Marcel de Castilho

QEEWeb: Um Sistema de Monitoramento de Qualidade de Energia Elétrica Utilizando o Framework Seguraweb

Florianópolis

2004

Marcel de Castilho

QEEWeb: Um Sistema de Monitoramento de Qualidade de Energia Elétrica Utilizando o Framework Seguraweb

Monografia apresentada à Universidade Federal de Santa Catarina, como parte dos requisitos para a obtenção do grau de Bacharel em Ciências da Computação.

Orientador:
Prof. Carlos Becker Westphall, Dr.

Universidade Federal de Santa Catarina

Florianópolis

2004

Monografia sob o título *"QEEWeb: Um Sistema de Monitoramento de Qualidade de Energia Elétrica Utilizando o Framework Seguraweb"* defendida por Marcel de Castilho e aprovada em 13 de fevereiro de 2004, em Florianópolis, Santa Catarina, pela banca examinadora constituída pelos doutores:

Prof. Carlos Becker Westphall, Dr.
Orientador

Prof. Carla Merkle Westphall, Dr.
Universidade Federal de Santa Catarina

Prof. Mario Dantas, Dr.
Universidade Federal de Santa Catarina

Aos meus pais, Celso e Marie

Agradecimentos

Agradeço ao professor Carlos Becker Westphall pela oportunidade concedida e pela confiança depositada.

Aos professores Carla Merkle Westphall e Mario Dantas pela participação na banca examinadora e pelo interesse neste trabalho.

Aos grandes companheiros e amigos da turma 00.1: Crineu Tres, Igor Tibes, Júlio Reis, Luis Fernando Jordan e Roberto Hartke Neto que estiveram sempre presentes ao longo do curso.

Ao inúmeros amigos que transformaram a Airgate em uma segunda família: Alex Mariano, André Carlucci, Cleidson Cavalcante, Daniel Terzella, Danilo Barbosa, Diego Ramos, Ivan Schuster, Roberta Cavalcanti, Ronaldo Pontes, Rubem dias e Sheila Ferreira.

À minha família pela compreensão, pelo incentivo e pelo carinho.

À Caro pela paciência e pelo amor.

À Deus que colocou todas estas pessoas no meu caminho.

“Todo trabalho importante - deves ter sentido em ti mesmo - exerce uma influência moral. O esforço para concentrar uma determinada matéria e dar-lhe uma forma harmoniosa, eu o comparo a uma pedra atirada em nossa vida interior: o primeiro círculo é estreito, mas amplos se destacam.”

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

ACL	Access Control List
API	Application Program Interface
ASP	Active Server Pages
CORBA	Common Object Request Broker Architecture
DAC	Discretionary Access Control
DCOM	Distributed Common Object
DSD	Dynamic Separation of Duty
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronics Engineer
IIS	Internet Information Server
IP	Internet Protocol
JSP	Java Server Pages
JVM	Java Virtual Machine
MAC	Mandatory Access Control
NIST	National Institute of Standards and Technology
OMG	Object Management Group
PBE	Password-Based Encryption
PHP	Hypertext Preprocessor
PPK	Password and Private Key
RBAC	Role-Based Access Control
RMI	Remote Method Invocation
SSD	Static Separation of Duty
TCP	Transmission Control Protocol

Resumo

As empresas de energia elétrica têm procurado implementar sistemas que permitam o diagnóstico e controle de qualidade de energia elétrica como uma forma de obter uma vantagem competitiva no mercado. O uso cada vez mais freqüente pelos consumidores industriais de equipamentos eletronicamente controlados e processos automatizados tem sofrido com o efeito da má qualidade de energia fornecida pelas distribuidoras. Tais equipamentos são extremamente sensíveis aos inerentes eventos de qualidade de energia elétrica, como as perturbações de curta e longa duração. Este trabalho apresenta a criação do QEEWEb: um sistema de monitoramento de qualidade de energia elétrica utilizando o framework Seguraweb. O Seguraweb é um framework orientado a objetos, baseado no modelo de segurança RBAC, que visa reduzir o tempo de desenvolvimento de aplicações Web, retirando dos desenvolvedores a preocupação com a implementação dos mecanismos de segurança, principalmente no que se refere ao controle de acesso.

Palavras-chave: monitoramento de qualidade de energia elétrica, controle de acesso baseado em papéis, frameworks

Abstract

The energy companies have implemented system that allow the diagnostic and control of power quality as an competitive advantage in market. The increasing use by the industrial consumers of electronic controlled equipments and automated processes have suffered with the effect of bad power quality provided by the distribution companies. This equipments are extremely sensitive to the inherent events of power quality, such as short and long duration variations. This work presents the creation of the QEEWeb: a power quality monitoring system using the Seguraweb framework. The Seguraweb is an object-oriented framework, based on the RBAC model, that aims to reduce the development time of Web applications, because the developers do not need to concern about the implementation of security mechanisms, mainly access control.

key-words: power quality monitoring, role-based access control, frameworks

1 Introdução

Neste capítulo são apresentados o contexto no qual se insere este trabalho, as justificativas que motivaram sua realização, o objetivo proposto, os métodos de pesquisa utilizados e, por fim, a forma de organização deste trabalho.

1.1 Contexto

A qualidade de fornecimento de energia elétrica[1] tem ocupado cada vez mais importância, tanto para as distribuidoras de energia elétrica quanto para os consumidores dessa energia. As distribuidoras têm procurado implementar sistemas que permitem o diagnóstico e controle de qualidade de energia, como uma forma de obter uma vantagem competitiva no mercado de energia elétrica.

Os processos industriais modernos cada vez mais utilizam equipamentos eletronicamente controlados e processos automatizados, com o objetivo de atingir as metas de produtividade e qualidade e competir na economia globalizada. Por outro lado, observa-se que tais equipamentos são extremamente sensíveis aos fenômenos associados à qualidade de energia, trazendo grandes prejuízos devido as interrupções de seus processos, perdas de produção, perdas de insumos e custos associados à mão-de-obra e a reparo de equipamentos. "Os dispositivos e equipamentos que vêm sendo utilizados atualmente nos sistemas de transmissão são mais sensíveis às flutuações de energia que os dispositivos utilizados nas décadas passadas. Alguns eventos que no passado ocorriam nos sistemas de energia elétrica e não causavam efeitos, atualmente resultam em desligamentos dos equipamentos" (MELO; CAVALCANTI, 2003, p. 477).

Embora a legislação do setor elétrico brasileiro não disponha de dispositivo jurídico que trate da maioria dos critérios de qualidade de energia, as distribuidoras vêm sofrendo

[1]Define-se que energia elétrica de boa qualidade, é aquela que garante o funcionamento contínuo, seguro e adequado dos equipamentos elétricos e processos associados, sem afetar o meio ambiente e o bem estar das pessoas.

desgastes na sua imagem, além dos custos com pedidos de ressarcimento de prejuízos sofridos por consumidores, na ordem de milhões de dólares, decorrentes da má qualidade de energia.

Uma alternativa para a implementação de um sistema de monitoração de qualidade de energia elétrica é o uso da Web como o mecanismo primário para trocar dados entre a aplicação e o usuário, a partir de um navegador Web (*Web browser*). Desde o nascimento da *World Wide Web*, os navegadores Web têm se tornado um padrão no acesso de dados pela Internet, em diferentes plataformas. Quando a aplicação é baseada na Web, não é preciso instalar nenhum software adicional na máquina do usuário, e uma vez que uma nova versão da aplicação é desenvolvida e atualizada no servidor, os usuários automaticamente começam a utilizar essa nova versão, sem a necessidade de atualização local.

O crescimento do uso de aplicações Web dentro das empresas é conseqüência, em grande parte, da popularização das Intranets no ambiente dessas empresas. A Intranet, assim como a Internet, é definida como uma rede de computadores baseada no conjunto de protocolos TCP/IP. No entanto, as Intranets pertencem a uma organização, geralmente uma empresa, e são acessíveis apenas por membros desta organização, funcionários, ou outras pessoas com autorização. Como as redes locais são hoje parte integrante da maioria das empresas, tanto a infra-estrutura de rede quanto a experiência no seu uso já se encontram instaladas. Assim, o uso de softwares baseados na Web se torna inevitável.

Por outro lado, "um dos grandes obstáculos no crescimento de Intranets é a inabilidade de gerenciar efetivamente a autorização às informações. As formas de controle de acesso existentes são custosas e propensas a erro" (FERRAIOLO; BARKLEY, 1997, p. 1). De fato, com o crescimento das empresas, ocorreu um aumento do volume de informação e do número de funcionários e, conseqüentemente, os problemas de segurança cresceram e se tornaram mais difíceis.

Nesse contexto, o modelo de segurança RBAC (*Role-Based Access Control*) representa uma tecnologia que tem chamado muita atenção, devido ao seu potencial na redução de complexidade e custo no gerenciamento de autorização em sistemas de grande porte. No modelo RBAC, o acesso de usuários a recursos computacionais é baseado na construção de papéis. Tais papéis definem um conjunto de atividades concedidas para usuários autorizados. Pode-se imaginar um papel como se fosse um cargo ou posição dentro de uma organização, que representa a autoridade necessária para conduzir as tarefas associadas. Segundo Ferraiolo e Barkley (1997, p. 1–2), o modelo RBAC permite aos administrado-

res especificar políticas de segurança complexas, que são geralmente impraticáveis ou até mesmo impossíveis de serem impostas através de outros modelos de controle de acesso.

1.2 Motivação

O Seguraweb é um framework orientado a objetos baseado no modelo de segurança RBAC. Como todo framework, o objetivo do Seguraweb é promover a reutilização de software como uma forma de aumentar a qualidade e a produtividade da atividade de desenvolvimento de software. O Seguraweb se encarrega da maioria dos mecanismos de segurança da aplicação, principalmente no que se refere ao controle de acesso, no qual é usado o modelo RBAC. Desta forma, toda a responsabilidade de implementar mecanismos de segurança é retirada dos desenvolvedores das aplicações que podem, assim, concentrar todas as suas atenções na solução do problema.

Uma das etapas do ciclo de desenvolvimento de frameworks, é a realização de testes, também chamada de manutenção do framework. Essa etapa visa determinar se o framework atende a todas as funcionalidades para o qual foi projetado, além de avaliar sua usabilidade.

De acordo com Silva (2000, p. 48–49), a finalidade básica de um framework é ser reutilizado na produção de diferentes aplicações, minimizando o tempo e esforço para isso. A construção de um framework é sempre precedida por um procedimento de análise e domínio em que são buscadas informações do domínio tratado. Porém, como um framework nunca é a descrição completa de um domínio, é possível que a construção de aplicações sob um framework leve à obtenção de novos conhecimentos do domínio tratado, indisponíveis durante a sua construção.

1.3 Objetivo

O objetivo principal deste trabalho é avaliar a usabilidade do framework Seguraweb através de um sistema de monitoramento de qualidade de energia elétrica via Internet, o QEEWeb.

1.3.1 Objetivos Específicos

Os objetivos específicos deste trabalho que podem ser citados são:

- Conhecer o modelo de segurança RBAC e seus aspectos mais importantes;
- Conhecer as diversas formas de reutilização de software, principalmente frameworks orientados a objetos;
- Conhecer o framework Seguraweb e aprender como utilizá-lo na construção de aplicações Web;
- Conhecer os principais fenômenos relacionados à qualidade de energia elétrica;
- Aprofundar o conhecimento da linguagem Java e da plataforma J2EE como base para a implementação do QEEWeb;
- Apresentar uma solução automatizada de gerenciamento e coleta de dados de qualidade de energia elétrica utilizando as tecnologias estudadas.

1.4 Metodologia de Pesquisa

A fim de atingir o objetivo proposto por este trabalho, faz-se uso de uma abordagem de pesquisa documental, que consiste no estudo de qualquer base de conhecimento acessível que esteja focada em metodologias de implementação de software, comparação de tecnologias e exemplificações. A maior fonte de pesquisa deste trabalho foram dissertações e artigos encontrados na Biblioteca Central da UFSC e na Internet.

1.5 Organização do Trabalho

Este trabalho está organizado em seis capítulos, cujos conteúdos são descritos a seguir.

O capítulo 2 apresenta uma visão geral do modelo de segurança RBAC, em especial o modelo unificado RBAC-NIST e suas características fundamentais.

O capítulo 3 apresenta o conceito de frameworks, as etapas de desenvolvimento do framework Seguraweb e como utilizá-lo na construção de aplicações Web.

O capítulo 4 apresenta a problemática da qualidade de energia elétrica em sistemas de distribuição e uma classificação dos fenômenos perturbadores de onda.

O capítulo 5 apresenta os aspectos relacionados à implementação do QEEWeb, os testes e resultados obtidos.

Finalmente, o capítulo 6 apresenta as conclusões e perspectivas de continuidade deste trabalho.

2 O Modelo de Segurança RBAC

O termo *Role-Based Access Control* (RBAC) é utilizado para descrever mecanismos de segurança que controlam o acesso de usuários a recursos computacionais, baseados na atribuição de papéis que usuários podem ter como parte de uma organização. Os usuários assumem papéis baseados nas suas responsabilidades e qualificações dentro da empresa, e todas as operações que eles têm permissão de executar são baseadas nesses papéis. Novos papéis podem ser facilmente atribuídos aos usuários e novas operações podem ser concedidas ou revogadas dos papéis, sem prejudicar a estrutura básica de acesso. Dessa forma, o modelo RBAC fornece um modo bastante intuitivo e eficaz de representar e gerenciar autorizações às informações.

Ferraiolo e Barkley (1997, p. 1) apontam como um dos grandes obstáculos no crescimento de Intranets a inabilidade de gerenciar efetivamente a autorização às informações. As formas de controle de acesso existentes são custosas e propensas a erro. Uma delas, as ACLs[1] (*Access Control Lists*), especificam para cada recurso protegido, uma lista de usuários ou grupo de usuários com seus respectivos modos de acesso a esse recurso. O uso de ACLs é problemático por uma variedade de razões. Por exemplo, supondo que um usuário receba novas responsabilidades ou mude de cargo dentro da empresa, essas mudanças ocasionaram uma revisão total de todos os privilégios do usuário sobre os recursos. Essa é uma tarefa trabalhosa e que consome tempo, principalmente em grandes organizações onde a quantidade de usuários é imensa. Sem a habilidade de executar uma revisão em cada usuário, a empresa corre o risco de manter direitos de acesso residuais inapropriados. Pelo contrário, no modelo RBAC, quando um usuário é membro de um papel, apenas os privilégios necessários para executar as funções correspondentes a esse papel são dadas a ele. Essa característica, chamada de princípio de mínimo privilégio, requer identificar as funções do usuário, determinar os mínimos privilégios necessários para executar as funções, e restringir o usuário a um domínio com esses privilégios e nada

[1]Existe uma classificação quanto as classes de modelos de controle de acesso, na qual resulta três tipos fundamentais: *Discretionary Access Control* (DAC), *Mandatory Access Control* (MAC) e *Role-Based Acess Control* (RBAC). As ACLs são são do tipo DAC.

mais. Para realizar uma mudança de cargo de um usuário na empresa, basta retirar as associações do usuário com os papéis atuais e associar novos papéis apropriados para o novo cargo. A independência lógica do modelo RBAC faz com que os administradores do sistema controlem o acesso em um nível de abstração que é natural e se aproxima bastante à forma que as empresas tipicamente conduzem seus negócios.

Nos últimos anos, segundo Ferraiolo et al. (2001, p. 225), os grandes fabricantes de software começaram a implementar as características do modelo RBAC no gerenciamento de banco de dados, gerenciamento de segurança e sistemas operacionais, sem nenhum acordo geral sobre um conjunto de características apropriadas do RBAC. "Alguns usuários e fabricantes de software reconheceram os benefícios potenciais do RBAC sem uma definição precisa do que o RBAC constituia" (FERRAIOLO; CUGINI; KUHN, 1995, p. 1). Embora os modelos RBAC implementados fossem relativamente similares nos conceitos RBAC fundamentais, eles se diferenciavam em detalhes significativos, uma vez que os modelos usavam terminologias diferentes para descrever os mesmos conceitos. Para amenizar os problemas de escopo e terminologia, o NIST (*National Institute of Standards and Technology*) propôs um padrão RBAC composto pelo Modelo de Referência RBAC (*RBAC Reference Model*), que define um conjunto de elementos RBAC básicos (usuários, papéis, permissões e objetos) e suas relações; e pela Especificação Funcional RBAC (*RBAC Functional Especification*), que define um conjunto de requisitos operacionais e funcionais de um sistema RBAC. O modelo unificado RBAC-NIST tem como objetivo validar o uso e a flexibilidade do RBAC.

Este capítulo apresenta uma visão geral do padrão RBAC-NIST, dando ênfase ao Modelo de Referência RBAC, pois é ele que define um vocabulário comum dos termos a serem usados na especificação dos requisitos e na definição do escopo das características RBAC incluídas nesse padrão. A Especificação Funcional RBAC, define requisitos sobre operações administrativas para criação e manutenção de elementos e relações; funções de revisões administrativas para executar *queries* administrativas; e funções de sistema para criar e manter atributo em sessões de usuários e fazer decisões de controle de acesso. A Especificação Funcional RBAC é mais voltada para a implementação do modelo RBAC e não é abordada neste trabalho.

Tanto o Modelo de Referência RBAC quanto a Especificação Funcional RBAC são organizados em quatro componentes:

- **RBAC Básico (*Core RBAC*):** define um conjunto mínimo de elementos e relações entre elementos, tais como as relações usuário-papel e permissão-papel, que são in-

dispensáveis em qualquer sistema RBAC;

- **RBAC Hierárquico (*Hierarchical RBAC*):** adiciona ao modelo RBAC relações de herança entre papéis;
- **Relações de Separação Estática de Tarefas (*Static Separation of Duty Relations*):** adiciona ao modelo RBAC relações de exclusividade entre papéis e usuários;
- **Relações de Separação Dinâmica de Tarefas (*Dynamic Separation of Duty Relations*):** adiciona ao modelo RBAC relações de exclusividade entre papéis ativados em sessões de usuários.

Com exceção do RBAC Básico, todos os componentes são independentes entre si e podem ser implementados separadamente.

2.1 RBAC Básico

O RBAC Básico (*Core RBAC*) incorpora os aspectos essenciais do modelo RBAC e é formado por um conjunto de cinco elementos fundamentais: usuários, papéis, permissões, operações e objetos. Um usuário é definido como uma pessoa, um papel é um conjunto de funções e responsabilidades atribuídas a um usuário, uma permissão é uma aprovação para executar uma operação em um ou mais objetos protegidos, uma operação é uma seqüência de comandos que executam uma função para o usuário e, por fim, um objeto é uma entidade que contém ou recebe uma informação. Percebe-se que operações e objetos dependem do tipo de sistema RBAC no qual são implementados. Por exemplo, em um sistema de arquivos, as operações poderiam ser ler, escrever e executar, e os objetos arquivos ou diretórios em um sistema operacional. Já em um sistema de gerenciamento de banco de dados, as operações poderiam ser inserir, atualizar e remover, e os objetos linhas, colunas, tabelas ou visões. A figura 1 ilustra o RBAC Básico.

De acordo com Ferraiolo et al. (2001, p. 228), o conceito básico do RBAC é que usuários são associados a papéis, permissões são associadas a papéis, e usuários adquirem permissões por serem membros de papéis. O RBAC Básico exige que relações usuário-papel e permissão-papel possam ser de muitos-pra-muitos. Desse modo, o mesmo usuário pode ser membro de muitos papéis e um único papel pode possuir muitos usuários. De forma similar, para permissões, uma única permissão pode ser atribuída a muitos papéis e um único papel pode possuir inúmeras permissões.

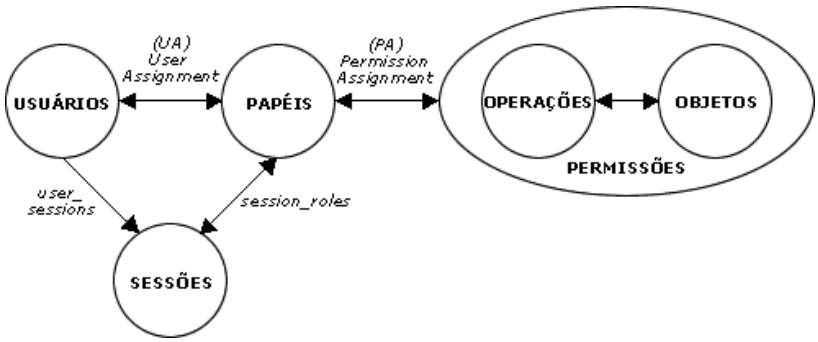

Figura 1: RBAC Básico

É importante destacar que os usuários podem utilizar permissões em múltiplos papéis simultaneamente. Isso evita que haja uma restrição em que usuários tenham que ativar um papel por vez. O conceito de sessão permite ativação e desativação seletiva de papéis. Uma sessão é definida como um mapeamento entre um usuário e um subconjunto de papéis associados ao usuário.

Finalmente, o RBAC Básico exige à revisão usuário-papel, que permite determinar, de maneira eficiente, quais os usuários associado a um papel e a quais papéis um usuário está associado. Uma exigência similar pode ser imposta na revisão permissão-papel, entretanto, considerando a dificuldade intríseca de se implementar esse mecanismo, a revisão permissão-papel é imposta apenas como uma função de revisão avançada.

2.2 RBAC Hierárquico

De acordo com Ferraiolo et al. (2001, p. 229), papéis podem ter sobreposição de responsabilidades, ou seja, permissões comuns podem ser adquiridas por usuários pertencentes a diferentes papéis. Além disso, em muitas empresas existem operações gerais que são executadas por um grande número de usuários. Como tal, se mostra ineficiente e administrativamente enfadonho especificar essas operações gerais. Para melhorar a eficiência e proporcionar uma estrutura organizacional, o modelo RBAC inclui o conceito de hierarquia de papéis. O RBAC Hierárquico (*Hierarchical RBAC*), ilustrado na figura 2, adiciona hierarquia de papéis ao RBAC Básico e oferece uma forma natural de estruturar papéis, autorizações e responsabilidades em uma organização.

