Universidade Federal de Juiz de Fora
Instituto de Ciências Exatas
Bacharelado em Ciência da Computação

Configuração de perfis de acesso com base na disponibilização de serviços web

Adriano Reiné Bueno

JUIZ DE FORA
AGOSTO, 2011

Configuração de perfis de acesso com base na disponibilização de serviços web

Adriano Reiné Bueno

Universidade Federal de Juiz de Fora

Instituto de Ciências Exatas

Departamento de Ciência da Computação

Bacharelado em Ciência da Computação

Orientador: Evaldo de Oliveira da Silva

JUIZ DE FORA

AGOSTO, 2011

Configuração de perfis de acesso com base na disponibilização de serviços web

Adriano Reiné Bueno

MONOGRAFIA SUBMETIDA AO CORPO DOCENTE DO INSTITUTO DE CIÊNCIAS EXATAS DA UNIVERSIDADE FEDERAL DE JUIZ DE FORA, COMO PARTE INTEGRANTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE BACHAREL EM CIÊNCIA DA COMPUTAÇÃO.

Aprovada por:

Evaldo de Oliveira da Silva
Mestre em Ciência da Computação

Ciro de Barros Barbosa
Doutor em Ciência da Computação

Giuliano Prado de Morais Giglio
Mestre em Computação

JUIZ DE FORA

27 DE AGOSTO, 2011

Agradeço aos meus pais e minha irmã que contribuíram de um modo muito especial para o meu crescimento como ser humano e como profissional, e a todos que estiveram presentes comigo e que de alguma forma me apoiaram nesta caminhada, em especial ao meu orientador Evaldo. Deixo a todos um grande abraço!

Resumo

A segurança entre as aplicações tem ganhado cada vez mais importância nas grandes organizações. A implantação das políticas de segurança da informação dentro das empresas previne a ocorrência de fraudes, roubo de informação e outros danos causados pela falta de segurança entre os sistemas. Nesse contexto, grandes organizações necessitam desenvolver aplicações capazes de fornecer mecanismos para o controle de nível de acesso entre sistemas heterogêneos e distribuídos. Este trabalho tem como objetivo a implementação de um *middleware* para autenticação e controle de acesso responsável por disponibilizar, por meio de serviços, os níveis de acesso de cada usuário no momento da autenticação no sistema. Utilizando o controle de acesso baseado em papéis, técnicas de integração de sistemas, *webservices* e arquitetura orientada a serviços podemos disponibilizar arquivos de fácil manipulação como o *XML* ou *JSON* para representar os níveis de acesso de um usuário. De acordo com as políticas de segurança da informação da empresa, será definido um padrão de representação do controle de acesso e as aplicações conhecendo esse padrão, podem fazer o controle de acesso do usuário.

Palavras-chave: Políticas de segurança da informação, controle de acesso baseado em papéis, serviços, *middleware*.

Abstract

The security between applications has earned increasing importance in large organizations. The implantation of information security polices inside the companies prevent fraud occurrences, information stealing and other damages caused by lack of security between systems. In this context, large organizations need to develop applications capable to provide ways for level access control between distributed and heterogenic systems. This work has as a goal a middleware implementation for authentication and access control responsible for provide, by services, access levels of each user at the system authentication(log in) moment. Using the access control based on roles, system integration techniques, web services and service-oriented architecture it can provide easy handling files like XML or JSON to represent the user's access levels. Based on company's information security politics, it will be defined a representation standard of access control and applications. Knowing this standard, it can control user access.

Keywords: Information security, role based access control, services, middleware.

Agradecimentos

A Deus, porque Dele, por Ele e para Ele são todas as coisas.

A todos que, direta ou indiretamente, contribuíram para a realização deste trabalho manifesto minha gratidão e, em particular,

Ao Professor e Orientador Msc. Evaldo de Oliveira da Silva, pelas valiosas e sábias orientações e, sobretudo, pela amizade, dedicação e tolerância o tempo todo.

Ao curso de Ciência de Computação da UFJF e aos Professores, pela colaboração e sugestões sempre valiosas.

Aos amigos Thiago Miranda do Couto, Thiago Soares e Diego Ricardo, pelo convívio e contribuição nesse trabalho.

A todos colegas de faculdade, por compartilharem comigo esta prazerosa caminhada.

Aos profissionais com os quais tive o privilégio de trabalhar.

Aos meus amigos pelo companheirismo e pelas horas de descontração. Sempre me lembrarei de todos vocês. Obrigado a todos.

À minha mãe Martha, meu pai Oswaldo e minha irmã Christianne, que sempre estiveram ao meu lado, cujo apoio e ajuda foram imprescindíveis à realização deste trabalho.

"Deus nos concede, a cada dia, uma página de vida nova no livro do tempo. Aquilo que colocarmos nela, corre por nossa conta".

Chico Buarque de Hollanda

Sumário

Lista de Figuras **8**

Lista de Tabelas **9**

Lista de Abreviações **10**

1 Introdução **11**

2 Fundamentação Teórica **14**
2.1 Políticas de Segurança da Informação 14
2.1.1 Classificação da Informação 15
2.1.2 Princípios da Segurança da Informação 16
2.1.3 Vulnerabilidades 17
2.2 Modelos de Controle de Acesso 18
2.2.1 Conceitos e Modelos de Controle de Acesso 18
2.2.1.1 Autenticação e Autorização 19
2.2.1.2 Usuários e Permissões em Transações 20
2.2.1.3 Modelo de Bell-LaPadula 20
2.2.1.4 Modelo Clark-Wilson 21
2.3 O Modelo RBAC 22
2.3.1 Visão Geral 22
2.3.2 Permissões 23
2.3.3 Ativação de Papéis 23
2.3.4 Família de modelos RBAC 24
2.4 *Web Services* 25
2.4.1 WSDL 26
2.4.2 SOAP 27
2.4.3 REST 27
2.5 Middlewares usados para o modelo RBAC 28
2.5.1 Visão Geral 28
2.5.2 MACA - Middleware de Autenticação e Controle de Acesso 29
2.5.2.1 RBAC baseado em WebServices 29
2.5.2.2 RBAC para o Modelo CORBA de Segurança 30

3 Proposta de um *Middleware* para Controle de Acesso Baseado em Papéis usando *Web Services* utilizando o sistema ABSecurity **33**
3.1 Modelo de Processo de Desenvolvimento de Software Utilizado para o AB-Security 33
3.2 Levantamento de Requisitos 34
3.3 Arquitetura da Aplicação 36
3.4 Padrão de Autenticação e Autorização 38

4 Aspectos de Implementação do Sistema ABSecurity **41**
4.1 Modelagem do Sistema 41
4.1.1 Modelagem do Banco de Dados 42

4.2 Implementação 43
4.3 Protótipos 45
4.4 Funcionamento 46
4.5 Testes 47

5 Considerações Finais **50**

Referências Bibliográficas **51**

Lista de Figuras

2.1 Tríade Grega (Spanceski, 2004) 15
2.2 Controle de acesso 19
2.3 Relacionamento entre usuários, papéis e permissões (Silva e Saldanha, 2006) 22
2.4 Exemplo de Hierarquia de Papéis 23
2.5 Arquitetura de *web services* (Ramalho, 2004) 25
2.6 Comunicação através de *middleware* (Maciel e Assis, 2004) 29
2.7 Arquitetura de um middleware RBAC baseado em *web services* (Sohr et al., 2006) 30
2.8 Modelo CORBA de segurança (Obelheiro et al., 2001) 31

3.1 Arquitetura do sistema ABSecurity 37
3.2 Padrão de Autenticação e Autorização 39

4.1 Diagrama das classes de autenticação e autorização 42
4.2 Diagrama da base de dados do ABSecurity 43
4.3 Funcionamento 46

Lista de Tabelas

4.1 Primeiro teste de carga 49
4.2 Segundo teste de carga 49

Lista de Abreviações

CORBA	*Common Object Request Broker Architecture*
COSS	*Common Object Service Specification*
CSS	*Cascading Style Sheets*
DAC	*Discretionary Access Control*
DAO	*Data Access Object*
HTML	*HyperText Markup Language*
JSON	*JavaScript Object Notation*
JSP	*JavaServer Faces*
MAC	*Mandatory Access Control*
MACA	*Middleware* de Autenticação e Controle de Acesso
MVC	*Model View Controler*
OMG	*Object Management Group*
ORB	*Object Request Broker*
RBAC	*Role Based Access Control*
REST	*Representational State Transfer*
SGBD	Sistema de Gerenciamento de Banco de Dados
SOAP	*Simple Object Access Protocol*
UDDI	*Universal Description, Discovery and Integration*
XML	*Extensible Markup Language*
WSDL	*Web Services Description Language*

1 Introdução

Grandes organizações mantêm seus sistemas ativos por muito tempo devido à importância que eles representam. Com o passar dos anos, o volume de dados aumenta e por isso esses sistemas não podem ser descartados. A falta de documentação das aplicações, alto grau de complexidade, baixa interoperabilidade, implantação de novos sistemas e a falta de profissionais capacitados nas diferentes tecnologias utilizadas são os principais problemas encontrados na hora de integrar esses sistemas.

Além disso, uma das grandes dificuldades dentro das organizações também é a manutenção e integração de sistemas legados. São aplicações consideradas antigas, e às vezes não possuem um controle do nível de acesso, mesmo os que têm algum tipo de controle de segurança, podem não seguir as atuais políticas de segurança da informação da empresa.

A segurança é um assunto muito importante a ser tratado dentro de uma organização. Toda informação deve ser protegida de maneira adequada, independente da forma que é apresentada. Para muitas empresas a informação é um bem valioso e protegê-la não é algo simples. É necessário que as aplicações sigam as políticas de segurança para não ocorrer fraudes, vazamento de informação, erros, etc (Spanceski, 2004).

Aliado à necessidade de integração de sistemas de informação, sendo eles legados ou não, as organizações necessitam de manter políticas de segurança da informação com base em regras do controle do nível de acesso às funcionalidades destes sistemas.

Deste modo, como implementar políticas de controle de acesso para os diversos sistemas de uma organização?

Segundo Silva e Saldanha (2006); Obelheiro et al. (2001) podemos utilizar os conceitos de controle de acesso baseado em papéis (*role-based access control* - RBAC) para definir o acesso de cada usuário nos sistemas, atribuindo papéis que representa um cargo ou função dentro da organização. Em cada papel é atribuído um conjunto de permissões. A permissão define o acesso a uma ou mais funcionalidades do sistema.

Os papéis podem ser organizados de forma hierárquica representando as respon-

sabilidades de cada usuário dentro da empresa. Os papéis atribuídos aos usuários que possuem cargos ou funções superiores herdam as permissões dos usuários de menor responsabilidade na organização (Obelheiro et al., 2001).

A autenticação é responsável por validar se o usuário tem acesso ao sistema. Após a autenticação, o sistema estabelece uma sessão contendo os papéis atribuídos ao usuário (Albuquerque Reis e Silva, 2004). Com os papéis ativos no sistema o usuário passa a ter acesso às funcionalidades que ele tem permissão.

Neste contexto, este trabalho propõe um *middleware* para autenticação e controle de acesso sendo responsável por disponibilizar, por meio de serviços, as informações de controle de acesso usando um arquivo JSON, no momento da autenticação do usuário no sistema. A aplicação, a partir da leitura do padrão do arquivo JSON, pode realizar a verificação dos papéis ativos para o usuário.

Um *middleware* (mediador) é uma aplicação responsável por integrar os sistemas. Ele cria um canal de comunicação entre as aplicações independente da plataforma, protocolos de comunicação ou sistemas operacionais (Maciel e Assis, 2004).

Outra justificativa para esse projeto, se deve a importância que os sistemas legados e distribuídos desempenham dentro de uma grande empresa. A substituição desses sistemas pode acarretar um grande impacto dentro da organização, tanto do ponto de vista estratégico quanto do econômico.

