado em conta, já que este elemento é garantido pelo protocolo SSL (*Secure Socket Layer*), já implementado na maioria dos servidores Web utilizados atualmente.

O banco de dados escolhido para a persistência das informações de segurança foi o Oracle 8i. Esta ferramenta foi escolhida por ser uma das mais utilizadas no desenvolvimento de aplicações de grande porte, como as que são implementadas na empresa Thermus, e por ter versões para várias plataformas, principalmente Windows, Unix e Linux[AKK02].

4.3.3 Projeto e Implementação do *Framework*

Os serviços de segurança dentro do contexto deste trabalho são autenticação, controle de acesso, auditoria, administração das informações de segurança e garantia de comunicação segura.

O *framework* está dividido em três camadas, como pode ser visto na Figura 5: básica, de persistência e de aplicação. A primeira camada possui classes básicas, que são utilizadas pelas outras camadas e podem ser utilizadas pelo usuário do *framework*. A camada de persistência possui classes que tem o objetivo administrar a materialização e desmaterialização dos objetos que necessitam de persistência em mecanismos de armazenamento. A camada de aplicação contém as classes que deverão ser utilizadas pelo usuário do *framework* na construção de aplicações Web. Esta camada utiliza as classes básicas e de persistência para implementar as características e regras do modelo RBAC.

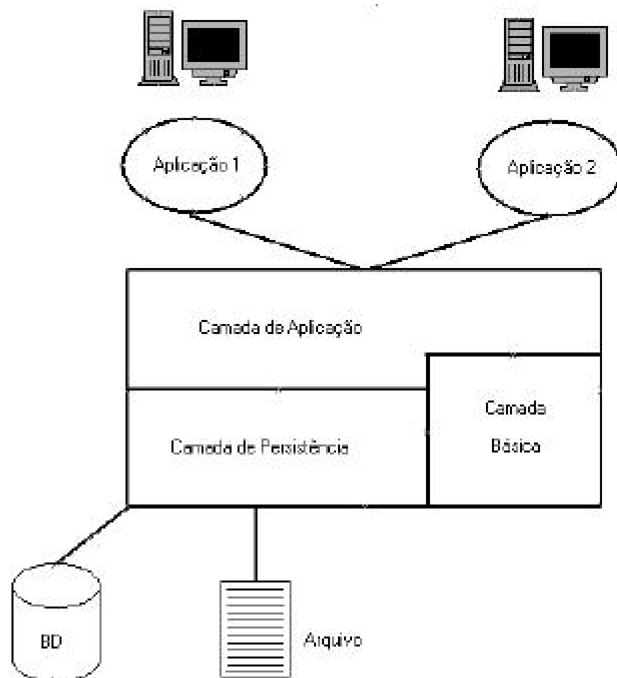

Figura 5: Esquema Geral do Framework Seguraweb.

Como a proposta deste trabalho é desenvolver um *framework* baseado no modelo RBAC de segurança, existem algumas classes que espelham as entidades principais deste modelo e que são utilizadas em vários pontos do *framework*. Essas classes são:

User, *Role* e *Permission*.

A classe *User*, que representa o elemento usuário do modelo RBAC, possui quatro atributos, como pode ser visto na Figura 6:

- id: identificador do usuário, que é uma *string* única por usuário;
- authenticationMethod: é o método de autenticação que será

utilizado para validar o usuário;

- fullName: nome completo do usuário;
- roles: lista de papéis que estão relacionados com o usuário, ou seja, a lista de papéis ao qual o usuário pertence.

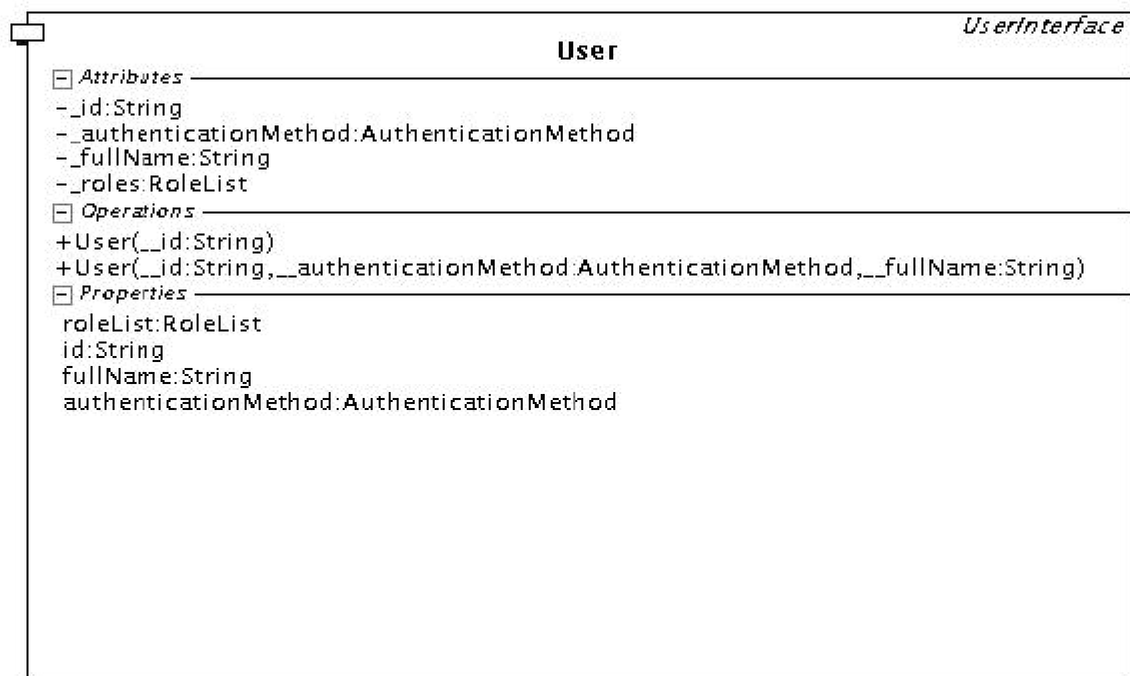

Figura 6: Diagrama da Classe User

A classe *Role*, que representa o elemento papel do modelo RBAC, possui cinco atributos, como pode ser visto na Figura 7:

- OID: identificador do papel, que deve ser único por papel;
- descr: descrição do papel;
- hRoleList: lista de papéis do qual esse papel herda as permissões. Isto significa que o *framework* suporta hierarquia de papéis;

- permissions: lista de permissões que estão relacionadas com o

papel;

- cardinality: cardinalidade do papel, ou seja, indica o número de usuários que podem estar relacionados com este papel ao mesmo tempo.

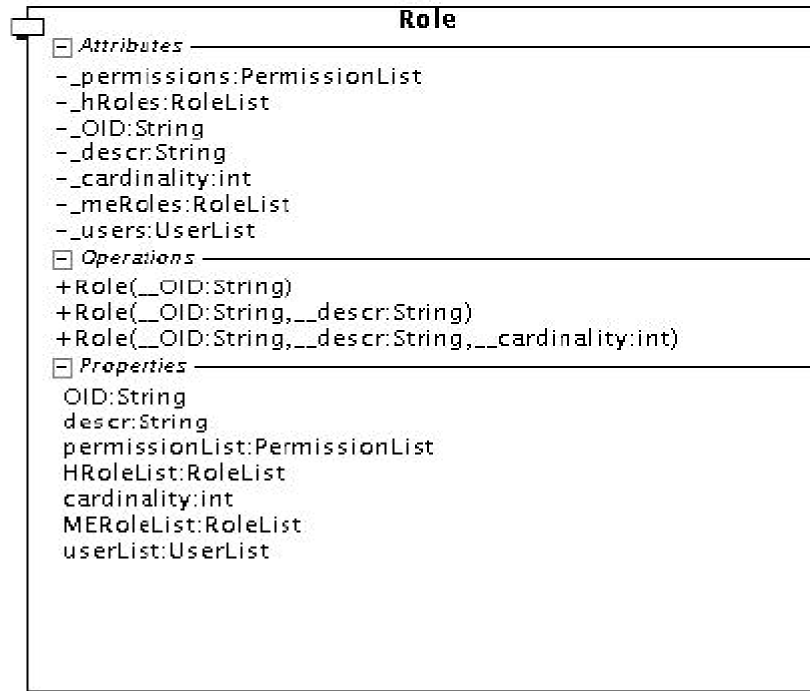

Figura 7: Diagrama da Classe Role

A classe *Permission*, que representa o elemento operação do modelo RBAC, possui apenas dois atributos, como pode ser visto na Figura 8:

- OID: identificador da operação, que deve ser único por operação;
- descr: descrição da operação.

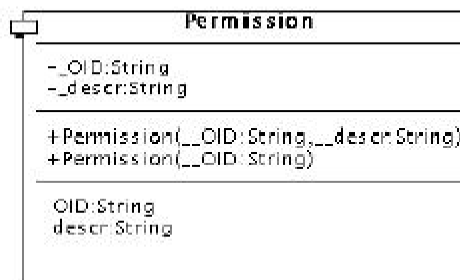

Figura 8: Diagrama da classe Permission

Outro componente da camada básica é a interface *AuthenticationMethod*, que funciona como um tipo abstrato, cujas classes concretas que implementam esta interface correspondem a métodos de autenticação, como *Password* (senha), *PrivateKey* (chave privada), PBE (*Password-Based Encryption*4) e PPK (*Password and Private Key*), que utiliza chave privada e senha conjuntamente, sendo um exemplo de criação de um novo método de autenticação utilizando outros métodos já existentes.

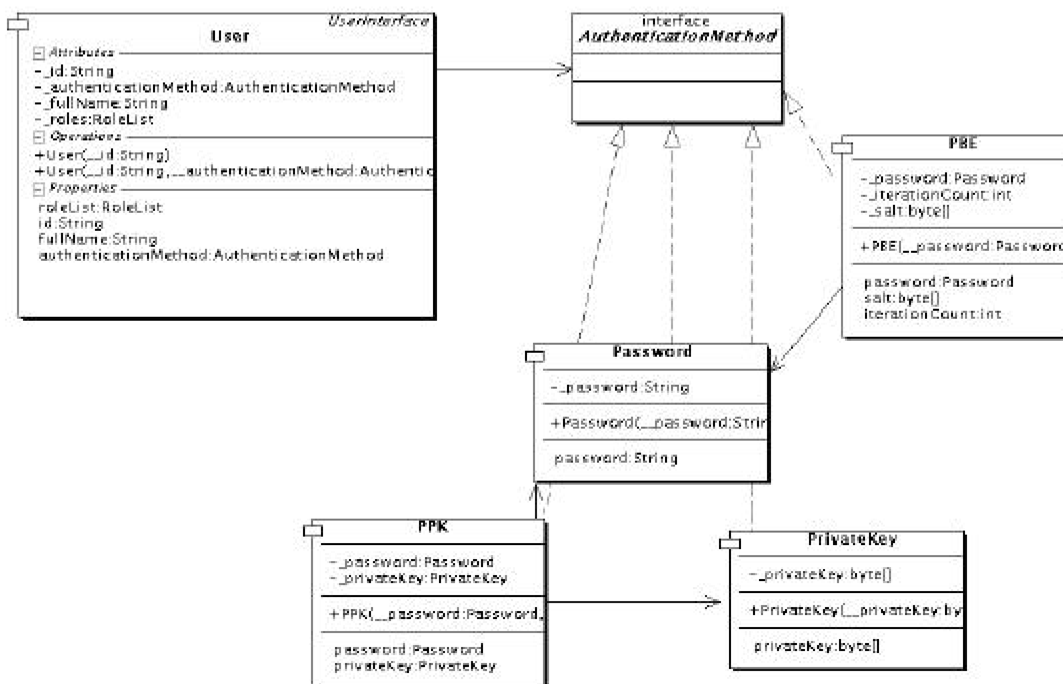

Figura 9: Classes correspondentes aos tipos de Autenticação

A camada de persistência é de grande importância, pois é através de sua utilização que os elementos do modelo RBAC poderão ser armazenados e recuperados de mecanismos de armazenamento persistentes. Uma qualidade desejável desta parte do *framework* é que ela seja extensível, ou seja, que possa suportar qualquer tipo de mecanismo de armazenamento persistente, principalmente banco de dados relacionais e arquivos comuns. A implementação da camada de persistência deste trabalho deu maior prioridade ao primeiro mecanismo, pois este oferece maior controle de segurança aos dados do que o segundo.

De acordo com [LAR00] , um *framework* para persistência é um conjunto de classes reutilizáveis, e que possam ser extensíveis, que forneçam serviços para objetos persistentes. Tipicamente, um *framework* para persistência tem que traduzir objetos para registros, salvá-los em um banco de dados e traduzir registros para objetos, quando for recuperar os mesmos do banco de dados.

A camada de persistência foi baseada no exemplo de *framework* de persistência de Craig Larman em [LAR00], que utiliza alguns padrões como *Object Identifier*, *DataBase Broker*, *Cache Management*, *Template Method*, *Virtual Proxy* e *Factory Method* de [BRO96] . Ele utiliza o mecanismo de materialização sob demanda. A materialização é o ato de transformar uma representação de dados não-orientada a objeto (por exemplo, registros), existente em um armazenamento persistente, em objetos. A desmaterialização é a atividade oposta, também conhecida como passivação.

A materialização sob demanda (também chamada de *Lazy*) é um mecanismo onde nem todos os objetos são materializados de uma só vez, uma instância em particular só é materializada sob demanda, quando necessária [LAR00].

É desejável ter uma forma consistente de relacionar objetos com registros e garantir que a materialização repetida de um objeto não resulte em objetos duplicados. Neste caso, o *framework* de persistência utiliza o padrão *Object Identifier* para atribuir um identificador de objeto (OID) para cada registro de objeto. Um OID é geralmente um valor alfanumérico, que deve ser único para cada instância de objeto. Se cada objeto está associado a um OID, e cada tabela tem um OID como chave básica, então cada objeto pode ser mapeado de forma unívoca para alguma linha em alguma tabela.

O padrão *Database Broker* propõe a criação de uma classe responsável pela materialização, desmaterialização e pelo *caching* (memorização prévia) dos objetos. De acordo com [LAR00], pode ser definida uma classe *broker* diferente para cada classe de objetos persistentes, e portanto, existirão diferentes tipos de *brokers* para diferentes tipos de armazenamento. O projeto de *Database Brokers* é baseado no padrão *Template Method*. A idéia deste padrão é definir um método gabarito (o *template method*) em uma superclasse que define o esqueleto de um algoritmo, com suas partes variantes e invariantes. O *template method* invoca outros métodos, alguns dos quais são operações que podem ser redefinidas em uma subclasse. Assim, as subclasses podem redefinir os

métodos que variam, de forma a acrescentar o seu próprio comportamento específico nos pontos de variação [LAR00].

Um outro requisito desejável é manter os objetos materializados em um *cache* (depósito temporário de objetos) local. Isso possibilita melhorar o desempenho, pois a materialização é relativamente lenta. O padrão *Cache Management* de [BRO96] propõe tornar os *Database Brokers* responsáveis pela manutenção do seu *cache*. Se for usado um *broker* diferente para cada classe de objeto persistente, cada um desses *brokers* poderá manter seu próprio *cache*.

Em algumas vezes é desejável adiar a materialização de um objeto até que seja absolutamente necessária. Isso é conhecido como materialização sob demanda. Esse tipo de materialização pode ser implementada utilizando o padrão *Virtual Proxy*.

Um *Virtual Proxy* é uma referência inteligente para o objeto real. Ele materializa o objeto real quando referenciado pela primeira vez, portanto implementa a materialização sob demanda. É um objeto "leve" segundo [LAR00] , que substitui um objeto real que pode, ou não, estar materializado. Um *Virtual Proxy* é considerado uma *smart reference* porque é tratado por um cliente como se fosse um objeto.