A hierarquia de papéis é basicamente uma ordem parcial definindo uma relação de herança entre papéis. Essa relação possibilita que papéis de maior nível na hierarquia adquiram permissões atribuídas aos papéis de menor nível. Segundo Obelheiro (2001, p. 34),

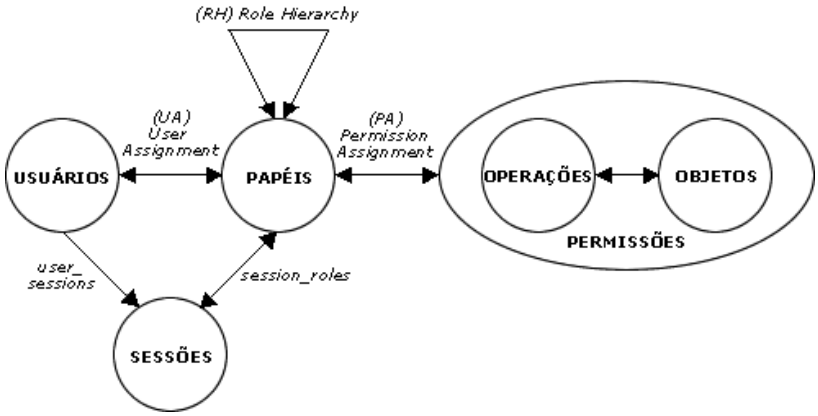

Figura 2: RBAC Hierárquico

embora hierarquias arbitrárias estejam mais próximas da realidade, a larga utilização de hierarquias limitadas levou a uma subdivisão do RBAC Hierárquico:

- **RBAC Hierárquico Geral:** uma hierarquia de papéis pode constituir qualquer tipo de ordem parcial;
- **RBAC Hierárquico Limitado:** existe algum tipo de restrição imposta na hierarquia de papéis. Neste caso, a hierarquia é limitada a estruturas simples como árvores ou árvores invertidas e não existe herança múltipla de papéis.

A figura 3 mostra um exemplo de hierarquia de papéis. Nesta hierarquia, os papéis mais privilegiados, como o papel Gerente, ocupam posições mais altas na árvore. Isso significa que os usuários associados ao papel Gerente herdam todas as permissões atribuídas aos papéis Caixa e Contador, sem a necessidade de listar explicitamente essas permissões para o papel Gerente. Por sua vez, os papéis Caixa e Contador herdam as permissões do papel Atendente, localizado abaixo deles na árvore, que indiretamente também são herdadas pelo papel Gerente . Isso é possível graças ao conceito de herança múltipla, que permite compor um papel a partir de outros papéis subordinados. Observa-se que nem todos os papéis precisam estar necessariamente relacionados. Neste mesmo exemplo, os papéis Caixa e Contador não estão hierarquicamente relacionados, apesar de compartilharem o papel Atendente.

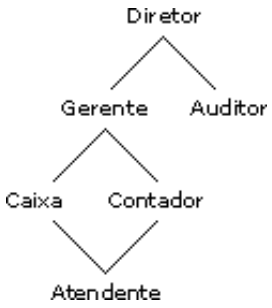

Figura 3: Exemplo de hierarquia de papéis

2.3 RBAC com Restrições

O RBAC com Restrições (*Constrained RBAC*) adiciona relações de separação de tarefas ao modelo RBAC. No modelo unificado RBAC-NIST, as relações de separação de tarefas podem ser de dois tipos: estáticas e dinâmicas.

2.3.1 Relações de Separação Estática de Tarefas

A separação de tarefas é usada para reforçar as políticas de conflito de interesses, isto é, impedir que usuários adquiram permissões associadas a papéis conflitantes. Uma forma de prevenir essa forma de conflito é através das Relações de Separação Estáticas de Tarefas (*Static Separation of Duty Relations* ou *SSD Relations*), que impõem restrições na associação entre usuários e papéis. Nessa abordagem, usuários associados a um papel não podem ser associados a um segundo papel, de acordo com restrições definidas pelo administrador de segurança. Por exemplo, suponha que um usuário esteja associado ao papel Caixa em um banco. Pode ser interessante para esse banco que tal usuário não esteja associado ao papel Auditor simultaneamente. Desta forma, os papéis Caixa e Auditor são ditos mutuamente exclusivos.

Outra restrição, que pode ser implementada, se refere à limitação numérica de usuários em um determinado papel, ou cardinalidade. A cardinalidade indica o número máximo de usuários que podem ocupar um papel em um determinado momento. Por exemplo, considere o papel Gerente. Embora outros empregados possam exercer esse papel, apenas um empregado pode assumir as responsabilidades do Gerente de cada vez.

Devido às potenciais inconsistências entre relações de separação estática de tarefas e relações de herança da hierarquia de papéis, o RBAC-NIST define separação estática de tarefas na presença e na ausência de hierarquia de papéis:

- **Separação Estática de Tarefas:** uma associação entre um usuário e um papel pode impedir que esse usuário se associe a outro papel;
- **Separação Estática de Tarefas na Presença de Hierarquia:** funciona de forma semelhante à separação estática de tarefas básica, porém, tanto os papéis herdados quanto os papéis diretamente associados ao usuário são considerados na imposição de restrições, como ilustrado na figura 4.

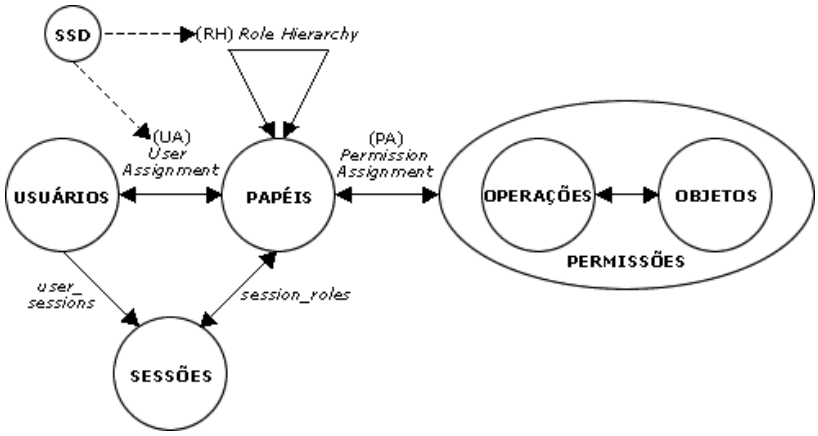

Figura 4: Separação Estática de Tarefas na Presença de Hierarquia

2.3.2 Relações de Separação Dinâmica de Tarefas

As Relações de Separação Dinâmica de Tarefas (*Dynamic Separation of Duty Relations* ou *DSD Relations*), como as Relações Estáticas de Separação de Tarefas, limitam as permissões disponíveis a um usuário. No entanto, as Relações de Separação Dinâmica de Tarefas se diferem das Relações Estáticas de Separação de Tarefas pelo contexto nas quais essas limitações são inseridas. Como pode ser visto na figura 5, as restrições são impostas aos papéis que podem ser ativados dentro de uma sessão do usuário. Essa característica agrega o fator tempo às políticas de conflito de interesses, pois permite que um usuário assuma papéis conflitantes, desde que esses papéis sejam ativados em sessões diferentes.

Outra característica das Relações de Separação Dinâmica de Tarefas é chamada de anulação de confiança na hora certa (*timely revocation of trust*), que é a extensão do princípio do mínimo privilégio: cada usuário pode ter diferentes níveis de privilégio em momentos distintos, dependendo dos papéis assumidos por ele e ativados em sua sessão. Isso assegura que as permissões não persistam por mais tempo do que o necessário para o usuário executar uma determinada tarefa.

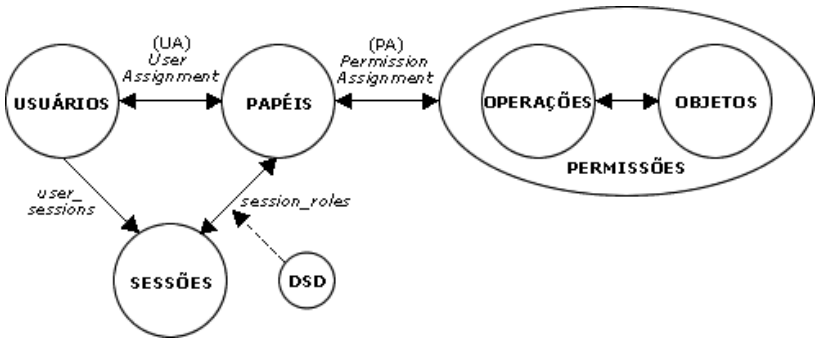

Figura 5: Relações de Separação Dinâmica de Tarefas

2.4 Conclusões do Capítulo

Este capítulo apresentou o controle de acesso baseado em papéis (Role-Based Access Control - RBAC) como um modelo de segurança flexível, e ao mesmo tempo centralizado, cuja configuração é dependente da política organizacional de proteção da informação. As características do modelo RBAC são um desejo de muitas organizações e são difíceis de serem obtidas com os modelos clássicos de segurança computacional.

O modelo unificado RBAC-NIST representa, segundo Obelheiro (2001, p. 43), uma primeira iniciativa de padronização das características básicas do controle de acesso baseado em papéis. Outros modelos RBAC possuem características que os distinguem do modelo RBAC-NIST, mas esta padronização representa um passo importante para o estabelecimento de um consenso sobre as diversas características fundamentais do RBAC e de um ponto de partida para novos desenvolvimentos na área.

3 Seguraweb: Um Framework RBAC Para Aplicações Web

O Seguraweb, proposto por Kátyra Kowalski Armanini (ARMANINI, 2002), é um framework orientado a objetos cujo objetivo principal é a diminuição do tempo de desenvolvimento de mecanismos de segurança de aplicações Web ou quaisquer outras que utilizam Java, principalmente no que se refere ao controle de acesso, no qual é usado o modelo de segurança RBAC.

Este capítulo apresenta as etapas de desenvolvimento do framework Seguraweb e como utilizá-lo na construção de aplicações Web. Para isso, é apresentada também a importância da reusabilidade de software e como essa pode ser obtida com a reutilização de frameworks.

3.1 O que é um Framework?

Nos últimos anos, o desenvolvimento de software mudou significativamente: os desenvolvedores, cada vez mais, vêm produzindo softwares mais complexos em menos tempo. Essa evolução da indústria de software, motivou os desenvolvedores a buscar novas técnicas, de modo a não só melhorar a qualidade do software, como também diminuir o tempo e o esforço necessários para produzí-los, uma vez que softwares mais complexos são mais caros e propensos a erros. Em particular, a crescente heterogeneidade de arquiteturas de hardware e a diversidade de sistemas operacionais e plataformas de comunicação dificultam bastante o desenvolvimento de aplicações desde o início.

Nesse contexto, segundo Silva (2000, p. 21), se encaixa a reutilização de software, em contraposição ao desenvolvimento de todas as partes de um sistema, como um fator que pode levar ao aumento da qualidade e da produtividade da atividade de desenvolvimento de software. Isso se baseia na perspectiva de que o reuso de artefatos de software[1] já de-

[1]Silva utiliza a expressão *artefato de software* de forma genérica, referindo-se não somente ao código,

senvolvidos e depurados, reduza o tempo de desenvolvimento, de testes e as possibilidades de introdução de erros na produção de novos artefatos.

Observa-se que a reutilização de artefatos de software não necessariamente precisa se restringir a uma simples utilização de trechos de código de aplicações já desenvolvidas (rotinas) ou na utilização de classes de uma biblioteca. A reutilização pode englobar, de forma mais significativa, as etapas de análise e projeto de um sistema. Na tecnologia orientada a objetos, "conhecer uma linguagem orientada a objetos (tal como Java) e ter acesso a uma biblioteca rica (tal como a biblioteca Java) é um primeiro passo necessário, porém insuficiente na criação de sistemas de objetos. Analisar e projetar um sistema a partir de uma perspectiva de objetos também é crítico" (LARMAN, 2000, p. 27).

Silva (2000, p. 22) descreve dois objetivos distintos na reutilização no nível das especificações de análise e projeto:

- Produzir uma aplicação utilizando o projeto de outra aplicação. Isso é o que ocorre em um processo de reengenharia, em que a especificação de uma aplicação é alterada, de modo a adaptar-se a novos requisitos, e é gerada uma nova aplicação.
- Produzir uma especificação adequada a um conjunto de aplicações (um domínio) e reutilizar esta especificação no desenvolvimento de diferentes aplicações. Essa é a abordagem dos frameworks orientados a objetos.

Atualmente, existem várias definições para frameworks orientados a objetos ou simplesmente frameworks[2]. Booch, Rumbaugh e Jacobson (2000, p. 380) propõe a seguinte definição: "um padrão de arquitetura que fornece um template extensível para aplicações dentro de um domínio". Ao especificar um framework, é especificado o esqueleto de uma arquitetura que é exposto aos usuários do framework. Os usuários utilizam essa micro-arquitetura para resolver um problema básico de um domínio comum, adaptando a estrutura do framework ao contexto do problema.

Trazendo o conceito de frameworks para mais próximo do conceito de classes, Silva (apud WIRFS-BROCK et al., 1991) define um framework como "um esqueleto de implementação de uma aplicação ou de um subsistema de aplicação, em um domínio de problema particular. É composto de classes abstratas e concretas e provê um modelo de interação ou colaboração entre as instâncias de classes definidas pelo framework. Um

como também às etapas de análise e projeto de aplicações, frameworks ou componentes.

[2]A expressão *frameworks orientados a objetos* se refere a abordagem do paradigma de orientação a objetos na descrição do domínio de um *framework*.

framework é utilizado através de configuração ou conexão de classes concretas e derivação de novas classes concretas a partir das classes abstratas do framework". O próprio Silva (2000, p. 31) simplifica a definição de Wirfs-Brock et al.: "um framework é uma estrutura de classes inter-relacionadas, que corresponde a uma implementação incompleta para um conjunto de aplicações de um domínio. Essa estrutura de classes deve ser adaptada para a geração de aplicações específicas".

Embora as definições de frameworks variem um pouco para cada autor, seu objetivo principal se mantém: promover o reuso do código e do projeto, de modo a diminuir o tempo e o esforço exigidos na produção de software e, conseqüentemente, melhorar a qualidade e aumentar a produtividade da atividade de desenvolvimento de software.

Os frameworks fornecem um grau muito elevado de reutilização de software e, como já citado, não devem ser confundidos com a simples reutilização de classes de uma biblioteca. A diferença fundamental é que na reutilização de classes de uma biblioteca são usadas classes isoladas. Cabe ao desenvolvedor da aplicação estabelecer suas interconexões. Dessa forma, os desenvolvedores continuam sendo responsáveis por fornecer toda infraestrutura e fluxo de controle da aplicação. No caso dos frameworks, a infraestrutura provém das etapas de análise e projeto, isto é, as interconexões preestabelecidas definem a arquitetura da aplicação, liberando o desenvolvedor dessa responsabilidade. Além disso, o fluxo de controle da aplicação também é fornecido pelo framework. Esse aspecto é conhecido por princípio de Hollywood: "Não nos chame, nós iremos chamá-lo" (LARMAN, 2000, p. 439). Isso significa que as classes definidas pelo usuário receberão mensagens das classes pré-definidas do framework. As figuras 6, 7 e 8 ilustram a diferença entre o desenvolvimento de uma aplicação reutilizando classes de biblioteca e o desenvolvimento de uma aplicação reutilizando um framework.

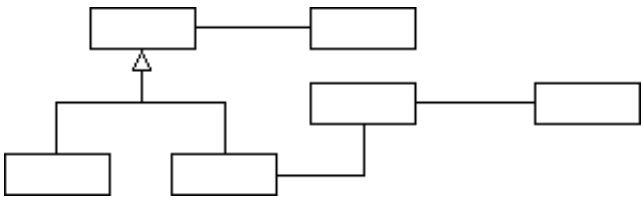

Figura 6: Aplicação desenvolvida totalmente

Larman (2000, p. 29) aponta a atribuição habilidosa de responsabilidades aos componentes de software como a habilidade individual mais importante na análise e projeto orientados a objetos. Em seguida à atribuição de responsabilidade e, segundo Larman, ocupando um segundo lugar bem próximo em termos de importância, está encontrar ob-

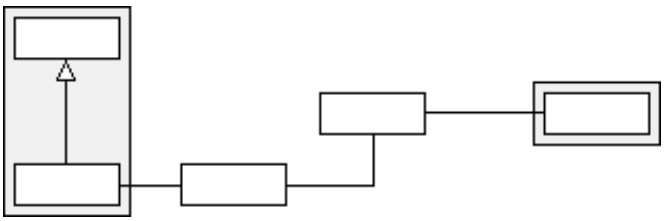

Figura 7: Aplicação desenvolvida reutilizando classes de biblioteca

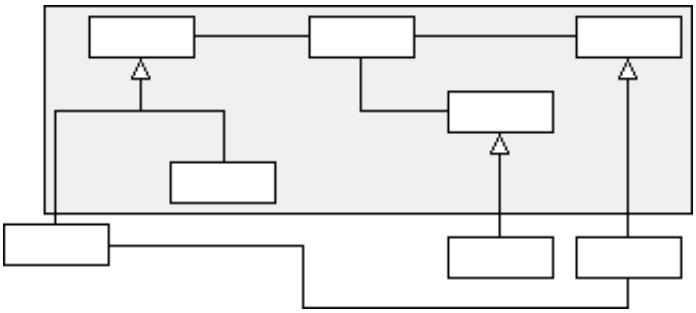

Figura 8: Aplicação desenvolvida reutilizando um framework

jetos ou abstrações adequadas[3]. Conclui-se, portanto, que frameworks maximizam os benefícios da tecnologia de orientação a objetos, uma vez que a atribuição de responsabilidades aos componentes do software e as abstrações, ou seja, a infraestrutura e o fluxo de controle da aplicação são fornecidas pelo framework. A partir de frameworks bem projetados, ou seja, frameworks completos, flexíveis, extensíveis e fáceis de usar, os desenvolvedores não precisam se preocupar com a arquitetura da aplicação, podendo assim concentrar seus esforços na particularização do comportamento do framework, de forma a moldá-lo a uma necessidade específica. Frameworks como o Microsoft DCOM (*Distributed Common Object*), Java RMI (*Remote Method Invocation*) e OMG CORBA (*Common Object Request Broker Architecture*), usados na comunicação entre objetos locais e remotos em sistemas distribuídos, são exemplos de frameworks de grande importância no desenvolvimento de software atual.

3.2 Desenvolvimento do Framework Seguraweb

Desenvolver um framework é um pouco diferente de desenvolver uma aplicação: "um framework de sucesso soluciona problemas que, superficialmente, são bastante diferentes do problema que justificou sua criação" (TALIGENT, 1994, p. 3). Silva (2000, p. 45)

[3]Larman enfatiza a atribuição de responsabilidades porque tende a ser a habilidade mais difícil de dominar.

aponta a principal característica buscada ao desenvolver um framework a generalidade em relação a conceitos e funcionalidades do domínio tratado. Portanto, pode-se pensar em um framework como uma solução independente, tanto do problema original quanto das futuras aplicações em que ele é usado. Ao mesmo tempo, cada uma dessas aplicações deve parecer aquela para qual o framework foi projetado.

Bosch et al. (1997, p. 6) descreve as seguintes atividades como parte de um modelo de desenvolvimento de frameworks simples:

- **Análise de domínio:** descreve o domínio que será abrangido pelo framework. Para capturar e identificar os conceitos relacionados com o domínio, pode-se consultar aplicações já desenvolvidas, especialistas e padrões existentes para esse domínio. O resultado dessa atividade é um modelo de análise de domínio, contendo os requisitos do domínio, os conceitos do domínio e as relações entre esses conceitos;

- **Projeto de arquitetura:** utiliza o modelo de análise de domínio como ponto de partida. O projetista deve definir um estilo de arquitetura [4] que se encaixe bem ao framework. Com base nisso é que o projeto de alto nível do framework é elaborado;

- **Projeto do framework:** o projeto de alto nível do framework é refinado e classes adicionais são projetadas. O resultado dessa 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 baseadas nas decisões de arquitetura e, finalmente, um documento com o histórico do projeto, descrevendo os problema encontrados e as soluções escolhidas juntamente com as argumentações dessas escolhas;

- **Implementação do framework:** se preocupa com a codificação das classes abstratas e concretas do framework;

- **Testes do framework:** não só determina se o framework atende a todas as funcionalidades para o qual foi projetado como também avalia a sua usabilidade. Porém, avaliar se uma entidade é utilizável não é trivial. No caso de frameworks, esse processo se resume a desenvolver aplicações que utilizem o framework;

[4] Um estilo de arquitetura (*architectural style*) caracteriza uma família de sistemas que estão relacionados por compartilharem propriedades estruturais e semânticas. Ele fornece um vocabulário de elementos de projeto (clientes e servidores, *pipes* e filtros, *blackboards* e *knowledge sources*, etc) e regras que descrevem como esses elementos podem ser combinados.

- **Geração de aplicação de teste:** para avaliar a usabilidade de um framework, é interessante desenvolver aplicações de teste baseadas nele. Dependendo do tipo da aplicação, ela pode testar diferentes aspectos do framework. Esta etapa visa verificar se o framework precisa ser reprojetado ou se está adequado para atender às necessidades das aplicações.

No desenvolvimento do Seguraweb, Armanini segue o modelo de desenvolvimento de frameworks proposto por Bosch et al.. Portanto, cada uma das etapas de desenvolvimento são descritas a seguir.

3.2.1 Análise de Domínio

A análise de domínio atua no desenvolvimento de descrições que generalizem uma família de aplicações com características comuns - um domínio de aplicações. É necessário obter os requisitos do domínio, os conceitos do domínio e as relações entre esses conceitos. Larman (2000, p. 60) define requisitos como uma descrição das necessidades ou dos desejos para um produto.

Após analisar aplicações já desenvolvidas e o modelo de segurança RBAC, definiu-se que os seguintes requisitos de segurança deveriam ser fornecidos pelo Seguraweb:

- **Autenticação:** assegurar que o usuário é quem ele diz ser;
- **Confidencialidade:** garantir a proteção das informações em trânsito;
- **Controle de acesso:** controlar o que o usuário pode acessar no sistema;
- **Auditoria:** registrar e analisar o que o usuário faz no sistema;
- **Administração:** gerenciar as informações de segurança incluindo as políticas de segurança (gerência de perfis de usuários, etc);
- **Persistência dos dados:** armazenar os dados de autenticação, controle de acesso, auditoria e administração.

O modelo RBAC é importante principalmente na implementação dos requisitos de controle de acesso e administração das informações de segurança das aplicações.

3.2.2 Projeto de Arquitetura

A maioria dos engenheiros de software consideram o projeto de arquitetura uma etapa crítica no processo de desenvolvimento de um software. "Escolher a arquitetura correta é geralmente crucial para o projeto de um software: uma escolha errada pode levar a um resultado desastroso" (GARLAN; SHAW, 1994, p. 2).