Conseguir manter os sistemas de uma organização dentro de suas políticas de segurança da informação é uma proposta que pode manter esses sistemas ativos por mais tempo. Com isso teremos um melhor controle das funcionalidades entre esses sistemas de acordo com o perfil de cada usuário.

Com um melhor controle de acesso nessas aplicações, fica mais fácil para o administrador gerenciá-las, bastaria definir os perfis de acesso dentro dessa aplicação e os sistemas por meio desses serviços poderiam definir as funcionalidades disponíveis para cada usuário.

Desta forma, este trabalho tem como objetivo geral apresentar um protótipo de sistema para controle de acesso baseado em papéis utilizando os conceitos de arquitetura orientada a serviços, e adotou os seguintes objetivos específicos:

- Realizar pesquisa bibliográfica sobre os principais conceitos sobre políticas de segurança da informação, técnicas de controle de acesso baseado em papéis.

- Pesquisar sobre tecnologias que permitem implementar webservices e arquitetura orientada a serviços.

- Desenvolver um protótipo de uma aplicação para fazer o controle de acesso em sistemas legados e distribuídos utilizando computação orientada a serviço.

- Implementação de rotinas desenvolvidas em outras tecnologias e arquiteturas de software a fim de servir como estudo de caso para acessar o controle de acesso criado como protótipo.

Esta monografia está organizada da seguinte forma:

Capítulo 1. Apresenta o tema e sua importância, a motivação, a justificativa e os objetivos deste trabalho.

Capítulo 2. Descreve os principais conceitos relacionados a políticas de segurança da informação, modelos de controle de acesso, *web services* e *middlewares* de autenticação e controle de acesso.

Capítulo 3. Apresenta o ABSecurity, um *middleware* para autenticação e controle de acesso. Descreve a análise de requisitos, projeto arquitetural e o padrão de Autenticação e Autorização.

Capítulo 4. Descreve os passos de modelagem, implementação e testes do ABSecurity, além de apresentar os protótipos implementados.

Capítulo 5. Contém uma síntese das contribuições deste trabalho e as sugestões de trabalhos futuros.

2 Fundamentação Teórica

2.1 Políticas de Segurança da Informação

A informação, independentemente da sua forma, é um dos maiores ativos de uma organização moderna, sendo vital para quaisquer níveis hierárquicos e dentro de qualquer organização que queira continuar competitiva no mercado. No passado a segurança da informação era algo mais simples, mas atualmente, vivemos em um mundo globalizado, fortemente interligado por redes, onde os problemas causados por *hackers*, vírus e ataques estão se tornando cada vez mais comuns, mais ambiciosos e incrivelmente mais sotisficados NBR ISO/IEC 17799 (2001).

Segundo NBR ISO/IEC 17799 (2001),

> "a informação é um ativo que, como qualquer outro ativo importante para os negócios, tem um valor para a organização e conseqüentemente necessita ser adequadamente protegida. A segurança da informação protege a informação de diversos tipos de ameaças para garantir a continuidade dos negócios, minimizar os danos aos negócios e maximizar o retorno dos investimentos e as oportunidades de negócio."

O objetivo principal de uma política de segurança é garantir que os usuários, equipes e gerentes tenham acesso a informações da empresa de acordo com seu perfil ou cargo (Spanceski, 2004). Essas informações podem existir de diversas maneiras em uma organização, como documentos impressos ou manuscritos, armazenadas eletronicamente, entre outros (NBR ISO/IEC 17799, 2001). Independente da forma como é apresentada, a informação deve ser protegida.

Uma política de segurança é conjunto de regras que devem ser seguidas para oferecer acesso aos recursos tecnológicos da empresa. De acordo com Spanceski (2004) uma boa política de segurança, em conjunto com os responsáveis pela segurança dos sistemas e equipamentos, é o que irá determinar o nível de segurança dos recursos tecnológicos da

empresa.

A segurança da informação é caracterizada pela preservação de confiabilidade, integridade e disponibilidade (NBR ISO/IEC 17799, 2001). É um recurso importante que busca diminuir ao máximo os riscos de vazamentos, fraudes, roubo de informações, paralisações, erros, uso indevido, sabotagens, ou qualquer outra ameaça que possa causar danos aos recursos tecnológicos da empresa (Spanceski, 2004).

A segurança é objetivo que se quer atingir, mas isso nem sempre é possível. O que pode ser feito é atingir um nível de segurança aceitável. Segurança é um processo que visa melhorar a segurança dos sistemas (Spanceski, 2004).

O processo de segurança consiste na análise do problema, síntese da solução e a avaliação da solução. Após todas essas etapas o processo deve ser reiniciado seguidamente. Esse processo é semelhante em muitos aspectos a tríade grega.

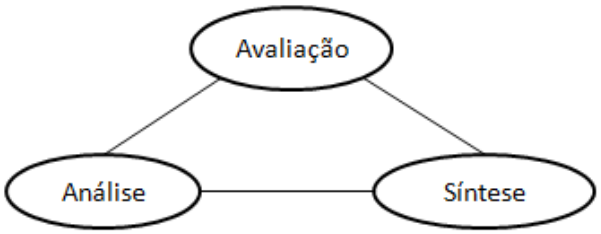

Figura 2.1: Tríade Grega (Spanceski, 2004)

2.1.1 Classificação da Informação

O objetivo da classificação da informação é garantir que os ativos da informação recebam um nível adequado de proteção. Classificar a informação é importante para indicar a prioridade, importância e o nível de proteção. Alguns itens podem necessitar de um tratamento diferenciado ou um grau maior de proteção de acordo com seu nível de criticidade e sensibilidade (NBR ISO/IEC 17799, 2001).

Para garantir que a informação seja protegida adequadamente é necessário algum sistema de classificação da informação para definir um conjunto apropriado de níveis de proteção. Segundo Spanceski (2004), nos dias de hoje a classificação mais comum é aquela dividida em quatro níveis:

- Secreta: são informações relevantes para a empresa, que são acessadas por um

número reduzido de pessoas e que devem ser muito bem protegidas. O acesso dessas informações por pessoas não autorizadas é extremamente crítico para a empresa.

- Confidencial: são informações restritas ao ambiente da empresa. Essas informações só podem ser acessadas se elas forem necessárias para o desempenho de alguma tarefa da empresa. Podem causar prejuízos financeiros para empresa caso sejam acessadas por usuários não autorizados.

- Interna: essas informações não devem sair da empresa, mas caso isso ocorra não será algo crítico para empresa.

- Públicas: são informações que podem ser disponibilizadas para um público em geral (clientes, fornecedores, imprensa, entre outros).

2.1.2 Princípios da Segurança da Informação

O usuário espera que suas informações estejam protegidas, no local determinado, que sejam confiáveis, corretas e sem que pessoas não autorizadas tenham acesso a seu conteúdo (Spanceski, 2004).

Os princípios de segurança representam os principais atributos que servem para orientar a análise, o planejamento e a implementação da segurança para proteger um determinado grupo de informações. De acordo com (Spanceski, 2004), esses princípios são:

- **Autenticidade**: A identificação de um usuário em um sistema está associado ao controle de autenticidade. Esse controle é responsável por proteger a informação contra possíveis intrusos. Geralmente é implementado utilizando um mecanismo de senhas ou assinatura digital.

- **Confiabilidade**: O objetivo da confiabilidade é proteger a informação contra pessoas não autorizadas. Para que uma pessoa acesse determinada informação ela deve estar autenticada e ter permissão para acessá-la.

- **Integridade**: A integridade tem como objetivo proteger que dados sejam alterados ou apagados por um usuário não autorizado. A integridade está ligada ao príncipio

de confiabilidade. Enquanto a integridade preocupa-se mais com a gravação, alteração e exclusão de dados, o príncipio da confiabilidade está mais voltado para a leitura dos dados.

- **Disponibilidade**: O princípio da disponibilidade tem como objetivo garantir que a informação sempre esteja disponível para um usuário autorizado sempre que necessário. A informação deve chegar ao usuário de forma integra e confiável. A disponibilidade da informação para usuários não autorizados e a degradação da informação podem causar danos graves para a empresa.

2.1.3 Vulnerabilidades

Segundo Spanceski (2004), vulnerabilidade é o ponto onde qualquer sistema pode sofrer um ataque, ou seja, o ponto onde uma fraqueza ou ausência de segurança que pode ser explorada para causar um incidente de segurança.

Para garantir a segurança da informação é necessário localizar os pontos vulneráveis e, se estes forem relevantes, a segurança nesse ponto deve ser revista ou implementada.

Segundo Spanceski (2004), as vulnerabilidades podem ser física, naturais, humanas, de software ou de hardware, entre outras. Alguns exemplos de vulnerabilidades:

- **Humana**: falta de comprometimento dos funcionários, compartilhamento de informações confidenciais por parte dos funcionários da empresa, falta de treinamento.
- **Física**: falta de infra-estrutura como instalações elétricas antigas, salas mal projetadas, falta de extintores, entre outros.
- **Naturais**: umidade, possibilidade de desastres naturais como terremotos, tempestades, entre outros.

As vulnerabilidades podem causar incidentes de segurança que podem afetar os négocios da empresa causando impactos negativos para os cliente e envolvidos. A principal causa de incidentes de segurança são as vulnerabilidades.

Os danos causados por um incidente de segurança acontecem pela má utilização ou implementação das medidas de segurança. Em alguns casos uma medida de segurança

pode ser boa e em outras não. Deve-se achar a melhor relação custo/benefício para garantir a segurança. Com medidas de segurança adequadas as vulnerabilidades diminuirão (Spanceski, 2004).

Segundo Spanceski (2004), alguns mecanismos de segurança podem ser implementados para diminuir a vulnerabilidade como políticas de segurança, cópia de segurança, controle de acesso segurança física, *firewall*, entre outros.

2.2 Modelos de Controle de Acesso

2.2.1 Conceitos e Modelos de Controle de Acesso

O controle de acesso define quais os acessos (criação, alteração, exclusão, consulta, entre outros) que um usuário terá a um determinado recurso de sua rede ou sistema. O controle de acesso pode ser implementado por meio de módulos de segurança, integrado a aplicações ou embutido no sistema operacional (Ferraiolo et al, 2001).

Um dos objetivos de uma organização no contexto de controle de acesso, é definir quem tem acesso a um recurso e qual será o tipo de acesso. O controle de acesso além de definir qual a permissão que um usuário tem ao recurso, também determina quando e como esse recurso pode ser utilizado. Comercialmente, esse objetivo pode ser descrito em termos de otimizar o compartilhamento de recursos. A otimização do compartilhamento de recursos é importante para conseguir melhor produtividade e utilidade do sistema (Silva e Saldanha, 2006).

O controle de acesso busca manter o sigilo e/ou disponibilidade dos recursos (Silva e Saldanha, 2006). Para isso é necessário utilizar conceitos de autenticação (*login*) e autorização (permissões de acesso). De acordo com a política de segurança e os recursos tecnológicos, o responsável pela segurança irá determinar como será implementado o controle de acesso.

Segundo Albuquerque Reis e Silva (2004), existem três classes quanto as políticas de controle de acesso:

- Controle de Acesso Discricionário (Discretionary Access Control - DAC): Os direitos de acesso a cada recurso são atribuídos livremente pelo proprietário do recurso.

- Controle de Acesso Obrigatório (Mandatory Access Control - MAC): A política de acesso é determinada pelo sistema e não pelo proprietário do recurso.

- Controle de Acesso Baseado em Papéis: O controle de acesso é baseado nos pápeis atribuídos aos usuários.