Outro padrão que também foi utilizado foi o *Factory Method*, que define uma *interface* para criação de um objeto, mas deixa para as subclasses decidirem quais classes irão instanciar.

A camada de persistência foi idealizada de forma que atenda a todos estes padrões apresentados anteriormente. Primeiramente criou-se a superclasse das classes *brokers*, denominada de *PersistentBroker.* Esta classe, como pode ser visto na Figura 10, possui apenas três métodos: *objectWith (OID)*, *inCache (OID)* e *materializeWith(OID)*, sendo que o primeiro é concreto e os outros dois são abstratos. O método *objectWith* é um *template method*, que retorna o objeto referente ao OID passado como parâmetro. O algoritmo deste método é simples, ele verifica se o objeto está em *cache* utilizando o método *inCache*, se estiver simplesmente retorna o objeto, e se não estiver, ele chama o método *materializeWith*, para realizar a materialização do objeto do banco de dados. O método *inCache* verifica se o objeto já foi trazido para a memória. Caso ele já esteja na memória, o próprio objeto é retornado, senão é retornado um objeto nulo. A implementação dos métodos abstratos *materializeWith* e *inCache* devem ser definidos

nas subclasses de *PersistentBroker*, pois dependem de como o objeto será armazenado e do tipo de objeto.

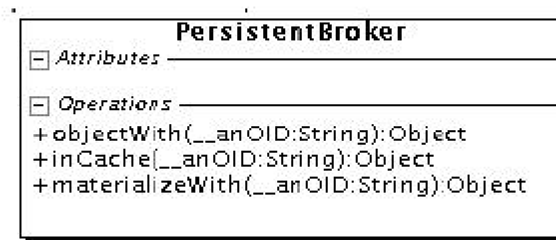

Figura 10: Diagrama da classe PersistentBroker

O principal mecanismo de armazenamento das informações referentes ao modelo RBAC é um banco de dados relacional. Deste modo, foi criada uma subclasse de *PersistentBroker*, denominada *RelationalPersistentBroker*, que é responsável pela materialização de objetos do banco de dados.

Esta subclasse possui também apenas três métodos: *materializeWith(OID)*, *selectFirst(OID)* e *currentRecordAsObject()*, sendo o primeiro concreto e os outros dois abstratos. O método *materializeWith* também é um *template method*, mas que implementa o método abstrato *materializeWith* de *PersistentBroker*. O algoritmo de *materializeWith* também é simples, ele faz uma chamada ao método *selectFirst* e depois uma chamada ao método *currentRecordAsObject*. O método *selectFirst* tem como objetivo executar a consulta do registro referente ao objeto no banco de dados e o método *currentRecordAsObject* tem como objetivo transformar o registro obtido na consulta em um objeto. Esses dois métodos deverão ser implementados pelas subclasses de *RelationalPersistentBroker* pois dependem do tipo de objeto que será materializado.

Como já foi dito na seção de projeto de arquitetura, o banco de dados escolhido para a implementação deste trabalho foi o Oracle 8i, mas a migração do *framework* para a utilização de outro banco de dados é simples, bastando apenas substituir o *driver* JDBC (*Java Database Connectivity*).

Como os três elementos principais do modelo RBAC são *User*, Role e *Permission*, foram incluídos no *Persistence Framework* as subclasses *brokers* de

RelationalPersistentBroker referentes a esses objetos: *RelationalBrokerUser*, *RelationalBrokerRole* e *RelationalBrokerPermission*. A principal função dessas subclasses é implementar os métodos abstratos *inCache*, *selectFirst* e *currentRecordAsObject* de acordo com o tipo de elemento (*User*, *Role* ou *Permission*) que representam, como pode ser visto na Figura 11.

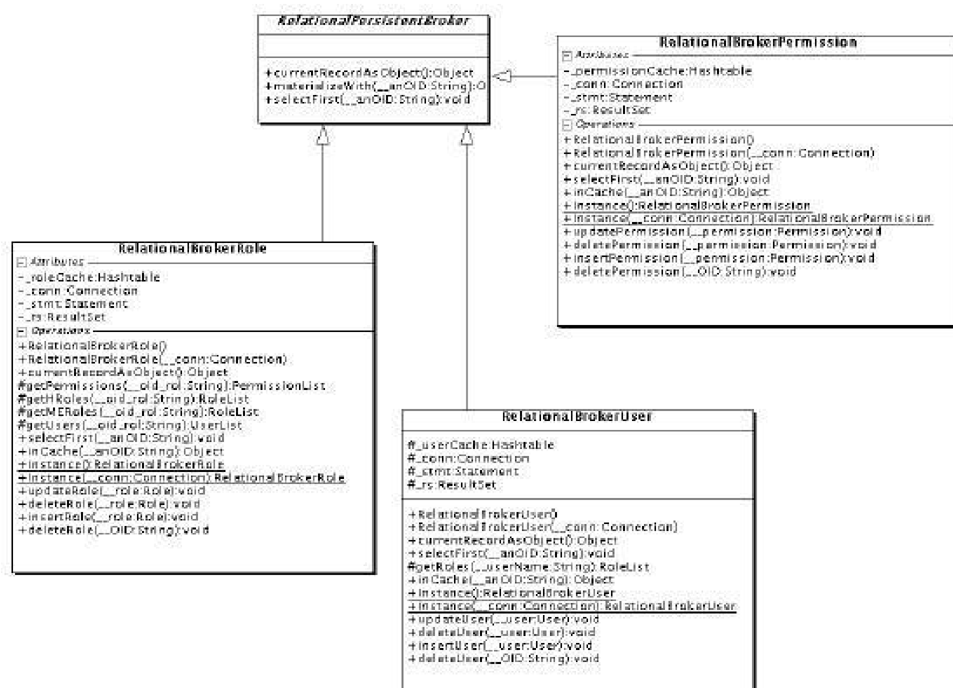

Figura 11: Diagramas das classes brokers dos elementos do modelo RBAC

Baseando-se nos padrões *Factory Method*, *Database Broker* e *Proxy*, foi criada a superclasse *VirtualProxy* (Figura 12). Esta classe possui atributos como o OID do objeto a qual se refere (*_OID*), o próprio objeto real (*_realSubject*), uma tabela *Hash* contendo os *brokers* que já foram utilizados (*_brokers*) e uma referência para o último *broker* que foi chamado (*_broker*).

Já que a classe *Virtual Proxy* é a superclasse dos objetos *proxy*, que irão substituir os objetos reais, é necessário que todas as classes herdeiras dessa classe implementem o método *materializeSubject*. Além desse método existe o método abstrato *createBroker*,

que deverá ser redefinido pelas subclasses de *VirtualProxy*, pois cada subclasse terá um tipo de *broker* diferente.

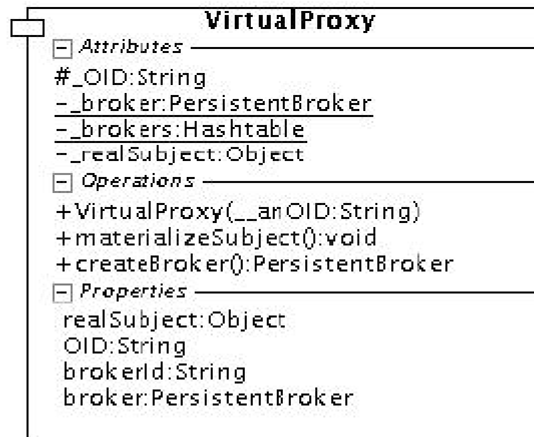

Figura 12: Diagrama de classe de Virtual Proxy

A partir deste ponto tornou-se necessário então a criação de um objeto *proxy* para cada classe representante de um elemento do modelo RBAC. Portanto, foram incluídas no *Persistence Framework*, as classes *UserProxy* (Figura 16), *RoleProxy* (Figura 17) e *PermissionProxy* (Figura 15). Todas as três classes herdam da superclasse *VirtualProxy*, e sobrescrevem o método *createBroker*, sendo que a classe *UserProxy* cria um *broker* de *RelationalBrokerUser*, a classe *RoleProxy* cria um *broker* de *RelationalBrokerRole* e por fim, a classe *PermissionProxy* cria um *broker* de *RelationalBrokerPermission*.

Além disso, cada uma dessas classes redefine alguns métodos das classes que irão substituir, como por exemplo, a classe *UserProxy* redefine o método *getFullName* da classe *User*, sendo que dentro deste método é invocada uma chamada do método de mesmo nome do objeto real.

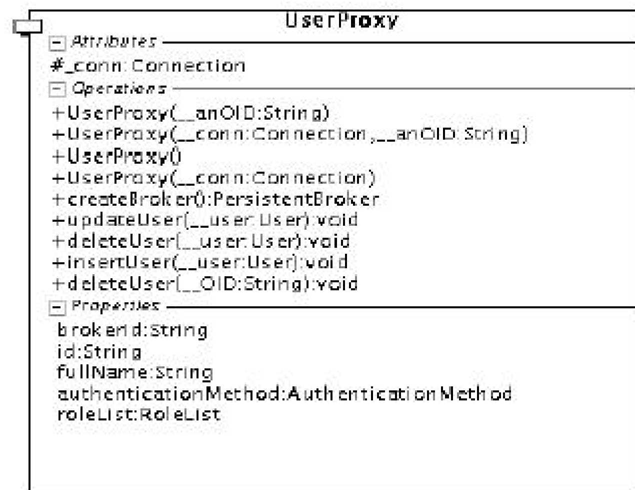

Figura 13: Diagrama da classe UserProxy

.

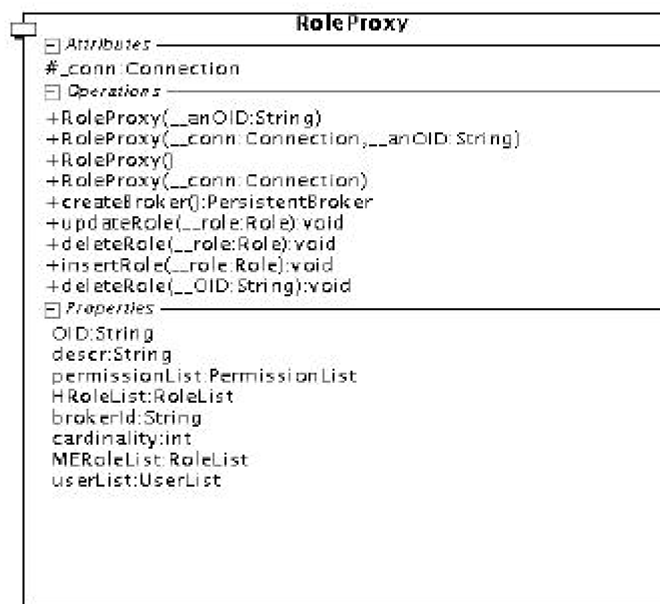

Figura 14: Diagrama da classe RoleProxy

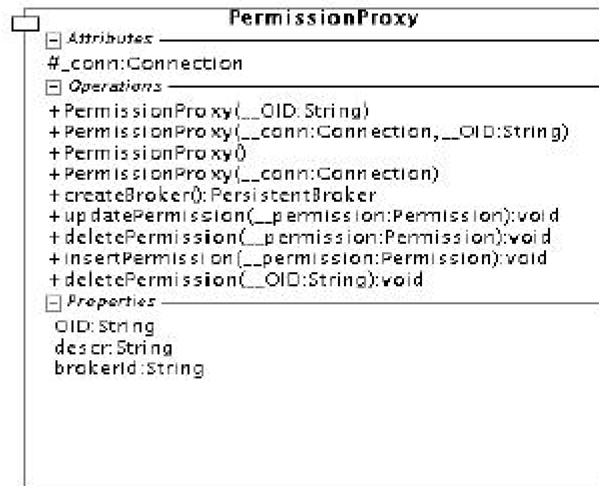

Figura 15: Diagrama da classe PermissionProxy

A camada de aplicação é aquela que contém as classes que utilizarão as camadas básica e de persistência para dar suporte à criação de aplicações de segurança seguindo o modelo de controle de acesso RBAC. Esta camada contém classes que são suporte à autenticação de usuários, ao controle de acesso, a auditoria e administração das informações de segurança. A camada de aplicação é aquela que contém a *interface* do *framework* com as aplicações que serão desenvolvidas a partir deste.

4.4 Autenticação

Existe uma variedade de métodos de autenticação de usuários, e estes métodos formam a base dos sistemas de controle de acesso. As três categorias de métodos para verificação da identidade de um usuário são baseadas em algo que o usuário sabe, tal qual uma senha; algo que o usuário possui, tal qual um *token* de autenticação; e alguma característica física do usuário, tal qual a impressão digital ou padrão de voz.

A autenticação pelo conhecimento é o modo mais utilizado para fornecer uma identidade a um computador, no qual destaca-se o uso de segredos, como senhas, chaves de criptografia, PIN *(Personal Identification Number*) e tudo mais que uma pessoa pode saber. Entretanto, este tipo de autenticação tem algumas limitações: elas podem ser adivinhadas, roubadas ou esquecidas. Soluções alternativas como perguntas randômicas e senhas descartáveis geralmente são simples de utilização e bem aceita pelos usuários,

baratas e fáceis de implementar. Além disso, não requerem hardware adicional como outras soluções baseadas em propriedade e características. Outra vantagem que vale destacar é que elas podem ser integradas em sistemas baseados em rede e na Web, além de diversos sistemas operacionais [FIO99]. Elas evitam vários problemas de ataques e problemas baseados em senha, mas não impedem que um usuário divulgue seu segredo para outro.

A utilização de perguntas randômicas, porém, adiciona uma dificuldade adicional ao usuário que quiser divulgar seu segredo, pois, ao contrário de contar apenas uma palavra (como no caso de senhas), terá que divulgar todas as informações constantes no questionário que serve de base para as perguntas randômicas.

As soluções de autenticação orientadas na propriedade se baseiam em um objeto físico que o usuário possui. Sua vantagem está relacionada com o princípio de que a duplicação do objeto de autenticação será mais cara que o valor do que está sendo guardado. As desvantagens deste tipo de autenticação são, que os objetos físicos podem ser perdidos ou esquecidos além do custo adicional do *hardware* [FIO99]. O problema da utilização destes dispositivos em aplicações Web em Intranets é que, assim como as senhas podem ser divulgadas para outras pessoas, os *tokens* também podem ser emprestados. Aliado ao custo do produto, isto faz com esta solução torne-se inviável para essas aplicações.

O último método de autenticação de usuários é baseado nas características físicas dos indivíduos. Os sistemas biométricos se baseiam em características fisiológicas e comportamentais de pessoas vivas. Os principais sistemas biométricos utilizados nos dias de hoje são baseados no reconhecimento de face, impressão digital, geometria da mão, íris, retina, padrão de voz, assinatura e ritmo de digitação. As vantagens desses sistemas são que eles não podem ser forjados nem tampouco esquecidos, obrigando que a pessoa a ser autenticada esteja fisicamente presente no ponto de autenticação. A desvantagem reside na falta de padrões, desconforto de usar alguns dispositivos biométricos e custos extras dos equipamentos envolvidos[FIO99].

Para dar suporte ao elemento de autenticação, foram criadas classes concretas que já implementam a categoria de autenticação através de senhas, por ser o tipo de autenticação atualmente mais utilizada em aplicações com interface Web. A autenticação

utilizando perguntas aleatórias não foi implementada, mas esta característica poderá ser feita por outro usuário do *framework*, pois este dará suporte para tal através de classes abstratas. O mesmo ocorre para os outros tipos de autenticação, como aqueles que utilizam objetos físicos ou sistemas biométricos.