Embora não haja uma terminologia ou notação bem definida para descrever um projeto de arquitetura, a descrição pode ser feita informalmente, através de diagramas ilustrando a estrutura básica do sistema. Eles servem como um componente crítico no desenvolvimento da maioria dos sistemas, tanto por fornecer um ponto de partida para determinar se o sistema irá atender os requisitos definidos quanto por guiar os programadores na construção do sistema.

Na prática, o projetista deve definir um estilo de arquitetura. No caso do Seguraweb foi usada a arquitetura *3-Tier* (figura 9) - um tipo especial de arquitetura cliente-servidor que consiste em três processos bem definidos, cada um executando em uma máquina diferente:

1. ***Client Tier*:** a interface com o usuário. Executa na máquina do cliente;
2. ***Application Tier*:** os módulos funcionais que processam os dados. Executa no servidor de aplicações (*Application Server*);
3. ***Data-storage Tier*:** armazena os dados utilizados pelas aplicações. Executa no servidor de banco de dados (*Database Server*).

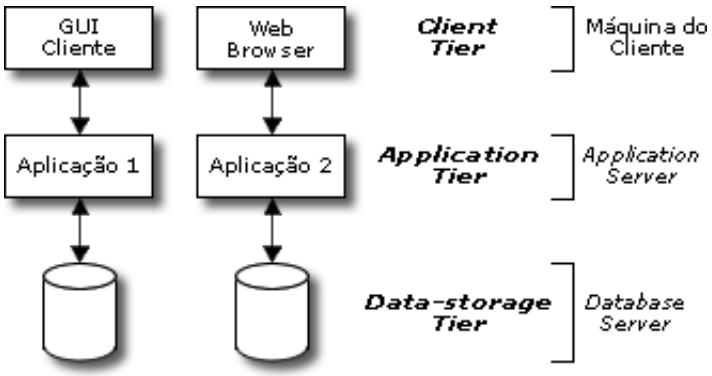

Figura 9: Arquitetura *3-tier*

O objetivo principal do Seguraweb, segundo Armanini (2002, p. 51), é que ele seja utilizado para gerar aplicações com interface Web. Procurou-se, portanto, obter informações

sobre as tecnologias existentes para a geração de conteúdo dinâmico na Web[5]. Chegou-se as seguintes tecnologias mais populares atualmente: ASP (*Active Server Pages*), PHP (*Hypertext Preprocessor*), JSP (*Java Server Pages*) e Servlets.

ASP, da Microsoft, permite optar entre múltiplas linguagens de *script*, incluindo PerlScript, Jscript e VBScript. No entanto, a linguagem padrão é VBScript: uma linguagem de *script* baseada na linguagem de programação VB (*Visual Basic*) da Microsoft. Essa linguagem permite acessar componentes ActiveX, que encapsulam virtualmente qualquer funcionalidade, incluindo acesso à banco de dados e manipulação de arquivos. A própria Microsoft disponibiliza uma grande variedade de componentes ActiveX, além de ferramentas e tutoriais para desenvolvê-los. A maior limitação da tecnologia ASP, porém, é que só pode ser usada no Microsoft IIS (*Internet Information Server*), executando no sistema operacional Windows.

PHP utiliza uma sintaxe similar a C e tem se tornado bastante popular nos últimos anos, pois é bastante simples e permite um fácil acesso à banco de dados. Além disso, possui extensões que permitem a comunicação com outros recursos de rede, como servidores de diretório e e-mail. Ao contrário da maioria das tecnologias Web existentes, PHP é uma tecnologia *Open Source* e pode ser usada em diversas plataformas e servidores HTTP, como o Apache e o Microsoft IIS.

JSP e Servlets integram a plataforma J2EE (*Java 2 Plataform, Enterprise Edition*) e, portanto, são baseadas na linguagem Java. Java possui algumas vantagens em relação às outras linguagens de programação como C e C++. Por ser uma linguagem orientada a objetos com tipagem forte, encapsulamento, tratamento de excessão e gerenciamento automático de memória, seu uso facilita bastante a produção de software. O principal atrativo de Java, entretanto, é que ela é uma tecnologia independente de plataforma, isto é, o código compilado Java (*bytecode*) é portável para qualquer plataforma que possua a máquina virtual Java (*Java Virtual Machine*) instalada. Além disso, o programador Java pode dispor de inúmeras APIs, incluindo aquelas utilizadas no acesso à banco de dados, serviços de diretórios, computação distribuída e criptografia.

Com relação ao armazenamento persistente das informações, foi escolhido o banco de dados relacional Oracle 8*i*. "Essa ferramenta foi escolhida por ser uma das mais utilizadas no desenvolvimento de aplicações de grande porte, e por ter versões para várias plataformas, principalmente Windows e Linux" (ARMANINI, 2002, p. 52). De fato, tratando-se

[5]Para as requisições Web mais simples, o browser solicita um documento HTML, o servidor Web procura o arquivo correspondente e o retorna. Requisições dinâmicas, porém, exigem que o servidor Web execute um processamento adicional de modo a gerar uma resposta customizada à requisição do cliente.

de desempenho, escalabilidade e segurança, o Oracle 8*i* oferece inúmeras soluções sob demanda para qualquer requisito de negócios. Além disso, possui uma extensão, chamada RAC (*Real Application Cluster*), que permite a criação de bancos em múltiplos nós. A tecnologia de *clustering*[6] oferece um novo nível de escalabilidade, disponibilidade e tolerância a falhas, sem a necessidade de modificação do código das aplicações.

3.2.3 Projeto e Implementação do Framework

A arquitetura do Seguraweb, ilustrada na figura 10, é dividida em três camadas: camada básica, camada de persistência e camada de aplicação. A primeira camada possui as classes básicas, usadas pelas outras camadas do framework. A camada de persistência utiliza um outro framework, proposto por Craig Larman (LARMAN, 2000), que tem como objetivo armazenar e recuperar objetos em um mecanismo de armazenamento persistente. Finalmente, a camada de aplicação contém as classes que deverão ser usadas pelos usuários do Seguraweb na construção das aplicações Web. Essa camada utiliza as camadas inferiores para implementar as características e regras gerais do modelo RBAC.

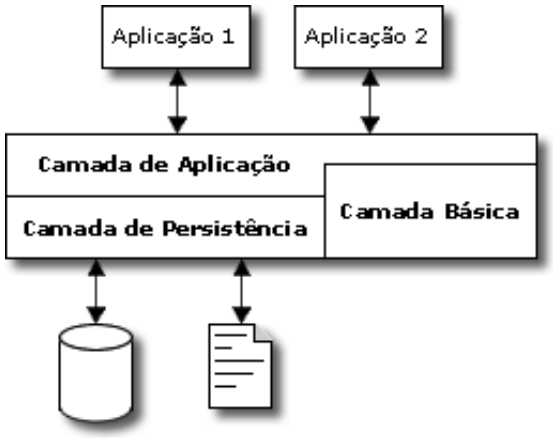

Figura 10: Arquitetura do Seguraweb

3.2.3.1 Camada Básica

A camada básica possui as classes principais que compõe o framework Seguraweb: *User*, *Role* e *Permission*. Essas classes, ilustradas na figura 11, espelham as entidades

[6] *Clustering* é o processo de conectar dois ou mais computadores de modo que eles se comportem como um único computador. No RAC, os servidores trabalham em cluster trocando informações através de discos compartilhados. Se um servidor falhar, outros servidores assumirão suas tarefas para que não haja falhas. Além disso, o RAC permite adicionar nós ao cluster à medida que aumenta a demanda por capacidade.

fundamentais do modelo RBAC (usuário, papel e permissão respectivamente). Elas são usadas pelas outras camadas do framework e, se necessário, pelos desenvolvedores de software que utilizarão o framework na construção de suas aplicações Web.

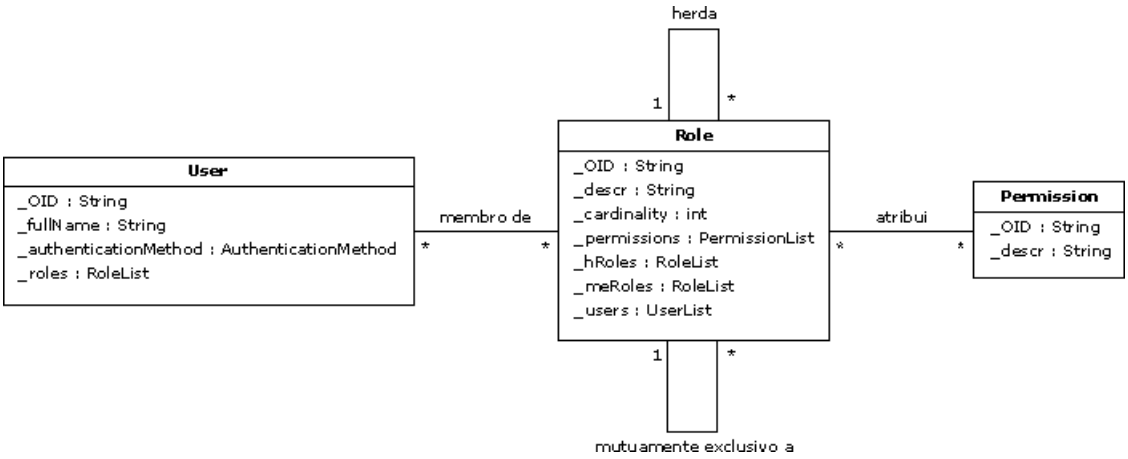

Figura 11: Diagrama de classes da camada básica

Percebe-se pelo diagrama que o Seguraweb implementa hierarquia de papéis e separação de tarefas (estáticas e dinâmicas). Isso é feito a partir dos atributos *_hRoleList*, *_meRoleList* e *_cardinality* da classe *Role*, que indicam a lista de papéis do qual o papel herda permissões, a lista de papéis mutuamente exclusivos ao papel e o número máximo de usuários que podem ser membros do papel respectivamente. Dessa forma o Seguraweb cobre aspectos importantes do modelo RBAC.

O atributo *AuthenticationMethod* da classe *User*, indica o método de autenticação que será utilizado para identificar o usuário no sistema, isto é, assegurar que o usuário é quem ele diz ser. A autenticação é o passo primordial na maioria dos sistema de segurança e não deve ser confundida com o conceito de autorização, que é o processo de conceder ou negar o acesso aos recursos computacionais. A autorização geralmente ocorre após a autenticação.

Para dar suporte a autenticação de usuários, o Seguraweb define a interface *AuthenticationMethod*. Essa interface representa os diversos métodos de autenticação disponíveis no framework: *Password* (senha), *PrivateKey* (chave-privada), PBE (*Password-Based Encryption*[7]) e PPK (*Password and Private Key*), que implementa a autenticação do usuário através de senha e chave-privada conjuntamente. Os métodos de autenticação são ilustrados na figura 12.

[7] *Password-Based Encryption* também implementa a autenticação por chave-privada, no entanto, essa chave é gerada a partir de uma senha do usuário e um número aleatório conhecido como *salt*

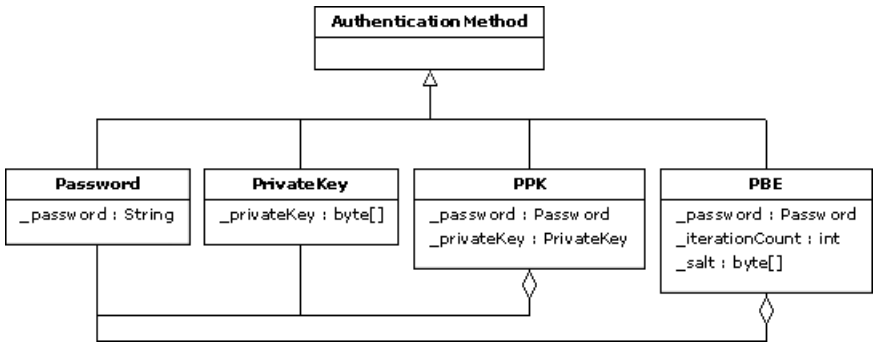

Figura 12: Diagrama de classes dos métodos de autenticação

3.2.3.2 Camada de Persistência

Como apontado na etapa de análise de domínio, a persistência dos dados é um dos requisito do Seguraweb. Segundo Larman (2000, p. 436), a maioria das aplicações requer o armazenamento e a recuperação de informações em um mecanismo de armazenamento persistente, tal como um banco de dados relacional.

Larman (2000, p. 437) descreve um framework para persistência como um conjunto de classes reutilizáveis - e usualmente extensíveis - que fornece 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.

No Seguraweb, a camada de persistência foi baseada no framework para persistência proposto por Larman. É através da camada de persistência que os elementos do modelo RBAC, implementados na camada básica, são armazenados e recuperados dos mecanismos de armazenamento persistentes.

Como a maioria dos frameworks, um framework para persistência é repleto de padrões. "Um padrão é uma solução comum para um problema básico em um determinado contexto" (BOOCH; RUMBAUGH; JACOBSON, 2000, p. 377). Dada uma categoria específica de problema, muitos padrões fornecem orientações sobre como as responsabilidades devem ser atribuídas aos objetos. Conseqüentemente, obtém-se estruturas de classes bem organizadas e mais aptas a modificações e extensões.

No caso do framework para persistência proposto por Larman, são utilizados os padrões *Object Identifier*, *Database Broker*, *Cache Management*, *Virtual Proxy*, entre outros.

O padrão *Object Identifier* propõe atribuir um identificador de objeto (OID) para cada registro de objeto. Um OID é usualmente um valor alfanumérico 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" (LARMAN, 2000, p. 442).

O padrão *Database Broker* propõe a criação de uma classe responsável pela materialização[8], desmaterialização e *caching* (memória prévia) dos objetos. Com relação ao *caching*, o padrão *Cache Management* propõe tornar os *Database Brokers* responsáveis pela manutenção do seu *cache*. "Quando os objetos são materializados, eles são colocados no *cache*, tendo como chave o seu OID. As solicitações subseqüentes do *broker* para um objeto farão com que o *broker*, primeiro, busque o objeto no *cache*, evitando, dessa forma, materializações desnecessárias" (LARMAN, 2000, p. 447).

O padrão *Virtual Proxy* é uma referência inteligente para o objeto real: um objeto cliente tem uma referência para o objeto Virtual Proxy, em vez de uma referência para o objeto real que pode, ou não, estar materializado. Segundo Larman (2000, p. 448), às vezes é desejável adiar a materialização de um objeto até que ela seja absolutamente necessária. Isso é conhecido como materialização sob demanda.

De modo a atender a esses padrões na camada de persistência do Seguraweb, criou-se uma superclasse para os *brokers* chamada *PersistentBroker*. Como o principal mecanismo de armazenamento das informações referentes ao modelo RBAC é um banco de dados relacional (Oracle 8*i*), foi criada uma subclasse de *PersistentBroker*, denominada *RelationalPersistentBroker*, que representa esse tipo de armazenamento persistente. Por fim, cada um dos elementos RBAC foi representado por uma subclasse correspondente: *RelationalBrokerUser*, *RelationalBrokerRole* e *RelationalBrokerPermission*. Como pode ser visto na figura 13, cada *broker* se responsbiliza pela manutenção de seu próprio *cache*, através dos atributos *_userCache*, *_roleCache* e *_permissionCache*.

De forma similar, para os *proxies*, criou-se uma superclasse *VirtualProxy* e subclasses correspondentes representando cada um dos elementos RBAC: *UserProxy*, *RoleProxy* e *PermissionProxy*. Como pode ser visto na figura 14, a classe *Virtual Proxy* possui um *_OID*, indicando o objeto real a qual esse *proxy* se refere, o próprio objeto real (*_realSubject*), uma lista contendo os *brokers* já utilizados (*_brokers*), e uma referência ao último *broker* instanciado (*_broker*).

[8] "*Materialização* é o ato de transformar uma representação de dados não orientada ao objeto (por exemplo, registros), existente em um armazenamento persistente, em objetos. A *desmaterialização* é a atividade oposta" (LARMAN, 2000, p. 440).

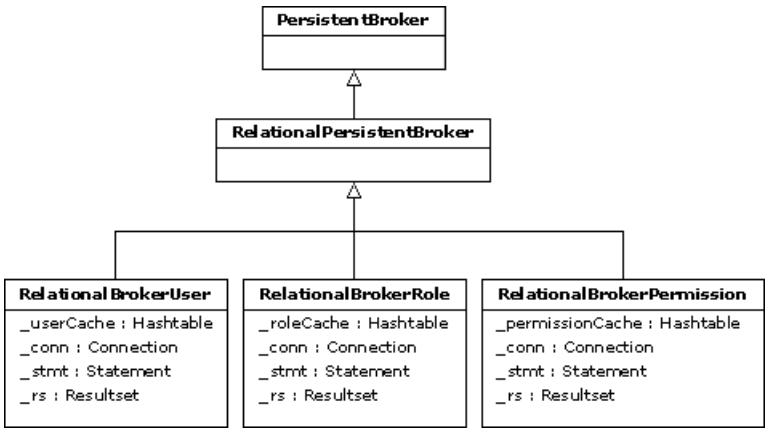

Figura 13: Diagrama de classes dos *brokers*

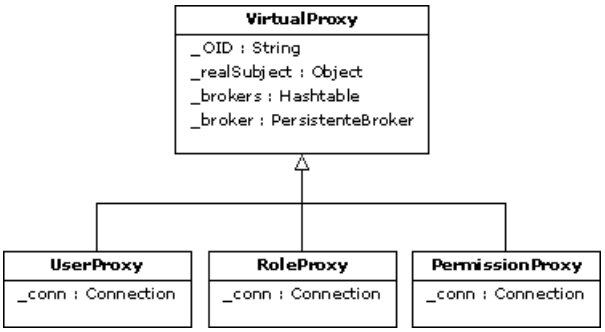

Figura 14: Diagrama de classes dos *proxies*

É importante destacar que cada *proxy* cria o *broker* correspondente ao elemento RBAC representado, ou seja, o *UserProxy* cria um *RelationalBrokerUser*, o *RoleProxy* cria um *RelationalBrokerRole* e o *PermissionProxy* cria um *RelationalBrokerPermission*. Além disso, os *proxies* implementam a mesma interface dos objetos reais. Isso significa que cada um dos métodos definidos nessa interface deverá ser implementado tanto pelo objeto real quanto pelo *proxy* desse objeto. A figura 15 ilustra as relações entre as classes *User*, *UserProxy* e *RelationalBrokerUser*. Para as outras classes básicas do Seguraweb (*Role* e *Permission*), as relações entre objetos reais, *proxies* e *brokers* são semelhantes.

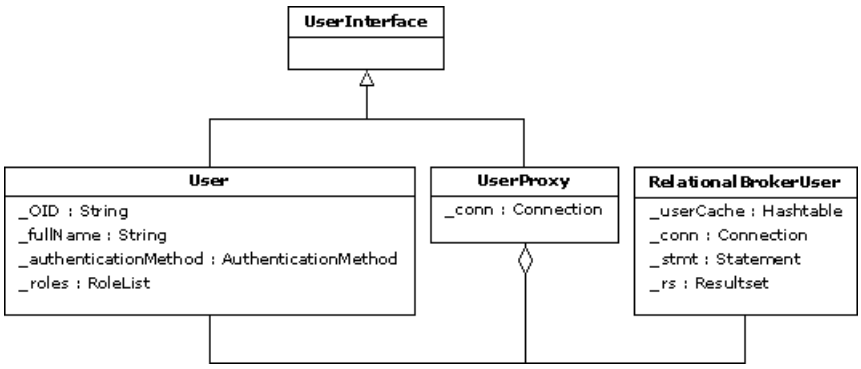

Figura 15: Relações entre as classes *User*, *UserProxy* e *RelationalBrokerUser*

3.2.3.3 Camada de Aplicação

A camada de aplicação contém as classes que dão suporte à autenticação de usuários, o controle de acesso, a auditoria e a administração das informações de segurança do modelo RBAC.

A autenticação é feita a partir da classe *Authentication* que verifica a autenticidade de usuários no sistema a partir de um dentre os diversos métodos de autenticação representados pela interface *AuthenticationMethod*.

O controle de acesso é feito a partir da classe *Authorization*, que verifica se o usuário tem ou não permissão de executar uma determinada operação no sistema, de acordo com os papéis assumidos por ele.

Finalmente, a administração das informações de segurança é feita a partir de classes responsáveis pela inserção, atualização e remoção de usuários, papéis e permissões; bem como classes responsáveis por gerenciar a criação de associações entre usuários e papéis, e entre papéis e permissões no sistema. Essas classes são: *DBUserManager*, *DBRolerManager*, *DBPermissionManager*, *RBACUserRoleAssociationManager* e *RBACRolePermissionAssociationManager*.

3.2.4 Testes do framework

Seguindo o modelo de desenvolvimento de frameworks proposto por Bosch et al., a última etapa de desenvolvimento de um framework são os testes. Uma forma eficiente de testar um framework é gerar aplicações de teste baseadas nele e, desta forma, verificar se o framework precisa ser reprojetado ou se está adequado para atender às necessidades das aplicações.

No trabalho de Armanini (ARMANINI, 2002), foram criadas duas aplicações de teste: uma para validar os requisitos de administração das informações de segurança e outra para validar o esquema de autorização baseada em papéis do Seguraweb. Este trabalho apresenta outra aplicação de teste, o QEEWeb.

3.3 Conclusões do Capítulo

Este capítulo apresentou cada uma das etapas do projeto e implementação do framework Seguraweb, e como são organizadas as camadas que compõe esse framework. Uma dessas camadas, a camada de persistência, baseada no framework para persistência proposto por Craig Larman (LARMAN, 2000), possui diversos padrões como uma forma de reutilização de projeto. Padrões são formas populares e eficientes de implementação de softwares flexíveis e reutilizáveis.

Na seção 3.1, apresentou-se a importância da reutilização de software, principalmente se ela englobar não somente trechos de código de aplicações já desenvolvidas (rotinas) ou classes de uma biblioteca, mas também as etapas de análise e projeto de um sistema, a partir da abordagem de frameworks orientados a objetos.

O Seguraweb facilita o desenvolvimento dos mecanismos segurança das aplicações, pois implementa requisitos importantes, como autenticação, confidencialidade, controle de acesso e auditoria. Esses requisitos apesar de necessários na maioria das aplicações nem sempre são fáceis de serem implementados.