2.2.1.1 Autenticação e Autorização

Autorização e autenticação são dois conceitos importantes na área de controle de acesso. São conceitos diferentes, mas um depende do outro, de forma que para conseguir autorização para um determinado recurso, o usuário deverá estar autenticado no sistema (Silva e Saldanha, 2006). Se o usuário não for reconhecido pelo sistema não terá como definir quais os recursos que esse usuário poderá acessar no sistema. Na figura 2.2 mostra uma representação desse processo.

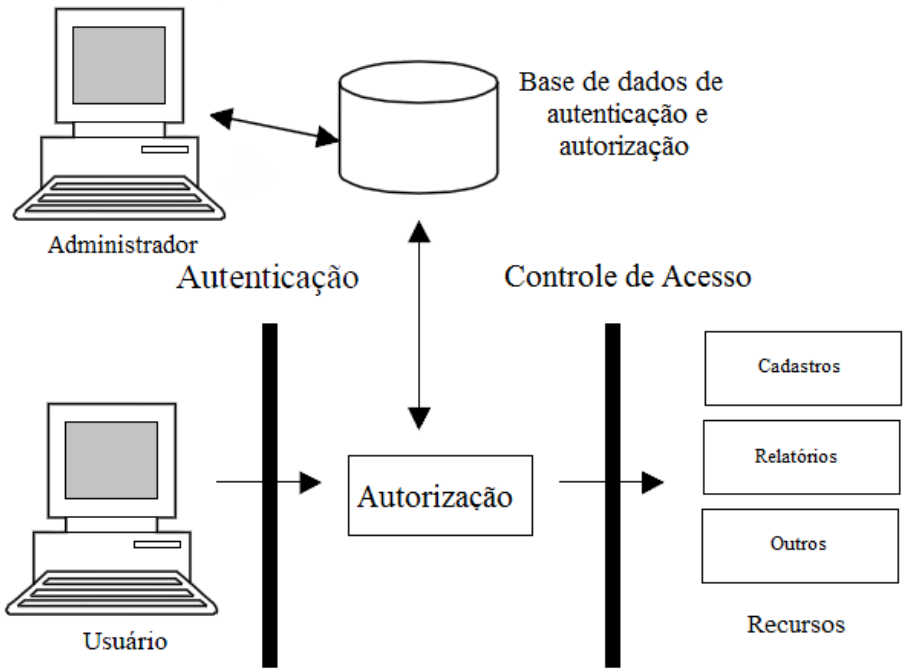

Figura 2.2: Controle de acesso

A autenticação é um processo que busca verificar a identidade do usuário no sistema, geralmente, no momento em que ele faz um login (acesso) em um sistema ou computador. Em outras palavras, a autenticação é que irá dizer quem somos dentro do sistema.

Autorização é utilizada para verificar se um determinado usuário previamente autenticado possui permissão para usar, manipular ou executar um recurso do sistema (Ferraiolo et al, 2001).

Um sistema deve atribuir a um usuário quais autorizações disponíveis para ele. Dessa forma, é possível identificar quais recursos um usuário poderá acessar no sistema. Se não houver como o sistema identificar um usuário, não será possível determinar de maneira correta quais recursos o usuário pode ou não pode acessar.

2.2.1.2 Usuários e Permissões em Transações

Os usuários em sistemas de informação são pessoas que utilizam o sistema para realizar um determinado trabalho (Silva e Saldanha, 2006). Podem ser desde os usuários comuns do sistema até administradores, programadores ou analistas de sistemas. Em muitos sistemas, é possível que um usuário único possua vários identificadores e estes podem estar ativos ao mesmo tempo. São mecanismos de autenticação que possibilitam o surgimento dessa situação.

Permissões (ou privilégios) são autorizações para realizar alguma ação no sistema, ou seja, refere-se a alguma forma de combinação entre um objeto e uma operação (Silva e Saldanha, 2006). As permissões definem quais áreas, tarefas e operações um usuário poderá executar no sistema.

Para realizar uma determina transação em um sistema o usuário deve possuir permissão para executar essa transação. Por exemplo, para que um usuário possa cadastrar um novo cliente será necessário ele ter permissão para essa transação.

2.2.1.3 Modelo de Bell-LaPadula

O modelo de *Bell-LaPadula* é uma descrição formal que usa um modelo de máquina de estados para determinar quais os caminhos permitidos para o fluxo da informação em um sistema. Esse modelo foi utilizado para determinar o controle de acesso em aplicações governamentais e militares.

Identificar as formas de comunicação disponíveis, em que é relevante a preservação do sigilo, é o principal objetivo desse modelo. Esse modelo é utilizado para tratar os

requisitos de segurança, de forma concorrente, dados com diversos níveis de sensibilidade. O modelo serve para formalizar políticas de segurança multinível (Silva e Saldanha, 2006).

É um modelo de transição formal do estado da política de segurança do computador que descreve um conjunto de regras de controle de acesso que utilizam etiquetas de segurança em objetos e autorizações para os indivíduos. No modelo, são adicionados níveis de segurança que refletem sistemas de segurança militar.

O modelo Bell e LaPadula adiciona controles de acesso obrigatórios ao sistema, impedindo o fluxo de informações de níveis de segurança mais altos (mais privilegiados) para os níveis de segurança mais baixos (Silva e Saldanha, 2006). Todo recurso recebe uma classificação e cada usuário um nível de segurança máximo.

2.2.1.4 Modelo Clark-Wilson

O modelo *Clark and Wilson* (1987) é um conjunto de regras gerais para satisfazer requisitos de sistemas comerciais. O modelo define dois princípios como os mais importantes na busca de garantia para manutenção da integridade: transações bem formadas e separação de responsabilidades (Silva e Saldanha, 2006).

O modelo de integridade *Clark-Wilson* fornece uma base para a especificação e análise de uma política de integridade de um sistema de computação. O modelo está principalmente preocupado com a formalização da noção de integridade das informações. A integridade das informações é mantido pela prevenção da corrupção de itens de dados em um sistema devido a qualquer erro ou má utilização.

Uma política de integridade descreve como os itens de dados no sistema devem ser mantidos válidos a partir de um estado do sistema para o outro e especifica os recursos de entidades diversas no sistema. Em outras palavras o modelo deve certificar que a informação só é modificada de forma autorizada por pessoas autorizadas.

De acordo com Silva e Saldanha (2006), o modelo de *Clark and Wilson* se difere do modelo de *Bell-LaPadula* por possuir uma abordagem que impõe o controle ao nível da aplicação. No modelo *Bell-LaPadula* a mediação de acesso fica no núcleo do sistema.

2.3 O Modelo RBAC

2.3.1 Visão Geral

O conceito de controle de acesso baseado em papéis (*role-based access control* - RBAC) surgiu com os primeiros sistemas computacionais multiusuários interativos, no início da década de 70 (Obelheiro et al., 2001). Uma grande parte dos sistemas desenvolvidos ultimamente utiliza algum modelo de RBAC para controlar o acesso (Guilheni, 2005).

De acordo com Ferraiolo et al (2001); Silva e Saldanha (2006) o controle de acesso baseado em papéis (RBAC), permite o acesso aos recursos de um sistema baseado no papel(*role*) que é definido para o usuário exercer numa determinada sessão.

Com o modelo RBAC, papéis são atribuídos a cada usuário baseado na suas responsabilidades, autoridade, cargo e competência. Essas associações de usuários a papéis podem ser anuladas e novas podem ser atribuídas facilmente. Com o RBAC, as permissões não são dadas aos usuários individualmente. As permissões são atribuídas a um determinado papel e ao usuário são atribuídos esses papéis de forma que o usuário pode ter acesso a essas permissões.

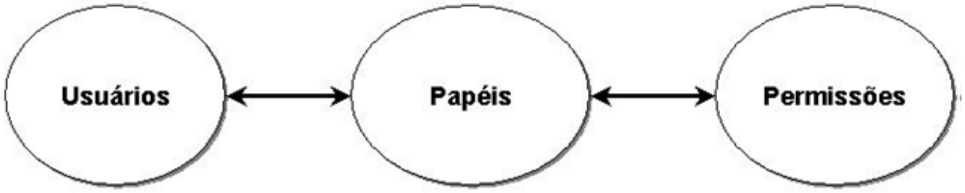

Figura 2.3: Relacionamento entre usuários, papéis e permissões (Silva e Saldanha, 2006)

Os papéis podem ser organizados de forma hierárquica representando as responsabilidades de cada usuário dentro da empresa. Os papéis atribuídos aos usuários que possuem cargos ou funções superiores herdam as permissões dos usuários de menor responsabilidade na organização (Obelheiro et al., 2001). Um exemplo de hierarquia de papéis é representado pela figura 2.4. No exemplo o papel Diretor Geral herda todas as permissões atribuídas aos outros papéis.

Segundo Guilheni (2005), a simplificação do processo de gerenciamento de permissões é a grande vantagem do uso do RBAC. Muitos usuários podem desempenhar o mesmo papel dentro da empresa. Para esses usuários podemos agrupá-los em papéis, que

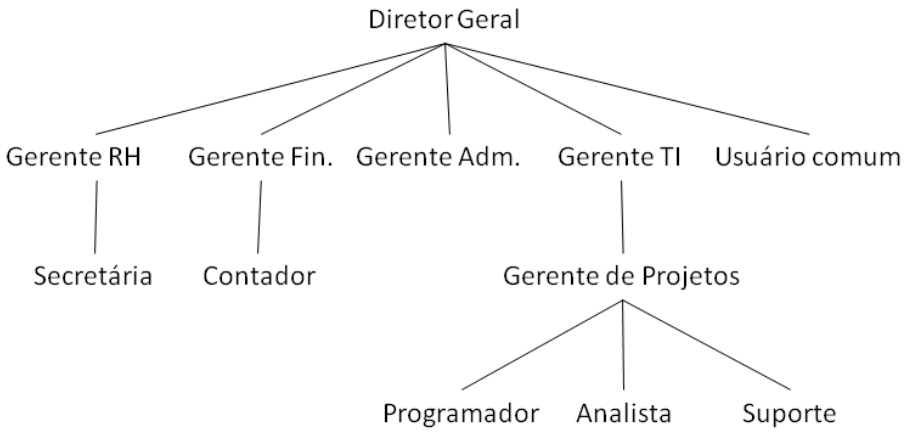

Figura 2.4: Exemplo de Hierarquia de Papéis

representam sua função na empresa junto ao sistema.

2.3.2 Permissões

As permissões definem quais tarefas, funções e responsabilidades poderão ser executadas por um usuário. Várias permissões podem ser atribuídas a um mesmo papel. processos e valores.

Segundo Obelheiro et al. (2001) ao modelar um sistema de controle de acesso, os administradores de sistemas podem tratar as permissões como um conceito abstrato que se refere à ligação arbitrária entre operações e objetos, levando em consideração, em alguns casos, processos e valores.

2.3.3 Ativação de Papéis

O RBAC exige primeiramente que o usuário esteja autorizado no sistema, bem como ser ativado em um papel, antes que possa executar alguma funcionalidade do sistema (Albuquerque Reis e Silva, 2004). Após a autenticação, o usuário estabelece uma sessão pela qual o usuário está associado a um subconjunto de papéis.

Uma autorização de papéis caracteriza a proibição de um usuário ter um papel ativo que não pertence a esse usuário. Somente a autorização de papel não é suficiente para o usuário ter permissão a algum recurso do sistema, é necessário que o papel seja ativado.

2.3.4 Família de modelos RBAC

O modelo possui sub-modelos que vão desde o mais simples ao mais complexo. Foi definida uma família de modelos que parte de um modelo básico que oferece as caractéristicas básicas do modelo RBAC. Componentes adicionais podem ser incluídos ao modelo básico para acrescentar novas funcionalidades e requisitos (Albuquerque Reis e Silva, 2004). Segundo Silva e Saldanha (2006) , o modelo RBAC possui quatro sub-modelos:

- RBAC Básico (*Core RBAC*): Inclui os aspectos essenciais presentes em todos modelos RBAC. No modelo básico, papéis são atribuídos aos usuários e os usuário herdam as permissões atribuídas a esse papel. O modelo exige que haja uma associação de muito-para-muitos entre usuários-papéis e papéis-permissões.