A interface *Authentication,* Figura 16, possui um único método, *isValidUser (id:String, auth:AuthenticationMethod)*, que serve para verificar se um determinado usuário é válido. A classe que implementa esta interface é *AuthenticationSeguraweb*, que é a classe da camada de aplicação responsável por realizar a autenticação do usuário. Esta classe implementa o método *isValidUser* da interface *Authentication* utilizando o mecanismo de senha como método de autenticação do usuário, por ser este método o mais utilizado atualmente para aplicações Web.

A parte de autenticação será completamente abrangida no capítulo seguinte, explanando principalmente em relação a autenticação com chave pública.

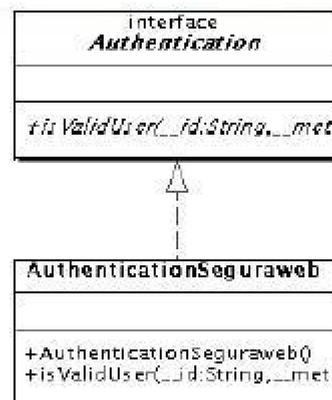

Figura 16: Componentes de Autenticação

4.5 Controle de Acesso

O objetivo desta parte do *framework* é dar suporte a verificação se os usuários possuem autorização ou não para realizar as operações, de acordo com o papel que o usuário possui no sistema. Esta parte é formada apenas por uma *interface* e uma classe concreta que implementa esta *interface*, como pode ser visto na Figura 17.

A *interface Authorization* possui apenas dois métodos que têm o mesmo objetivo, retornar se o usuário possui permissão ou não de executar uma operação. A classe *AuthorizarionSeguraweb* implementa esta *interface* utilizando as classes da camada de persistência para obter as informações do banco de dados. Por este motivo um dos atributos da classe é a conexão com o banco de dados.

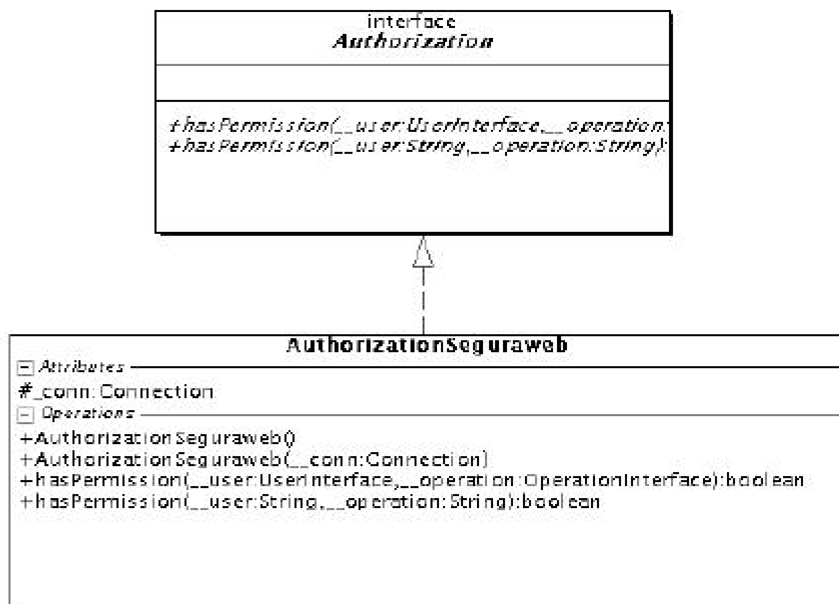

Figura 17: Diagrama de classes de Autorização

4.6 Administração

Esta parte do *framework* tem como objetivo dar suporte à gerência das informações de segurança de acordo com as regras do modelo de segurança RBAC.
Para alcançar este objetivo foram criadas classes responsáveis pela inserção, remoção e atualização de usuários, papéis e permissões; bem como classes responsáveis por gerenciar a criação de relacionamentos entre usuários e papéis e entre papéis e permissões.

A classe *DBUserManager* é a classe do *framework* que poderá ser utilizada para inserir e remover usuários, além de obter e alterar informações dos mesmos no banco de dados relacional. Esta classe pode ser vista na Figura 18.

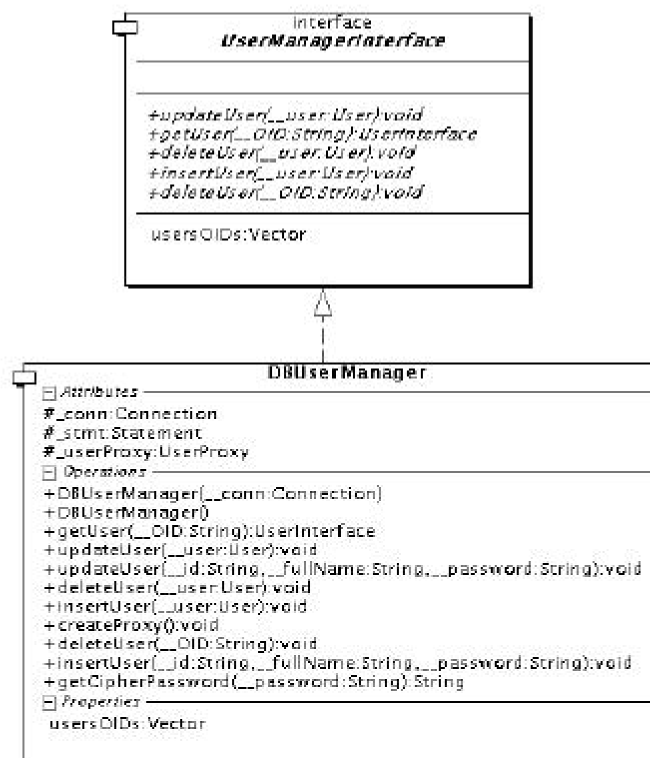

Figura 18: Diagrama da Classe DBUserManager

Como pode ser visto na Figura 18, a classe *DBUserManager* possui os seguintes métodos que são acessíveis ao usuário do *framework* Seguraweb:

- *getUser*: método para obter o objeto *proxy* referente ao usuário cujo identificador é passado como parâmetro;
- *updateUser*: atualiza as informações do usuário no banco de dados;
- *insertUser*: insere um novo usuário no banco de dados;
- *deleteUser*: remove o usuário no banco de dados.

A classe *DBRoleManager* dá suporte a inserção e remoção de papéis, bem como obtenção e alteração das informações dos mesmos no banco de dados relacional. Esta classe pode ser vista na Figura 19.

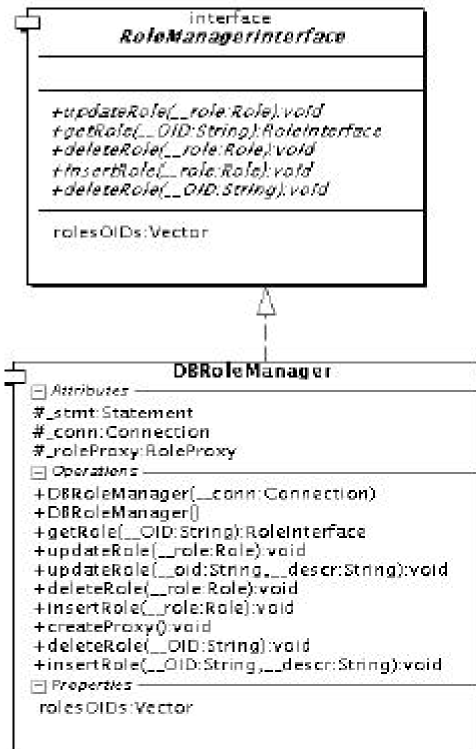

Figura 19: Diagrama da Classe Role Manager

Como pode ser visto na Figura 19, a classe *DBRoleManager* possui os seguintes métodos com acesso público:

- *getRole*: obtém o objeto *proxy* referente ao papel cujo identificador é passado como parâmetro;
- *updateRole*: atualiza as informações do papel no banco de dados;
- *insertRole*: insere um novo papel no banco de dados;
- *deleteRole*: remove o papel no banco de dados.

A classe *DBPermissionManager* possui métodos para inserir e remover permissões, além de obter e alterar as informações das mesmas no banco de dados relacional. Esta classe é apresentada na Figura 20.

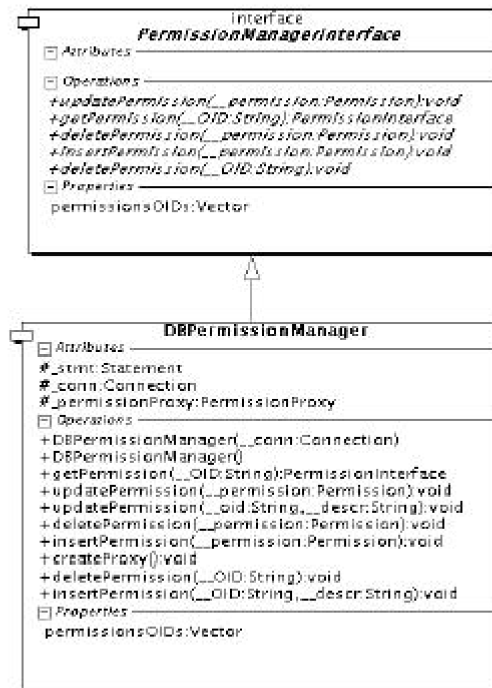

Figura 20: Diagrama da Classe DBPermissionManager

Como pode ser visto na Figura 20, a classe *DBPermissionManager* os métodos a seguir:

- *getPermission*: obtém o objeto *proxy* referente à permissão cujo identificador é passado como parâmetro;
- *updatePermission*: atualiza as informações da permissão no banco de dados;
- *insertPermission*: insere uma nova permissão no banco de dados;
- *deletePermission*: remove a permissão no banco de dados.

A classe *RBACUserRoleAssociationManager* (Figura 21) é a classe que dá suporte a criação e remoção de autorizações para usuários se tornarem membros de papéis, bem como da ativação e desativação de papéis para esses usuários, sempre seguindo algumas regras do modelo de segurança RBAC. No decorrer do texto será apresentado como essas regras do modelo de segurança RBAC foram implementadas dentro do *framework* Seguraweb.

Os métodos da classe *RBACUserRoleAssociationManager* são descritos abaixo:

- *makeAssociation*: estes métodos têm como objetivo autorizar um

usuário a se tornar membro do papel;

- *numberUsersAssociated*: retorna o número de usuários que estão atualmente autorizados ao papel cujo identificador é passado como parâmetro;
- *isAssociated*: retorna verdadeiro se o usuário já está autorizado a ser membro de um papel e falso caso contrário;
- *deleteAssociation*: retira a autorização do usuário ser membro do papel;
- *isME*: retorna se o papel cujo identificador é passado como parâmetro é mutuamente exclusivo a algum papel da lista de papéis ativos do usuário;
- *activateAssociation*: ativa o papel para o usuário, verificando se este é autorizado ou não;
- *disactivateAssociation*: desativa o papel para o usuário autorizado.

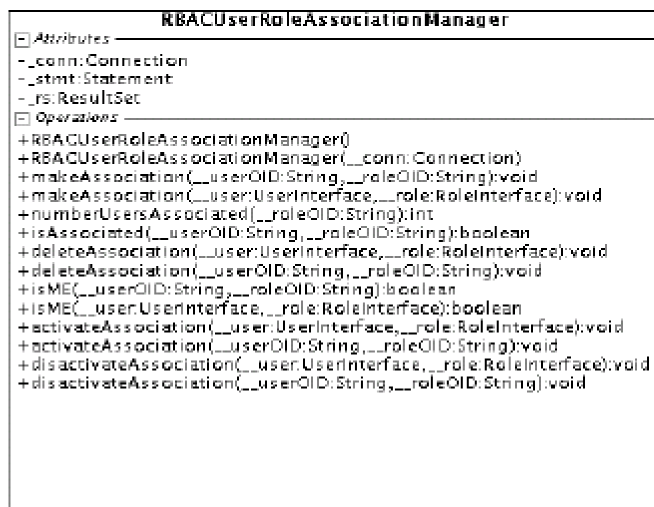

Figura 21: Diagrama da classe RBACUserRoleAssociationManager

A classe responsável pela associação de papéis e permissões é a classe *RBACRolePermissionAssociationManager* (Figura 22). Os métodos dessa classe são descritos a seguir:

- *makeAssociation*: realiza a associação entre o papel e a permissão;

- *isAssociated*: retorna verdadeiro caso a associação entre o papel e permissão já existe, e falso caso a associação não exista;
- *deleteAssociation*: remove a associação entre o papel e a permissão.

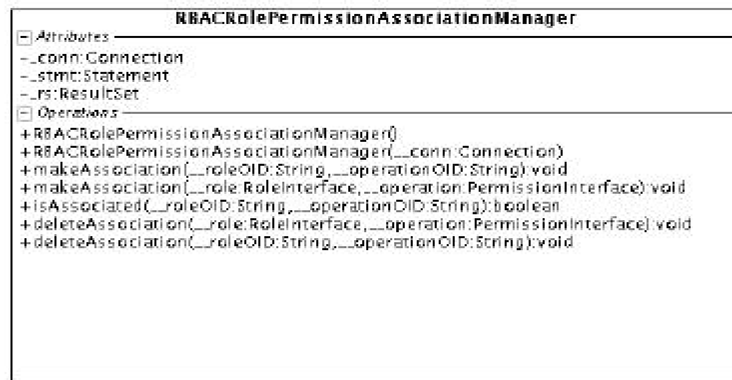

Figura 22: Diagrama da classe RBACRolePermissionAssociationManager

4.7 Implementação das Regras do Modelo RBAC

Como a proposta do *framework* Seguraweb é implementar o modelo de segurança RBAC é necessário apresentar como esta implementação foi feita. A seguir serão apresentadas as regras do modelo e como estas foram implementadas no *framework* Seguraweb.

4.7.1 Hierarquia de Papéis

A regra da hierarquia de papéis foi implementada criando o atributo *_hRoles* do tipo *RoleList* na classe *Role* (Figura 7). Esse atributo contém a lista dos *proxies* dos papéis na qual o papel herda as permissões. Existe uma outra lista, *_permissions*, que contém as permissões que estão relacionadas com o papel. No momento da materialização do papel, as permissões relacionadas com este e com os papéis que estão na lista de hierarquia *_hRoles*, são carregadas para a memória.

a) Cardinalidade

A propriedade de cardinalidade foi implementada através da criação do atributo *_cardinality* na classe *Role* (Figura 7). Quando um papel é criado, deve ser definida uma cardinalidade para ele. Antes de realizar uma associação entre um usuário e um papel, é verificado a quantidade de usuários que já estão relacionados com este papel e comparado com a cardinalidade dele. Se o número de usuários já relacionados ao papel for igual à cardinalidade deste, a associação entre o usuário e o papel não é realizada.