4 Monitoramento de Qualidade de Energia Elétrica

No brasil, o mercado de energia elétrica tem sofrido diversas mudanças nos últimos anos, à medida que as indústrias em geral têm investido em automatização de seus sistemas produtivos na busca de obter uma melhoria nos seus processos e enfrentar a concorrência advinda da globalização da economia. Essa transformação, embora contribua para um aumento da produtividade industrial e para um uso mais eficiente da energia elétrica, mudou os requisitos de qualidade da energia. Enquanto os sistema eletromecânicos são insensíveis a interrupções do fornecimento de energia da ordem de segundos, os sistemas eletroeletrônicos são sensíveis a interrupções do fornecimento da ordem de milisegundos, além da sensibilidade à sub-tensões. Além disso, embora os equipamentos eletroeletrônicos sejam mais econômicos para a indústria, eles podem provocar o aparecimento de distorções harmônicas na tensão, ocasionadas pela presença de cargas com características não-lineares no sistema elétrico. Cargas não-lineares, segundo Jesus, Nonenmacher e Oliveira (2003, p. 13) afetam a operação dos componentes e equipamentos, podendo causar sobreaquecimentos e redução da vida útil de transformadores e máquinas elétricas, ressonâncias com capacitores, erros em medidores, atuação intempestiva de sistemas de produção, etc. Sendo assim, limites são recomendados quanto aos valores das distorções das tensões e correntes, visando manter a conformidade da onda e a qualidade do fornecimento de energia.

A questão da qualidade de energia elétrica apesar de não ser nova, só recentemente, com a mudança de regulamentação no setor elétrico brasileiro, começou a ser amplamente discutida. Ainda não foram definidos indicadores de desempenho do sistema que leve em conta todos os aspectos relacionados a qualidade de energia elétrica entregue aos consumidores, nem há um sistema de monitoramento e estatística de distúrbios que permita definir ou estabelecer com precisão padrões de qualidade a serem seguidos. “Ressalta-se que mesmo nos países industrializados, se observa a necessidade de se rever os critérios de gestão do mercado e da qualidade de energia. Esses tópicos são de grande importância

no atual contexto de mudanças em que passa o setor elétrico brasileiro onde a presença do Estado será cada vez menor e conseqüentemente uma maior atuação do setor privado. Nesse contexto a oferta de energia com boa qualidade será um dos fatores de competição entre as própria concessionárias" (MELO; CAVALCANTI, 2003, p. 478).

Este capítulo apresenta uma classificação dos principais fenômenos que afetam a qualidade da onda na tensão de alimentação. A qualidade de energia elétrica, na maioria dos casos, pode ser representada pela qualidade da tensão elétrica do ponto onde a carga está ligada. Como já existem normas para tensão, cujas características podem ser facilmente medidas, a medição da sua qualidade torna-se uma tarefa menos complicada.

4.1 Fenômenos de Qualidade de Energia Elétrica

Parada (1999, p. 35) define que uma onda ideal de tensão deve ter uma forma senoidal de freqüência 50 ou 60 Hz, com amplitude especificada e invariável e deve existir simetria entre as tensões de fases nos casos de sistemas trifásicos. Quando essas características são alteradas, a onda sofre perturbações e a onda ideal já não existe. Num sistema de distribuição, não é possível ter uma onda ideal porque são muitas as fontes de distorção, mas sim é possível ter uma boa qualidade da onda, isto é, ter um suprimento que cumpra uma conformidade determinada.

A energia elétrica possui características que a diferencia dos demais insumos industriais. Ela precisa ser gerada concomitantemente com o consumo, não pode ser armazenada pelos consumidores, não pode ser transportada pelos meios usuais de transporte, e mais importante, sua qualidade depende tanto da concessionária que a produz como do consumidor.

A recomendação IEEE Std 1159-1995 para monitoramento de qualidade de energia elétrica (INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEER, 1995, p. 12) caracteriza os fenômenos de qualidade como:

- **Transitórios:** impulsivos ou oscilatórios;
- **Perturbações de curta duração:** interrupções, *sags* ou *swells*;
- **Perturbações de longa duração:** interrupções longas, subtensões ou sobretensões;
- **Desequilíbrios de tensão**;

- **Distorções da forma de onda:** componente DC, harmônicos, interharmônicos, *notching* ou ruídos;
- **Flutuações de tensão**;
- **Variações de freqüência**.

4.1.1 Transitórios

Os transitórios são fenômenos conseqüentes de alterações súbitas nas condições operacionais de um sistema de energia elétrica. Geralmente, a duração de um transitório é muito pequena, mas de grande importância, uma vez que submetem equipamentos a grandes solicitações de tensão, corrente, ou ambas. Existem dois tipos de transitórios: os impulsivos, causados por descargas atmosféricas, e os oscilatórios, normalmente decorrentes de chaveamentos na rede. Exemplos de um transitório impulsivo e um transitório oscilatório são mostrados nas figuras 16 e 17 respectivamente.

Figura 16: Transitório impulsivo

Figura 17: Transitório oscilatório

4.1.2 Perturbações de Curta Duração

Geralmente, as perturbações de curta duração são variações de tensão causadas por falhas (curto circuito) no sistema elétrico. Dependendo do local da falha e das condições do sistema, o resultado pode ser um afundamento de tensão (*sag*), uma elevação de tensão (*swell*), ou mesmo uma interrupção completa da rede.

Uma interrupção de curta duração ocorre quando a tensão de suprimento é interrompida por um período de tempo não superior a 1 minuto. No passado, as interrupções

de curta duração não eram consideradas preocupantes. Entretanto, com o aumento da sensibilidade dos equipamentos dos consumidores, a interrupção de curta duração passou a ser evitada.

Um *sag* é uma redução na magnitude da tensão de curta duração (0,5 ciclos até 1 minuto). *Sags* são causados por falhas no sistema elétrico, partidas de motores e de cargas de grande potência.

Um *swell* é uma elevação na magnitude da tensão de curta duração (0,5 ciclos até 1 minuto). Como no caso do *sag*, o *swell* está relacionado com a ocorrência de falhas no sistema elétrico mas pode ocorrer também devido ao desligamento de cargas ou a conexão de banco de capacitores de grande potência.

A presença de *sags* e *swells* podem, entre outros problemas, causar o mau funcionamento de equipamentos eletrônicos, parada de motores e danificar os sensíveis circuitos de sistemas computacionais.

As figuras 18, 19 e 20 são exemplos de perturbações de curta duração.

Figura 18: Interrupção de curta duração

Figura 19: *Sag*

Figura 20: *Swell*

4.1.3 Perturbações de Longa Duração

As perturbações de longa duração são variações de tensão com duração superior a 1 minuto. Essas perturbações podem ser classificadas como interrupções longas, subtensões ou sobretensões.

As subtensões e sobretensões são similares a *sags* e *swells* respectivamente. Entretanto, não são resultados de falhas no sistema elétrico, e sim de desvios de carga e chaveamentos na rede.

Uma interrupção de longa duração ocorre quando a tensão de suprimento é interrompida por um período de tempo que excede 1 minuto. Ao contrário da interrupção de curta duração, que é de caráter temporário, a interrupção de longa duração é considerada uma falha permanente. Isto significa que é necessário uma intervenção manual para restaurar o sistema elétrico.

4.1.4 Desequilíbrio de Tensão

Os desequilíbrios de tensão, segundo Parada (1999, p. 44–45), surgem em função das cargas no sistema elétrico serem distribuídas desigualmente entre as fases, provocando o aparecimento de tensões desequilibradas. O nível de desequilíbrio de tensão é definido como o quociente entre o valor de tensão com maior desvio em relação ao valor médio das três fases, e o valor médio das fases expresso em porcentagem.

As tensões desequilibradas comprometem os motores de indução e os conversores estáticos. Já as correntes desequilibradas, têm influência significativa no funcionamento de geradores.

4.1.5 Distorções na Forma de Onda

Um dos problemas freqüentemente encontrados em sistemas elétricos é o aparecimento de harmônicas, ocasionadas pela presença de cargas especiais na rede. As harmônicas distorcem a forma de onda da tensão e da corrente, e são resultado da instalação de equipamentos com características não-lineares no sistema elétrico.

As distorções harmônicas, como pode ser visto na figura 21, vêm contra os objetivos da qualidade do fornecimento promovido por uma concessionária de energia elétrica, a qual deve fornecer aos seus consumidores uma tensão puramente senoidal, com amplitude e freqüências constantes.

Figura 21: Distorção harmônica

4.1.6 Flutuações de Tensão

Alguns tipos de carga, segundo Parada (1999, p. 45), em função das correntes absorvidas, provocam flutuações de tensão. Essas flutuações podem ser variações aleatórias, repetitivas ou esporádicas.

O efeito mais comum das flutuações de tensão é o *flicker* ou cintilação luminosa, causado por fornos a arco, máquinas de solda, laminadores, ferrovias e partidas repetitivas de motores. O *flicker*, ilustrado na figura 22, provoca sensação de incômodo visual, dependendo de sua amplitude e freqüência.

Figura 22: *Flicker*

4.1.7 Variações de Freqüência

Variações de freqüência são definidas como desvios no valor da freqüência fundamental (50 ou 60 Hz) de um sistema elétrico. Esses distúrbios são resultado principalmente de falhas na rede.

4.2 Conclusões do Capítulo

Este capítulo apresentou como a qualidade no fornecimento de energia elétrica tem ocupado cada vez mais importância tanto para as distribuidoras quanto para os consumidores. De acordo com Melo e Cavalcanti (2003, p. 477), uma interrupção no fornecimento de energia elétrica pode causar muitos prejuízos e transtornos aos consumidores, como perda de produtos manufaturados, negócios, sistemas de informações e outros. Essa pre-

ocupação tem se tornado mais evidente devido a diversos fatores, entre os quais podemos citar:

- O uso cada vez mais freqüente pelos consumidores industriais de equipamentos eletronicamente controlados e processos automatizados;
- Os dispositivos e equipamentos que vêm sendo utilizados atualmente nos sistemas de transmissão são mais sensíveis às flutuações de energia do que os dispositivos utilizados nas décadas passadas;
- As mudanças ocorridas no perfil do mercado de energia devido às instalações nas últimas décadas de indústrias no Brasil que são consumidoras eletrointensivas.

Programas de monitoramento de qualidade de energia elétrica, segundo Medeiros et al. (2003, p. 392), são de grande importância para as distribuidoras, pois a determinação de indicadores que expressem a qualidade de energia elétrica, como os apresentados na seção 4.1, tanto nos pontos de conexão com a transmissora quanto nos pontos de entrega aos consumidores, permitem estabelecer relações de causa-efeito que venham subsidiar ações de caráter preventivo ou corretivo para operação e planejamento do sistema elétrico.

5 *QEEWeb: Um Sistema de Monitoramento de Qualidade de Energia Elétrica Utilizando o Framework Seguraweb*

O QEEWeb é uma aplicação Web criada com o objetivo de avaliar a usabilidade do framework Seguraweb. A partir do QEEWeb, que é a primeira aplicação completa construida a partir do Seguraweb, é possível testar diferentes aspectos do framework e verificar se ele atende às necessidades da aplicação.

Este capítulo apresenta os aspectos relacionados à implementação do QEEWeb, que permite o diagnóstico e controle da qualidade de energia através do acompanhamento do sistema elétrico. O QEEWeb consiste na apresentação dos resultados da operação de medição de modo a obter um meio eficiente de monitorar o sistema elétrico e direcionar as ações voltadas para a manutenção da qualidade de energia elétrica fornecida, identificando problemas e responsabilidades.

5.1 Ambiente de Desenvolvimento

O QEEWeb é uma aplicação J2EE (*Java 2 Plataform, Enterprise Edition*) e utiliza um modelo chamado *J2EE Application Model*, ilustrado na figura 23. Nesse modelo, a lógica da aplicação é dividida em componentes de acordo com sua função, e cada um desses componentes é instalado em máquinas diferentes, dependendo da camada a qual ele pertence. Embora uma aplicação J2EE possa ter três ou quatro camadas, ela geralmente é considerada *3-tier* pois os componentes são distribuídos em apenas três máquinas: a máquina do cliente, o servidor de aplicações (*J2EE Server*) e o servidor de banco de dados (*Database Server*).

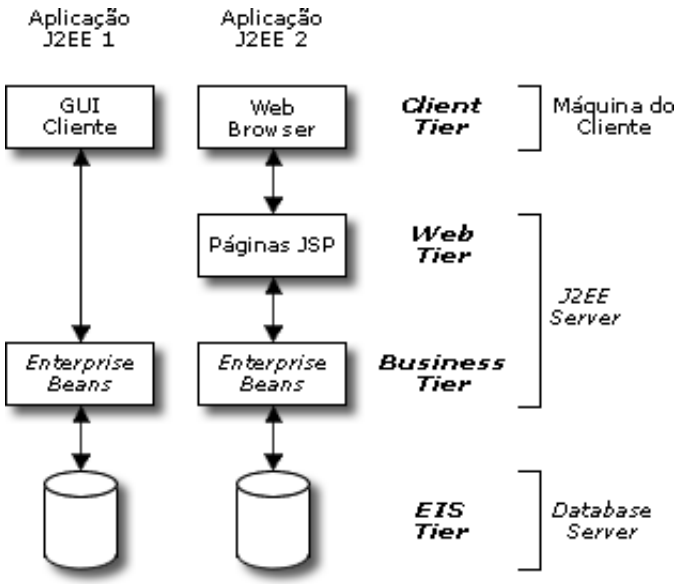

Figura 23: *J2EE Application Model*

5.1.1 Plataformas

O QEEWeb, como já citado anteriormente, é uma aplicação J2EE e, portanto, baseada na linguagem Java. Java é mais que uma linguagem de programação, ela é uma plataforma portável. Isso significa que o QEEWeb pode ser executado em qualquer sistema operacional que possua a máquina virtual Java instalada, incluindo o Windows e o Linux.

Neste trabalho utilizou-se os seguintes sistemas operacionais: Microsoft Windows 2000, na máquina de desenvolvimento e no servidor de aplicações, e Red Hat Linux 7.3, no servidor de banco de dados.

5.1.2 Linguagens de Programação

Aplicações J2EE são feitas de componentes, que podem ser desde *applets* executando na máquina do cliente a páginas JSP, *Enterprise JavaBeans*, ou ambos executando no servidor. Componentes J2EE são escritos em Java e compilados da mesma forma que qualquer programa nessa linguagem.

Neste trabalho optou-se pelo uso de páginas JSP, devido a seu bom desempenho e facilidade na criação de conteúdo dinâmico na Web. Seu funcionamento é simples: páginas JSP são arquivos de texto que contém a linguagem HTML tradicional junto com o código compilado Java. Quando o servidor recebe um requisição, apenas o código Java

é executado. Só então o conteúdo dinâmico gerado por esse código é combinado com o HTML e retornado para o usuário.

5.1.3 Ferramentas

No desenvolvimento do QEEWeb utilizou-se as seguintes ferramentas citadas abaixo:

- **NetBeans IDE (versão 3.5.1):** ambiente de desenvolvimento integrado (*Integrated Development Environment* - IDE) para Java;
- **Java 2 Platform, Standard Edition (J2SE - versão 1.4.2):** compilador, máquina virtual e APIs Java que permitem desenvolver e executar aplicações;
- **Java 2 Platform, Enterprise Edition (J2EE - versão 1.4):** serviços e APIs Java adicionais que simplificam o desenvolvimento de aplicações Web;
- **Java Modbus Library (jamod - versão 1.0):** implementação Java do protocolo Modicon Modbus;
- **Chart FX for Java:** componente Java para a criação de gráficos na Web;
- **Apache Tomcat (versão 4.1):** *container*[1] Java para JSP e Servlets;
- **Oracle 8*i* Database:** banco de dados relacional da Oracle.

5.2 Implementação

Uma característica importante do QEEWeb é o uso da Web como mecanismo primário de acesso dos usuários à aplicação, a partir de um navegador Web (*Web browser*). O uso das aplicações Web tem se tornado comum dentro das empresas devido a sua facilidade na implantação, uma vez que a maioria dos computadores possui um navegador Web instalado, tal como o Internet Explorer ou Netscape. Dessa forma, não é necessário a instalação de nenhum software adicional na máquina do usuário. Além disso, a atualização do software pode ser feita diretamente no servidor, de forma transparente para o usuário.

A figura 24 ilustra a arquitetura do QEEWeb. Basicamente, o servidor de aplicações (*J2EE Server*) coleta os dados de qualidade de energia elétrica dos medidores e os armazena no servidor de banco de dados (*Database Server*). O servidor de aplicações também

[1] *Containers* são entidades que provêm algum tipo de serviço para componentes Java.

publica os dados coletados através de páginas JSP, acessadas pelos usuários a partir de qualquer ponto na rede corporativa.

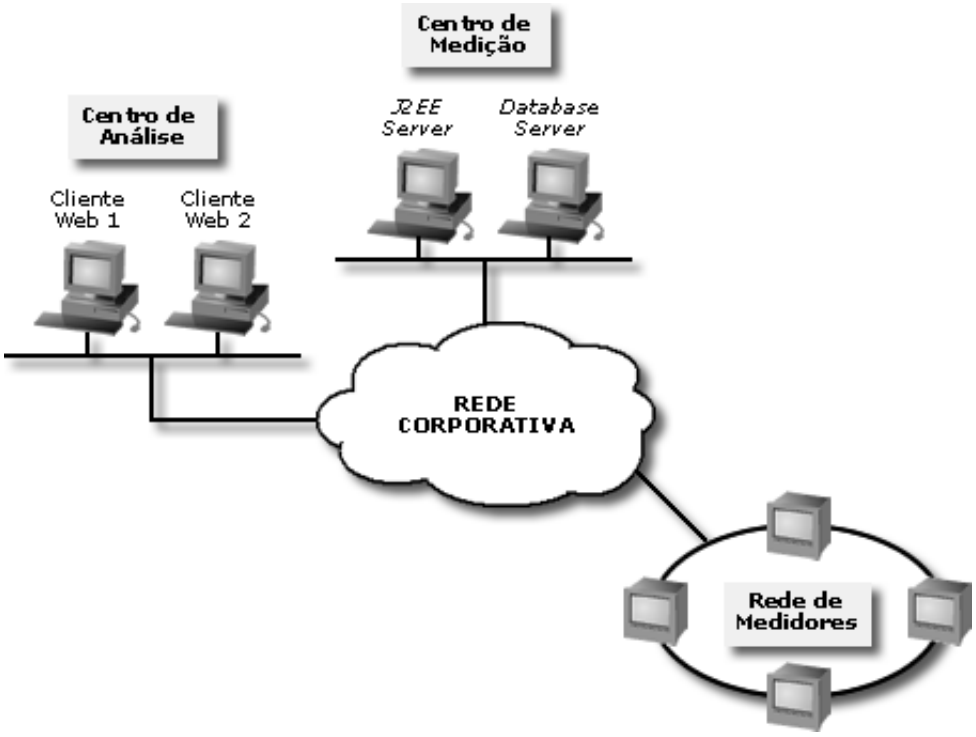

Figura 24: Arquitetura do QEEWeb

5.2.1 Aquisição de Dados

A aquisição dos dados de qualidade de energia elétrica do QEEWeb é feita a partir do protocolo de comunicação Modicon Modbus. O Modbus é um protocolo muito utilizado pela indústria na comunicação com dispositivos de controle. Ele foi originalmente criado para transferir dados entre controladores e sensores usando a porta serial RS-232. Atualmente, ele suporta tanto as portas seriais (RS-232 e RS-485) quanto a interface Ethernet RJ45.

O Modbus é um protocolo mestre/escravo (*master/slave*) onde o mestre inicia uma requisição e o escravo responde com os dados solicitados. Um mestre Modbus (*Modbus master*) é geralmente um software de controle e o escravo Modbus (*Modbus slave*) um dispositivo remoto, como os medidores de energia elétrica.

Os medidores de energia elétrica utilizados neste trabalho foram medidores ION 7500, da Power Measurement, especialmente desenvolvidos para o armazenamento de dados de qualidade de energia.

Uma vantagem dos medidores ION 7500 é a variedade de possibilidades de comunicação que ele oferece, como é mostrado na tabela 1. Assim, os dados podem ser coletados diretamente através de conexões na porta Ethernet ou, caso o medidor não esteja localizado na Internet, através de um modem conectado na porta serial (RS-232) do medidor. O modem utilizado pode ser tanto um modem convencional quanto um modem *wireless*, dependendo da situação.

Tabela 1: Portas de comunicação do ION 7500

Porta	Tipo	Padrão/Opcional
1	Porta selecionável RS-232/RS-485	Padrão
2	Porta óptica lrDA	Padrão
3	Modem interno	Opcional
4	Porta Ethernet 10BaseT (ou FL)	Opcional

Um módulo de software chamado *QEEWeb Data Collector*, implementado em Java, coleta os dados de todos os medidores cadastrados no sistema, através da porta Ethernet, e os armazena no banco de dados. O *QEEWeb Data Collector* utiliza a *Java Modbus Library* (jamod), que é uma implementação Java do protocolo Modbus.

5.2.2 Armazenamento Persistente

Como foi citado na seção anterior, os dados de qualidade de energia elétrica coletados pelo *QEEWeb Data Collector* são armazenados em um banco de dados. Neste trabalho, utilizou-se o banco de dados relacional Oracle 8*i* para o armazenamento persistente das informações. A escolha pelo Oracle foi feita considerando que esse banco de dados é o mais recomendado para aplicações de grande porte. Supondo que o QEEWeb colete mensalmente cerca de 1 MByte de dados por ponto de medição, e que podem existir sistemas na ordem de milhares de pontos, é fundamental um banco de dados com alto desempenho, escalabilidade e segurança. Além disso, o Seguraweb também utiliza esse banco de dados.

5.2.3 Interface Web

O QEEWeb publica relatórios e gráficos de qualidade de energia elétrica através de páginas JSP, que executam no servidor de aplicações do QEEWEb, mais especificamente em um *container* Java para JSP e Servlets, o Apache Tomcat. O Tomcat é um dos

containers mais utilizados para o desenvolvimento e publicação de aplicações e serviços Java na Internet.

É importante ressaltar que as páginas JSP não servem apenas para a publicação de gráficos e relatórios, todas as operações administrativas do sistema são feitas a partir da Web. Tais operações incluem cadastro, edição e remoção de pontos de medição, usuários e pastas.

A figura 25 ilustra a tela principal do QEEWeb. Nesta tela, o usuários podem acessar as operações administrativas do sistema, efetuar *login* e *logout*, fazer pesquisas, e gerar gráficos e relatórios.