- RBAC Hierárquico (*Hierarchical RBAC*): Inclui o conceito de hierarquia de papéis. Nesse sub-modelo existe o conceito de herança de papéis.

- RBAC com Restrições Estáticas(*Static Constrained RBAC*): Inclui restrições impostas na designação de papéis, neste caso, se um papel se um usuário está incluído na lista de membros de um papel A e há um relacionamento de restrição estática entre o papel A e um papel B, este usuário não poderá ser incluído na lista de membros do papel B. Este relacionamento de restrição estática também deve ser respeitado pelo RBAC Hierárquico quando este for utilizado.

- RBAC com Restrições Dinâmicas (*Dynamic Constrained RBAC)*): Impõe restrições na ativação de conjunto de papéis que podem ser adicionados como atributos do sujeito de um usuário. Este modelo é semelhante ao RBAC de Restrições estáticas, porém de forma menos restritiva. Enquanto as restrições estática são aplicadas no momento da designação de papés, as restrições dinâmicas são aplicadas no momento da ativação do papel por um usuário.

É importante destacar que os modelos descritos acima apenas descrevem formalmente o modelo RBAC. O RBAC é apenas uma especificação e sua implementação é livre para definir como a política de controle de acesso a um recurso deve ser feita, como papéis devem ser descritos e como atribuir papéis aos usuários.

2.4 *Web Services*

Web services é uma solução utilizada para integrar e comunicar sistemas. É possível com essa tecnologia que novas aplicações troquem informações com sistemas distribuídos e heretogêneos existentes. De acordo com Furtado et al. (2009), os *web services* podem ser definidos como programas modulares, independentes de plataforma e auto-descritivos que podem ser invocados através da internet. Geralmente são contruídos utilizando algumas especificações como XML, SOAP, WSDL, REST e UDDI.

De acordo (Furtado et al., 2009), *web services* disponibilizam padrões de desenvolvimento para implementar funções de negócios que possam ser invocadas remotamente. *Web services* utiliza como plataforma básica o XML com HTML. O XML (*Extensible Markup Language*), é uma linguagem de marcação, de fácil manipulação, independente de plataforma ou linguagem de programação e com ela é possível descrever diversos tipos de dados, funções e mensagens.

Uma alternativa ao XML é o JSON (*JavaScript Object Notation*). O JSON é um formato de texto para serialização de dados estruturados que tem como objetivo ser simples, portável e textual. Pode representar quatro tipos primários(*strings*, booleanos, números e nulos) e dois tipos estruturados(listas e objetos) (Fonseca e Simões, 2007).

Existem muitas maneiras de se contruir e usar *web services*, por possuir diversas tecnologias associadas e múltiplas camadas (Furtado et al., 2009). A figura 2.5 ilustra a arquitetura de *web services* e algumas famílias dessas tecnologias.

Figura 2.5: Arquitetura de *web services* (Ramalho, 2004)

De acordo com Ramalho (2004) o ciclo de vida de web services possui quatros estados distintos:

- **Publicação**: Responsável por registrar seu serviço no repositório de *web services*.
- **Descoberta**: Processo pelo qual a aplicação toma conhecimento da existência do *web service* pesquisando num repositório.
- **Descrição**: Nesses processo o *web service* disponibiliza sua API, desta forma a aplicação cliente consegue visualizar toda a interface do *web service*, onde estão descritas todas suas funcionalidades.
- **Invocação**: Processo responsável pela troca de mensagens entre aplicações cliente e servidor.

De acordo com Ramalho (2004), a utilização desses quatro estados permite constituir os ciclo de vida do *web service* descritos a seguir:

- Contrução do *web service* utilizando alguma linguagem de programação;
- Especificação da interface do serviço definido em um documento WSDL;
- Registro do serviço em um diretório UDDI;
- A aplicação cliente busca em um repositório UDDI e encontra o serviço;
- A aplicação cliente estabelece uma conexão com o serviço e trocas mensagens utilizando SOAP.

2.4.1 WSDL

O WSDL (*Web Services Description Language* ou Linguagem de Descrição de Serviços Web) é utilizada para descrever *web services* baseado na linguagem XML. Esse documento descreve o serviço, como acessá-lo e quais são as operações ou métodos disponíveis (Furtado et al., 2009).

De acordo com Furtado et al. (2009), o WSDL é dividido nas seguintes camadas:

- Descrição de interface do serviço: responsável por descrever quais as operações e seus parâmetros;
- Conexão do serviço: indica qual protocolo e formato as operações dos serviços são disponibilizadas;
- Localização física: onde o serviço está disponível.

2.4.2 SOAP

SOAP (*Simple Object Access Protocol* ou Protocolo Simples de Acesso a Objeto) é um protocolo baseado na linguagem XML responsável pela troca de informações de estruturadas em uma plataforma descentralizada e distribuída. Permite baixo acoplamento entre cliente e servidor e permite comunicação, independente de protocolo, entre serviços de diferentes organizações (Furtado et al., 2009).

Segundo Furtado et al. (2009), este protocolo possuir três partes:

- Um envelope, que define a mensagem e como processá-la;
- Um conjunto de regras codificadas para expressar instâncias de tipo de dados definidos pela aplicação;
- Uma convenção para representar chamadas de métodos e respostas.

Durante uma requisição SOAP o *Listener* aceita a mensagem SOAP, extrai o XML do corpo da mensagem, transforma a mensagem XML em um protocolo nativo, delega a requisição ao processo, extrai do documento XML o corpo da mensagem, transforma a mensagem XML em um protocolo nativo, passa a requisição ao serviço corrente e retorna a resposta em um documento no formato XML (Furtado et al., 2009).

2.4.3 REST

O protocolo REST(Representational State Transfer - Transferência do Estado Representacional),também conhecido como RESTful http, possui muitos princípios de arquiteturas de rede que foca em acesso a recursos de formas simples e sem manutenção de estado.

Possui uma série de restrições de arquitetura com o objetivo de melhorar a escalabilidade, desempenho e abstração de recursos em sistemas distribuídos (Furtado et al., 2009).

De acordo com Furtado et al. (2009), os princípios de arquitetura incluem:

- Cada requisição de um cliente ao servidor possui toda a informação necessária para compreender o pedido.
- Os recursos possuem a mesma interface que o cliente, como utiliza o protocolo HTTP, as únicas operações permitidas são: GET, POST, PUT DELETE.
- Cada recurso possui uma identificação única representada por uma URI.
- Componentes REST manipulam recursos utilizando troca de representações de recursos. Esta representação pode ser um documento HTML, XML ou JSON.

Web services com SOAP e *web services* com REST possuem algumas diferenças. Rest usa XML sobre HTTP sem uma definição de interface WSDL, o formato de mensagem é um documento XML e utiliza apenas o HTTP como protocolo de comunicação. SOAP define um interface WSDL, o formato da mensagem é um documento XML em um envelope SOAP e roda sobre diversos protocolos de comunicação como o HTTP, FTP, SMTP, entre outros (Furtado et al., 2009).

2.5 Middlewares usados para o modelo RBAC

2.5.1 Visão Geral

Um *middleware* (mediador) no âmbito de computação distribuída é um sistema que faz a intermediação entre sistemas. Sua principal função é criar um canal de comunicação entre sistemas de plataformas, protocolos de comunicação e sistemas operacionais diferentes (Maciel e Assis, 2004).

Um *middleware* tem como objetivo diminuir a complexidade e heterogeneidade de diversos sistemas existentes, disponibilizando serviços, de forma transaparente, entre aplicações distribuídas. Este tipo de componente é muito utilizado quando um sistema atual deve interoperar com sistemas legados e heterogêneos (Maciel e Assis, 2004).

Todo o processo é transparente entre as aplicações e a heterogeneidade existente é tratada também pelo middleware. A figura 2.6 ilustra a comunicação entre aplicações distribuídas através de um *middleware*.

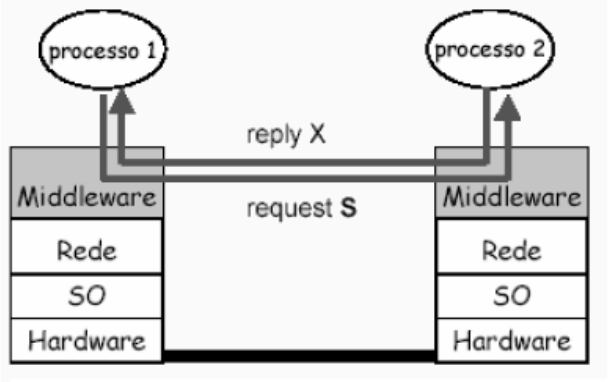

Figura 2.6: Comunicação através de *middleware* (Maciel e Assis, 2004)

Poucos *middlewares* suportam integração entre aplicações implementadas em diferentes linguagens. Uma tecnologia que permite integração entre diferentes linguagens é o CORBA (*Common Object Request Broker Architecture*), pois neste *middleware* é possível fazer o mapeamento através de uma interface comum a essas aplicações.

A utilização de *middlewares* para definir controle de acesso entre sistemas é muito importante, pois esses sistemas utilizam tecnologias diferentes e estão distribuídos geograficamente. Com o *middleware* essas aplicações podem ser integradas facilmente.

2.5.2 MACA - Middleware de Autenticação e Controle de Acesso

O objetivo do *Middleware* de Autenticação e Controle de Acesso (MACA) é prover os serviços de autenticação de usuário e de autorização de acesso para aplicações legadas ou em desenvolvimento, independente de plataforma e de linguagem de programação, através de uma API padronizada.

2.5.2.1 RBAC baseado em WebServices

De acordo com Bhatti et al. (2003) o aumento do número de sistemas distribuídos que exigem maior segurança, e a demanda para o compartilhamento de informações de conteúdo online em várias aplicações Internet, serviços de segurança da Web está se tornando uma tarefa cada vez mais importante.

Esses serviços da Web introduzem um novo conjunto de desafios de segurança que não são previstas em modelos tradicionais de segurança. Com utilização de webservices é necessário representar os elementos do RBAC em XML ou JSON, definições de esquemas são gerados para o usuário, papel e permissão. Através desses documentos XML que é definido quais usuários tem acesso ao sistemas e quais recursos ele terá acesso.

No trabalho descrito em (Sohr et al., 2006), foi desenvolvido um *framework* de autenticação e autorização baseado em webservices utilizando o modelo RBAC. A arquitetura desse sistema está representado na figura 2.7. A comunicação entre os componentes desse *framework* é feito utilizando *web services* através de trocas de mensagens SOAP.

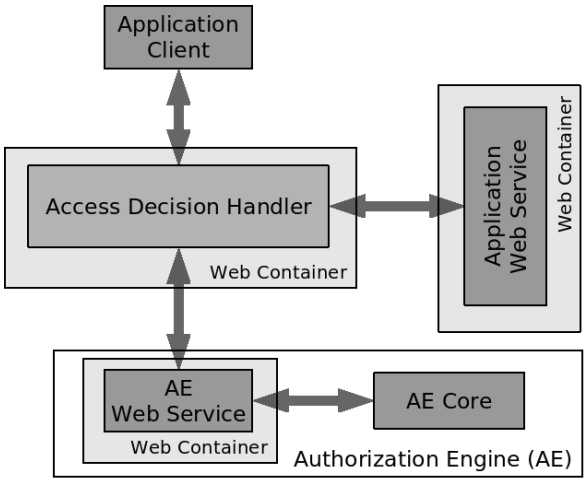

Figura 2.7: Arquitetura de um middleware RBAC baseado em *web services* (Sohr et al., 2006)

O componente principal dessa arquitura é o *Access Decision Handler*, implementado utilizando um *interceptor*, um componente de *middleware*. O mecanismo de autorização é feito utilizando esse *interceptor* que tem como objetivo impor todas as políticas de segurança baseado no modelo RBAC. Um dos benefícios dessa abordagem é que a lógica de autenticação e autorização não precisa estar contida na aplicação e qualquer mudança nas políticas de segurança não requer qualquer modificação da aplicação (Sohr et al., 2006).