Uma associação entre um usuário e um papel pode ser realizada utilizando os métodos *makeAssociation* da classe *RBACUserRoleAssociationManager* (Figura 21). O método *_numberUsersAssociated* é o responsável por obter o número de usuários já relacionados com o papel cujo identificador é passado como parâmetro.

b) Autorização de Papéis

Esta propriedade é implementada no método *activateAssociation*, que tem como objetivo ativar um papel para um usuário. Antes de realizar a ativação, é verificado se o papel está autorizado para o usuário utilizando o método *isAssociated*. Se o papel não estiver autorizado para o usuário a sua ativação não é permitida.

c) Execução de Papéis, Autorização de Operação e Autorização de Acesso a Objeto

As regra de execução de papéis, autorização de operação e autorização de acesso a objeto foram implementadas nos métodos *hasPermission* da classe *AuthorizationSeguraweb* (Figura 20). Neste método é obtida a lista de papéis ativos do usuário. Para cada papel ativo é obtida a lista de permissões relacionadas. Neste momento é verificado se a permissão está contida na lista de permissões. Se a permissão estiver contida em alguma lista de permissões de algum papel ativo indica que o usuário pode executar a operação.

d) Separação Dinâmica de Tarefas

A regra da Separação Dinâmica de Tarefas foi implementada com a criação do atributo *_meRoles* do tipo *RoleList* na classe *Role*. Esse atributo contém a lista de papéis na qual o papel é mutuamente exclusivo. Assim, no momento em que é feita a ativação entre um usuário e um papel, é obtida a lista de papéis ativos do usuário (atributo *_roles* da classe *User*) e para cada papel desta lista é verificado se não está contido na lista de papéis mutuamente exclusivos do papel que se deseja fazer a ativação. Se existir pelo menos um papel mutuamente exclusivo, a ativação do papel para o usuário não é realizada. Outro ponto a ser considerado é que, caso exista em um determinado momento, um usuário com dois papéis ativos e posteriormente estes papéis são definidos como mutuamente exclusivos, os dois papéis são automaticamente desativados para o usuário. Isto é realizado para evitar inconsistências, já que de acordo com a regra de Separação Dinâmica de Tarefas, um usuário não pode ter dois papéis mutuamente exclusivos ativos no mesmo momento.

Optou-se por implementar apenas a Separação Dinâmica de Tarefas por esta ser considerada menos rígida que a regra de Separação Estática de Tarefas, pois permite que o usuário esteja autorizado ao mesmo tempo a papéis mutuamente exclusivos, mas não permite que esses papéis sejam ativados ao mesmo tempo. A Separação Operacional de Tarefas também não foi implementada.

4.8 Aplicações de Teste

Seguindo o esquema de [BOS97] , a última etapa do desenvolvimento de um *framework* é a geração de uma aplicação de teste. Neste trabalho foi criada uma aplicações de teste, sendo que foi utilizado o segundo teste da autora do framework. No teste seguinte foi colocada a autenticação via assinatura digital e testada a parte de auditoria, criando os logs de autenticação, além disto os testes servem para validar a parte de administração das informações de segurança e outra para validar o esquema de autorização.

4.8.1 A Aplicação SecServer

Uma chamada telefônica é definida, segundo a ANATEL [ANA02], como uma ação realizada pelo chamador a fim de obter comunicação com o equipamento terminal desejado. Chamador é quem origina a chamada, e é também conhecido como Assinante A ou Originador. Quem recebe a chamada é conhecido como Chamado, Asssinante B ou Destino [SOU02].

Para que as prestadoras possam cobrar dos seus usuários estas chamadas é necessário registrar os dados das chamadas telefônicas, a fim de permitir que posteriormente seja feito o cálculo dos custos das chamada e emissão de contas. O responsável por registrar todas as informações necessárias chama-se sistema de bilhetagem. As centrais telefônicas, além de servir como nó de comutação de chamadas, também podem executar a função de bilhetagem, mas nem todas executam esta função. As centrais com função de bilhetagem serão chamadas de centrais bilhetadoras [SOU02].

A central registra estas e outras informações da chamada num registro chamado bilhete ou CDR (Call Detail Record). A central bilhetadora armazena então este CDR juntamente com outros numa memória de massa. É este conjunto de CDR, que constituem um arquivo e que devem ser filtrados, dependendo das informações neles contidas [SOU02].

Este mesmo arquivo pode alimentar diferentes sistemas dentro de uma prestadora, mas geralmente contendo apenas informações úteis ao sistema em questão. Deste modo, é indesejável à área de faturamento receber bilhetes que não são faturáveis, provocando um processamento desnecessário. Os bilhetes não faturáveis, por exemplo por congestionamento na rede, são extremamente úteis na área de rede detectar os motivos e horários de maior congestionamento, enquanto que os faturáveis não possuem muita utilidade. Daí a importância da filtragem de bilhetes evitando o processamento de informação desnecessária [SOU02].

É neste ponto que o trabalho de mestrado de [SOU02] atua. O trabalho se trata de um mecanismo para a filtragem de CDRs. Este mecanismo é formado por um Servidor

de Segurança, um Servidor de Aplicação e um Servidor de Compilação. Resumidamente, os filtros de CDRs são construídos pelo usuário responsável pelas definições destes filtros. Após a definição dos filtros, estes serão enviados ao Servidor de Compilação, onde serão transformados em código fonte. A partir deste código fonte,
são geradas *shared libraries* que estarão disponíveis para serem executadas pelo
Servidor de Aplicação. Isso é feito através de uma requisição do usuário para aplicação
de um determinado filtro sobre o fluxo de dados, ou seja, o conjunto de CDRs.

No trabalho de [SOU02], surgiu a necessidade de integração com um sistema de segurança, utilizando CORBA. Diante disto, bem como da necessidade de implementação de uma aplicação para validação do *framework* Seguraweb, foi criada a aplicação *SecServer*. Esta aplicação se trata de um servidor de segurança que atende a requisições de pedidos de autenticação de usuários e de verificações de controle de acesso, utilizando a tecnologia CORBA.

4.9 Conclusões do Capítulo

Este capítulo apresentou alguns trabalhos relacionados com o trabalho realizado nesta dissertação, bem como foi feita uma breve comparação entre estes trabalhos e o *framework* Seguraweb. Após foram apresentadas as etapas de análise e projeto do *framework* Seguraweb, como análise de domínio, o projeto de arquitetura, o projeto do *framework* e a criação de aplicações de teste para a validação do *framework*. No momento da apresentação do projeto do *framework*, foi mostrada a estrutura deste e suas características, detalhando a implementação da persistência das informações de segurança e mostrando como cada regra do modelo de segurança RBAC foi implementada dentro do *framework*[AKK02].

Capítulo 5

Introdução

A criptografia por chave privada tem algumas desvantagens, a principal dela provém do perigo de ter a chave privada perdida ou roubada, toda a segurança fica comprometida. Na criptografia com chave pública pode-se evitar este problema, visto que são utilizadas chaves públicas e privadas. A assinatura digital permite autenticação, requisito fundamental do sistema, pois se saberá que usuários esta entrando no sistema. Serviços de auditoria permitem ao sistema gerenciar o monitoramento de eventos para posterior análise e investigação

5.1 Autenticação por chave pública

As tecnologias (Servlets,JSP e Applets), por serem baseadas na linguagem Java possuem a vantagem de serem independente de plataforma, ou seja, podem ser executadas tanto em plataforma Unix, Linux ou Windows. Este é o principal motivo pela escolha de Java como linguagem de implementação do framework, bem como também por Java ser uma linguagem orientada a objetos. Deste modo o framework se torna mais abrangente, pois pode ser utilizado por aplicações baseadas em Servlets, JSP ou em Applets. Estas características fazem com que Java seja a linguagem mais escolhida para implementação de aplicações na Internet. Além disso, a linguagem Java possui um modelo de segurança, dispondo de algumas APIs que implementam funções de segurança, como por exemplo, JCE (Java Cryptography), JSSE (Java Secure Socket Extension) e JAAS (Java Authentication and Authorization Service) [GON01]. Outra vantagem da implementação em Java é a facilidade de portar a aplicação para utilização de CORBA.

A autenticação é a garantia que um usuário é quem ele diz ser, provando assim que ele está ou não autorizado no sistema. A autenticação é uma questão fundamental quando duas entidades trocam dados entre si (sessão). Em grande parte trata-se de um

problema de distribuição de chaves. Como foi referida, uma implementação segura de distribuição de chaves obriga a:

- Criptografia convencional: a existência de um centro de distribuição de chaves (KDC - "Key Distribution Center")
- Criptografia de chave pública: a existência de um emissor de certificados de chave pública

Existe uma variedade de métodos de autenticação de usuários, e estes métodos formam a base dos sistemas de controle de acesso. As três categorias de métodos para verificação de identidade de um usuário são baseadas em algo que o usuário sabe, tal qual uma senha; algo que o usuário possui, tal qual um token de autenticação; e alguma característica física do usuário, tal qual a impressão digital ou padrão de voz.A autenticação pelo conhecimento é o modo mais utilizado para fornecer uma identidade a um computador, no qual destaca-se uso de segredos, como senhas, chaves de criptografia, PIN (Personal Identification Number).

A solução para problemas de identificação, autenticação e privacidade em sistemas de computador normalmente reside no campo da criptografia.

Criptografia é caracterizada como a ciência de escrever em códigos ou em cifras, ou seja, é um conjunto de métodos que permite tornar incompreensível uma mensagem (ou informação), de forma a permitir que apenas as pessoas autorizadas consigam decifrá-la e compreendê-la. É o embaralhamento dos dados referentes a um arquivo, e-mail ou transação. Desta forma, as informações são protegidas, antes de serem enviadas de um computador para outro, de maneira que somente o cliente e o destinatário dos dados possam lê-los [STA99].

O uso de criptografia surge pela necessidade de se enviar informações sigilosas através de um meio de comunicação não confiável, ou seja, da necessidade de confidencialidade das informações. Seu objetivo é garantir que um intruso não tenha acesso à informação (ataque passivos) ou não possa modificá-la de alguma forma (ataques ativos).

Um bom método de criptografia deve garantir que seja, senão impossível, pelo menos muito difícil, que um intruso recupere, a partir do texto criptografado e do conhecimento deste método, o valor das chaves. Assim, a confidencialidade do texto

transmitido será assegurada enquanto as chaves se mantiverem secretas. Outra forma de se avaliar a confidencialidade obtida por um método de criptografia é afirmando que ele é computacionalmente seguro. Para isso ele deve atender os seguintes critérios: que o custo de se tentar decifrar o texto criptografado seja superior ao valor da informação criptografado, e, que o tempo necessário para decifrá-lo seja superior à vida útil da informação.

Existem dois tipos principais de criptografia: a simétrica e a assimétrica.

5.1.1 Criptografia simétrica

Quando a chave de cifragem e a de decifragem são iguais, tem-se a criptografia simétrica. Os algoritmos de chave simétrica são baseados em operações simples, de fácil implementação (tanto em hardware como em software) e com grande velocidade de processamento. Entre estas operações podem ser citadas substituição, permutação, operações aritméticas simples e operações booleanas (em especial a operação de ou exclusivo). Entre os principais algoritmos utilizados atualmente podem ser citados:

- Triple DES
- IDEA
- Blowfish
- RC5
- CAST-128
- RC2

A proteção da privacidade usando um algoritmo simétrico, como o contido no DES (o Data Encryption Standard, patrocinado pelo governo americano) é relativamente fácil em pequenas redes, exigindo a troca de chaves de criptografia secreta entre cada parte. À medida que uma rede prolifera, a troca segura de chaves secretas se torna cada vez mais dispendiosa e desajeitada. Em conseqüência, esta solução por si só é impraticável mesmo para redes moderadamente grandes [SCH96].

O DES requer o compartilhamento de uma chave secreta, mesmo caso de todos os algoritmos simétricos. Cada pessoa precisa confiar na outra que guarda a chave secreta do par, sem revelá-la para ninguém. Como o usuário deve ter uma chave diferente para cada

pessoa com que se comunica, ele deve confiar em cada pessoa com uma das suas chaves secretas. Isto significa que em implementações práticas, a comunicação segura somente pode ocorrer entre pessoas com algum tipo de relacionamento anterior, seja ele pessoal ou profissional.

Questões fundamentais não tratadas pelo DES são a autenticação e o não-repúdio:

- Autenticação para se garantir a identificação das pessoas ou organizações envolvidas na comunicação e apurar suas responsabilidades, funcionários são responsáveis por suas ações relacionadas à segurança;
- Não-repúdio, garantia que o emissor de uma mensagem ou a pessoa que executou determinada transação de forma eletrônica, não poderá, posteriormente negar sua autoria, e também funcionários autorizados não podem ter seu acesso negado maliciosamente

A autenticação não pode ser assegurada uma vez que chaves secretas compartilhadas impedem que uma das partes comprove quem esta na oura extremidade, pois a decifragem dos dados pode ser utilizando por qualquer um que tenha a chave secreta em seu poder, sendo assim, a mesma chave que possibilita a comunicação segura poderia ser usada para criar falsificações no nome do outro usuário.

Os problemas de autenticação e proteção de privacidade em redes grandes foram tratados teoricamente em 1976 por Whitfield Diffie e Martin Hellman quando públicaram seus conceitos para um método de trocar mensagens secretas sem trocar chaves secretas. Utilizando esta idéia Ronald Rivest, Adi Shamir e Len Adleman, na ocasião professores do Massachusetts Institute of Technology., inventaram o RSA Public Key Cryptosystem em 1977.

Em vez de usar a mesma chave igualmente para criptografar e decifrar os dados, o sistema RSA utiliza um par coincidente de chaves de criptografia e decriptografia. Um par é definido como coincidente quando uma chave é capaz de criptografar a mensagem e através de operações inversas a outra chave é capaz de recuperar a informação original. Na criptorgrafia simétrica RSA cada chave executa uma transformação unidirecional sobre os dados. Cada chave é a função inversa da outra; o que uma faz, somente a outra pode desfazer.

5.1.2 Criptografia assimétrica

A criptografia usando chave Pública ou Assimétrica consiste na utilização de um par de chaves, uma pública e outra privada, diferentemente da criptografia convencional, que utilizava a mesma chave para cifrar e decifrar. A pública, como o próprio nome diz é de conhecimento público e é divulgada em diversas maneiras (LDAP, e-mail...). A chave Privada é de conhecimento somente do próprio usuário. O que for cifrado utilizando uma das chaves somente poderá ser visualizado com a outra.

Com a chave pública é possível prover os serviços de confidencialidade, autenticação e distribuição de chaves.

A Chave Pública RSA é disponibilizada publicamente por seu proprietário, enquanto a Chave Privada RSA é mantida secreta. Para enviar uma mensagem secreta, um autor embaralha a mensagem com a Chave Pública do destinatário pretendido. Estando assim criptografada, a mensagem somente pode ser decodificada com a Chave Privada do destinatário [RSA98].

Inversamente, o usuário também pode embaralhar os dados usando sua Chave Privada; em outras palavras, as chaves RSA funcionam em qualquer uma das duas direções. Isto fornece a base para a "assinatura digital”, pois se o usuário pode desembaralhar uma mensagem com a Chave Pública de alguém, o outro usuário deve ter usado sua Chave Privada para embaralhá-la em primeiro lugar. Como somente o proprietário pode utilizar sua própria chave privada, a mensagem embaralhada se torna um tipo de assinatura eletrônica -- um documento que ninguém mais pode produzir.

Uma assinatura digital é criada produzindo-se um valor de hash para um texto de uma mensagem.Isto gera um resumo da mensagem. O resumo da mensagem é em seguida criptografado com uso da chave privativa do indivíduo que está enviando a mensagem, transformando-a em uma assinatura digital. A assinatura digital somente pode ser decriptografia pela chave pública do mesmo indivíduo. O destinatário da mensagem decifra a assinatura digital e, em seguida, recalcula o resumo da mensagem. O valor deste resumo da mensagem recém-calculado é comparado ao valor do resumo de mensagem encontrado na assinatura. Se os dois coincidirem, a mensagem não foi adulterada. Como

a chave pública do remetente foi usada para verificar a assinatura, o texto deve ter sido assinado com a chave privada conhecida somente pelo remetente.

5.2 Auditoria

Serviços de auditoria permitem ao sistema gerenciar o monitoramento de eventos do ORB, como por exemplo, tentativas de violação de segurança através de registros dos detalhes de eventos relevantes a segurança do sistema. Um grande numero de eventos pode ser registrados. Entretanto, políticas de auditoria são usadas para restringir quais os tipos de eventos devem ser examinados e em quais circunstâncias [MAY97].