Observa-se na figura 26 que ao cadastrar um usuário no sistema é preenchido o papel assumido por ele (cargo) e a pasta que ele pode acessar. O papel é necessário para o controle de acesso baseado em papéis fornecido pelo Seguraweb. Já a pasta é um mecanismo adicional de controle de acesso, que permite que o usuário acesse apenas os pontos de medição contidos nas pastas da qual ele tem acesso.

Como pode ser visto nas figuras 27 e 28, ao cadastrar um ponto de medição no sistema a pasta a qual ele pertence é informada. Caso o usuário tenha acesso a essa pasta, ele verá esse ponto de medição em sua estrutura de pastas.

O usuário pode então gerar diversos gráficos de qualidade de energia elétrica como os ilustrados nas figuras 29, 30, 31, 32, 33 e 34. Os gráficos do QEEWeb são gerados utilizando a ferramenta gráfica Chart FX for Java.

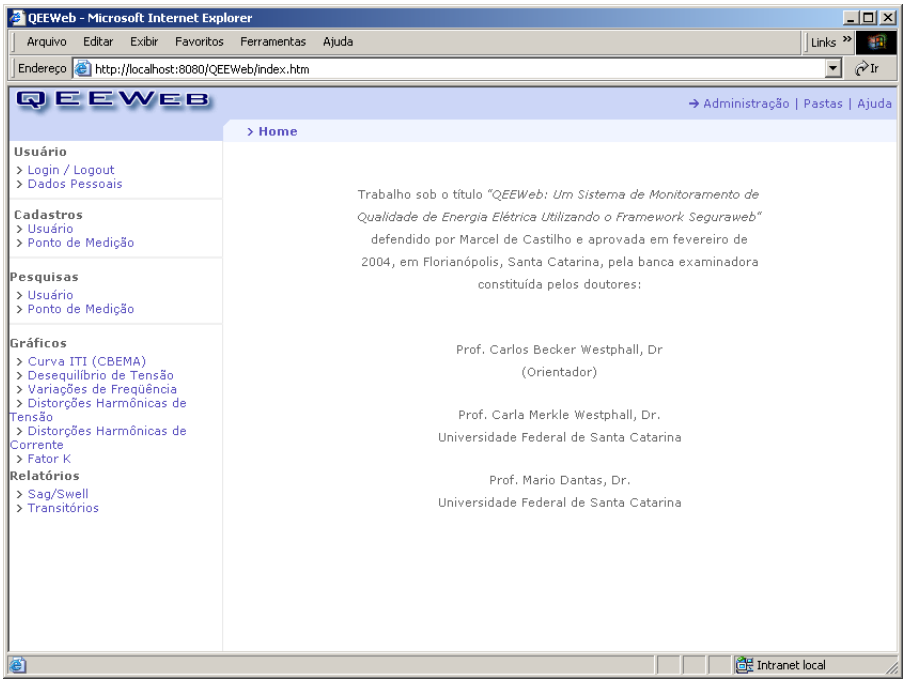

Figura 25: Tela principal do QEEWeb

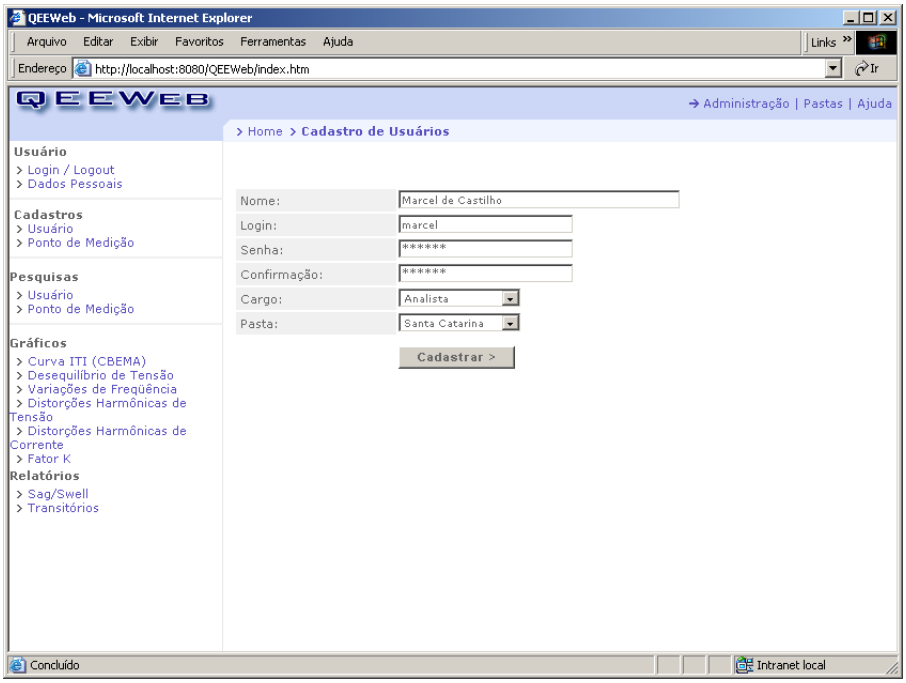

Figura 26: Formulário de cadastro de usuário

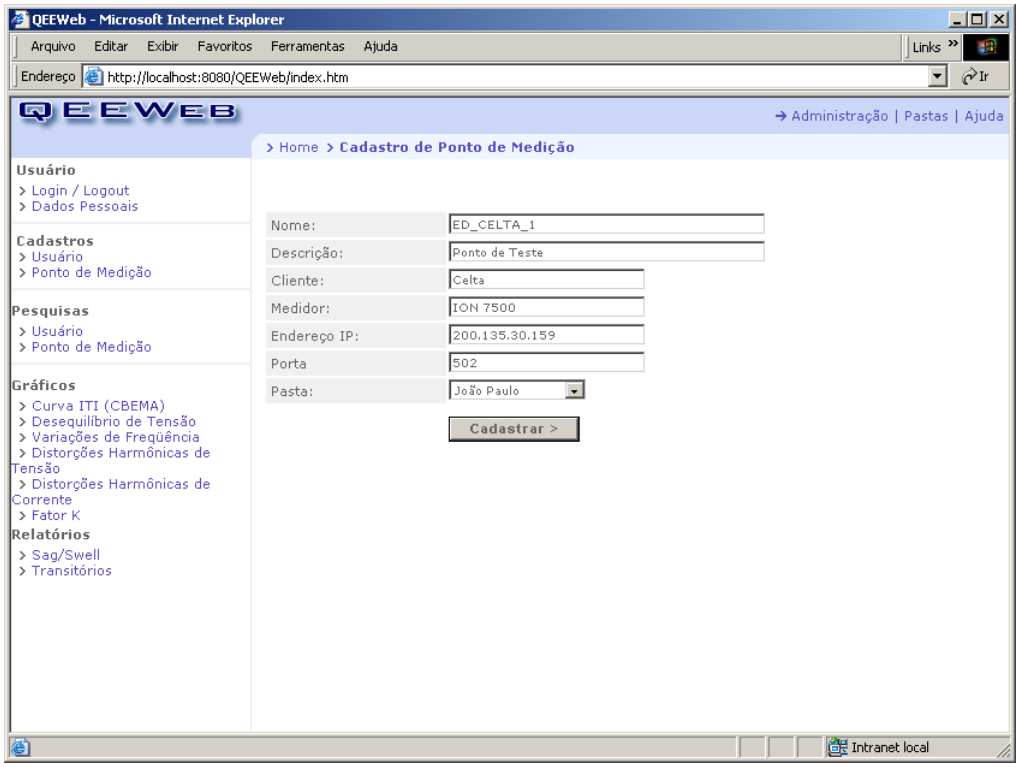

Figura 27: Formulário de cadastro de ponto de medição

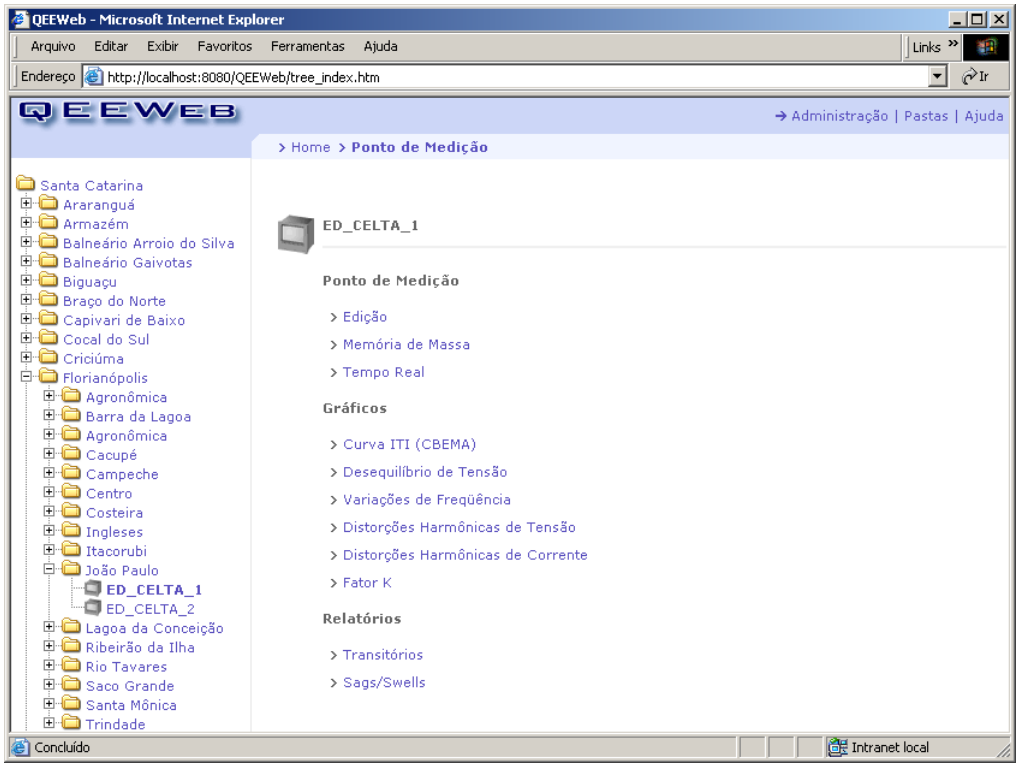

Figura 28: Pastas do usuário

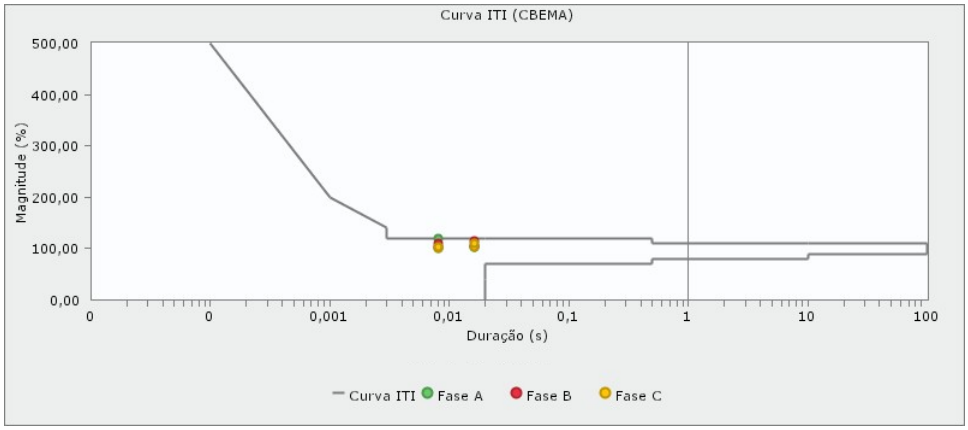

Figura 29: Gráfico da curva ITI (CBEMA)

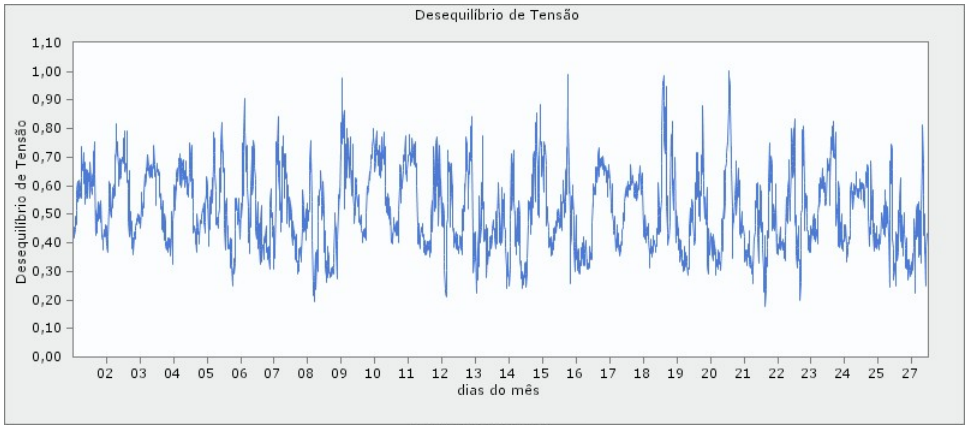

Figura 30: Gráfico de desequilíbrio de tensão

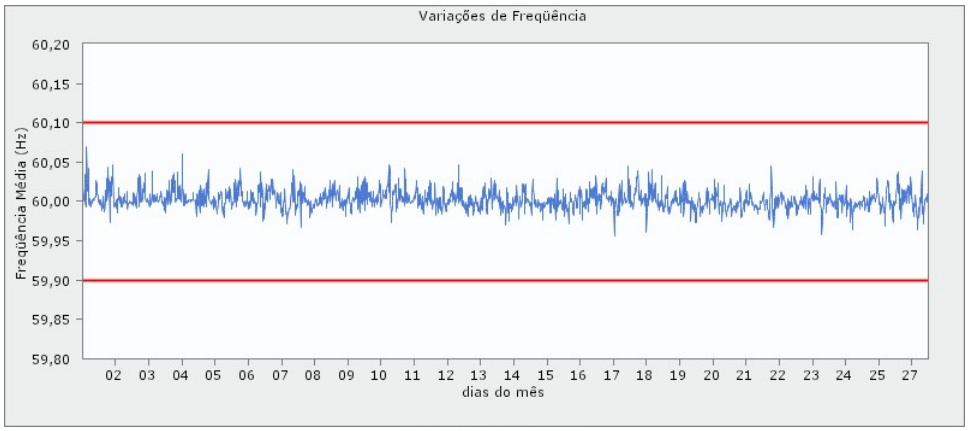

Figura 31: Gráfico de variações de freqüência

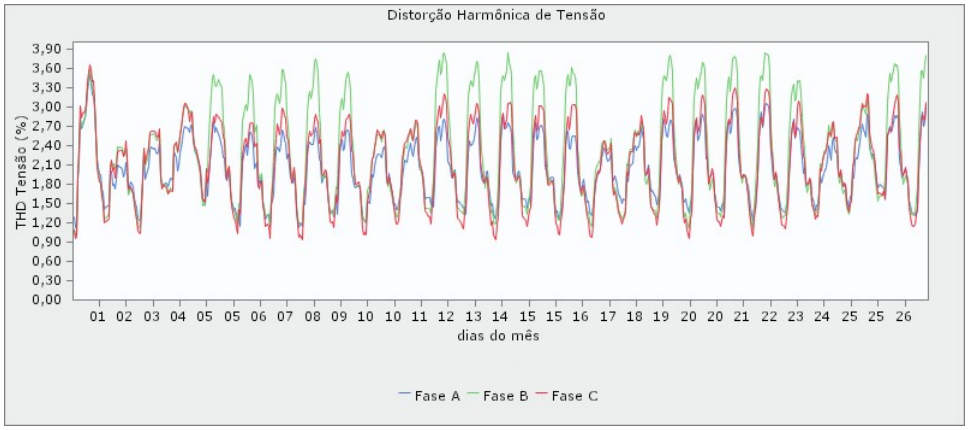

Figura 32: Gráfico de distorções harmônicas de tensão

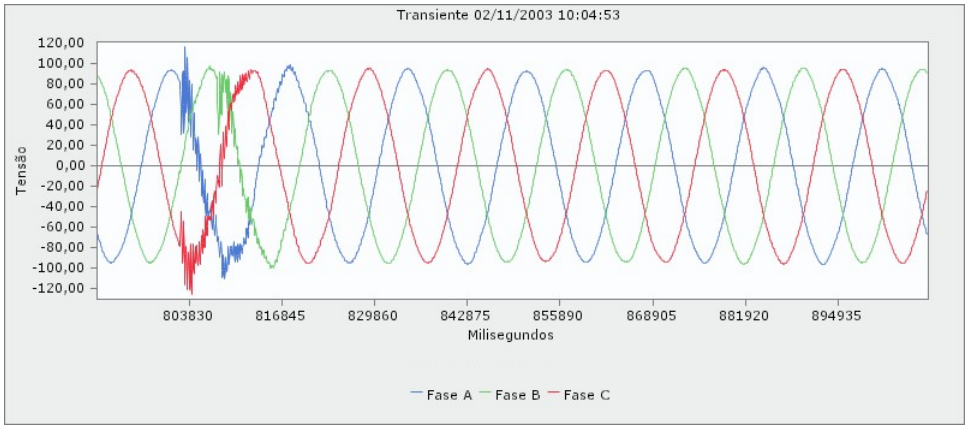

Figura 33: Forma de onda de um transitório

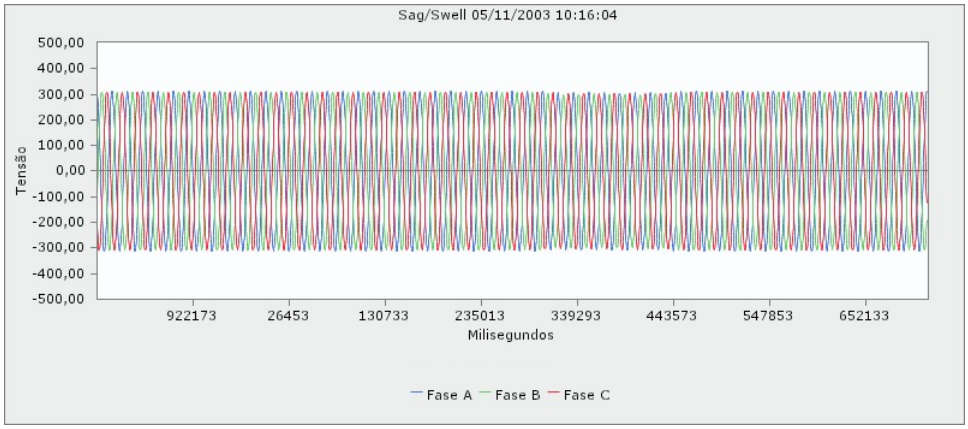

Figura 34: Forma de onda de um *sag*

5.3 Testes Realizados

Terminada a implementação do QEEWeb, foram feitos inúmeros testes não só com o objetivo de validar se o Seguraweb atende aos requisitos para o qual foi projetado mas também para testar as funcionalidades do sistema.

Para testar o esquema de autorização baseada em papéis do Seguraweb, foi criada uma estrutura hierárquica simples de papéis que o usuário pode assumir e operações que podem ser executadas por esses papéis no sistema. Os seguintes papéis foram criados: Visitante, Analista, Gerente e Administrador. Foi definido que todo usuário que acesse o QEEWeb é um visitante e não pode executar nenhuma operação no sistema, exceto *login*. Os analistas podem gerar gráficos e relatórios e gerentes podem executar operações de cadastro, edição e remoção de pontos de medição. Finalmente, o papel Administrador engloba todos os papéis e além das permissões herdadas, um administrador tem permissão para cadastrar, editar e remover usuários e pastas no sistema.

A autenticação de usuários no QEEWeb foi feita através de senhas escolhidas por esses usuários. Esse método de autenticação, apesar de bastante simples, é suficiente para garantir a segurança na maioria das aplicações Web. Com excessão dos visitantes, todos os usuários precisam se identificar no sistema para adquirir as permissões a eles concedidas.

Por fim, para testar as funcionalidades do QEEWeb, foram cadastrados dois pontos de medição que tiveram seus dados coletados por um período de dois meses.

5.4 Resultados Obtidos

Os testes realizados obtiveram o resultado esperado, isto é, tanto os mecanismos de segurança do Seguraweb quanto as funcionalidades do QEEWEb se comportaram como o desejado. Com relação ao controle de acesso, embora houvessem poucos papéis, as permissões foram facilmente atribuídas a esses papéis, utilizando principalmente o conceito de herança, que permite a sobreposição de responsabilidades. No futuro, se houvesse uma alteração, como a adição de novas funcionalidades ao sistema ou uma mudança na estrutura organizacional, ela seria tratada com um mínimo de modificação na aplicação.

O QEEWeb provou corresponder às expectativas e, com um melhor projeto, pode ser implementado comercialmente para uso das distribuidoras de energia elétrica. Entretanto, uma avaliação de desempenho do sistema não pode ser feita devido a pequena quantidade

de pontos de medição e, conseqüentemente, o pequeno volume de dados.

5.5 Conclusões do Capítulo

Este capítulo apresentou os aspectos relacionados à implementação do QEEWeb, uma aplicação Web que utiliza o framework Seguraweb para implementar seus mecanismos de segurança, principalmente o controle de acesso. Mostrou-se desde o ambiente de desenvolvimento da aplicação, até testes realizados de modo a verificar a usabilidade do framework utilizado, além dos resultados obtidos.

A maioria das ferramentas e tecnologias utilizadas no desenvolvimento do QEEWeb baseiam-se em padrões abertos, o que é um fator vantajoso e importante pois permite a portabilidade do software.

O QEEWeb possui a maioria das funcionalidades dos softwares de qualidade de energia elétrica existentes atualmente no mercado e permite um diagnóstico preciso da qualidade de energia nos pontos de medição, com a vantagem de ser uma aplicação inteiramente baseada na Web e independente de plataforma.

6 Conclusão

Modelos de segurança, segundo Obelheiro (2001, p. 76) desempenham um papel prepoderante na definição de políticas de sergurança, que são uma peça-chave em sistemas de informação modernos. O uso crescente de arquiteturas distribuídas baseadas em redes de larga escala, como a Internet e as grandes Intranets coporativas, aumentam a necessidade de políticas de controle de acesso mais rígidas do que as tradicionais políticas baseadas em modelos discricionários. O modelo unificado RBAC-NIST representa a classe de modelos baseados em papéis, que surge como uma alternativa ao mesmo tempo flexível e rigorosa.