2.5.2.2 RBAC para o Modelo CORBA de Segurança

No sentido de minimizar os problemas de segurança encontrados em sistemas de objetos distribuídos, um modelo de referência foi elaborado pela OMG (*Object Management*

Group) para segurança em sistemas de objetos distribuídos que utilizam a arquitetura CORBA (*Common Object Request Broker Architecture*). Esse modelo quando aplicado corretamente provê alto nível de segurança para essas aplicações.

A especificação de segurança CORBA define um conjunto de objetos e os relacionamentos entre esses objetos em um modelo capaz de fornecer funcionalidades como identificação e autenticação de principais, controle de acesso, comunicação segura entre objetos auditoria e administração de segurança (Obelheiro et al., 2001).

Segundo a especificação, o modelo CORBA de segurança é representado em quatros níveis, apresentados na figura 2.8. O nível de aplicação é composto pelos objetos de aplicação (cliente e servidor). O nível de *middleware* contém o serviços ORB, objetos de serviços COSS(*Common Object Service Specification*) que são contruídos sobre o núcleo ORB (*Object Request Broker*) e estendem funcionalidades básicas, adicionando novas características, implementando a segurança dos objetos distribuídos no nível de *middleware*. Os serviços subjacentes de segurança se encontram no nível de tecnologia. Esses serviços definem protocolos de implementação de funcionalidades como integridade e confiabilidade na comunicação cliente-servidor. O nível de proteção básica é o nível inferior, que representa o sistema operacional e o hardware (Obelheiro et al., 2001).

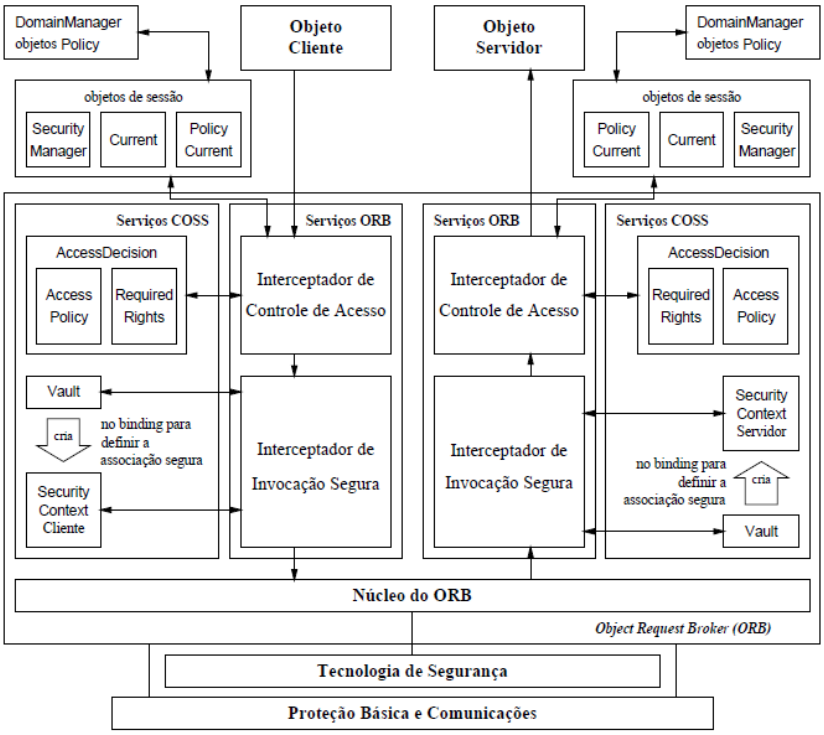

Figura 2.8: Modelo CORBA de segurança (Obelheiro et al., 2001)

No modelo CORBA de segurança é definido o conceito de principal responsável por mediar as invocações de operações de objetos. O modelo especifica um conjunto de objetos de serviços que implementam os controle de segurança em um sistema CORBA.

3 Proposta de um *Middleware* para Controle de Acesso Baseado em Papéis usando *Web Services* utilizando o sistema ABSecurity

Este trabalho apresenta um *middleware* para autenticação e controle de acesso baseado em papéis utilizando *web services*. O *middleware* foi implementado e disponibilizado juntamente com o sistema chamado ABSecurity. O desenvolvimento do ABSecurity foi realizado através do levantamento de requisitos discutidos junto ao orientador, e das reuniões realizadas semanalmente.

Todas as funcionalidades do sistema se basearam nos princípios da segurança da informação e no modelo RBAC hierárquico. Esse modelo foi escolhido por oferecer um melhor controle das permissões de um sistema, além de contemplar, os papéis existentes na organização, facilitando sua compreensão (Silva e Saldanha, 2006).

Foi utilizado *web services*, para a disponibilização dos perfis de acesso cadastrados na aplicação ABSecurity, pela facilidade de integração de sistemas distribuídos e heterôgeneos (Sohr et al., 2006). Foi criado um padrão para que as aplicações possam se comunicar com o serviço de autenticação e autorização do ABSecurity.

Após o processo da coleta de informações necessárias, a próxima etapa foi realizar um estudo sobre a arquitetura, modelagem do sistema e da sua respectiva base de dados.

3.1 Modelo de Processo de Desenvolvimento de Software Utilizado para o ABSecurity

Segundo Sommerville (2004), um processo é um conjunto de atividades e resultados associados, que tem como objetivo gerar um produto de software. Geralmente são executados

por engenheiros de software e não existe um processo ideal, onde cada empresa utiliza uma abordagem diferente. Um modelo de processo de software é uma representação abstrata de um processo de software (Sommerville, 2004). O modelo de processo utilizado no desenvolvimento desse trabalho foi o modelo de desenvolvimento em cascata. De acordo com Sommerville (2004), este modelo leva em consideração as atividades mostradas anteriormente e as representa como fases separadas do processo, como especificações de requisitos, projeto de software, implementação e testes. Após a aprovação da cada estágio, o desenvolvimento passa para o próximo estágio.

O sistema foi desenvolvido em um período de quatro meses e as etapas do processo de desenvolvimento são descritas a seguir:

1. **Análise de Requisitos:** Realizada junto com o orientador para definição e compreensão da arquitetura e funcionalidades do sistema.
2. **Modelagem do Sistema:** Definição das entidades e seus relacionamentos.
3. **Implementação:** Implementação dos serviços e funcionalidades do sistema.
4. **Fase de teste:** Nesta fase foram realizados testes de carga para verificar quantos usuários o sistema suporta.

3.2 Levantamento de Requisitos

Durante a fase de levantamento de requisitos foi realizado um estudo do modelo RBAC para conhecer os processos envolvidos e identificar os requisitos funcionais e não funcionais do sistema. Os requisitos funcionais descrevem as ações que um sistema pode executar, sem levar em consideração restrições físicas. (Sommerville, 2004). Requisitos não funcionais definem apenas os atributos do sistema como desempenho, segurança, usabilidade e confiabilidade (Sommerville, 2004).

Nesta fase foram identificados os seguintes requisitos funcionais do sistema:

- **Cadastro de Usuários:** Incluir, alterar, consultar e excluir usuários que utilizarão o sistema ABSecurity e os sistemas gerenciados por ele. Na inclusão e alteração é possível atribuir papéis cadastrados ao usuário.

- **Cadastro de Papéis:** Incluir, alterar, consultar e excluir papéis. Nesta funcionalidade também é possível atribuir transações cadastradas aos papéis e visualizar os usuários atribuídos a este papel.

- **Cadastro de Transações:** Incluir, alterar, consultar e excluir transações. Cada transação é uma permissão que um usuário terá no sistema. Transações são formadas por funcionalidades e tipos de transações. Exemplo: Uma transação de consulta a dados do cliente.

- **Cadastro de Funcionalidades:** Incluir, alterar, consultar e excluir funcionalidades. Esta funcionalidade tem como objetivo cadastrar as funcionalidades presentes em cada sistema que utiliza o ABSecurity. Exemplo: Cadastro de Clientes, Cadastro de Fornecedores, entre outros.

- **Cadastro de Tipos de Transações:** Incluir, alterar, consultar e excluir tipos transações. Tipo de transação é uma ação presente em uma ou mais funcionalidades. Exemplo: Incluir, excluir, alterar e consultar.

- **Cadastro de Sistemas:** Incluir, alterar, consultar e excluir dos sistemas que utilizarão o ABSecurity para gerenciar os perfis de acesso.

- **Geração de *logs*:** Funcionalidade responsável por gerar *logs* dos usuários que fazem *login* no ABSecurity e nos sistemas gerenciados por ele. Exemplo: Sistema de Gestão(PHP), Controle Financeiro (Java), entre outros.

Durante o levantamento de requisitos de um software o usuário não se lembra de informar os requisitos não funcionais. Ele está preocupado com as funcionalidades do sistema. A pessoa responsável pela análise de requisitos deve explorar e questionar esse assunto. Os requisitos não funcionais do sistema estão descritos a seguir.

- **Desempenho**: O tempo de resposta do sistema durante seu funcionamento deve ser rápido.

- **Usabilidade**: O sistema deve apresentar facilidade de uso.

- **Confiabilidade**: O sistema deve ser confiável para que não ocorram erros e desvios de informação.

- **Segurança**: Requisito não funcional mais importante do sistema. Está associado a privacidade e integridade dos dados dos usuários.

3.3 Arquitetura da Aplicação

Nesta etapa do processo de desenvolvimento, foi feito o projeto da arquitetura, que tem como objetivo, estabelecer uma estrutura global do sistema. É muito importante para o sucesso de qualquer projeto de *software*, uma boa estruturação, com uma descrição consistente de cada parte do sistema e suas propriedades, além dos processos de comunicação e relacionamento entre elas (Sommerville, 2004).

No projeto foi utilizado a arquitetura em camadas, baseado neste modelo foram definidas as camadas do sistema, que se sobrepõe, partindo das camadas de nível mais baixo, que fazem a comunicação com o banco de dados, até as camadas mais elevadas, responsáveis pela interface com o usuário. Nas camadas intermediárias se localizam a lógica, funcionalidades e serviços dos sistema (Sommerville, 2004). Na figura 3.1 estão representados os principais componentes da arquitetura do ABSecurity.

No projeto da arquitetura foram utilizados alguns padrões de projeto. Padrões descrevem soluções para problemas que se repetem em um determinado contexto. Um padrão de projeto determina um nome e define o problema, a solução, quando aplicar esta solução e seus resultados (Santos, 2008).

O *Core* é o núcleo do sistema ABSecurity. Ele é responsável pela lógica, comunicação e persistência de dados no banco, além de disponibilizar os serviços, para a aplicação Web e para o *web service*. O *Core* possui três componentes principais:

- ***Entity* (Entidade)**: Representa as entidades do sistema, ou seja, a grosso modo é um registro de uma tabela em um banco de dados. Ao invés de manipular os dados diretamente no banco, foi utilizado *entities* para realizar essa tarefa.

- **DAO (*Objeto de acesso a dados*)**: Abstrai o mecanismo de persistência utilizado na aplicação. A camada de serviços acessa os dados persistidos sem saber se

os dados estão em um banco de dados ou em um arquivo (Berger, 2005).

- ***Service* (Serviço)**: Responsável pela lógica de negócio e disponibilização de serviços para os outros componentes do sistema. O Service é o único componente da aplicação que tem acesso ao DAO.