Existem duas categorias de políticas de auditoria:

- As políticas de auditoria de sistema que controlam quais eventos são registrados como resultado das atividades relevantes do sistema, como autenticação, mudanças de privilégios, sucesso ou falha de invocação de objetos e administração de políticas de segurança.
- As políticas de auditoria da aplicação que controla quais eventos são examinados pelas aplicações cientes da segurança.

Os eventos podem ser registrados sobre rastros de auditoria para depois serem analisados ou surgem alarmes que podem ser enviados aos administradores.

O *framework* RBAC SeguraWeb permite que todos os eventos significativos possam ser registrados para subsidiar uma posterior auditoria. Para tanto, os objetos ou classes de objetos também podem ser descritos para que sejam passíveis de registros de eventos ou não dependendo do nível de segurança que se quer. Desta maneira, pode-se, seletivamente configurar o acesso a quais objetos que serão passíveis de registros, limitando o registro aos eventos realmente significativos para aplicação.

Tais atitudes são importantes para a manutenção preventiva e corretiva. Podem por exemplo criar logs de tudo que o usuário faz, durante o tempo que está usando os recursos no sistema, desde quando ele se loga até o quando ele sai do sistema, como

também podem ser criados logs de alarme quando um usuário teve de alguma forma seus papéis alterados dentro do sistema.

A proposta de introdução de outras classes ao Seguraweb como as classe relativas a auditoria teve como finalidade facilitar seu uso dando suporte para o uso de logs no Seguraweb.

Num sistema de auditoria, uma boa maneira de ser extrair tudo de um bom sistema de auditoria e que ele permita que o acesso ao *software* de sistema seja restrito a um número limitado de pessoas, cujas responsabilidades exijam esse acesso. Criando logs de alarmes [DOR02]**.**

O sistema de auditoria deve apresentar logs quando o usuário ultrapassar o limite de tentativas de se logar ao sistema Os sistemas não permitem mais que três tentativas de *log-on* (entrada em comandos para iniciar uma sessão) com senhas inválidas. Trilhas de auditoria são mantidas e todas as atividades envolvendo acesso e modificação de arquivos vulneráveis ou críticos são registradas [DOR02]**.**

Violações de segurança e atividades suspeitas, tais como tentativas frustradas de entrada no sistema, são relatadas para a gerência e investigadas (examinar os relatórios sobre atividades suspeitas).Os gerentes de segurança investigam as violações de segurança e relatam os resultados para a alta administração.Medidas de disciplina são tomadas para corrigir as violações de segurança detectadas.Políticas de controle de acesso são modificadas quando violações de segurança são detectadas [DOR02].

O controle sobre o acesso e a alteração do *software* de sistema são essenciais para oferecer uma garantia razoável de que os controles de segurança baseados no sistema operacional não estão comprometidos, prejudicando o bom funcionamento do sistema computacional como um todo.

Numa auditoria deve-se averiguar se o um indivíduo tem o controle de todos os estágios críticos de um processo (por exemplo, um programador com permissão para independentemente escrever, testar e aprovar alterações de programa).

Funções distintas são desempenhadas por diferentes indivíduos. Com os logs pode-se prevenir que possíveis funcionários recebam papéis além da sua função.

A camada de auditoria foi implementada dando a opção do administrador escolher de que forma ele fará os logs do sistema. Foram criadas duas classes que vão permitir que

seja feitos os logs onde o administrador achar mais conveniente para implementação do sistema que ele fará usando o Seguraweb. Esta classe procura facilitar o uso da parte da auditoria cabendo ao administrador somente definir quais serão os eventos mais relevantes, que serão registrados para posterior análise através de um arquivo de logs ou através de uma consulta ao banco de dados contendo todos os dados ou eventos que foram considerados relevantes para o sistema. Na figura 23, o diagrama de classes da arquitetura da camada de auditoria ela mostra o relacionamento das classes criadas com a classe LogsInterace, da qual elas são classes-filhas. Foram implementadas duas classes a FileLogs tem como responsabilidade criar logs em arquivo texto e a classe DBlogs permite a criação de logs em banco de dados.

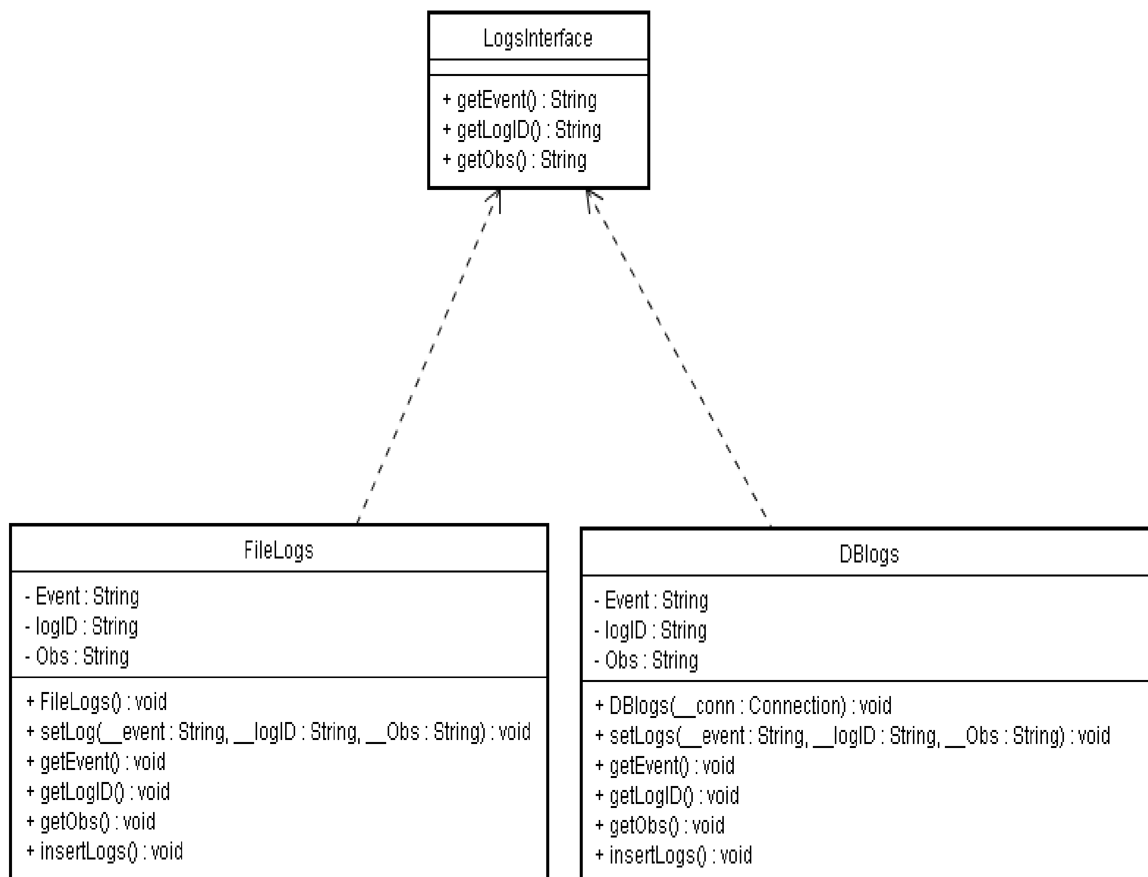

Figura 23: Arquitetura da camada de auditoria

A classe FileLogs apresentada acima fará os logs em registro criando documentos de logs em arquivos de texto simples, sendo então possível para o administrador ou para

o auditor fazer suas analise sobre possíveis falhas e entre outras coisas dependendo do que o auditor considerar relevante.

A classe DBLogs foi implementada para registrar os logs num banco de dados, foi necessário criar uma tabela para conter os logs do sistema. Nesta tabela terá a coluna event, onde será colocado que evento vai ser relevante de auditoria, coluna LogID, para colocar o nome ou o id do usuário conforme for implementado o sistema usando o RBAC, coluna Obs, que vai registrar observações a mais que o log precisará dependendo do que será feito ou do que será registrado e uma coluna DATALOGS que vai registrar a data com hora que foi gerado aquele evento, para depois poder ser feito a analise através do dia ou hora de registro do log.

A classe Dblogs terá também ligação com a classe RelationalPersistentDBlogs que será a classe responsável por criar os logs no banco de dado. A camada de persistência criada para o Dblogs foi baseada no exemplo de framework de persistência e seguindo a tendência do trabalho SeguraWeb. Ele usará a classe já implementada no SeguraWeb, que usa o padrão Database Broker que propõe a criação de uma classe responsável pela materialização, desmaterialização e pelo caching dos objetos. O objeto em questão será o Dblogs onde será feito os registro no banco de dados.

No método *setlog(__event:String,__logID:String,__obs:String,),* tanto na classe FileLogs ou DBlogs, vai ser usado quando o administrador, depois de definido quais os eventos relevante para o sistema, ele vai indicar que evento(primeiro parâmetro) será passível de auditoria. Ele pode definir que a autenticação será um evento relevante de auditoria toda vez que o usuário for ser cadastrar é feito um log, então ele define como *__event* igual à autenticação , sendo possível depois para fazer a análise dos logs e ver que o evento foi autenticação. No segundo parâmetro ele passa o logID que dependendo a finalidade pode ser por exemplo o nome do usuário ou que usuário teve sua permissão retirada. No terceiro parâmetro é definido o que o administrador ou auditor achar conveniente que os logs tenham, por exemplo, que a autenticação do usuário falhou.

5.3 Classes adicionadas ao *framework*

5.3.1 Classes de administração:

Esta classe contém o método responsável em inserção de novos usuários, onde o usuário vai escolher a sua senha ou será definida sua senha pública que vai ficar armazenado no banco para quando ele for se logar fará sua assinatura com a chave privada e será validado caso esteja a sua assinatura de acordo com a chave pública. A camada de persistência é implementada no banco de dados ORACLE.

A classe que contém as tarefas de administração trás a classe DBUserManager que é implementa a classe UserManagerInterface e ele é superclasse de mais duas classes que são a:

- *DBUserManagerPBE*: ela é subclasse da DBUserManager implementa autenticação através de quatro categorias que são:
  - Password (senha)
  - ,PrivateKey (chave privada),
  - PBE (Password-Based Encryption) é usado para que quando o password digitado (na autenticação) e criptografado e comparado com o password criptgrafado que esta armazenada em um banco de dados na tabela do usuário no campo password.
  - PPK (Password and Private Key) é utilizado para que quando se fizer a autenticação será utilizado um password digitado pelo usuário este password será criptografado pela chave privada arma zenada no banco de dados, aumentando a segurança no momento que passará o password para a autenticação. É utilizada a chave e privada e senha conjuntamente. Quando o usuário for cadastrado através da superclasse DBUserManager a subclasse

DBUserManagerPBE vai gerar um chave privada que vai ficar registrado na camada de persistência.

- *DBUserManagerSIG:* ela é subclasse da DBUserManager, ela vai implementar a autenticação por assinatura digital usando o algoritmo de assinatura digital DSA. É nesta classe que o usuário vai ser cadastrado, sendo colocado, entre outras coisas, sua senha que vai servir para gerar a chave pública esta vai ficar armazenada no banco de dados. Esta chave pública armazenada vai ser utilizada para quando o usuário for se autenticar ela fará a verificação da assinatura digital.

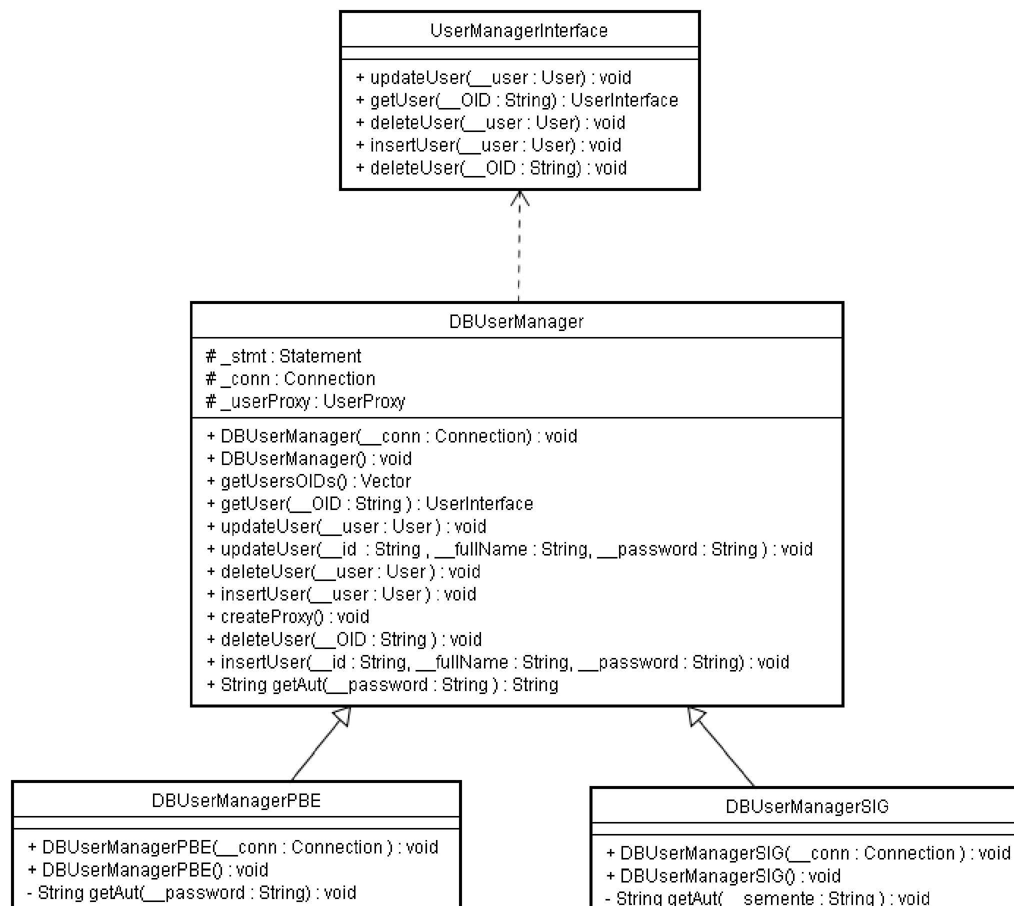

Figura 24: Diagrama de classe de administração de usuário e sua chaves Autenticação

5.3.2 Classes de autenticação:

Uma assinatura digital é o criptograma resultante da cifragem de um determinado bloco de dados (*documento*) pela utilização da chave privada em um algoritmo assimétrico. A verificação da assinatura é feita "decifrando-se" o criptograma (*assinatura*) com a suposta chave pública correspondente. Se o resultado for "válido", a assinatura é considerada "válida", ou seja, autêntica, uma vez que apenas o detentor da chave privada, par da chave pública utilizada, poderia ter gerado aquele criptograma.Na figura é ilustrado este procedimento.