O framework Seguraweb, retira dos desenvolvedores de software a preocupação de implementar os mecanismos de segurança das aplicações, principalmente o controle de acesso, que adota políticas baseadas em papéis, mais especificamente o modelo unificado RBAC-NIST.

O objetivo principal deste trabalho foi avaliar a usabilidade do framework Seguraweb através de uma aplicação Web. O protótipo implementado, baseado em um sistema de monitoramento de qualidade de energia elétrica, enfatiza a viabilidade do framework e a adequação do modelo RBAC à aplicação.

Embora a proposta inicial deste trabalho fosse apenas avaliar o framework Seguraweb, desde o início procurou-se apresentar uma aplicação que abordasse um problema na área de energia elétrica, como a qualidade de energia. A qualidade hoje, segundo Parada (1999, p. 23), é uma preocupação de todos os agentes do setor: os clientes manifestam sua sensibilidade e demandam melhores níveis de qualidade de serviço; fabricantes de equipamentos precisam de normas que limitem as perturbações na rede e que especifiquem os níveis de imunidade dos equipamentos que fabricam; as empresas distribuidoras tem necessidade de sinais para investir em operação, manutenção e qualidade de serviço, além de precisar monitorar os clientes perturbadores; a regulamentação deve outorgar as regras e limites que possibilitem um nível adequado de qualidade de serviço.

O QEEWEb permite o diagnóstico e controle da qualidade de energia através do

acompanhamento do sistema elétrico. Assim, é possível identificar os principais distúrbios tanto nos pontos de conexão com a transmissora quanto nos pontos de entrega aos consumidores, facilitando o trabalho das distribuidoras que podem então direcionar as ações voltadas para a manutenção da qualidade de serviço. Por ser uma aplicação inteiramente baseada na Web e independente de plataforma, o QEEWeb oferece mais flexibilidade do que os outros softwares de monitoramento existentes no mercado.

Existem diversas possibilidades de continuação deste trabalho. Com relação ao Seguraweb, alguns mecanismos de segurança poderiam ser melhorados, com a adição de novos métodos de autenticação e um subsistema de auditoria. Embora o Seguraweb tenha atendido todas as necessidades do QEEWeb, novas funcionalidades poderia ser criadas de modo a contribuir com a evolução do framework.

Outra perspectiva de continuidade é no próprio QEEWeb, que poderia ser melhor estudado e aperfeiçoado. Alguns fenômenos de qualidade de energia importantes ainda não são tratados pelo QEEWeb, como os índices de continuidade de fornecimento, que refletem o comportamento do sistema em relação às interrupções de longa duração. Além disso, a aquisição de dados ainda é bastante simples pois permite a coleta apenas de pontos de medição diretamente conectados na Intranet. A coleta de dados de pontos de medição localizados fora do ambiente da empresa é essencial, uma vez que os pontos de medição dos clientes (consumidores livres) não são alcançáveis pela rede. A coleta de dados nesse caso geralmente é feita usando linha discada convencional ou redes *wireless*.

Referências

ARMANINI, Kátyra Kowalski. *Seguraweb*: um Framework RBAC para Aplicações Web. 128 p. Dissertação (Mestrado em Ciência da Computação) — Universidade Federal de Santa Catarina, Florianópolis, 2002.

BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. *UML*: Guia do Usuário. Rio de Janeiro: Editora Campus, 2000. 472 p.

BOSCH, Jan et al. *Object-Oriented Frameworks*: Problems & Experiences. 18 p. — University of Karlskrona/Ronneby, Ronneby, 1997.

FERRAIOLO, David F.; BARKLEY, John. Specifying and Managing Role-Based Access Control within a Corporate Intranet. In: SECOND ACM WORKSHOP ON ROLE-BASED ACCESS CONTROL. Fairfax: ACM Press, 1997. p. 77–82.

FERRAIOLO, David F.; CUGINI, Janet A.; KUHN, D. Richard. Role-Based Access Control: Features and Motivations. In: COMPUTER SECURITY APPLICATIONS CONFERENCE. Nova Orleans: IEEE Computer Society Press, 1995.

FERRAIOLO, David F. et al. Proposed NIST Standart for Role-Based Access Control. In: ACM TRANSACTIONS ON INFORMATION AND SYSTEM SECURITY. [S.l.]: ACM Press, 2001. p. 224–274.

GARLAN, David; SHAW, Mary. *An Introduction to Software Architecture*. 39 p. — Carnegie Mellon University, Pittsburgh, 1994.

INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEER. *IEEE Std 1159-1995*: IEEE Recommended Practice for Monitoring Electric Power Quality. Nova York, 1995. 70 p.

JESUS, Nelson C. de; NONENMACHER, Cassio F. B.; OLIVEIRA, Hermes R. P. M. de. Análise da Influência de Cargas Não-lineares: Avaliação dos Efeitos e Limites de Cortes de Tensão (Voltage Notching). In: V SEMINÁRIO BRASILEIRO SOBRE QUALIDADE DA ENERGIA ELÉTRICA. Aracaju, 2003. p. 13–18.

LARMAN, Craig. *Utilizando UML e Padrões*: Uma Introdução à Análise e ao Projeto Orientado a Objetos. Porto Alegre: Editora Bookman, 2000. 492 p.

MEDEIROS, Marcos Oriano B. de et al. Monitoração da Qualidade de Energia de um Sistema de Distribuição. In: V SEMINÁRIO BRASILEIRO SOBRE QUALIDADE DA ENERGIA ELÉTRICA. Aracaju, 2003. p. 387–392.

MELO, Miguel O. B. C.; CAVALCANTI, Guilherme A. Avaliação do Impacto da Qualidade de Energia Elétrica no Mercado e na Produção Industrial: Análise e Metodologia. In: V SEMINÁRIO BRASILEIRO SOBRE QUALIDADE DA ENERGIA ELÉTRICA. Aracaju, 2003. p. 477–482.

OBELHEIRO, Rafael Rodrigues. *Modelos de Segurança Baseados em Papéis para Sistemas de Larga Escala*: A Proposta RBAC-JaCoWeb. 89 p. Dissertação (Mestrado em Engenharia Elétrica) — Universidade Federal de Santa Catarina, Florianópolis, 2001.

PARADA, Gabriel Maurício Olguín. *Proposta de Regulamentação da Qualidade de Serviço em Sistemas de Distribuição*. 145 p. Dissertação (Mestrado em Engenharia Elétrica) — Universidade Federal de Santa Catarina, Florianópolis, 1999.

SILVA, Ricardo Pereira e. *Suporte ao desenvolvimento e uso de frameworks e componentes*. 262 p. Tese (Doutorado em Ciência da Computação) — Universidade Federal do Rio Grande do Sul, Porto Alegre, 2000.

TALIGENT. *Building Object-Oriented Frameworks*. 1994. White Paper.

WIRFS-BROCK, Allen et al. Designing reusable designs (panel session): experiences designing object-oriented frameworks. In: EUROPEAN CONFERENCE ON OBJECT-ORIENTED PROGRAMMING ADDENDUM : SYSTEMS, LANGUAGES, AND APPLICATIONS. Ottawa: ACM Press, 1991. p. 19–24.