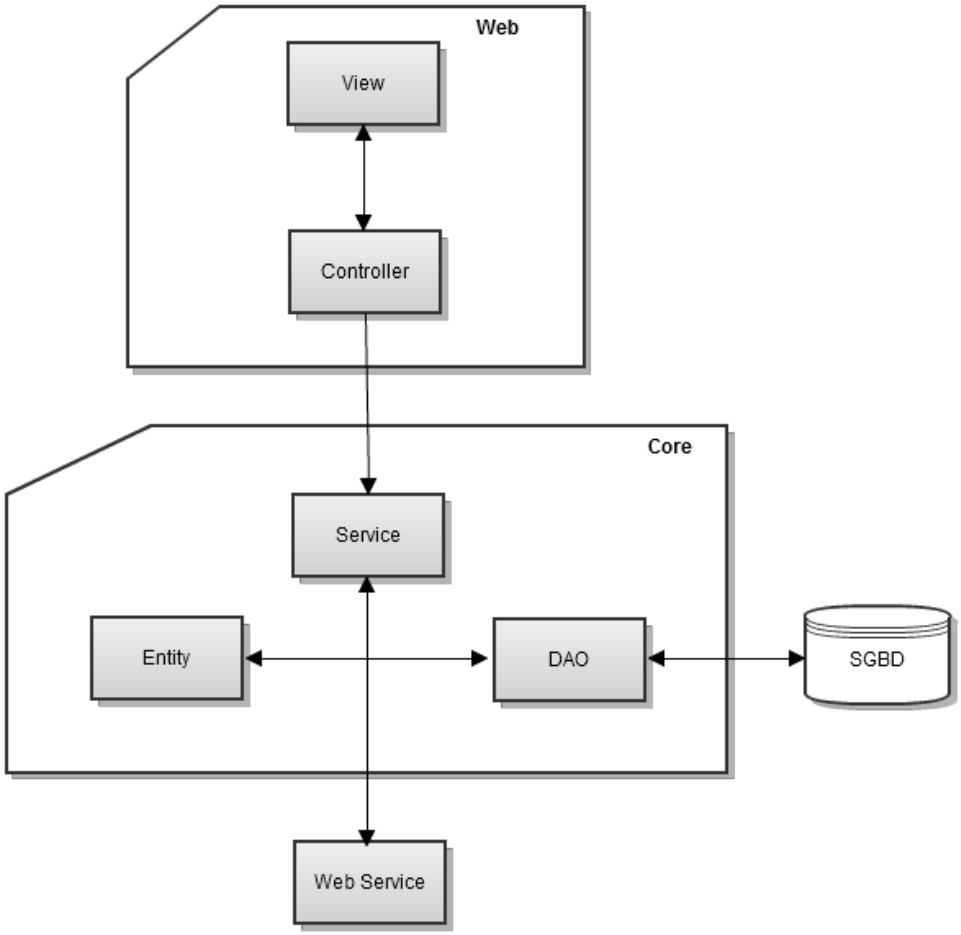

Figura 3.1: Arquitetura do sistema ABSecurity

A aplicação *Web* fornece uma interface gráfica para o usuário manipular as informações necessárias para criação de perfis de usuário, além de disponibilizar um relatório de acesso do usuário às aplicações. Utiliza o padrão de arquitetura de software MVC, que visa separar a lógica de negócio da lógica de apresentação (Santos, 2008). O padrão MVC está dividido em três camadas:

- **Camada Lógica da Aplicação (*Model*)**: Representa os dados da aplicação e as regras de negócio que administra o acesso e a atualização dos dados. Ela também é responsável por guardar o estado de persitência e disponibilizar as funcionalida-

des e serviços encapsuladas pelo próprio modelo (Santos, 2008). Esta camada é representada pelo componente *Core* da aplicação.

- **Camada de apresentação (*View*)**: Apresenta as informações de forma apropriada para o usuário e transmite para camada de controle, os eventos de ações do usuário. Acessa os dados da camada lógica por meio da camada de controle, além de fornecer componentes, para entrada e saída de dados (Santos, 2008).

- **Camada de Controle da Aplicação (*Controller*)**: Especifica o comportamento da aplicação, recebe a entrada de dados e inicia a resposta ao usuário ao chamar objetos da camada lógica. Ela também é responsável por validar e filtrar a entrada de dados (Santos, 2008).

O *web service* é responsável por fornecer o perfil de acesso do usuário, para as aplicações no momento do *login*, por meio do serviço de autenticação e autorização. Para que as aplicações acessem esse serviço foi utilizado um padrão, responsável por acessar e disponibilizar o perfil do usuário ao restante da aplicação, além de fornecer métodos para verificação de permissões do usuário.

O Sistema de Gerenciamento de Banco de Dados (SGBD) é responsável por gerenciar a base de dados da aplicação, retirando da aplicação a responsabilidade de gerenciar o acesso, a organização e manipulação dos dados.

3.4 Padrão de Autenticação e Autorização

Para que as aplicações possam se comunicar com o sistema ABSecurity foi criado o padrão de Autenticação e Autorização. A utilização de um padrão fará com que todos os sistemas falem a mesma língua, melhorando a qualidade, facilitando a implementação e manutenção da aplicação.

Este padrão centraliza todas as informações e métodos responsáveis pelo controle de acesso da aplicação. Deve ser implementado utilizando uma classe ou módulo visível em todo sistema. Nele deve conter os campos com as informações e as transações disponíveis do usuário, além de fornecer métodos, que acessem o *web service*, converta o retorno do serviço para um formato apropriado da linguagem e forneça métodos de autenticação e

verificação de transações. A figura 3.2 ilustra o padrão apresentado, com seus campos e métodos. Novos métodos e campos podem ser acrescentados caso seja necessário.

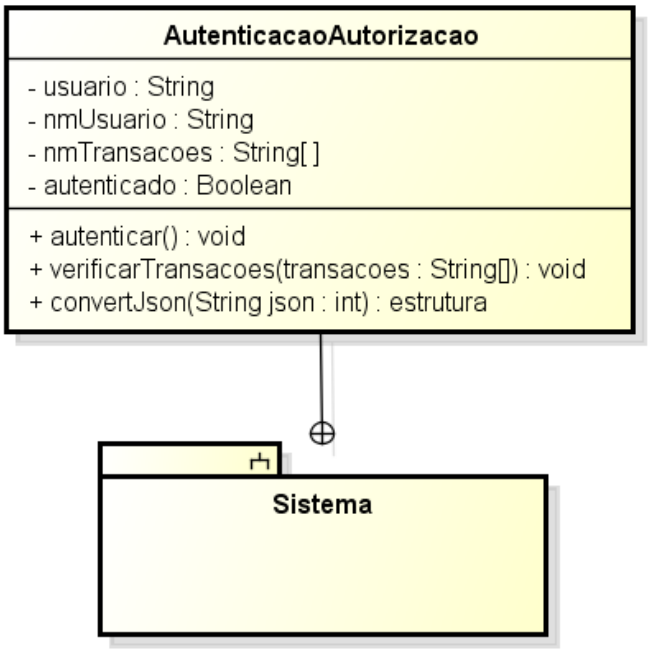

Figura 3.2: Padrão de Autenticação e Autorização

O método de autenticação faz uma requisição ao serviço de Autenticação e Autorização, passando como parâmetros, o usuário, a senha e o identificador do sistema. O serviço verifica a autenticidade do usuário e suas transações de acordo com os papéis atribuídos a ele. Essas informações são retornados pelo serviço no formato JSON.

Os campos com as informações e as transações do usuário são semelhantes ao do retorno do serviço de Autenticação e Autorização e possuem os seguintes campos:

- **nmUsuario**: Nome completo do usuário.
- **usuario**: Usuário do sistema.
- **nmTransacoes**: Lista com os nomes das transações disponíveis para o usuário no sistema. Através da lista com os nomes das transações disponíveis que é feito o controle de acesso do usuário no sistema.
- **autenticado**: Campo que diz ao sistema se o usuário está autenticado.

Um exemplo do retorno do *web service* no formato JSON é exemplificado a seguir:

```
{
    "nmTransacoes" :
    [
        "SGE_CAD_CONTP_INC",
        "SGE_CAD_CONRE_ALT",
        "SGE_CAD_CONTP_CON",
        "SGE_CAD_CONTP_EXC",
        "SGE_CAD_CONRE_INC",
        "SGE_CAD_CONRE_EXC",
        "SGE_CAD_PC_INC",
        "SGE_CAD_PC_ALT",
        "SGE_CAD_PC_CON",
        "SGE_CAD_PC_EXC",
          ],
    "usuario" : "adriano.bueno",
    "nmUsuario" : "Adriano Reiné Bueno",
    "autenticado" : true
}
```

A aplicação ao receber o retorno do serviço o converte para alguma estrutura de dados da linguagem. Existem muitas bibliotecas que convertem JSON para alguma estrutura da linguagem. O programador é livre pra decidir se implementa um conversor ou utiliza uma biblioteca.

O método de verificação de transações é utilizado para controlar o acesso as funcionalidades. Ele recebe como parâmetros as transações do usuário e verifica se o usuário tem acesso aquele recurso. De acordo com necessidades de cada aplicação, novos métodos de verificação podem ser implementados.

4 Aspectos de Implementação do Sistema ABSecurity

Neste capítulo são descritos alguns aspectos de implementação do sistema ABSecurity. O sistema foi modelado e implementado com base na análise de requisitos, no projeto arquitetural e no padrão de Autenticação e Autorização descritos no capítulo 3. Nesta fase, também foram desenvolvidos três protótipos de aplicação em plataformas e tecnologias diferentes para controlar o acesso dos usuários utilizando o sistema ABSecurity. Por último, foram realizados alguns testes de carga para verificar o desempenho das aplicações em um ambiente distribuído.

4.1 Modelagem do Sistema

Nesta etapa do processo de desenvolvimento foram construídos alguns modelos, com base nas informações coletadas na análise de requisitos e no projeto arquitetural do sistema. A modelagem de sistema tem como objetivo criar modelos que expliquem as propriedades e o comportamento de um sistema. Na construção da aplicação os modelos são usados na identificação das características e funcionalidades, e no planejamento de sua contrução (Sommerville, 2004).

A figura 4.1 exemplifica um diagrama de classes de uma parte do sistema responsável pelo *login* na aplicação ABSecurity e pela disponibilização do serviço de autenticação. Nesse diagrama é possível identificar todos os componentes do projeto arquitetural do sistema. A classe AutenticacaoAutorizacaoDAO é responsável pelo acesso ao banco de dados, através da *entities*, para obter as informações necessárias para o controle de acesso do usuário. A classe AutenticacaoAutorizacaoService é responsável por oferecer os serviços para a classe AutenticacaoAutorizacaoController, responsável por controlar o *login* do sistema ABSecurity, e AutenticacaoAutorizacaoWebService, responsável por disponibilizar o perfil de acesso do usuário para as outras aplicações.