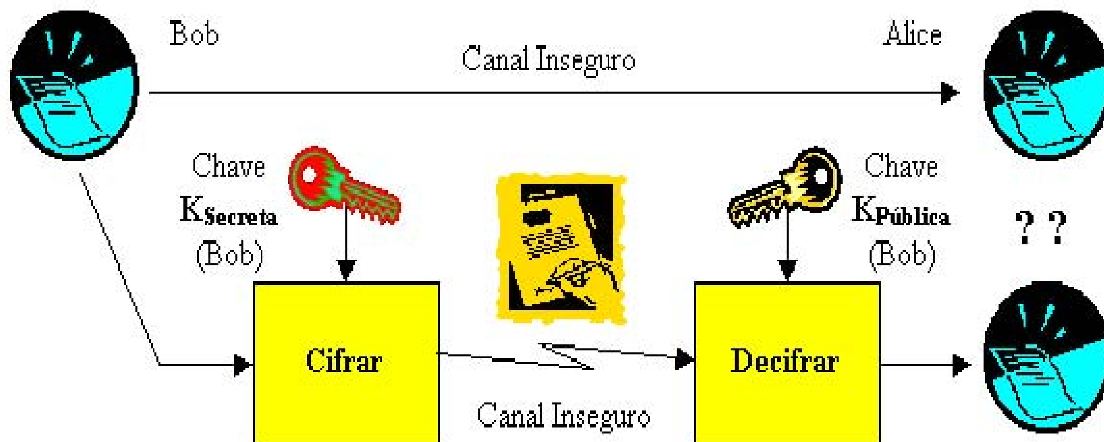

Figura 25: Procedimento de assinatura digital

A mensagem é cifrada com uma chave secreta K, que é do conhecimento somente da fonte "Bob". Ela é confidencial porque somente a fonte tem a chave privada podendo criar a mensagem com sua chave privada. Já sua chave pública e de conhecimento de todos o que permite que "Alice" possa decifrar a mensagem que "Bob" mandou e ele não pode negar a autoria da mensagem porque somente ele poderia ter cifrado a mensagem com sua chave privada. E a mensagem por "Bob" só poderá ser decifrada pela sua chave pública [SCH96].

Este procedimento é capaz de garantir tanto a origem (autenticação do emissor), tendo em vista que supostamente somente Bob conhece sua chave privada e, portanto somente ele é capaz de gerar uma assinatura que possa ser verificada com sua chave pública, como também a integridade do documento, já que, se o mesmo for alterado, a verificação da assinatura irá indicar isto, caso tenha vindo efetivamente do pretenso emissor.

As assinaturas digitais são possíveis de serem verificadas usando chaves públicas, uma das características da assinatura digital é poder assinar informações em um sistema de computador e depois provar sua autenticidade sem se preocupar com a segurança do sistema que as armazena, este é um dos fatores mais importantes de se optar por algoritmos de criptografia assimétrica em relação à simétrica que se deve armazenar a chave privada dificultando a conservação da segurança da chave.

As assinaturas digitais, como outras convencionais, podem ser forjadas. A diferença é que a assinatura digital pode ser matematicamente verificada. Dado um documento e sua assinatura digital, pode-se facilmente verificar sua integridade e autenticidade.

Propriedades da Assinatura Digital:

- Verificar o Autor e a data/hora da assinatura, sendo que as assinaturas devem ser únicas para cada usuário do sistema;
- Autenticar o conteúdo original;
- A assinatura deve poder ser verificável por terceiros (resolver disputas);
- O emissor de uma mensagem não pode invalidar a assinatura de uma mensagem. Ou seja, não pode negar o envio de uma mensagem com sua assinatura;
- O receptor da mensagem não pode modificar a assinatura contida na mensagem;
- Um usuário não pode ser capaz de retirar a assinatura de uma mensagem e colocar em outra.

O algoritmo DSA, usado na implementação da parte de autenticação, com chave pública, é o acrônimo de padrão de assinatura digital (Digital Signature Standard), criado

pelo NIST, e específica o DSA para a assinatura digital e SHA-1 para hashing. O DSA é um algoritmo assimétrico e a chave privada opera sobre o hash da mensagem SHA-1.

Para verificar a assinatura, um pedaço do código calcula o hash e outro pedaço usa chave pública para decifrar a assinatura, e por fim ambos comparam os resultados garantindo a autoria da mensagem.

Para fazer a implementação da assinatura digital, que é uma das formas e autenticação, foram utilizados o JCA(Java Cryptography Architecture) e JCE(Java Crytography Architecture) que são framework que facilitam a incorporação de funcionalidades criptográficas em aplicações onde use Java. O JCA e o JCE, no framework SeguraWeb, são as classes responsáveis por fazer a autenticação, elas geram as chaves quando o usuário se autentica no sistema. *Frameworks* permitem criar aplicações mais facilmente, sem exigir um grande conhecimento do domínio em que se aplicam. Fornecendo uma solução parcial para um problema (como neste caso) permitindo reduzir o esforço de desenvolvimento, pois apenas é necessário acrescentar a implementação correspondente a aplicação desejada.

As vantagens de se usar o JCA são:

- Independência das implementações específica e dos algoritmos, compatibilidade entre as implementações e possibilidade de extensão;
- Pertence à API base do Java e, como tal, permite escrever aplicações com técnicas criptográficas com todas as vantagens inerentes a esta linguagem (por exemplo, independência da plataforma);
- Sendo um framework permite disponibilizar novas técnicas criptográficas com o mínimo de esforços.

Uma das vantagens mencionadas acima é uma das mais importantes do JCA que torna o framework extensivo. A independência dos algoritmos significa que os utilizadores podem utilizar as diversas técnicas criptográficas independentemente do algoritmo que as implementa. Esta independência é conseguida através de classes abstratas que especificam cada uma das técnicas criptográficas. Estas classes são denominadas a Engine Class.

A JCE foi criada separadamente devido às restrições de exportação de tecnologias criptográficas dos EUA/CANADÁ.

A classe usada para geração da chave Signature é a Engine Class dedicada a operação criptográfica de assinatura digital e verificação de assinaturas.

A chave Signature utiliza uma chave privada que é gerada a partir de uma semente, que pode ser uma frase, uma palavra ou um conjunto de números, e a partir dessa semente vai ser gerada, através do framework JCA, a chave privada e uma chave pública. A chave pública é transformada em String onde vai ser armazenada numa camada de persistência que pode ser um banco de dados relaciona ou um arquivo comum. Sendo que quando o usuário for se autenticar com a chave privada, será criada a assinatura digital que vai ser verificado através da chave privada armazenada.

A camada de autenticação do RBAC usando todas esta tecnologia (JCA e JCE) explicadas acima é mostrada na figura 26:

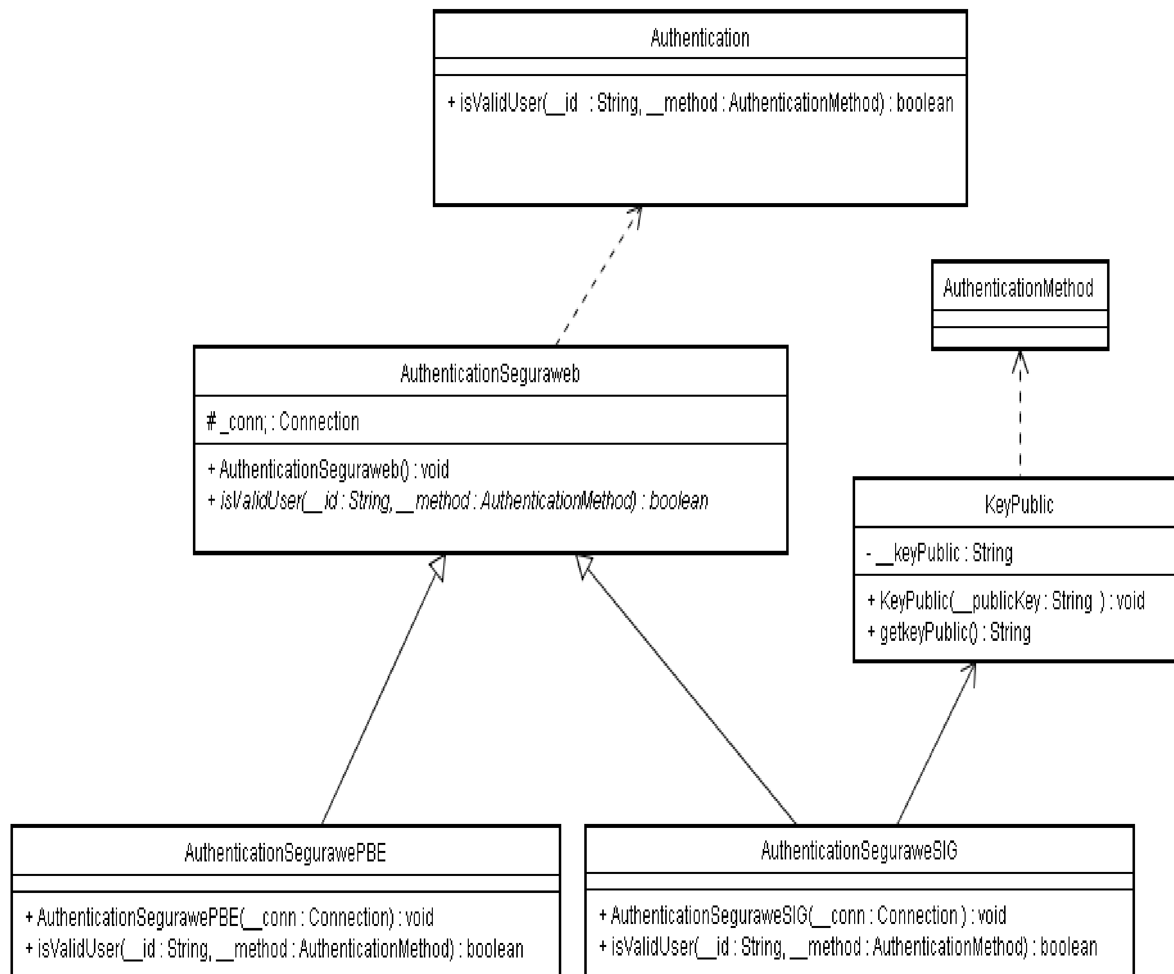

Figura 26: Autenticação através da chave pública

Esta camada, que implementa a parte de autenticação, foi expandida para que fosse mais abrangente e implementasse além da autenticação comum através de senha e chave privada a autenticação baseada em chave pública

A chave pública é gerada na classe DBUserManagerSIG classe esta que vai cadastrar novos usuários. Quando o usuário for cadastrado ele digitará uma senha que será usada como chave privada do usuário e vai servir para gerar a chave pública dele. Quando o usuário for se autenticar com sua senha fará a assinatura digital e com a chave pública que esta armazenada em um banco de dados verificará a autenticidade da assinatura.

A interface Authentication(figura 26), possui um único método, *isValidUser(id:String,auth:AuthenticationMethod)*, que serve para verificar se um determinado usuário é válido. A classe AuthenticationSeguraweb implementa a interface Authentication, esta classe AuthenticationSeguraweb ela possui um construtor e método abstrato que é *isValidUser* este método vai ser rescrito nas classes:

- AuthenticationSegurawebPBE e AuthenticationSegurawebSIG.

A classe AuthenticationSegurawebPBE implementa o método *isValidUser* ele utiliza o mecanismo de senha como método de autenticação do usuário, por ser este método o mais utilizado atualmente para aplicações Web.

A classe AuthenticationSegurawebSIG que possui o seu construtor vai também implementar o método *isValidUser* utilizando a chave privada criando uma assinatura digital e com a chave pública vai verificar a autoria, sendo que o usuário não poderá negar a sua assinatura. A vantagem do método de chave pública está no fato de o administrador o RBAC não fica com a responsabilidade de cuidar da chave privada do usuário como acontece na autenticação por senhas ou chave privadas, pois será armazenado, na camada de persistência, somente a chave pública que será de domínio público.

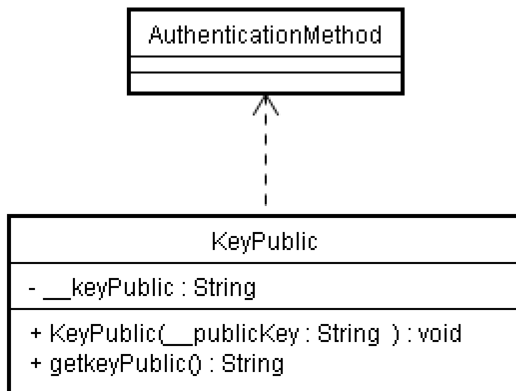

Figura 27: Classe da chave pública

A classe que implementa a interface AuthenticationMethod, possui um atributo que vai ser a chave pública do usuário. Esta classe será acessada quando for feito a autenticação do usuário a classe KeyPublic vai trazer a chave através do método
get KeyPublic().

5.4 Conclusões do Capítulo

Este capítulo apresentou conceitos sobre criptografia simétrica e assimétrica, assinatura digital e sobre explanou sobre as alterações feitas no framework SeguraWeb em relação a criptografia por chave pública que no trabalho foi desenvolvida com assinatura digital, apesar de definir a assinatura digital como assinatura por chave pública n não ser apreciado pela maioria dos autores.

A auditoria também foi mostrada e a infraestrutura para que ela aconteça foi explicada e implementada, podendo assim, os eventos do sistema serem gravados em bancos de dados ou arquivos. Foram mostradas e explicada as classes concretas criadas, sendo estas novas classe herdeiras das classe concretas e abstratas já presentes no *framework*.

Capítulo 6

Conclusões

6.1 Revisão das motivações e objetivos

O objetivo principal deste trabalho foi o estudar um *framework* para fazer o controle de acesso de aplicações com interface Web ou quaisquer outras aplicações Java utilizadas no ambiente corporativo de uma empresa e expandi-lo, implementando métodos de autenticação por chave pública. O *framework* utilizado tem como característica principal retirar a preocupação do desenvolvedor destas aplicações de implementar a parte de controle de acesso de suas aplicações.

A política de segurança do *framework* é a política baseada em papéis, pois esta é considerada atualmente como a mais adequada para utilização em um ambiente corporativo. Para tal, foram realizados estudos do modelo de segurança RBAC (*Role-Based Access Control*) do NIST. Além do estudo do modelo RBAC e do framework, foram estudados o método de chaves públicas, assim como, os seus diversos algoritmos, como o RSA e o DSA.

6.2 Visão geral do trabalho

O texto do trabalho inicialmente apresentou as características do modelo de segurança RBAC, bem como as regras definidas neste modelo e algumas características de cada modelo pertencente à família de modelos RBAC do NIST.

Depois é discutida a importância da reusabilidade de *software* e como esta pode ser obtida com a utilização de *frameworks*. Foram expostos alguns conceitos de *frameworks*, suas características principais, os tipos de classificação e o ciclo de vida. Após foram recomendadas algumas atividades que devem ser realizadas no desenvolvimento de *frameworks*, bem como as metodologias de desenvolvimento

existentes e a utilização de padrões de projeto. Foram ainda expostos aspectos de como se deve documentar um *framework*, como estudá-lo e como utilizá-lo.

Alguns trabalhos relacionados a este foram mostrados,
bem como foi feita uma breve comparação entre estes e o *framework* Seguraweb. Foram discutidas as etapas de análise e projeto do *framework* Seguraweb, como análise de domínio, o projeto de arquitetura, o projeto do *framework* (mostrando como algumas
dos modelos RBAC foram implementadas) e a criação de aplicações de teste para a
validação do *framework*.

Por fim foram estudados os meios de autenticação por chave pública (assinatura digital no caso deste trabalho) e por chave privada, as vantagens e desvantagens de cada método e descreve a maneira que os métodos de auditoria e autenticação foram implementados no *framework*.

6.3 Contribuições e escopo do trabalho

Considerando os objetivos iniciais, algumas contribuições deste trabalho podem ser citadas:

1. O estudo e utilização do *framework* SeguraWeb, verificando se o framework realmente dispõe aquilo que foi explicitado;

2. A expansão do *framework* através da implementação da autenticação por meio de assinatura digital, permitindo o uso deste tipo de autenticação;

3. Implementação de classes que provêm serviços de auditoria

6.4 Perspectivas futuras

Existem algumas possibilidades de continuidade deste trabalho. A principal delas é a evolução da estrutura do *framework*, onde poderão ser incluídas novas e diversas funcionalidades, de acordo com a necessidade dos desenvolvedores que irão utilizá-lo.