ANEXO A – Código Fonte

```
package qeeweb;

public interface Authentication {
    public boolean isValidUser(String _OID, AuthenticationMethod _authenticationMethod);
}

package qeeweb;

public interface AuthenticationMethod {}

package qeeweb;

import java.sql.Connection; import java.security.MessageDigest;

public class AuthenticationQEEWeb implements Authentication {
    protected Connection conn;

    /**
     * Construtor
     */
    public AuthenticationQEEWeb(Connection _conn) {
        conn = _conn;
    }

    /**
     * Metodos herdados de Authentication
     */
    public boolean isValidUser(String _OID, AuthenticationMethod _authenticationMethod) {
        String password = ((Password) _authenticationMethod).getPassword();
        UserQEEWebProxy user = new UserQEEWebProxy(_OID, conn);
        MessageDigest md5 = null;
```

```
        try {
            md5 = MessageDigest.getInstance("MD5");
            md5.update(password.getBytes());
        } catch (Exception e) {
            System.err.println(e.toString());
        }

        if (user.getSenha().equals(new String(md5.digest()))) {
            return true;
        }

        if (user.getSenha().equals(password)) {
            return true;
        }

        return false;
    }
}
```

```
package qeeweb;

public interface Authorization {
    public boolean hasPermission(UserInterface _user, PermissionInterface _permission);

    public boolean hasPermission(String _OID, String _operation);
}
```

```
package qeeweb;

import java.sql.Connection; import java.util.Enumeration;

public class AuthorizationQEEWeb implements Authorization {
    protected Connection conn;

    /**
     * Construtor
     */
    public AuthorizationQEEWeb(Connection _conn) {
        conn = _conn;
    }

    /**
     * Metodos herdados de Authorization
```

```
     */
    public boolean hasPermission(UserInterface _user, PermissionInterface _permission) {
        return hasPermission(_user.getOID(), _permission.getOID());
    }

    public boolean hasPermission(String _userOID, String _permissionOID) {
        UserQEEWebProxy user = new UserQEEWebProxy(_userOID, conn);
        RoleList roles = user.getRoleList();
        RoleProxy roleProxy = null;
        PermissionProxy permissionProxy = null;
        PermissionList permissions = null;

        for (Enumeration e = roles.elements(); e.hasMoreElements();) {
            roleProxy = (RoleProxy) e.nextElement();
            permissions = roleProxy.getPermissionList();

            for (Enumeration e2 = permissions.elements(); e2.hasMoreElements();) {
                permissionProxy = (PermissionProxy) e2.nextElement();

                if (permissionProxy.getOID().equals(_permissionOID)) {
                    return true;
                }
            }
        }

        return false;
    }
}
```

```
package qeeweb;

public class DBConstants {
    public static final String DRIVER = "oracle.jdbc.driver.OracleDriver";
    public static final String CONN_URL = "jdbc:oracle:thin:@192.168.0.2:1521:dbdois";
    public static final String LOGIN = "marcel";
    public static final String SENHA = "marcel";
}
```

```
package qeeweb;

import java.sql.Connection;

import java.sql.Statement;
```

```
import java.sql.DriverManager;

import java.sql.SQLException;

import oracle.jdbc.driver.*;

public class DBManager {
    protected Connection conn;
    protected Statement stmt;

    /**
     * Construtores
     */
    public DBManager() {
        try {
            Class.forName(DBConstants.DRIVER);
        } catch (ClassNotFoundException e) {
            System.err.println(e.toString());
        }

        try {
            conn = DriverManager.getConnection(DBConstants.CONN_URL,
            DBConstants.LOGIN, DBConstants.SENHA);
            stmt = conn.createStatement();
        } catch (SQLException e) {
            System.err.println(e.toString());
        }
    }
}

package qeeweb;

import java.sql.ResultSet;

public class DBPontoManager extends DBManager {
    PontoProxy pontoProxy;

     /**
     * Construtores
     */
    public DBPontoManager() {
        super();
```

```
    createProxy();
}

/**
 * Get Methods
 */
public PontoInterface getPonto(String _OID) {
    return new PontoProxy(_OID, conn);
}

/**
 * Metodos de acesso ao proxy
 */
public void insertPonto(Ponto _ponto) {
    pontoProxy.insertPonto(_ponto);
}

public void insertPonto(String _OID, String _descricao, String _cliente,
String _medidor, String _enderecoIP, String _porta, int _cdPasta, boolean
_habilitado) {
    Ponto ponto = new Ponto(_OID, _descricao, _cliente, _medidor,
    _enderecoIP, _porta, _cdPasta, _habilitado);
    insertPonto(ponto);
}

public void updatePonto(Ponto _ponto) {
    pontoProxy.updatePonto(_ponto);
}

public void updatePonto(String _OID, String _nome, String _descricao, String
_cliente, String _medidor, String _enderecoIP, String _porta, int _cdPasta,
boolean _habilitado) {
    Ponto ponto = new Ponto(_OID, _descricao, _cliente, _medidor,
    _enderecoIP, _porta, _cdPasta, _habilitado);
    updatePonto(ponto);
}

public void deletePonto(Ponto _ponto) {
    pontoProxy.deletePonto(_ponto);
}

public void deletePonto(String _OID) {
    pontoProxy.deletePonto(_OID);
```

```
}

/**
 * Metodos protegidos
 */
protected void createProxy() {
    pontoProxy = new PontoProxy(conn);
}

/**
 * Outros
 */
public ResultSet pesqMedia(String _OID) {
    ResultSet rs = null;
    String sql = "select to_char(dt_hr_instante, 'dd') dia, vl_v_des,
    vl_freq from tb_media where oid_ponto='" + _OID + "'";

    try {
        rs = stmt.executeQuery(sql);
    } catch (Exception e) {
        System.err.println(e.toString());
    }

    return rs;
}

public ResultSet pesqHarmonica(String _OID) {
    ResultSet rs = null;
    String sql = "select to_char(dt_hr_instante, 'dd') dia, vl_v1_thd,
    vl_v2_thd, vl_v3_thd, vl_i1_thd, vl_i2_thd, vl_i3_thd, vl_i1_k, vl_i2_k,
    vl_i3_k from tb_harmonica where oid_ponto='" + _OID + "'";

    try {
        rs = stmt.executeQuery(sql);
    } catch (Exception e) {
        System.err.println(e.toString());
    }

    return rs;
}

public ResultSet pesqSagsSwells(String _OID) {
    ResultSet rs = null;
```

```
    String sql = "select to_char(dt_hr_instante, 'dd/mm/yyyy hh24:mi:ss')
    dt_hr_instante, vl_dist_dur, vl_v1_min, vl_v1_max, vl_v1_avg, vl_v2_min,
    vl_v2_max, vl_v2_avg, vl_v3_min, vl_v3_max, vl_v3_avg from tb_sag_swell
    where oid_ponto='" + _OID + "'";

    try {
        rs = stmt.executeQuery(sql);
    } catch (Exception e) {
        System.err.println(e.toString());
    }

    return rs;
}

public ResultSet pesqTransitorios(String _OID) {
    ResultSet rs = null;
    String sql = "select to_char(dt_hr_instante, 'dd/mm/yyyy hh24:mi:ss')
    dt_hr_instante, vl_v1_dur, vl_v1_max, vl_v2_dur, vl_v2_max, vl_v3_dur,
    vl_v3_max from tb_transitorio where oid_ponto='" + _OID + "'";

    try {
        rs = stmt.executeQuery(sql);
    } catch (Exception e) {
        System.err.println(e.toString());
    }

    return rs;
}

public ResultSet consOndaSagSwell(String _OID) {
    ResultSet rs = null;
    String sql = "select to_char(dt_hr_instante, 'dd/mm/yyyy hh24:mi:ss')
    dt_hr_instante, vl_milisegundos, vl_v1, vl_v2, vl_v3, vl_i1, vl_i2,
    vl_i3 from tb_sag_swell_onda where oid_ponto='" + _OID + "'";

    try {
        rs = stmt.executeQuery(sql);
    } catch (Exception e) {
        System.err.println(e.toString());
    }

    return rs;
}
```

```
    public ResultSet consOndaTransitorio(String _OID) {
        ResultSet rs = null;
        String sql = "select to_char(dt_hr_instante, 'dd/mm/yyyy hh24:mi:ss')
        dt_hr_instante, vl_milisegundos, vl_v1, vl_v2, vl_v3, vl_i1, vl_i2,
        vl_i3 from tb_transitorio_onda where oid_ponto='" + _OID + "'";

        try {
            rs = stmt.executeQuery(sql);
        } catch (Exception e) {
            System.err.println(e.toString());
        }

        return rs;
    }
}

package qeeweb;

import java.security.MessageDigest;

public class DBUserManager extends DBManager {
    UserProxy userProxy;

     /**
     * Construtores
     */
    public DBUserManager() {
        super();
        createProxy();
    }

    /**
     * Get Methods
     */
    public UserInterface getUser(String _OID) {
        return new UserProxy(_OID, conn);
    }

    /**
     * Metodos de acesso ao proxy
     */
    public void insertUser(User _user) {
```

```
    userProxy.insertUser(_user);
}

public void insertUser(String _OID, String _password, String _name) {
    User user = new User(_OID, new Password(getCipherPassword(_password)),
    _name);
    insertUser(user);
}

public void updateUser(User _user) {
    userProxy.updateUser(_user);
}

public void updateUser(String _OID, String _password, String _name) {
    User user = new User (_OID, new Password(getCipherPassword(_password)),
    _name);
    updateUser(user);
}

public void deleteUser(User _user) {
    userProxy.deleteUser(_user);
}

public void deleteUser(String _OID) {
    userProxy.deleteUser(_OID);
}

/**
 * Metodos protegidos
 */
protected void createProxy() {
    userProxy = new UserProxy(conn);
}

protected String getCipherPassword(String _password) {
    MessageDigest md5 = null;

    try {
        md5 = MessageDigest.getInstance("MD5");
        md5.update(_password.getBytes());
    } catch (Exception e) {
        System.err.println(e.toString());
    }
```

```
        return new String(md5.digest());
    }
}

package qeeweb;

import java.sql.ResultSet;

public class DBUserQEEWebManager extends DBUserManager {
     /**
     * Construtores
     */
    public DBUserQEEWebManager() {
        super();
    }

    /**
     * Get Methods
     */
    public UserInterface getUser(String _OID) {
        return new UserQEEWebProxy(_OID, conn);
    }

    /**
     * Metodos de acesso ao proxy
     */
    public void insertUser(String _OID, String _senha, String _nome, int _cdPasta) {
        UserQEEWeb user = new UserQEEWeb(_OID, getCipherPassword(_senha), _nome,
        _cdPasta);
        insertUser(user);
    }

    public void updateUser(String _OID, String _senha, String _nome, int _cdPasta) {
        UserQEEWeb user = new UserQEEWeb(_OID, getCipherPassword(_senha), _nome,
        _cdPasta);
        updateUser(user);
    }

    /**
     * Metodos protegidos
     */
    protected void createProxy() {
```

```
        userProxy = new UserQEEWebProxy(conn);
    }

    /**
     * Outros
     */
    public boolean isValidUser(String _OID, String _senha) {
        AuthenticationQEEWeb authentication = new AuthenticationQEEWeb(conn);
        Password password = new Password(_senha);

        return authentication.isValidUser(_OID, password);
    }

    public boolean hasPermission(String _userOID, String _permissionOID) {
        AuthorizationQEEWeb authorization = new AuthorizationQEEWeb(conn);

        return authorization.hasPermission(_userOID, _permissionOID);
    }

    public ResultSet pesqLog() {
        ResultSet rs = null;
        String sql = "select to_char(dt_hr_instante, 'dd/mm/yyyy hh24:mi:ss')
        dt_hr_instante, oid_usuario, oid_permissao, str_descricao,
        str_endereco_ip from tb_log";

        try {
            rs = stmt.executeQuery(sql);
        } catch (Exception e) {
            System.err.println(e.toString());
        }

        return rs;
    }
}

package qeeweb;

public class Password implements AuthenticationMethod {
    private String password;

    /**
     * Construtores
     */
```

```
    public Password(String _password) {
        password = _password;
    }

    /**
     * Get Methods
     */
    public String getPassword() {
        return password;
    }
}

package qeeweb;

public class PBE implements AuthenticationMethod {
    private Password password;
    private byte[] salt;
    private int iterationCount;

    /**
     * Construtores
     */
    public PBE(Password _password, byte[] _salt, int _iterationCount) {
        password = _password;
        salt = _salt;
        iterationCount = _iterationCount;
    }

    /**
     * Get Methods
     */
    public Password getPassword() {
        return password;
    }

    public byte[] getSalt() {
        return salt;
    }

    public int getIterationCount() {
        return iterationCount;
    }
}
```

```
package qeeweb;

public class Permission implements PermissionInterface {
    private String OID;
    private String descr;

    /**
     * Construtores
     */
    public Permission(String _OID) {
        OID = _OID;
    }

    public Permission(String _OID, String _descr) {
        OID = _OID;
        descr = _descr;
    }

    /**
     * Get Methods
     */
    public String getOID() {
        return OID;
    }

    public String getDescr() {
        return descr;
    }
}

package qeeweb;

public interface PermissionInterface {
    public String getOID();

    public String getDescr();
}

package qeeweb;

import java.util.Vector;
```

```
import java.util.Enumeration;

public class PermissionList {
    private Vector permissions;

    public PermissionList() {
        permissions = new Vector();
    }

    public PermissionInterface get(int _index) {
        return (PermissionInterface) permissions.elementAt(_index);
    }

    public Enumeration elements() {
        return permissions.elements();
    }

    public void put(PermissionInterface _permission) {
        permissions.addElement(_permission);
    }

    public void put(PermissionInterface _permission, int _index) {
        permissions.insertElementAt(_permission, _index);
    }
}

package qeeweb;

import java.sql.Connection;

public class PermissionProxy extends VirtualProxy implements
PermissionInterface {
    protected Connection conn;

    /**
     * Construtores
     */
    public PermissionProxy(String _OID, Connection _conn) {
        super(_OID);
        conn = _conn;
    }

    /**
```

```
     * Get Methods
     */
    public String getDescr() {
        return ((Permission) getRealSubject()).getDescr();
    }

    /**
     * Metodos herdados de VirtualProxy
     */
    public String getBrokerId() {
        return "RelationalBrokerPermission";
    }

     /**
     * Metodos de acesso ao broker
     */
    public PersistentBroker createBroker() {
        return RelationalBrokerPermission.Instance(conn);
    }

    public void insertPermission(Permission _permission) {
        ((RelationalBrokerPermission) getBroker()).insertPermission(_permission);
    }

    public void updatePermission(Permission _permission) {
        ((RelationalBrokerPermission) getBroker()).updatePermission(_permission);
    }

    public void deletePermission(Permission _permission) {
        ((RelationalBrokerPermission) getBroker()).deletePermission(_permission);
    }

    public void deletePermission(String _OID) {
        ((RelationalBrokerPermission) getBroker()).deletePermission(_OID);
    }
}

package qeeweb;

abstract public class PersistentBroker {
    public Object objectWith(String _OID) {
        Object obj = inCache(_OID);
```

```
        if (obj != null) {
            return obj;
        } else {
            return materializeWith(_OID);
        }
    }

    /**
     * Metodos a serem implementados por subclasses
     */
    public Object inCache(String _OID) {
        return null;
    }

    public Object materializeWith(String _OID) {
        return null;
    }
}

package qeeweb;

import java.util.Hashtable;

public class Ponto implements PontoInterface {
    private String OID;
    private String descricao;
    private String cliente;
    private String medidor;
    private String enderecoIP;
    private String porta;
    private int cdPasta;
    private boolean habilitado;
    private Hashtable erros;

    /**
     * Construtores
     */
    public Ponto() {
        OID = "";
    }

    public Ponto(String _OID) {
        OID = _OID;
```

```
}

public Ponto(String _OID, String _descricao, String _cliente, String
_medidor, String _enderecoIP, String _porta, int _cdPasta, boolean
_habilitado) {
    OID = _OID;
    descricao = _descricao;
    cliente = _cliente;
    medidor = _medidor;
    enderecoIP = _enderecoIP;
    porta = _porta;
    cdPasta = _cdPasta;
    habilitado = _habilitado;
    erros = new Hashtable();
}

/**
 * Get Methods
 */
public String getOID() {
    return OID;
}

public String getDescricao() {
    return descricao;
}

public String getCliente() {
    return cliente;
}

public String getMedidor() {
    return medidor;
}

public String getEnderecoIP() {
    return enderecoIP;
}

public String getPorta() {
    return porta;
}
```

```
public int getCdPasta() {
    return cdPasta;
}

public boolean getHabilitado() {
    return habilitado;
}

public String getErro(String s) {
    String erro = (String) erros.get(s);

    if (erro == null) {
        erro = "";
    }

    return erro;
}

/**
 * Set Methods
 */
public void setOID(String _OID) {
    OID = _OID;
}

public void setDescricao(String _descricao) {
    descricao = _descricao;
}

public void setCliente(String _cliente) {
    cliente = _cliente;
}

public void setMedidor(String _medidor) {
    medidor = _medidor;
}

public void setEnderecoIP(String _enderecoIP) {
    enderecoIP = _enderecoIP;
}

public void setPorta(String _porta) {
    porta = _porta;
```

```
    }

    public void setCdPasta(int _cdPasta) {
        cdPasta = _cdPasta;
    }

    public void setHabilitado(boolean _habilitado) {
        habilitado = _habilitado;
    }

    /**
     * Verifica se o usuario e valido
     */
    public boolean ehValido() {
        if (OID.equals("")) {
            erros.put("OID", "*");
            return false;
        }

        if (cdPasta < 0) {
            erros.put("pasta", "*");
            return false;
        }

        return true;
    }
}

package qeeweb;

public interface PontoInterface {
    public String getOID();

    public String getDescricao();

    public String getCliente();

    public String getMedidor();

    public String getEnderecoIP();

    public String getPorta();
```

```
    public int getCdPasta();

    public boolean getHabilitado();

    public boolean ehValido();
}

package qeeweb;

import java.sql.Connection;

public class PontoProxy extends VirtualProxy implements
PontoInterface {
    protected Connection conn;

    /**
     * Construtores
     */
    public PontoProxy() {
        super("");
    }

    public PontoProxy(Connection _conn) {
        super("");
        conn = _conn;
    }

    public PontoProxy(String _OID) {
        super(_OID);
    }

    public PontoProxy(String _OID, Connection _conn) {
        super(_OID);
        conn = _conn;
    }

    /**
     * Metodos do objeto real
     */
    public String getDescricao() {
        return ((Ponto) getRealSubject()).getDescricao();
    }
```

```
public String getCliente() {
    return ((Ponto) getRealSubject()).getCliente();
}

public String getMedidor() {
    return ((Ponto) getRealSubject()).getMedidor();
}

public String getEnderecoIP() {
    return ((Ponto) getRealSubject()).getEnderecoIP();
}

public String getPorta() {
    return ((Ponto) getRealSubject()).getPorta();
}

 public int getCdPasta() {
    return ((Ponto) getRealSubject()).getCdPasta();
}

public boolean getHabilitado() {
    return ((Ponto) getRealSubject()).getHabilitado();
}

public boolean ehValido() {
    return ((Ponto) getRealSubject()).ehValido();
}

/**
 * Metodos herdados de VirtualProxy
 */
public String getBrokerId() {
    return "RelationalBrokerPonto";
}

public PersistentBroker createBroker() {
    return RelationalBrokerPonto.Instance(conn);
}

/**
 * Metodos de acesso ao broker
 */
public void insertPonto(Ponto _ponto) {
```

```
        ((RelationalBrokerPonto) getBroker()).insertPonto(_ponto);
    }

    public void updatePonto(Ponto _ponto) {
        ((RelationalBrokerPonto) getBroker()).updatePonto(_ponto);
    }

    public void deletePonto(Ponto _ponto) {
        ((RelationalBrokerPonto) getBroker()).deletePonto(_ponto);
    }

    public void deletePonto(String _OID) {
        ((RelationalBrokerPonto) getBroker()).deletePonto(_OID);
    }
}

package qeeweb;

public class PPK implements AuthenticationMethod {
    private Password password;
    private PrivateKey privateKey;

    /**
     * Construtores
     */
    public PPK(Password _password, PrivateKey _privateKey) {
        password = _password;
        privateKey = _privateKey;
    }

    /**
     * Get Methods
     */
    public Password getPassword() {
        return password;
    }

    public PrivateKey getPrivateKey() {
        return privateKey;
    }
}

package qeeweb;
```

```
public class PrivateKey implements AuthenticationMethod {
    private byte[] privateKey;

    public PrivateKey(byte[] _privateKey) {
        privateKey = _privateKey;
    }

    public byte[] getPrivateKey() {
        return privateKey;
    }
}

package qeeweb;

import java.util.Hashtable;

import java.sql.*;

public class RelationalBrokerPermission extends
RelationalPersistentBroker {
    private Connection conn;
    private Statement stmt;
    private ResultSet rs;
    private Hashtable permissionCache;

    /**
     * Construtores
     */
    public RelationalBrokerPermission(Connection _conn) {
        conn = _conn;

        try {
            stmt = conn.createStatement();
        } catch (Exception e) {
            System.err.println(e.toString());
        }

        permissionCache = new Hashtable();
    }

    /**
     * Metodo herdado de PersistentBroker
```

```
 */
public Object inCache(String _OID) {
    return permissionCache.get(_OID);
}

/**
 * Metodos herdados de RelationalPersistentBroker
 */
public void selectFirst(String _OID) {
    try {
        rs = stmt.executeQuery("select oid_permissao, str_descricao from
        tb_permissao where oid_permissao='" + _OID + "'");
    } catch (Exception e) {
        System.err.println(e.toString());
    }
}

public Object currentRecordAsObject() {
    Permission permission = null;

    try {
        if (rs.next()) {
            String oid = rs.getString("oid_permissao");
            String descr = rs.getString("str_descricao");

            permission = new Permission(oid, descr);
            permissionCache.put(oid, descr);
        }

        rs.close();
    } catch (Exception e) {
        System.err.println(e.toString());
    }

    return permission;
}

/**
 * Metodos de acesso ao broker
 */
public static RelationalBrokerPermission Instance(Connection _conn) {
    return new RelationalBrokerPermission(_conn);
}
```

```
public void insertPermission(Permission _permission) {
    String oid = _permission.getOID();
    String descr = _permission.getOID();
    String sql = "insert into tb_permissao (oid_permissao, str_descricao)
    values ('" + oid + "', '" + descr + "')";

    try {
        if (stmt.execute(sql)) {
            permissionCache.put(oid, _permission);
        }
    } catch (Exception e) {
        System.err.println(e.toString());
    }
}

public void updatePermission(Permission _permission) {
    String oid = _permission.getOID();
    String operation = _permission.getOID();

    String sql = "update tb_permissao set oid_permissao='" + oid + "',
    str_operacao='" + operation + "' where oid_permissao='" + oid + "'";

    try {
        if (stmt.execute(sql)) {
            permissionCache.remove(oid);
        }
    } catch (Exception e) {
        System.err.println(e.toString());
    }
}

public void deletePermission(Permission _permission) {
    deletePermission(_permission.getOID());
}

public void deletePermission(String _OID) {
    String sql = "delete from tb_permissao where oid_permissao='" + _OID +
    "'";

    try {
        if (stmt.execute(sql)) {
            permissionCache.remove(_OID);
```

```
            }
        } catch (Exception e) {
            System.err.println(e.toString());
        }
    }
}

package qeeweb;

import java.sql.Connection;

import java.sql.Statement;

import java.sql.ResultSet;

import java.util.Hashtable;

public class RelationalBrokerPonto extends
RelationalPersistentBroker {
    protected Connection conn;
    protected Statement stmt;
    protected ResultSet rs;
    protected Hashtable pontoCache;

    /**
     * Construtores
     */
    public RelationalBrokerPonto(Connection _conn) {
        conn = _conn;

        try {
            stmt = conn.createStatement();
        } catch (Exception e) {
            System.err.println(e.toString());
        }

        pontoCache = new Hashtable();
    }

    /**
     * Metodo herdado de PersistentBroker
     */
    public Object inCache(String _OID) {
```

```
    return pontoCache.get(_OID);
}

/**
 * Metodos herdados de RelationalPersistentBroker
 */
public void selectFirst(String _OID) {
    try {
        rs = stmt.executeQuery("select oid_ponto, str_descricao,
        str_cliente, str_medidor, str_endereco_ip, str_porta, cd_pasta,
        fl_habilitado from tb_ponto where oid_ponto='" + _OID + "'");
    } catch (Exception e) {
        System.err.println(e.toString());
    }
}

public Object currentRecordAsObject() {
    Ponto ponto = null;

    try {
        if (rs.next()) {
            String oid = rs.getString("oid_ponto");
            String descricao = rs.getString("str_descricao");
            String cliente = rs.getString("str_cliente");
            String medidor = rs.getString("str_medidor");
            String enderecoIP = rs.getString("str_endereco_ip");
            String porta = rs.getString("str_porta");
            int cdPasta = rs.getInt("cd_pasta");
            boolean habilitado = rs.getBoolean("fl_habilitado");

            ponto = new Ponto(oid, descricao, cliente, medidor, enderecoIP,
            porta, cdPasta, habilitado);
            pontoCache.put(oid, ponto);
        }

        rs.close();
    } catch (Exception e) {
        System.err.println(e.toString());
    }

    return ponto;
}
```

```
/**
 * Metodos de acesso ao broker
 */
public static RelationalBrokerPonto Instance(Connection _conn) {
    return new RelationalBrokerPonto(_conn);
}

public void insertPonto(Ponto _ponto) {
    String oid = _ponto.getOID();
    String descricao = _ponto.getDescricao();
    String cliente = _ponto.getCliente();
    String medidor = _ponto.getMedidor();
    String enderecoIP = _ponto.getEnderecoIP();
    String porta = _ponto.getPorta();
    int cdPasta = _ponto.getCdPasta();
    boolean habilitado = _ponto.getHabilitado();
    String sql = "insert into tb_ponto (oid_ponto, str_descricao,
    str_cliente, str_medidor, str_endereco_ip, str_porta, cd_pasta,
    fl_habilitado) values ('" + oid + "', '" + descricao + "', '" + cliente
    + "', '" + medidor + "', '" + enderecoIP + "', '" + porta + "', " +
    cdPasta + ", '" + habilitado + "')";

    try {
        if (stmt.execute(sql)) {
            pontoCache.put(oid, _ponto);
        }
    } catch (Exception e) {
        System.err.println(e.toString());
    }
}

public void updatePonto(Ponto _ponto) {
    String oid = _ponto.getOID();
    String descricao = _ponto.getDescricao();
    String cliente = _ponto.getCliente();
    String medidor = _ponto.getMedidor();
    String enderecoIP = _ponto.getEnderecoIP();
    String porta = _ponto.getPorta();
    int cdPasta = _ponto.getCdPasta();
    boolean habilitado = _ponto.getHabilitado();

    String sql = "update tb_ponto set oid_ponto='" + oid + "',
    str_descricao='" + descricao + "', str_cliente='" + cliente + "',
```

```
        str_medidor='" + medidor + "', str_endereco_ip='" + enderecoIP + "',
        str_porta='" + porta + "', cd_pasta=" + cdPasta + ", fl_habilitado='" +
        habilitado + "' where oid_ponto='" + oid +"'";

        try {
            if (stmt.execute(sql)) {
                pontoCache.remove(oid);
            }
        } catch (Exception e) {
            System.err.println(e.toString());
        }
    }

    public void deletePonto(Ponto _ponto) {
        deletePonto(_ponto.getOID());
    }

    public void deletePonto(String _OID) {
        String sql = "delete from tb_ponto where oid_ponto='" + _OID + "'";

        try {
            if (stmt.execute(sql)) {
                pontoCache.remove(_OID);
            }
        } catch (Exception e) {
            System.err.println(e.toString());
        }
    }
}

package qeeweb;

import java.util.Hashtable;

import java.util.Enumeration;

import java.util.Vector;

import java.sql.*;

public class RelationalBrokerRole extends
RelationalPersistentBroker {
    private Connection conn;
```

```
private Statement stmt;
private ResultSet rs;
private Hashtable roleCache;

/**
 * Construtores
 */
public RelationalBrokerRole(Connection _conn) {
    conn = _conn;

    try {
        stmt = conn.createStatement();
    } catch (Exception e) {
        System.err.println(e.toString());
    }

    roleCache = new Hashtable();
}

/**
 * Metodo herdado de PersistentBroker
 */
public Object inCache(String _OID) {
    return roleCache.get(_OID);
}

/**
 * Metodos herdados de RelationalPersistentBroker
 */
public void selectFirst(String _OID) {
    try {
        rs = stmt.executeQuery("select oid_papel, str_descricao,
        num_cardinalidade from tb_papel where oid_papel='" + _OID + "'");
    } catch (Exception e) {
        System.err.println(e.toString());
    }
}

public Object currentRecordAsObject() {
    Role role = null;
    RoleProxy roleProxy = null;
    PermissionProxy permissionProxy = null;
    PermissionList permissions = null;
```

```
    RoleList hRoles = null;
    RoleList meRoles = null;
    UserList users = null;

    try {
        if (rs.next()) {
            String oid = rs.getString("oid_papel");
            String descr = rs.getString("str_descricao");
            int cardinality = rs.getInt("num_cardinalidade");

            role = new Role(oid, descr, cardinality);
            permissions = getPermissions(oid);
            hRoles = getHRoles(oid);
            meRoles = getMERoles(oid);
            users = getUsers(oid);

            for (Enumeration e = hRoles.elements(); e.hasMoreElements();) {
                roleProxy = (RoleProxy) e.nextElement();

                for (Enumeration e2 =
                roleProxy.getPermissionList().elements();
                e2.hasMoreElements();) {
                    permissionProxy = (PermissionProxy) e2.nextElement();
                    permissions.put(permissionProxy);
                }
            }

            role.setPermissionList(permissions);
            role.setHRoleList(hRoles);
            role.setMERoleList(meRoles);
            role.setUserList(users);
            roleCache.put(oid, role);
        }

        rs.close();
    } catch (Exception e) {
        System.err.println(e.toString());
    }

    return role;
}

/**
```

```
 * Metodos de acesso ao broker
 */