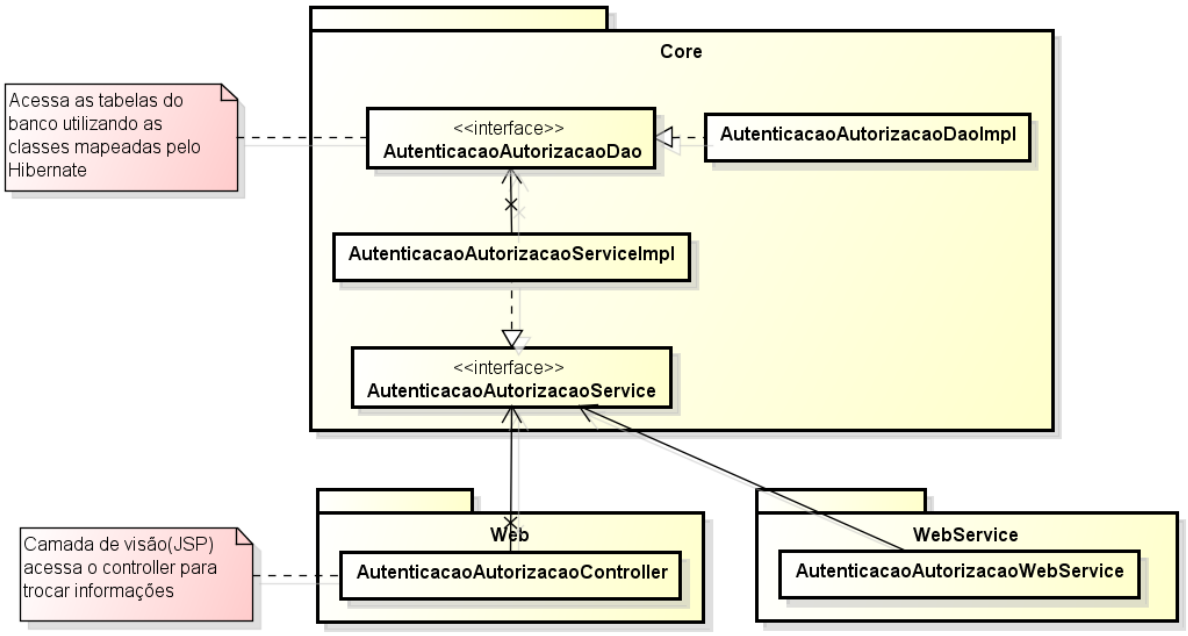

Figura 4.1: Diagrama das classes de autenticação e autorização

Todas as funcionalidades implementadas no sistema foram modeladas de forma semelhante a da figura 4.1. A fase de modelagem do sistema foi fundamental para compreensão dos serviços e funcionalidades do sistema, e esteve em constante mudança no processo de desenvolvimento do sistema ABSecurity.

4.1.1 Modelagem do Banco de Dados

A modelagem do banco de dados foi desenvolvida com base nos requisitos coletados e com ajuda de uma ferramenta CASE, essa modelagem não é feita de acordo com o modelo conceitual entidade relacionamento, ela é mais ampla, mais próxima de como foi implementado o banco de dados.

A figura 4.2 mostra a modelagem do banco de dados. No modelo é possível identificar todas as entidades do sistema e seus relacionamentos. Também é possível identificar os componentes principais do modelo RBAC.

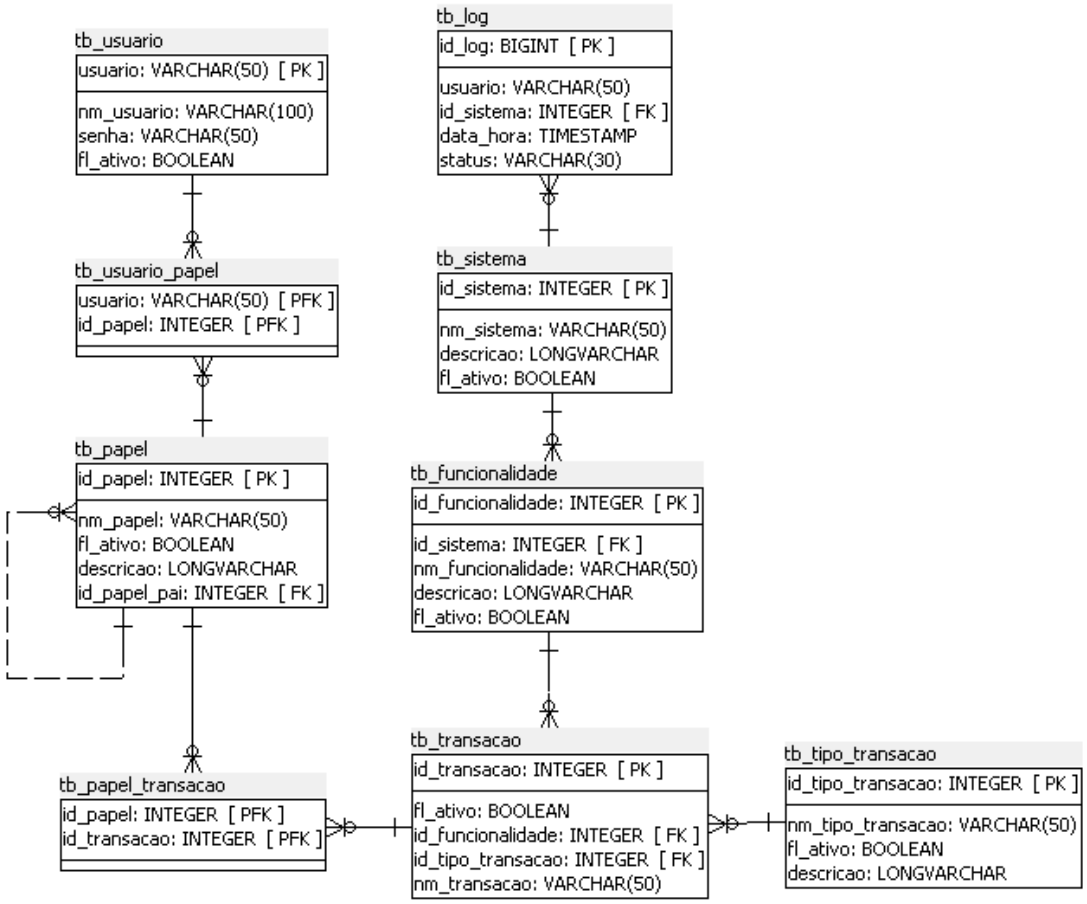

Figura 4.2: Diagrama da base de dados do ABSecurity

4.2 Implementação

Neste estágio do processo de desenvolvimento foi realizada a implementação do ABSecurity. A fase de implementação do desenvolvimento do sistema é o processo de transformação de uma especificação de sistema em um sistema executável. Essa fase envolve processos de projeto e programação de sotware (Sommerville, 2004).

Foi utilizada a linguagem de programação Java em conjunto com os *frameworks* *Spring* e *Hibernate*. O *Spring Framework* facilitou muito o desenvolvimento da aplicação, pois oferece diversos módulos que podem ser utilizados de acordo com as necessidades do projeto, como módulos voltados para desenvolvimento Web, MVC, persistência, controle de transação, acesso remoto e programação orientada a aspectos. O módulo *core* do *Spring* deu suporte a injeção de dependência e inversão de controle, muito comum em *frameworks*. O módulo MVC foi essencial para a implementação do padrão MVC descrito na seção 3.3. Também foram utilizados os módulos de persistência e controle de transação,

que em conjunto com o Hibernate agilizaram o desenvolvimento do componente *Core* do ABSecurity e para aplicação do padrão DAO.

O Hibernate é uma ferramenta de mapeamento objeto/relacional para Java. Ela transforma os dados de tabelas de um banco de dados em objetos definido pelo desenvolvedor. Esses objetos são os *entities* definido no projeto arquitetural do sistema. Com o Hibernate a velocidade de desenvolvimento melhora muito, pois livra o desenvolvedor de escrever muito código de acesso a banco de dados e de SQL que seria escrito usando outra ferramenta (King et al., 2011).

Na construção do *web service* foi utilizado serviços REST e o JSON para fazer o intercâmbio de dados entre as aplicações. A implementação do serviço foi simplificado pelo *Spring Framework*, pois facilita sua configuração, além de converter o objeto de retorno do serviço para o formato JSON. Foi utilizado serviços REST por ser mais simples de ser implementado, pois utiliza o protocolo HTTP de maneira semelhante a utilizada no desenvolvimento de aplicações *web*. Outra vantagem do uso de serviços REST é a performance, pois as mensagens trocadas são menores, por não precisarem ser envelopadas em um pacote SOAP (Ramalho, 2004).

A segurança do sistema ABSecurity foi feita de forma semelhante a utilizada nos protótipos descritos na seção 4.3. Suas funcionalidades foram cadastradas no próprio sistema para gerar as transações disponíveis para o programador implementar o controle de acesso. O padrão discutido na seção 3.4 foi implementado, com a diferença que o sistema acessa as informações de acesso do usuário de forma direta, sem usar o serviço de autenticação e autorização. Dessa maneira, foi necessário implementar apenas os métodos de verificação de transações, com o auxílio da sessão e de filtros de requisições.

No desenvolvimento das interfaces foram utilizadas tecnologias como JSP, HTML, JavaScript, JQuery, CSS, entre outras. Nesta fase foi implementada a camada de apresentação do padrão MVC.

O Sistema de Gerenciamento de Banco de Dados utilizado foi o Postgresql por ser um SGBD de código aberto e por possuir algumas características encontradas apenas em banco de dados comerciais. Ele permitiu a implementação do modelo de base de dados ilustrado na figura 4.2.

4.3 Protótipos

No presente trabalho foram desenvolvidos três protótipos em tecnologias e plataformas diferentes. Nos protótipos foram implementadas as interfaces e o padrão apresentado na seção 3.4, deixando de lado a persistência de dados, entre outras funcionalidades. O objetivo dos protótipos é testar as aplicações funcionando em conjunto, fazendo o controle do usuário em cada sistema, restringindo ou liberando o acesso a determinadas funcionalidades, de acordo com seu perfil de acesso. Todos os protótipos são sistema de gestão contendo módulos de recursos humanos, administrativo, financeiro e comercial.

Os protótipos foram cadastrados no ABSecurity, juntamente com suas funcionalidades, dessa forma foi possível criar as transações disponíveis para cada protótipo. O programador conhecendo essas transações, consegue implementar rotinas para verificação de permissão para cada funcionalidade da aplicação. O acesso ao serviço de Autenticação e Autorização é feito no *login* do usuário no sistema. Atualmente, a maioria da linguagens de programação tem bibliotecas que permitem fazer o acesso a serviços *web*.

No primeiro protótipo foi implementado um sistema de gestão *web*, desenvolvido em PHP, rodando no servidor de aplicação Apache e sistema operacional Windows 7. Por ser uma linguagem orientada a objetos, o padrão foi implementado utilizando uma classe, contendo os campos e métodos necessários para o controle de acesso. O método de acesso ao *web service* e a conversão do objeto JSON, foram implementados com o auxílio de bibliotecas escritas na própria linguagem, simplificando sua implementação. O método de verificação de transações foi feito com o auxílio da sessão, criada no momento da autenticação do usuário no sistema. Estas verificações são feitas nos menus e URLs do protótipo.

Foi implementando um sistema de gestão para dispositivos móveis, rodando no sistema operacional Android 2.1, utilizando a linguagem Java. Foi utilizado o *framework Spring* para Android que auxiliou na implementação dos métodos descritos no padrão de Autenticação e Autorização. Em cada item de menu é feito a verificação acesso do usuário ao recurso.

No último protótipo implementado, foi criada uma aplicação rodando em um console, em linguagem C, rodando no sistema operacional Linux Ubuntu. O objetivo desse

protótipo é mostrar o acesso de uma linguagem mais antiga ao serviço de Autenticação e Autorização. Após o *login* do usuário, a lista de transações disponíveis para o usuário é impressa no console. O padrão foi implementado utilizando uma estrutura da linguagem. O método de acesso ao *web service* foi o único método implementado utilizando uma biblioteca da linguagem.

Os resultados obtidos com o desenvolvimento dos protótipos foram satisfatórios, visto que todas conseguiram acessar o serviço e fazer o controle de acesso do usuário, viabilizando o uso desse tipo de solução em sistemas distribuídos e heterogêneos.

4.4 Funcionamento

Nesta seção são descritos os passos do funcionamento das aplicações. A figura 4.3, ilustra o funcionamento do ABSecurity e dos protótipos.