A parte de auditoria poderia ser explorada, utilizando exemplos de auditoria nos sistemas operacionais utilizados, mostrando exemplos de auditoria.

Outro trabalho que poderia ser realizado é a implementação da camada de persistência utilizando arquivos textos ou banco de dados orientados a objetos para o armazenamento das informações de segurança.

Por fim, outra sugestão, proveniente da autora do *framework* SeguraWeb é a realização de testes de importação do *framework* na versão Oracle 9i e utilização do mesmo na implementação de *stored-procedures* em Java, ou mesmo a utilização de outros bancos de dados juntamente com este *framework* [AKK02].

Referências Bibliográficas

[AKK02] ARMANINI, Kátyra Kowalski. **Seguraweb: Um Framework RBAC Para Aplicações Web.** 2002. Universidade Federal de Santa Catarina.
Dissertação submetida ao Curso de Pós-Graduação em Ciência da Computação.

[WES00] WESTPHALL, Carla Merckle. **Um Esquema de Autorização para a Segurança em Sistemas Distribuídos de Larga Escala.** 2000. Curso de Pós-Graduação em Engenharia Elétrica. Universidade Federal de Santa Catarina.

[NIBU98] **An Introduction to Role-Based Access Control.** NIST CSL Bulletin on RBAC.

[SAN94] SANDHU, Ravi; SAMARATI, Pierangela. **Access Control: Principle and Practice.** IEEE Communications Magazine, Volume 32 Issue: 9 Sept. 1994. Pg. 40-48.

Johnson, Ralph E. & Foote, Brian**.** Designing Reusable Classes. **Journal of Object-Oriented Programming.** June/July 1988, Volume 1, Number 2, pages 22-35

MELLOR, S. **Automatic code generation from UML models**. 1999**.**

Mohamed Fayad, Douglas Schmidt**. Object-Oriented Application Frameworks.** CACM Vol.40, No. 10 , Outubro **1997.**

Ralph E. Johnson**. Components, Frameworks, Patterns**. In ACM SIGSOFT Symposium on Software Reusability, 1997.

[BEZ99] BEZNOSOV, Konstantin; DENG Yi. **A Framework Implementing Role-Based Access Control Using CORBA Security Service.** 1999. Center of

Advanced Distributed Systems Engineering. School of Computer Science – Florida International University.

[KEL00] KELATH, Jayasankar. **Role Based Access Control.** Dept. of Computer Science, University of Idaho, Moscow, ID.

[OBE01] OBELHEIRO, Rafael Rodrigues. **Modelos de Segurança Baseados em Papéis para Sistemas de Larga Escala: A Proposta RBAC-JACOWEB.** 2001.Curso de Pós-Graduação em Engenharia Elétrica. Universidade Federal de Santa Catarina.

[OBE02] OBELHEIRO, Rafael Rodrigues; FRAGA, Joni S. **Role-Based Access Control for CORBA Distributed Object Systems.** 2002. Departamento de Automação e Sistemas. Universidade Federal de Santa Catarina. IEEE - Object-Oriented Real-Time Dependable Systems, 2002. (WORDS 2002). Proceedings of the Seventh International Workshop on , 2002 Pg: 53 –60.

[SHI00] SHIN, Michael E.; AHN, Gail-Joon. **UML-Based Representation of Role-Based Access Control.** ISE Department. George Mason University, USA. Enabling Technologies: Infrastructure for Collaborative Enterprises, 2000. (WET ICE2000). Proceedings. IEEE 9th International Workshops on 2000. Pg. 195-200.

[AHN01] AHN, Gail-Joon; SHIN, Michael E. **Role-Based Authorization Constraints Specification Using Object Constraint Language.** Department of Computer Science. University of North Carolina at Charlotte. Department of Information and Software Engineering. George Mason University. Enabling Technologies: Infrastructure for Collaborative Enterprises, 2001. WET ICE 2001. Proceedings. Tenth IEEE International Workshops on 2001. Pg. 157-162.

[BAR97] BARKLEY, John F.; CINCOTTA, Anthony V.; FERRAIOLO, David F.; GRAVILLA, Serban; KUHN, D. Richard. **Role-Based Access Control for the**

World Wide Web. 1997. NIST – National Institute of Standards and Technology.

[BEZ99] BEZNOSOV, Konstantin; DENG Yi. **A Framework Implementing Role-Based Access Control Using CORBA Security Service.** 1999. Center of Advanced Distributed Systems Engineering. School of Computer Science – Florida International University.

[BOS97] BOSCH, Jan; MOLIN, Peter; MATTSSON, Michael, BENGTSSON, PerOlof. **Object-Oriented Frameworks – Problems & Experiences.** 1997. Department of Computer Science and Business Administration. University of Karlskrona/Ronneby. Suécia. http://www.ipd.hk-r.se/michaelm/papers/ex-frame.ps

[BRO96] BROWN, K. WHITENACK, B. **Crossing Chasms. Pattern Languages of Program Design**. 1996. vol. 2. Reading, MA. Editora: Addison-Wesley.

[BUN01] BUNDY, Alan; BLEWITT, Alex; STARK, Ian. **Automatic Verification of Java Design Patterns.** Division of Informatics. University of Edinburg. IEEE- Automated Software Engineering, 2001. (ASE 2001). Proceedings. 16th Annual International Conference on, 2001. Pg. 324-327.

[BUS96] BUSCHMANN, Frank et al.1996.**Pattern - Oriented Software Architecture: A System of Patterns.** New York : J. Wiley.

[BUT00] BUTLER, Gregory. **Object-Oriented Application Frameworks.** Department of Computer Science. Concordia University. Montreal. http://www.cs.concordia.ca/~faculty/gregb/

[EDE98] EDEN, A. **LePUS – A Declarative Pattern Specification Language.** PhD Thesis, Department of Computer Science, TelAviv University, 1998. http://cs.concordia.ca/~faculty/eden/lepus/.

[FAY99] FAYAD, Mohamed; SCHMIDT, Douglas C. **Building Application Frameworks: Object-Oriented Foundations of Framework Design.** Setembro 1999. Editora John Wiley Professio. 1a Edição. New York.

[FER95] FERRAIOLO, David F.; CUGINI, Janet A.; KUHN, D. Richard. **Role-Based Access Control (RBAC): Features and Motivations.** 1995. U. S. Department of Commerce. NIST – National Institute of Standards and Technology.

[FER99] FERRAIOLO, David F.; BARKLEY, John F., KUHN, D. Richard. **A Role-Based Access Control Model and Reference Implementation within a Corporate Intranet.** 1999. NIST – National Institute of Standards and Technology.
[FIO99] FIORESE, Maurício. **Uma Solução na Autenticação de Usuários para Ensino à Distância.** http://www.inf.ufrgs.br/pos/SemanaAcademica/Semana99/fiorese/fiorese.html
.
[GAM95] GAMMA, Erich; HELM, Richard; JOHNSON, Ralph; VLISSIDES, John. **Design Patterns: Elements of Reusable Object-Oriented Software.** 1994. Reading: Addison-Wesley.

[GUS96] GUSTAFSSON, Mats; SHAHMEHRI, Nahid. **A Role Description Framework and its Applications to Role-Based Access Control.** Maio 1996. Laboratory for Intelligent Information Systems. Department of Computer and Information Science. Linkping University. Sweden.

[JAN98] JANSEN, W. A. **A Revised Model for Role-Based Access Control.**1998. NIST 6192.

[JOH91] JOHNSON, Ralph E.; RUSSO, Vincent F. **Reusing Object-Oriented Designs**.Maio 1991.Department of Computer Science. University of Illinois. EUA. ftp://st.cs.uiuc.edu/pub/papers/frameworks/reusable-oo-design.ps

[JOH97] JOHNSON, Ralph E. **Components, Frameworks, Patterns.** Fevereiro 1997. Disponível em: ftp://st.cs.uiuc.edu/pub/papers/frameworks/framework97.ps

[LAN95] LANDIN, Nicklas; NIKLASSON, Axel. **Development Of Object Oriented Frameworks.** 1995. Ericsson Software Technology.
[LAR00] LARMAN, Craig. **Utilizando UML e Padrões: Uma Introdução à Análise e Ao Projeto Orientado a Objetos.** 2000. Porto Alegre. Editora Bookman. 126

[MON97] MONROE, Robert T.; KOMPANEK, Andrew; MELTON, Ralph; GARLAN, David. **Architectural Styles, Design Patterns, and Objects.** Janeiro 1997. *IEEE Software,pp.43-22.*http://www-2.cs.cmu.edu/afs/cs/project/able/www/paper_abstracts/ObjPatternsArch-ieee.html
[NIBU98] **An Introduction to Role-Based Access Control.** NIST CSL Bulletin on RBAC.

[OBE01] OBELHEIRO, Rafael Rodrigues. **Modelos de Segurança Baseados em Papéis para Sistemas de Larga Escala: A Proposta RBAC-JACOWEB.** 2001. Curso de Pós-Graduação em Engenharia Elétrica. Universidade Federal de Santa Catarina.

[OBE02] OBELHEIRO, Rafael Rodrigues; FRAGA, Joni S. **Role-Based Access Control for CORBA Distributed Object Systems.** 2002. Departamento de Automação e Sistemas. Universidade Federal de Santa Catarina. IEEE - Object-Oriented Real-Time Dependable Systems, 2002. (WORDS 2002). Proceedings of the Seventh International Workshop on , 2002 Pg: 53 –60.

[OH00] OH, Sejong; PARK, Seog. **Enterprise Model as a Basis of Administration on Role-Based Access Control.** Dept. of Computer Science, Sogang University , Seoul, Korea. IEEE – Cooperative Database Systems for Advanced Applications, 2001, CODAS 2001. The Proceedings of the Third International

Symposium on 2000. Pg. 150-158.

[PAP00] PAPA, M.; BREMER, O.; CHANDIA, R.; HALE, J.; SHENOI, S. **Extending Java for Package Based Access Control.** Center for Information Security. Department of Computer Science, Keplinger Hall. University of Tulsa, Oklahoma. Computer Security Applications, 2000. ACSAC'00. 16th Annual Conference 2000. Pg. 67-76.

[RAM94] RAMOS, Alexandre Moraes. **Interface de Controle de Acesso Para o Modelo de Gerenciamento OSI.** 1994. Universidade Federal de Santa Catarina. Dissertação submetida ao Curso de Pós-Graduação em Ciência da Computação.

[SAN00] SANDHU, Ravi; FERRAIOLO, David F.; JUN, Richard. **The NIST Model for Role-Based Access Control: Towards a Unified Standard.** 2000. Proceedings of the 5th ACM Workshop on Role-Based Access Control (RABC´2000). Berlin. Alemanha.

[STA99] STALLINGS, William. **Cryptography and Network Security**: Principles and Practice. Prentice Hall, 1999. 569p.

[RSA98] RSA Data Security, Inc. "**Frequently Asked Questions about Today's Cryptography**". 1998. http://www.rsa.com.

[SCH96] SCHNEIER, Bruce. **Applied Cryptography**: Protocols, Algorithms, and Source Code in C. 2sd Edition, New York: John Wiley & Sons, 1996. 758p.

[GON01] GONG, Li, ELLISON, Gary, DAGEFORDE, Mary. **InsideJava 2 Platform Security:** Architechture API Design, anda Implemantation. 2001. http://java.sun.com/docs/books/index.html

[MAY97] MAYNARD, Jack. **Unix Security Auditing: A Pratical Guide**. Sys Adming, v. 6, n. 6, p. 67-72, Jun. 1997

[DOR02] Dora, Daniel Seleme **Tutorial sobre Auditoria de Segurança,** Jun. 2002 http://redes.ucpel.tche.br/documentos/auditoria/conteudo.html