public static RelationalBrokerRole Instance(Connection _conn) {
    return new RelationalBrokerRole(_conn);
}

public void insertRole(Role _role) {
    String oid = _role.getOID();
    String descr = _role.getDescr();
    int cardinality = _role.getCardinality();
    String sql = "insert into tb_papel (oid_papel, str_descricao,
    num_cardinalidade) values ('" + oid + "', '" + descr + "', " +
    cardinality + ")";

    try {
        if (stmt.execute(sql)) {
            roleCache.put(oid, _role);
        }
    } catch (Exception e) {
        System.err.println(e.toString());
    }
}

public void updateRole(Role _role) {
    String oid = _role.getOID();
    String descr = _role.getDescr();
    int cardinality = _role.getCardinality();
    String sql = "update tb_papel set oid_papel='" + oid + "',
    str_descricao='" + descr + "', num_cardinalidade=" + cardinality + "
    where oid_papel='" + oid + "'";

    try {
        if (stmt.execute(sql)) {
            roleCache.remove(oid);
        }
    } catch (Exception e) {
        System.err.println(e.toString());
    }
}

public void deleteRole(Role _role) {
    deleteRole(_role.getOID());
}
```

```
public void deleteRole(String _OID) {
    String sql = "delete from tb_papel where oid_papel='" + _OID + "'";

    try {
        if (stmt.execute(sql)) {
            roleCache.remove(_OID);
        }
    } catch (Exception e) {
        System.err.println(e.toString());
    }
}

public void setHierarchyRoles(RoleInterface _role1, RoleInterface _role2) {
    setHierarchyRoles(_role1.getOID(), _role2.getOID());
}

public void setHierarchyRoles(String _OID1, String _OID2) {
    String sql = "insert into tb_papel_papel_hier values ('" + _OID1 + "', '"
    + _OID2 + "')";

    try {
        if (stmt.execute(sql)) {
            roleCache.remove(_OID1);
            roleCache.remove(_OID2);
        }
    } catch (Exception e) {
        System.err.println(e.toString());
    }
}

public void setMERoles(RoleInterface _role1, RoleInterface _role2) {
    setMERoles(_role1.getOID(), _role2.getOID());
}

public void setMERoles(String _OID1, String _OID2) {
    String sql = "insert into tb_papel_papel_me values ('" + _OID1 + "', '"
    + _OID2 + "')";

    try {
        if (stmt.execute(sql)) {
            roleCache.remove(_OID1);
            roleCache.remove(_OID2);
```

```
        }
    } catch (Exception e) {
        System.err.println(e.toString());
    }
}

public void deleteHierarchyRoles(RoleInterface _role1, RoleInterface _role2)
{
    deleteHierarchyRoles(_role1.getOID(), _role2.getOID());
}

public void deleteHierarchyRoles(String _OID1, String _OID2) {
    String sql = "delete from tb_papel_papel_hier where (oid_papel='" +
    _OID1 + "' and oid_papel_hier='" + _OID2 + "') or (oid_papel_hier='" +
    _OID1 + "' and oid_papel='" + _OID2 + "')";

    try {
        if (stmt.execute(sql)) {
            roleCache.remove(_OID1);
            roleCache.remove(_OID2);
        }
    } catch (Exception e) {
        System.err.println(e.toString());
    }
}

public void deleteMERoles(RoleInterface _role1, RoleInterface _role2) {
    deleteMERoles(_role1.getOID(), _role2.getOID());
}

public void deleteMERoles(String _OID1, String _OID2) {
    String sql = "delete from tb_papel_papel_me where (oid_papel='" + _OID1
    + "' and oid_papel_me='" + _OID2 + "') or (oid_papel_me='" + _OID1 + "'
    and oid_papel='" + _OID2 + "')";

    try {
        if (stmt.execute(sql)) {
            roleCache.remove(_OID1);
            roleCache.remove(_OID2);
        }
    } catch (Exception e) {
        System.err.println(e.toString());
    }
```

```
}

/**
 * Metodos protegidos
 */
protected PermissionList getPermissions(String _OID) {
    PermissionList permissions = new PermissionList();
    PermissionProxy proxy = null;
    String sql = "select oid_permissao from tb_papel_permissao where oid_papel='" + _OID + "'"

    try {
        rs = stmt.executeQuery(sql);

        while (rs.next()) {
            proxy = new PermissionProxy(rs.getString("oid_permissao"), conn);
            permissions.put(proxy);
        }

        rs.close();
    } catch (Exception e) {
        System.err.println(e.toString());
    }

    return permissions;
}

protected RoleList getHRoles(String _OID) {
    RoleList hRoles = new RoleList();
    RoleProxy proxy = null;
    String sql = "select oid_papel_hier from tb_papel_papel_hier where
    oid_papel='" + _OID + "'";

    try {
        rs = stmt.executeQuery(sql);

        while (rs.next()) {
            proxy = new RoleProxy(rs.getString("oid_papel_hier"), conn);
            hRoles.put(proxy);
        }

        rs.close();
    } catch (Exception e) {
        System.err.println(e.toString());
```

```
    }

    return hRoles;
}

protected RoleList getMERoles(String _OID) {
    RoleList meRoles = new RoleList();
    RoleProxy proxy = null;
    String sql = "select oid_papel_me from tb_papel_papel_me where
    oid_papel='" + _OID + "'";

    try {
        rs = stmt.executeQuery(sql);

        while (rs.next()) {
              proxy = new RoleProxy(rs.getString("oid_papel_me"), conn);
              meRoles.put(proxy);
        }

        rs.close();
    } catch (Exception e) {
        System.err.println(e.toString());
    }

    return meRoles;
}

protected UserList getUsers(String _OID) {
    UserList users = new UserList();
    UserProxy proxy = null;
    String sql = "select oid_usuario from tb_usuario_papel where
    oid_papel='" + _OID + "'";

    try {
        rs = stmt.executeQuery(sql);

        while (rs.next()) {
            proxy = new UserProxy(rs.getString("oid_usuario"), conn);
            users.put(proxy);
        }

        rs.close();
    } catch (Exception e) {
```

```
            System.err.println(e.toString());
        }

        return users;
    }
}

package qeeweb;

import java.sql.Connection;

import java.sql.Statement;

import java.sql.ResultSet;

import java.util.Hashtable;

public class RelationalBrokerUser extends
RelationalPersistentBroker {
    protected Connection conn;
    protected Statement stmt;
    protected ResultSet rs;
    protected Hashtable userCache;

    /**
     * Construtores
     */
    public RelationalBrokerUser(Connection _conn) {
        conn = _conn;

        try {
            stmt = conn.createStatement();
        } catch (Exception e) {
            System.err.println(e.toString());
        }

        userCache = new Hashtable();
    }

    /**
     * Metodo herdado de PersistentBroker
     */
    public Object inCache(String _OID) {
```

```
    return userCache.get(_OID);
}

/**
 * Metodos herdados de RelationalPersistentBroker
 */
public void selectFirst(String _OID) {
    try {
        rs = stmt.executeQuery("select oid_usuario, str_senha, str_nome from
        tb_usuario where oid_usuario='" + _OID + "'");
    } catch (Exception e) {
        System.err.println(e.toString());
    }
}

public Object currentRecordAsObject() {
    User user = null;
    RoleList roles = null;

    try {
        if (rs.next()) {
            String oid = rs.getString("oid_usuario");
            String password = rs.getString("str_senha");
            String name = rs.getString("str_nome");

            user = new User(oid, new Password(password), name);
            roles = getRoles(oid);

            user.setRoleList(roles);
            userCache.put(oid, user);
        }

        rs.close();
    } catch (Exception e) {
        System.err.println(e.toString());
    }

    return user;
}

/**
 * Metodos de acesso ao broker
 */
```

```
public static RelationalBrokerUser Instance(Connection _conn) {
    return new RelationalBrokerUser(_conn);
}

public void insertUser(User _user) {
    String oid = _user.getOID();
    String password = ((Password)
    _user.getAuthenticationMethod()).getPassword();
    String name = _user.getName();
    String sql = "insert into tb_usuario (oid_usuario, str_senha, str_nome)
    values ('" + oid + "', '" + password + "', '" + name + "')";

    try {
        if (stmt.execute(sql)) {
            userCache.put(oid, _user);
        }
    } catch (Exception e) {
        System.err.println(e.toString());
    }
}

public void updateUser(User _user) {
    String oid = _user.getOID();
    String password = ((Password)
    _user.getAuthenticationMethod()).getPassword();
    String name = _user.getName();
    String sql = "update tb_usuario set oid_usuario='" + oid + "',
    str_senha='" + password + "', str_nome='" + name + "' where
    oid_usuario='" + oid +"'";

    try {
        if (stmt.execute(sql)) {
            userCache.remove(oid);
        }
    } catch (Exception e) {
        System.err.println(e.toString());
    }
}

public void deleteUser(User _user) {
    deleteUser(_user.getOID());
}
```

```
    public void deleteUser(String _OID) {
        String sql = "delete from tb_usuario where oid_usuario='" + _OID + "'";

        try {
            if (stmt.execute(sql)) {
                userCache.remove(_OID);
            }
        } catch (Exception e) {
            System.err.println(e.toString());
        }
    }

    /**
     * Metodos protegidos
     */
    protected RoleList getRoles(String _OID) {
        RoleList roles = new RoleList();
        RoleProxy proxy = null;
        String sql = "select oid_papel from tb_usuario_papel where
        oid_usuario='" + _OID + "'";

        try {
            rs = stmt.executeQuery(sql);

            while (rs.next()) {
                proxy = new RoleProxy(rs.getString("oid_papel"), conn);
                roles.put(proxy);
            }

            rs.close();
        } catch (Exception e) {
            System.err.println(e.toString());
        }

        return roles;
    }
}

package qeeweb;

import java.sql.Connection;

import java.sql.Statement;
```

```
import java.sql.ResultSet;

import java.util.Hashtable;

public class RelationalBrokerUserQEEWeb extends
RelationalBrokerUser {
    /**
     * Construtores
     */
    public RelationalBrokerUserQEEWeb(Connection _conn) {
        super(_conn);
    }

    /**
     * Metodos herdados de RelationalPersistentBroker
     */
    public void selectFirst(String _OID) {
        try {
            rs = stmt.executeQuery("select oid_usuario, str_senha, str_nome,
            cd_pasta from tb_usuario where oid_usuario='" + _OID + "'");
        } catch (Exception e) {
            System.err.println(e.toString());
        }
    }

    public Object currentRecordAsObject() {
        User user = null;
        RoleList roles = null;

        try {
            if (rs.next()) {
                String oid = rs.getString("oid_usuario");
                String senha = rs.getString("str_senha");
                String nome = rs.getString("str_nome");
                int cdPasta = rs.getInt("cd_pasta");

                user = new UserQEEWeb(oid, senha, nome, cdPasta);
                roles = getRoles(oid);

                user.setRoleList(roles);
                userCache.put(oid, user);
            }
```

```
            rs.close();
        } catch (Exception e) {
            System.err.println(e.toString());
        }

        return user;
    }

    /**
     * Metodos de acesso ao broker
     */
    public static RelationalBrokerUser Instance(Connection _conn) {
        return new RelationalBrokerUserQEEWeb(_conn);
    }

    public void insertUser(User _user) {
        String oid = _user.getOID();
        String senha = ((Password)
        _user.getAuthenticationMethod()).getPassword();
        String nome = _user.getName();
        int cdPasta = ((UserQEEWeb) _user).getCdPasta();
        String sql = "insert into tb_usuario (oid_usuario, str_senha, str_nome,
        cd_pasta) values ('" + oid + "', '" + senha + "', '" + nome + "', " +
        cdPasta + ")";

        try {
            if (stmt.execute(sql)) {
                userCache.put(oid, _user);
            }
        } catch (Exception e) {
            System.err.println(e.toString());
        }
    }

    public void updateUser(User _user) {
        String oid = _user.getOID();
        String senha = ((Password)
        _user.getAuthenticationMethod()).getPassword();
        String nome = _user.getName();
        int cdPasta = ((UserQEEWeb) _user).getCdPasta();
        String sql = "update tb_usuario set oid_usuario='" + oid + "',
        str_senha='" + senha + "', str_nome='" + nome + "', cd_pasta=" + cdPasta
```

```
        + " where oid_usuario='" + oid +"'";

        try {
            if (stmt.execute(sql)) {
                userCache.remove(oid);
            }
        } catch (Exception e) {
            System.err.println(e.toString());
        }
    }

    public void deleteUser(User _user) {
        deleteUser(_user.getOID());
    }

    public void deleteUser(String _OID) {
        String sql = "delete from tb_usuario where oid_usuario='" + _OID + "'";

        try {
            if (stmt.execute(sql)) {
                userCache.remove(_OID);
            }
        } catch (Exception e) {
            System.err.println(e.toString());
        }
    }
}

package qeeweb;

abstract public class RelationalPersistentBroker extends
PersistentBroker {
    /**
     * Metodo herdado de PersistentBroker
     */
    public Object materializeWith(String _OID) {
        selectFirst(_OID);
        Object obj = currentRecordAsObject();
        return obj;
    }

    /**
     * Metodos a serem implementados por subclasses
```

```
     */
    public void selectFirst(String _OID) {
    }

    public Object currentRecordAsObject() {
        return null;
    }
}

package qeeweb;

import java.util.Enumeration;

public class Role implements RoleInterface {
    private String OID;
    private String descr;
    private int cardinality;
    private PermissionList permissions;
    private RoleList hRoles;
    private RoleList meRoles;
    private UserList users;

    /**
     * Construtores
     */
    public Role(String _OID) {
        OID = _OID;
    }

    public Role(String _OID, String _descr) {
        OID = _OID;
        descr = _descr;
    }

    public Role(String _OID, String _descr, int _cardinality) {
        OID = _OID;
        descr = _descr;
        cardinality = _cardinality;
    }

    /**
     * Get Methods
     */
```

```
public String getOID() {
    return OID;
}

public String getDescr() {
    return descr;
}

public int getCardinality() {
    return cardinality;
}

public PermissionList getPermissionList() {
    return permissions;
}

public RoleList getHRoleList() {
    return hRoles;
}

public RoleList getMERoleList() {
    return meRoles;
}

public UserList getUserList() {
    return users;
}

/**
 * Set Methods
 */
public void setCardinality(int _cardinality) {
    cardinality = _cardinality;
}

public void setPermissionList(PermissionList _permissions) {
    permissions = _permissions;
}

public void setHRoleList(RoleList _hRoles) {
    hRoles = _hRoles;
}
```

```
public void setMERoleList(RoleList _meRoles) {
    meRoles = _meRoles;
}

public void setUserList(UserList _users) {
    users = _users;
}

/**
 * Adiciona os papeis as listas
 */
public void addHierarchyRole(RoleInterface _role) {
    hRoles.put(_role);
}

public void addMERole(RoleInterface _role) {
    meRoles.put(_role);
}

/**
 * Remove os papeis das listas
 */
public void delHierarchyRole(RoleInterface _role) {
    hRoles.delRole(_role);
}

public void delHierarchyRole(String _OID) {
    hRoles.delRole(_OID);
}

public void delMERole(RoleInterface _role) {
    meRoles.delRole(_role);
}

public void delMERole(String _OID) {
    meRoles.delRole(_OID);
}

/**
 * Verifica se os papeis pertencem as listas
 */
public boolean isHierarchyRole(RoleInterface _role){
    return isHierarchyRole(_role.getOID());
```

```
    }

    public boolean isHierarchyRole(String _OID) {
        RoleProxy role = null;

        for (Enumeration e = hRoles.elements(); e.hasMoreElements();) {
           role = (RoleProxy) e.nextElement();

           if (_OID.equals(role.getOID())) {
               return true;
           }
        }

        return false;
    }

    public boolean isMERole(RoleInterface _role) {
        return isMERole(_role.getOID());
    }

    public boolean isMERole(String _OID) {
        RoleProxy role = null;

        for (Enumeration e = meRoles.elements(); e.hasMoreElements();) {
           role = (RoleProxy) e.nextElement();

            if (_OID.equals(role.getOID())) {
                return true;
            }
        }

        return false;
    }
}

package qeeweb;

public interface RoleInterface {
    public String getOID();

    public String getDescr();

    public int getCardinality();
```

```
    public PermissionList getPermissionList();

    public RoleList getHRoleList();

    public RoleList getMERoleList();

    public void addHierarchyRole(RoleInterface _role);

    public void addMERole(RoleInterface _role);

    public void delHierarchyRole(RoleInterface _role);

    public void delHierarchyRole(String _OID);

    public void delMERole(RoleInterface _role);

    public void delMERole(String _OID);

    public boolean isHierarchyRole(RoleInterface _role);

    public boolean isHierarchyRole(String _OID);

    public boolean isMERole(RoleInterface _role);

    public boolean isMERole(String _OID);
}

package qeeweb;

import java.util.Vector;

import java.util.Enumeration;

public class RoleList {
    private Vector roles;

    public RoleList() {
        roles = new Vector();
    }

    public RoleInterface get(int _index) {
        return (RoleInterface) roles.elementAt(_index);
```

```
    }

    public Enumeration elements() {
        return roles.elements();
    }

    public void put(RoleInterface _role) {
        roles.addElement(_role);
    }

    public void put(RoleInterface _role, int _index) {
        roles.insertElementAt(_role, _index);
    }

    public void delRole(RoleInterface _role) {
        delRole(_role.getOID());
    }

    public void delRole(String _OID) {
        Enumeration e = roles.elements();
        int index = 0;
        boolean res = true;
        RoleProxy role = null;

        while (e.hasMoreElements() && res) {
            role = (RoleProxy) e.nextElement();

            if (role.getOID().equals(_OID)) {
                roles.removeElementAt(index);
                res = false;
            } else {
                index++;
            }
        }
    }
}

package qeeweb;

import java.sql.Connection;

public class RoleProxy extends VirtualProxy implements
RoleInterface {
```

```
protected Connection conn;

/**
 * Construtores
 */
public RoleProxy(String _OID, Connection _conn) {
    super (_OID);
    conn = _conn;
}

/**
 * Get Methods
 */
public String getDescr() {
    return ((Role) getRealSubject()).getDescr();
}

public int getCardinality() {
    return ((Role) getRealSubject()).getCardinality();
}

public PermissionList getPermissionList() {
    return ((Role) getRealSubject()).getPermissionList();
}

public RoleList getHRoleList() {
    return ((Role) getRealSubject()).getHRoleList();
}

public RoleList getMERoleList() {
    return ((Role) getRealSubject()).getMERoleList();
}

public UserList getUserList() {
    return ((Role) getRealSubject()).getUserList();
}

/**
 * Adiciona papeis as listas
 */
public void addHierarchyRole(RoleInterface _role) {
    ((Role) getRealSubject()).addHierarchyRole(_role);
}
```

```
public void addMERole(RoleInterface _role) {
    ((Role) getRealSubject()).addMERole(_role);
}

/**
 * Remove papeis das listas
 */
public void delHierarchyRole(RoleInterface _role) {
    ((Role) getRealSubject()).delHierarchyRole(_role);
}

public void delHierarchyRole(String _OID) {
    ((Role) getRealSubject()).delHierarchyRole(_OID);
}

public void delMERole(RoleInterface _role) {
    ((Role) getRealSubject()).delMERole(_role);
}

public void delMERole(String _OID) {
    ((Role) getRealSubject()).delMERole(_OID);
}

/**
 * Verifica se papeis pertencem as listas
 */
public boolean isHierarchyRole(RoleInterface _role) {
    return ((Role) getRealSubject()).isHierarchyRole(_role);
}

public boolean isHierarchyRole(String _OID) {
    return ((Role) getRealSubject()).isHierarchyRole(_OID);
}

public boolean isMERole(RoleInterface _role) {
    return ((Role) getRealSubject()).isMERole(_role);
}

public boolean isMERole(String _OID) {
    return ((Role) getRealSubject()).isMERole(_OID);
}

/**
```

```
 * Metodos herdados de VirtualProxy
 */
public String getBrokerId() {
    return "RelationalBrokerRole";
}

public PersistentBroker createBroker() {
    return RelationalBrokerRole.Instance(conn);
}

/**
 * Metodos de acesso ao broker
 */
public void insertRole(Role _role) {
    ((RelationalBrokerRole) getBroker()).insertRole(_role);
}

public void updateRole(Role _role) {
    ((RelationalBrokerRole) getBroker()).updateRole(_role);
}

public void deleteRole(Role _role) {
    ((RelationalBrokerRole) getBroker()).deleteRole(_role);
}

public void deleteRole(String _OID) {
    ((RelationalBrokerRole) getBroker()).deleteRole(_OID);
}

public void setHierarchyRoles(RoleInterface _role1, RoleInterface _role2) {
    ((RelationalBrokerRole) getBroker()).setHierarchyRoles(_role1, _role2);
}

public void setHierarchyRoles(String _OID1, String _OID2) {
    ((RelationalBrokerRole) getBroker()).setHierarchyRoles(_OID1, _OID2);
}

public void setMERoles(RoleInterface _role1, RoleInterface _role2) {
    ((RelationalBrokerRole) getBroker()).setMERoles(_role1, _role2);
}

public void setMERoles(String _OID1, String _OID2) {
    ((RelationalBrokerRole) getBroker()).setMERoles(_OID1, _OID2);
```

```
    }

    public void deleteHierarchyRoles(RoleInterface _role1, RoleInterface _role2)
    {
        ((RelationalBrokerRole) getBroker()).deleteHierarchyRoles(_role1,
        _role2);
    }

    public void deleteHierarchyRoles(String _OID1, String _OID2) {
        ((RelationalBrokerRole) getBroker()).deleteHierarchyRoles(_OID1, _OID2);
    }

    public void deleteMERoles(RoleInterface _role1, RoleInterface _role2) {
        ((RelationalBrokerRole) getBroker()).deleteMERoles(_role1, _role2);
    }

    public void deleteMERoles(String _OID1, String _OID2) {
        ((RelationalBrokerRole) getBroker()).deleteMERoles(_OID1, _OID2);
    }
}

package qeeweb;

public class User implements UserInterface {
    protected String OID;
    protected AuthenticationMethod authenticationMethod;
    protected String name;
    protected RoleList roles;

    /**
     * Construtores
     */
    public User() {
        OID = "";
    }

    public User(String _OID) {
        OID = _OID;
    }

    public User(String _OID, AuthenticationMethod _authenticationMethod, String
    _name) {
        OID = _OID;
```

```
        authenticationMethod = _authenticationMethod;
        name = _name;
    }

    /**
     * Get Methods
     */
    public String getOID() {
        return OID;
    }

    public AuthenticationMethod getAuthenticationMethod() {
        return authenticationMethod;
    }

    public String getName() {
        return name;
    }

    public RoleList getRoleList() {
        return roles;
    }

    /**
     * Set Methods
     */
    public void setOID(String _OID) {
        OID = _OID;
    }

    public void setAuthenticationMethod(AuthenticationMethod
    _authenticationMethod) {
        authenticationMethod = _authenticationMethod;
    }

    public void setName(String _name) {
        name = _name;
    }

    public void setRoleList(RoleList _roles) {
        roles = _roles;
    }
}
```

```
package qeeweb;

public interface UserInterface {
    public String getOID();

    public AuthenticationMethod getAuthenticationMethod();

    public String getName();

    public RoleList getRoleList();
}

package qeeweb;

import java.util.Vector;

import java.util.Enumeration;

public class UserList {
    private Vector users;

    public UserList() {
        users = new Vector();
    }

    public UserInterface get(int _index) {
        return (UserInterface) users.elementAt(_index);
    }

    public Enumeration elements() {
        return users.elements();
    }

    public void put(UserInterface _user) {
        users.addElement(_user);
    }

    public void put(UserInterface _user, int _index) {
        users.insertElementAt(_user, _index);
    }
}
```

```
package qeeweb;

import java.util.Vector;
import java.util.Enumeration;

public class UserList
{
    private Vector users;

    public UserList() {
        users = new Vector();
    }

    public UserInterface get(int _index) {
        return (UserInterface) users.elementAt(_index);
    }

    public Enumeration elements() {
        return users.elements();
    }

    public void put(UserInterface _user) {
        users.addElement(_user);
    }

    public void put(UserInterface _user, int _index) {
        users.insertElementAt(_user, _index);
    }
}

package qeeweb;

import java.sql.Connection;

public class UserProxy extends VirtualProxy implements
UserInterface {
    protected Connection conn;

    /**
     * Construtores
     */
    public UserProxy(Connection _conn) {
        super("");
```

```
    conn = _conn;
}

public UserProxy(String _OID, Connection _conn) {
    super(_OID);
    conn = _conn;
}

/**
 * Metodos do objeto real
 */
public AuthenticationMethod getAuthenticationMethod() {
    return ((User) getRealSubject()).getAuthenticationMethod();
}

public String getName() {
    return ((User) getRealSubject()).getName();
}

public RoleList getRoleList() {
    return ((User) getRealSubject()).getRoleList();
}

/**
 * Metodos herdados de VirtualProxy
 */
public String getBrokerId() {
    return "RelationalBrokerUser";
}

public PersistentBroker createBroker() {
    return RelationalBrokerUser.Instance(conn);
}

/**
 * Metodos de acesso ao broker
 */
public void insertUser(User _user) {
    ((RelationalBrokerUser) getBroker()).insertUser(_user);
}

public void updateUser(User _user) {
    ((RelationalBrokerUser) getBroker()).updateUser(_user);
```

```
    }

    public void deleteUser(User _user) {
        ((RelationalBrokerUser) getBroker()).deleteUser(_user);
    }

    public void deleteUser(String _OID) {
        ((RelationalBrokerUser) getBroker()).deleteUser(_OID);
    }
}

package qeeweb;

import java.util.Hashtable;

public class UserQEEWeb extends User {
    private int cdPasta;
    private Hashtable erros;

    /**
     * Construtores
     */
    public UserQEEWeb() {
        super();
    }

    public UserQEEWeb(String _OID) {
        super(_OID);
    }

    public UserQEEWeb(String _OID, String _senha, String _nome, int _cdPasta) {
        super(_OID, new Password(_senha), _nome);

        cdPasta = _cdPasta;
        erros = new Hashtable();
    }

    /**
     * Get Methods
     */
    public String getSenha() {
        return ((Password) getAuthenticationMethod()).getPassword();
    }
```

```
public int getCdPasta() {
    return cdPasta;
}

public String getErro(String s) {
    String erro = (String) erros.get(s);

    if (erro == null) {
        erro = "";
    }

    return erro;
}

/**
 * Set Methods
 */
public void setSenha(String _senha) {
    authenticationMethod = new Password(_senha);
}

public void setCdPasta(int _cdPasta) {
    cdPasta = _cdPasta;
}

/**
 * Verifica se o usuario e valido
 */
public boolean ehValido() {
    if (OID.equals("")) {
        erros.put("OID", "*");
        return false;
    }

    if (getSenha() == "") {
        erros.put("senha", "*");
        return false;
    }

    if (name.equals("")) {
        erros.put("nome", "*");
        return false;
```

```
        }

        if (cdPasta < 0) {
            erros.put("pasta", "*");
            return false;
        }

        return true;
    }
}

package qeeweb;

public interface UserQEEWebInterface {
    public String getSenha();

    public int getCdPasta();

    public String getErro(String s);

    public boolean ehValido();
}

package qeeweb;

import java.sql.Connection;

public class UserQEEWebProxy extends UserProxy implements
UserQEEWebInterface {
    /**
     * Construtores
     */
    public UserQEEWebProxy(Connection _conn) {
        super(_conn);
    }

    public UserQEEWebProxy(String _OID, Connection _conn) {
        super(_OID, _conn);
    }

    /**
     * Metodos do objeto real
     */
```

```
public String getSenha() {
    return ((UserQEEWeb) getRealSubject()).getSenha();
}

public int getCdPasta() {
    return ((UserQEEWeb) getRealSubject()).getCdPasta();
}

public String getErro(String s) {
    return ((UserQEEWeb) getRealSubject()).getErro(s);
}

public boolean ehValido() {
    return ((UserQEEWeb) getRealSubject()).ehValido();
}

/**
 * Metodos herdados de VirtualProxy
 */
public String getBrokerId() {
    return "RelationalBrokerUserQEEWeb";
}

public PersistentBroker createBroker() {
    return RelationalBrokerUserQEEWeb.Instance(conn);
}

/**
 * Metodos de acesso ao broker
 */
public void insertUser(User _user) {
    ((RelationalBrokerUserQEEWeb) getBroker()).insertUser(_user);
}

public void updateUser(User _user) {
    ((RelationalBrokerUserQEEWeb) getBroker()).updateUser(_user);
}

public void deleteUser(User _user) {
    ((RelationalBrokerUserQEEWeb) getBroker()).deleteUser(_user);
}

public void deleteUser(String _OID) {
```

```
        ((RelationalBrokerUserQEEWeb) getBroker()).deleteUser(_OID);
    }
}

package qeeweb;

import java.util.Hashtable;

public class VirtualProxy {
    protected String OID;
    private Object realSubject;
    static private PersistentBroker broker;
    static private Hashtable brokers;

    /**
     * Construtores
     */
    public VirtualProxy(String _OID) {
        OID = _OID;
        brokers = new Hashtable();
    }

    /**
     * Get Methods
     */
    public String getOID() {
        return OID;
    }

    public Object getRealSubject() {
        if (realSubject == null) {
            materializeSubject();
        }

        return realSubject;
    }

    public PersistentBroker getBroker() {
        if (broker == null) {
            broker = createBroker();
            brokers.put(broker.getClass().getName(), broker);
        } else if (!broker.getClass().getName().equals(getBrokerId())) {
            if (brokers.containsKey(getBrokerId())) {
```

```
                broker = (PersistentBroker) brokers.get(getBrokerId());
            } else {
                broker = createBroker();
                brokers.put(broker.getClass().getName(), broker);
            }
        }

        return broker;
    }

    /**
     * Materializa realSubject
     */
    public void materializeSubject() {
        realSubject = getBroker().objectWith(OID);
    }

    /**
     * Metodos a serem implementados por subclasses
     */
    public String getBrokerId() {
        return null;
    }

    public PersistentBroker createBroker() {
        return null;
    }
}
```