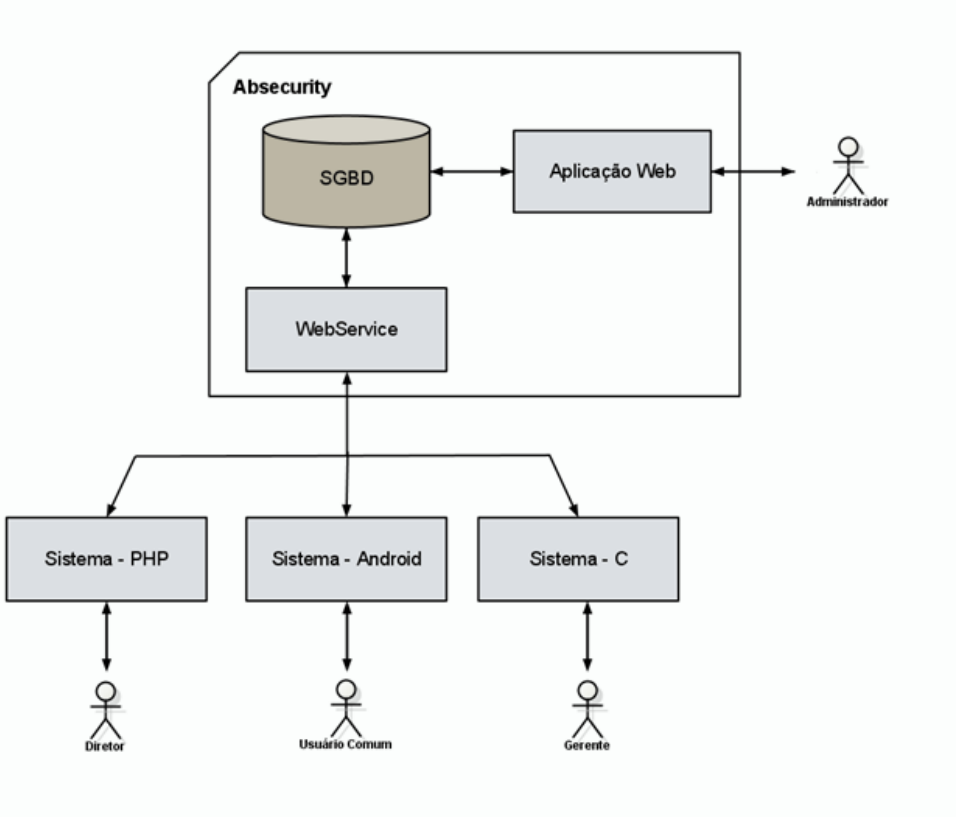

Figura 4.3: Funcionamento

1. O administrador das políticas de segurança é responsável por cadastrar os sistemas que utilizam o ABSecurity. Exemplo: Sistema de Gestão (PHP), Sistema Contatos

(Android), entre outros.

2. Para cada sistema é necessário cadastrar suas funcionalidades. Exemplo: Cadastro de Clientes, Cadastro de Produtos.

3. Transações são geradas a partir da funcionalidade e tipo de transação. Exemplo: Incluir cliente, alterar cliente, consultar cliente, entre outros.

4. O administrador atribui as transações a um dado papel. Exemplo: Gerente, Diretor, Administrador.

5. Esses papéis são atribuídos aos usuários de acordo com seu perfil ou cargo.

6. Suponha que a um determinado usuário, foi atribuído o papel de gerente de recursos humanos. Esse papel é responsável pelos cadastros do módulo de Recursos Humanos do Sistema de Gestão da Empresa. Ele terá acesso a todas as transações atribuídas a esse papel.

7. Ao tentar fazer o *login*, o sistema irá fazer uma requisição ao serviço.

8. O serviço consulta a base de dados para verificar se o usuário foi cadastrado e se ele possui transações para esse sistema.

9. O serviço retorna um objeto AutenticacaoAutorizacao, no formato JSON, convertido automaticamente no retorno do *web service*, utilizando uma biblioteca Java.

10. O sistema terá disponível o perfil de acesso do usuário no sistema, caso esse usuário possua transações de acesso as funcionalidades. Se o usuário não estiver cadastrado ou não possuir transações, ele não será autenticado.

4.5 Testes

Nesta fase foram feitos alguns testes de carga. Existem muitos motivos para relizar testes de carga em aplicações web. Geralmente são feitos teste de carga para determinar o comportamento da aplicação em condições normais e altos picos de carga. É recomendado

que o teste comece com um pequeno grupo de usuários virtuais e repita várias vezes aumentando o número de usuários até atingir o pico (Campos, 2011).

Os testes de carga foram realizados utilizando a ferramenta Apache JMeter. No JMeter foi criado um grupo de usuários que acessam o *web service* de autenticação e autorização por um determinado período de tempo. Durante os testes foram analisados os seguintes itens:

- Tempo médio de resposta;
- Tempo mínimo de resposta;
- Tempo máximo de resposta;
- Porcentagem de erro.

Foi criado um ambiente para simular a rede numa organização, para verificar o comportamento das aplicações, em um ambiente corporativo. Foi configurado um servidor de aplicação rodando em uma máquina virtual onde fica hospedada a aplicação ABSecutity. Os protótipos rodaram em plataformas e sistemas operacionais diferentes.

O servidor de aplicação possui as seguintes características:

- **Máquina virtual**: Vmware Player.
- **Sistema operacional**: Linux Ubuntu 11.04.
- **Processador**: Um núcleo de processamento de um Intel Core i5 2.4 GHz
- **Memória**: 1 GB de memória RAM
- **Servidor de aplicação**: Apache Tomcat 6

No primeiro teste executado o sistema está sendo pouco utilizado e no intervalo de um segundo um grupo de usuários tenta acessar o sistema. Com os dados obtidos, representados na tabela 4.1, verificou-se o comportamento da aplicação em condições normais recebendo um número razoável de usuários. O sistema se manteve estável até 500 usuários, a partir desse número o sistema começou a não conseguir tratar todas as requisões e o tempo médio de resposta aumentou muito.

Tabela 4.1: Primeiro teste de carga

Número de Usuários	Porcentagem de erros	Tempo médio de resposta (ms)	Tempo mínimo de resposta (ms)	Tempo máximo de resposta (ms)
100	0%	542	24	995
200	0%	1965	137	3383
300	0%	2850	75	5162
400	0%	4366	287	9137
500	0,28%	4737	81	9198
600	0,95%	7201	2750	12513
700	16,10%	7559	2915	19286

Na execução do segundo teste, representado pela tabela 4.2, um grupo grande de usuários acessa o sistema em um período de sessenta segundos. Nesse teste verificou-se que o sistema conseguiria atender até 4000 usuários por minuto sem apresentar erros e com um tempo médio de resposta satisfatório.

Tabela 4.2: Segundo teste de carga

Número de Usuários	Porcentagem de erros	Tempo médio de resposta (ms)	Tempo mínimo de resposta (ms)	Tempo máximo de resposta (ms)
3500	0%	66	13	428
4000	1,30%	315	13	3157
4500	11,40%	5994	38	16252
5000	19,87%	5819	39	28887

Os resultados do teste de carga foram satisfatórios, visto que o servidor de aplicação utilizado possui uma configuração inferior aos servidores utilizados pelas empresas.

5 Considerações Finais

O objetivo geral proposto neste trabalho de construir um protótipo de sistema utilizando o modelo de controle de acesso baseado em papéis com *web services* foi alcançado com êxito. A escolha deste modelo em sistema procura facilitar e agilizar o controle de acesso a recursos, bem como a gerência de políticas de segurança do mesmo. A utilização de *web services* promoveu a interoperabilidade entre os sistemas, reduzindo sua complexidade.

A adoção desse tipo de solução demonstrou-se extremamente viável e benéfica, pois é possível garantir, de forma centralizada, que as políticas de segurança entre as aplicações de uma empresa sejam seguidas.

Outro objetivo era a implementação de protótipos desenvolvidos em outra tecnologias e arquiteturas, para testar o comportamento das aplicações em um ambiente distribuído. Foi criado o padrão de Autenticação e Autorização responsável por integrar os protótipos, facilitando e padronizando o acesso ao sistema ABSecurity.

Os resultados dos testes comprovaram que é possível utilizar esse tipo de sistema em ambientes corporativos com um número grande de usuários, desde que a empresa possua uma infraestrutura de rede adequada.

Existem várias propostas de continuidade deste trabalho. A principal delas é a melhoria da segurança e da estrutura do ABSecurity, com a inclusão de novas funcionalidades, de acordo com as necessidades das empresas.

A parte de auditoria poderia ser melhorada, com o objetivo de verificar a eficiência dos controles e procedimentos de segurança existentes e a correta utilização dos recursos, facilitando a administração dos sistemas, mostrando deficiências e irregularidades que comprometam a segurança e o desempenho das aplicações.

Com isso, concluído o objetivo do trabalho, cria-se um número maior de opções, onde possíveis soluções de problemas na área de segurança de sistemas podem agora ser analisadas com uma visão maior, levando em conta sistemas distribuídos e heterogêneos.

Referências Bibliográficas

de Albuquerque Reis, J. A.; da Silva, D. Q. Ampliação do seguraweb, um framework rbac para aplicações web. **Universidade Federal de Santa Catarina**, 2004.

Berger, M. *Data Access Object Pattern*. 2005.

Bhatti, R.; Joshi, J. B. D.; Bertino, E. ; Ghafoor, A. *Access Control in Dynamic XML-based Web-Services with X-RBAC*. **International Conference on Web Services**, 2003.

Campos, F. M. **Testes de performance - testes de carga, stress e virtualização**, 2011.

Ferraiolo, D. F.; Sandhu, R.; Gavrila, S.; Kuhn, D. R. ; Chandramouli, R. *A proposed standard for role based access control*. **ACM Transactions on Information and System Security**, 2001.

Fonseca, R.; oes, A. S. Alternativas ao xml: Yaml e json. **Universidade do Minho**, 2007.

Furtado, C.; Pereira, V.; Azevedo, L.; Baião, F. ; Santoro, F. Arquitetura orientada a serviço - conceituação. **Universidade Federal do Estado do Rio de Janeiro**, 2009.

Guilheni, S. N. Controle de acesso baseado em papéis e certificados de atributos x.509. **Universidade de São Paulo**, 2005.

Gavin King, Christian Bauer, E. B. S. E. **Hibernate getting started guide**, 2011.

Maciel, R. S. P.; de Assis, S. R. *Middleware*: Uma solução para o desenvolvimento de aplicações distribuídas. **CienteFico**, 2004.

Associação Brasileira de Normas Técnicas, Rio de Janeiro. **NBR ISO/IEC 17799 - Tecnologia da informação - Código de prática para a gestão da segurança da informação**, 2001.

Obelheiro, R. R.; da Silva Fraga, J. ; Westphall, C. M. Controle de acesso baseado em papéis para o modelo corba de segurança. **19º Simpósio Brasileiro de Redes de Computadores**, 2001.

Ramalho, C. J. F. L. J. C. Web services: Metodologias de desenvolvimento. **Artigo**, 2004.

Santos, G. P. Aplicação do padrão de projeto mvc com jsf. **Faculdade de Jaguariúna**, 2008.

da Silva, A. T. S.; Saldanha, H. V. **Controle de Acesso Baseado em Papéis na Informatização de Processos Judiciais**. Universidade de Brasília, 2006.

Sohr, K.; Mustafa, T. ; Bao, X. *Enforcing Role-Based Access Control Policies in Web Services with UML and OCL*. ***Center for Computing Technologies (TZI), Universitat Bremen***, 2006.

Sommerville, I. **Engenharia de Software**. Addison Wesley, 2004.

Spanceski, F. R. Política de segurança da informação - desenvolvimento de um modelo voltado para instituições de ensino. **Instituto Superior Tupy**, 2004.