Anexos

Código fonte

```
//Autenticacao

import java.security.*;
import java.sql.*;
import javax.crypto.*;
import javax.crypto.spec.*;
/**
 * Classe abstrata que implementa a interface de autenticação.
 * @author Daniel Quadros Da Silva, Júlio Alexandre de Albuquerque Reis
 */
public abstract class AuthenticationSeguraweb implements Authentication
{
     /**
     * Construtor.
     * @param __conn: Objeto de conexão com o banco de dados
     */
    public AuthenticationSeguraweb(Connection __conn) {
        _conn = __conn;
      /*Provider sunJCE = new com.sun.crypto.provider.SunJCE();
      Security.addProvider (sunJCE);*/
    }

    /**
    * Retorna verdadeiro se o usuário é válido, caso contrário retorna
falso.
    * @param __id: Identificador único do usuário.
    * @param __method: método de autenticação (senha).
    */
    public abstract boolean isValidUser(String __id,
AuthenticationMethod __method) {

     }
    /**
     * Conexão com o banco de dados.
     */
    protected Connection _conn;
}



import javax.crypto.*;
import javax.crypto.spec.*;
import java.security.*;
import java.sql.*;

/**
```

```
 * Classe concreta que implementa a interface de autenticação.
 * Autenticacao atraves de senha e/ou chave privada.
 * @author Daniel Quadros da Silva, Júlio Alexandre de Albuquerque Reis
 */
public class AuthenticationségurawePBE extends AuthenticationSeguraweb
{

    public AuthenticationSegurawePBE(Connection __conn) {
        super(__conn);

    }


    public boolean isValidUser(String __id, AuthenticationMethod
__method) {
      try {
        UserProxy user = new UserProxy (_conn,__id);

        PBEKeySpec pbeKeySpec;
        PBEParameterSpec pbeParamSpec;
        SecretKeyFactory keyFac;
          //PBE pbe = (PBE) __method;
          PBE pbe = new PBE
((Password)__method,"julianjt".getBytes(),20);
          //PBE pbe = new PBE
(((String)((Password)__method).getPassword()).getCharArray());

          pbeParamSpec = new
PBEParameterSpec(pbe.getSalt(),pbe.getIterationCount());
          pbeKeySpec = new
PBEKeySpec(pbe.getPassword().getPassword().toCharArray());
          keyFac = SecretKeyFactory.getInstance("PBEWithMD5AndDES");
          SecretKey pbeKey = keyFac.generateSecret(pbeKeySpec);

          Cipher pbeCipher = Cipher.getInstance("PBEWithMD5AndDES");

          pbeCipher.init(Cipher.ENCRYPT_MODE, pbeKey, pbeParamSpec);

          byte[] ciphertext =
pbeCipher.doFinal(pbe.getPassword().getPassword().getBytes());

          if ((Hexadecimal.getHexa(new
String(ciphertext))).equals(((Password)user.getAuthenticationMethod()).
getPassword())) return true;
          return false;
      }
        catch (java.security.NoSuchAlgorithmException e) {
            System.err.println (e.toString());
          return false;
        }
        catch (javax.crypto.NoSuchPaddingException e1) {
            System.err.println (e1.toString());
          return false;
        }
        catch (java.security.InvalidKeyException e2) {
            System.err.println (e2.toString());
          return false;
```

```
        }
        catch (javax.crypto.IllegalBlockSizeException e3) {
            System.err.println (e3.toString());
          return false;
        }
        catch (javax.crypto.BadPaddingException e4) {
            System.err.println (e4.toString());
          return false;
        }
        catch (java.security.spec.InvalidKeySpecException e5) {
            System.err.println (e5.toString());
          return false;
        }
        catch (java.security.InvalidAlgorithmParameterException e6) {
            System.err.println (e6.toString());
          return false;
        }
      catch (Exception e7) {
          System.err.println (e7.toString());
          return false;
      }
    }

}
```

```
import javax.crypto.*;
import javax.crypto.spec.*;
import java.security.*;
import java.sql.*;

/**
 * Classe concreta que implementa a interface de autenticação.
 * Autenticacao atraves de chave publica criando um assinatura digital.
 * @author Daniel Quadros da Silva, Júlio Alexandre de Albuquerque Reis
 */
public class AuthenticationSegurawеSIG extends AuthenticationSeguraweb
{

    public AuthenticationSegurawеSIG(Connection __conn) {
        super(__conn);

    }

   // Utiliza chave publica para assinatura digital
    public boolean isValidUser(String __id, AuthenticationMethod
__method) {
      try {

        UserProxy user = new UserProxy (_conn,__id);
        Password chavePrivada = (Password)__method;
        KeyPublic keypublica =  new KeyPublic(((Password)
user.getAuthenticationMethod()).getPassword());

        // Gerar chaves usando RSA
```

```
KeyPairGenerator kpg = KeyPairGenerator.getInstance("DSA");
SecureRandom random = new SecureRandom();
byte[] semente = chavePrivada.getPassword().getBytes();
random.setSeed(semente);
kpg.initialize(1024,random);
KeyPair kp = kpg.generateKeyPair();

// Extrair componentes privada
DSAPrivateKey  ks = (DSAPrivateKey)kp.getPrivate();
DSAParams dsaParams = ks.getParams();

//Buscar parametros
BigInteger p =  dsaParams.getP();
BigInteger q =  dsaParams.getQ();
BigInteger g =  dsaParams.getG();
BigInteger y =  new BigInteger(keypublica.getkeyPublic());

// Recuperar a chave Publica atraves dos parametros.
DSAPublicKeySpec publicKeySpec = new DSAPublicKeySpec(y,p,q,g);
KeyFactory keyFactory = KeyFactory.getInstance("DSA");
PublicKey pubKeyD = keyFactory.generatePublic(publicKeySpec);

//Assinatura da mensangens.
Signature dss = Signature.getInstance("SHA1withDSA");

// Mensagem que faram parte da assinatura.
String m1 = "Mensagem         ";
String m2 = " que eu vou assinar que vai ser assinada";


// 3. Assinar (usando chave de assinatura - ks)
dss.initSign(ks);
dss.update(m1.getBytes());
dss.update(m2.getBytes());
byte[] s = dss.sign();

// 4. Verificar (usando chave de verificação - kv)
dss.initVerify(pubKeyD);
dss.update(m1.getBytes());
dss.update(m2.getBytes());
boolean v = dss.verify(s);

return v;

}

catch (java.security.InvalidKeyException e2) {
    System.err.println (e2.toString());
  return false;
}
catch (javax.crypto.IllegalBlockSizeException e3) {
    System.err.println (e3.toString());
  return false;
}
catch (javax.crypto.BadPaddingException e4) {
    System.err.println (e4.toString());
  return false;
```

```
        }

      catch (Exception exc) {
      exc.printStackTrace();

      return false;
    }
  }
}



/**
 * Classe que utiliza o mecanismo de chave publica atraves
 * da assinatura digital, sendo guarda apenas a chave publica
 * do usuario.
 * @author Daniel Quadros da Silva, Júlio Alexandre de Albuquerque Reis
 */
public class KeyPublic implements AuthenticationMethod {
      /**
     * Construtor que cria um objeto PublicChave
     * que é passada como parâmetro.
     */
    public KeyPublic(String  __publicKey) {
        __keyPublic = __publicKey;

    }

     /**
     * Método que retorna a Chave Publica.
     */
    public String getkeyPublic() {
        return __keyPublic;
    }

    /**
     * Chave Publica.
     */
    private String __keyPublic;

}

//Administração




import java.util.Vector;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.ResultSet;
import javax.crypto.*;
import javax.crypto.spec.*;
import java.security.*;
```

```
/**
 * Classe concreta que implementa a interface de gerenciamento das
informações do usuário.
 * @author Daniel Quadros Da Silva, Júlio Alexandre de Albuquerque Reis
 */
public abstract class DBUserManager implements UserManagerInterface {

     /**
     * Construtor.
     * @param __conn: Objeto de conexão com o banco de dados
     */
    public DBUserManager(Connection __conn) {
        _conn = __conn;
        try {
         System.out.println (_conn);
           _stmt = _conn.createStatement();
        }
        catch (Exception e) {
            System.err.println (e.toString());
        }
        this.createProxy();
    }

    /**
    * Construtor.
    */
    public DBUserManager() {
        this.createProxy();
    }

    /**
    * Obtém a lista de identificadores dos usuários existentes.
    */
    public Vector getUsersOIDs() throws Exception {
        Vector oids = new Vector();
        ResultSet rs = _stmt.executeQuery("select login_use from
user_use");
        while (rs.next())
            oids.addElement(rs.getString ("login_use"));
        rs.close();
        return oids;
    }

    /**
    * Obtém o usuário cujo identificador é passado como parâmetro.
    */
    public UserInterface getUser(String __OID) {
            return new UserProxy(_conn,__OID);
    }

    /**
    * Atualiza as informações do usuário que são passadas como
parâmetro.
    * @param __user: um objeto usuário contendo as informações novas.
    */
    public void updateUser(User __user) throws Exception {
        _userProxy.updateUser (__user);
```

```
    }

    /**
    * Atualiza as informações do usuário cujo identificador é passado
como parâmetro, com as informações que são passadas como parâmetro.
      @param __id: identificador único do usuário.
      @param __fullName: o novo nome completo do usuário.
      @param __password: a nova senha do usuário.
    */
    public void updateUser(String __id,String __fullName, String
__password) throws Exception {
      User user = new User (__id,new
Password(this.getCipherPassword(__password)),__fullName);
      this.updateUser (user);
    }

    /**
    * Remove o usuário que é passado como parâmetro.
    * @param __user: o objeto referente ao usuário que deve ser
removido.
    */
    public void deleteUser(User __user) throws Exception {
        _userProxy.deleteUser(__user);
    }

    /**
    * Cadastra o usuário que é passado como parâmetro.
    * @param __user: o objeto usuário que deve ser cadastrado.
    */
    public void insertUser(User __user) throws Exception {
        _userProxy.insertUser(__user);
    }

    public void createProxy() {
        _userProxy = new UserProxy (_conn);
    }

    /**
    * Remove o usuário cujo identificador é passado como parâmetro.
    * @param __OID: identificador do usuário que deve ser removido.
    */
    public void deleteUser(String __OID) throws Exception {
        _userProxy.deleteUser(__OID);
    }

    /**
    * Cadastra um novo usuário com as informações que são passadas como
parâmetro.
    * @param __id: identificador do usuário
    * @param __fullName: nome completo do usuário
    * @param __password: senha do usuário.
    */
    public void insertUser(String __id, String __fullName, String
__password) throws Exception {
        User user = new User (__id,new
Password(this.getAut(__password)),__fullName);
        this.insertUser(user);
```

```
    }

    public abstract String getAut(String __password) throws Exception {

    }

    protected Statement _stmt;
    protected Connection _conn;
    protected UserProxy _userProxy;
}
```

```
import java.util.Vector;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.ResultSet;
import javax.crypto.*;
import javax.crypto.spec.*;
import java.security.*;

/**
 * Classe concreta que implementa a interface de gerenciamento das
informações do usuário.
 * @author Daniel Quadros Da Silva, Júlio Alexandre de Albuquerque Reis
 */
public class DBUserManagerPBE extends DBUserManager {

    /**
    * Construtor.
    */
    public DBUserManagerPBE(Connection __conn) {
        super(__conn);
    }

     /**
    * Construtor.
    */
    public DBUserManagerPBE() {
        super();
    }

    private String getAut(String __password) throws Exception {
        PBEKeySpec pbeKeySpec;
        PBEParameterSpec pbeParamSpec;
        SecretKeyFactory keyFac;
          PBE pbe = new PBE(new
Password(__password),"julianjt".getBytes(),20);

          pbeParamSpec = new
PBEParameterSpec(pbe.getSalt(),pbe.getIterationCount());
          pbeKeySpec = new
PBEKeySpec(pbe.getPassword().getPassword().toCharArray());
          keyFac = SecretKeyFactory.getInstance("PBEWithMD5AndDES");
          SecretKey pbeKey = keyFac.generateSecret(pbeKeySpec);
```

```
          Cipher pbeCipher = Cipher.getInstance("PBEWithMD5AndDES");

          pbeCipher.init(Cipher.ENCRYPT_MODE, pbeKey, pbeParamSpec);

          byte[] ciphertext =
pbeCipher.doFinal(pbe.getPassword().getPassword().getBytes());

          return Hexadecimal.getHexa(new String(ciphertext));
    }


}
```

```
import java.math.*;
import java.util.Vector;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.ResultSet;
import javax.crypto.*;
import javax.crypto.spec.*;
import java.security.*;
import java.security.spec.*;
import java.security.interfaces.*;

/**
 * Classe concreta que implementa a interface de gerenciamento das
informações do usuário.
 * faz o gerenciamento e cria a chave publica para fazer assinatura
 * no login.
 * @author Daniel Quadros da Silva, Júlio Alexandre de Albuquerque Reis
 */
public class DBUserManagerSIG extends DBUserManager {

      /**
    * Construtor.
    */
    public DBUserManagerSIG(Connection __conn) {
        super(__conn);
    }

     /**
    * Construtor.
    */
    public DBUserManagerSIG() {
        super();
    }
```

```
      /**
    * Cadastra um novo usuário com as informações que são passadas como
parâmetro.
    * @param __id: identificador do usuário
    * @param __fullName: nome completo do usuário
    * @param __password: senha do usuário.
    */

    private String getAut(String __semente) throws Exception {

      // Gerar chaves usando DSA
      KeyPairGenerator kpg = KeyPairGenerator.getInstance("DSA");
      SecureRandom random = new SecureRandom();
      // Sera a semente que vai gerar as chaves publicas e privada

      byte[] semente = __semente.getBytes();
      random.setSeed(semente);
      kpg.initialize(1024,random);
      KeyPair kp = kpg.generateKeyPair();

      // Extrair componentes publica e privada
      DSAPublicKey   kv = (DSAPublicKey)kp.getPublic();
      DSAPrivateKey  ks = (DSAPrivateKey)kp.getPrivate();

      // Extraindo os paramentros da chave privada e recuperando a
chave publica
      DSAParams dsaParams = ks.getParams();
      BigInteger pi = dsaParams.getP();
      BigInteger qi = dsaParams.getQ();
      BigInteger gi = dsaParams.getG();
      BigInteger KeyPublic = kv.getY(); // Extraindo a chave publica

       // Transformando de BigInteger para String para guardar a chave
no Banco de dados.
        String ParamPublic =  KeyPublic.toString();


       // Retorna um array de String guardando todos os paramentros
        return ParamPublic;
    }

}

//Auditoria


/**
 * Interface referente ao elemento usuário do modelo RBAC.
 */
public interface LogsInterface {
    /**
    * Retorna o tipo do evento.
    */
    String getEvent() throws Exception;

    /**
    * Retorna um identificador do evento relevante.
```

```
    */
    String getLogID() throws Exception;

    /**
    * Retorna o uma observacao a mais do evento.
    */
    String getObs() throws Exception;


}



import java.util.*;
/**
 * Classe concreta que implementa a interface de gerenciamento das
informações do usuário.
 * @author Daniel Quadros Da Silva, Júlio Alexandre de Albuquerque Reis
 */
public class DBLogs implements LogsInterface {


      public DBLogs(){

      }

      public setLog(String __event,String _logID,String __Obs ){

            __event = _event;
            __logID = _logID;
            __Obs   = _Obs;
      }
    /**
    * Retorna o tipo do evento.
    */
    public String getEvent() {
      return __Event
    }

    /**
    * Retorna um identificador do evento relevante.
    */
    public String getLogID() {
      return __logID
    }


    public String getObs() {
      return __Obs
    }

       public String getDate() {

       Date dia = new Date();
      String dialog  =dia.toString();
       Return dialog  ;
    }
```

```
    protected String __Event;

    protected String __logID;

    protected String __Obs;

}
```

```
import java.sql.*;
/**
    *Classe  concreta  que inserir os logs.
  * @author Daniel Quadros Da Silva, Júlio Alexandre de Albuquerque Reis
    */
public class RelationalBrokerDBlogs {

    public RelationalBrokerUser() {

      try {
                DriverManager.registerDriver(new
oracle.jdbc.driver.OracleDriver());
                _conn =
DriverManager.getConnection("jdbc:oracle:thin:@192.168.1.139:1521:thsea
rch","katyra","jdmep94i");
                _stmt = _conn.createStatement();
        }
      catch (Exception e) {  // (ClassNotFoundException and
SQLException)
                System.err.println (e.toString());
      }
    }


    public void insertDBlogs (DBLogs __dblogs) throws Exception {
        String sql = "insert into log_events_log
(event,LogID,Obs,Datalogs ) values
('"+__dblogs.getEvent()+"','"+__dblogs.getLogID()+"','__dblogs.getdate'
,)";
        int lines = _stmt.executeUpdate (sql);

      }


    public Vector getLogID(String __type) throws Exception {
        Vector oids = new Vector();
        ResultSet rs = _stmt.executeQuery("select LogID from
log_events_log");
        while (rs.next())
            oids.addElement(rs.getString ("LogID"));
        rs.close();
        return oids;
```

```
    }


    public void deleteDBlogs(DBLogs __dblogs) throws Exception {

        String sql = "delete from log_events_log where LogID
='"+__dblogs.getLogID()+"'";
        int lines = _stmt.executeUpdate (sql);


     }

    protected Connection _conn;
    protected Statement _stmt;
    protected ResultSet _rs;
}
```

```
import java.util.*;
import java.io.*;


public class FileLogs implements LogsInterface {

      public FileLogs(){

      }

      public setLog(String __event,String _logID,String __Obs ){

            __event = _event;
            __logID = _logID;
            __Obs   = _Obs;
      }
    /**
    * Retorna o tipo do evento.
    */
    public String getEvent() {
      return __Event
    }

    /**
    * Retorna um identificador do evento relevante.
    */
    public String getLogID() {
      return __logID
    }


    public String getObs() {
      return __Obs
    }
```

```
    public insertLogs(){

     String logs;
     Date dia = new Date();
     logs = this.getEvent()+" "+this.getLogID()+" "+this.getObs();
     logs = logs +" "+= dia.toString();
     File = new File("c:\");
     FileOutputStream NewFile = FileOutputStream(File);
     NewFile.write(logs.getBytes();

    }

    protected String __Event;

    protected String __logID;

    protected String __Obs;


}



//TESTE


import org.omg.CORBA.*;
import java.sql.*;

public class SecServerImpl
    extends SecurityServer.SecServerPOA
{

    private Connection _conn;

    public SecServerImpl(Connection __conn)
    {
      _conn = __conn;
    }

    public boolean isValidUser(java.lang.String username,
java.lang.String password) {
       try {
           AuthenticationSegurawesIG  auth = new
AuthenticationSegurawesIG  (_conn);
            boolean isvalid = auth.isValidUser(username,new
Password(password));
           DBLogs  logs = DBLogs();
           logs
.setLog("autenticacao","username",isValidUser.isvalid.toString();
           return isvalid;
        }

        catch (Exception e) {
           e.printStackTrace();
         return false;
        }
```

```
    }

   public String getServerIOR (String serverId) {
      return null;
   }

   boolean setServerIOR (String serverId, org.omg.CORBA.StringHolder
serverIOR) {
      return false;
   }

   public boolean hasPermissionTo (String username, String permission)
{
      return false;
   }

}
```