**UNIVERSIDADE REGIONAL DE BLUMENAU**

**CENTRO DE CIÊNCIAS EXATAS E NATURAIS**

**CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO**

# MIDDLEWARE PARA A COMUNICAÇÃO DE DADOS ENTRE SISTEMAS DISTRIBUÍDOS COM WS SECURITY

**CAIO RENAN HOBUS**

**BLUMENAU**
**2013**

**2013/2-02**

**CAIO RENAN HOBUS**

# MIDDLEWARE PARA A COMUNICAÇÃO DE DADOS ENTRE SISTEMAS DISTRIBUÍDOS COM WS SECURITY

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciência da Computação — Bacharelado.

Prof. Jhony Alceu Pereira - Orientador

**BLUMENAU**
**2013**

**2013/2-02**

# MIDDLEWARE PARA A COMUNICAÇÃO DE DADOS

# ENTRE SISTEMAS DISTRIBUÍDOS COM WS SECURITY

Por

**CAIO RENAN HOBUS**

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

| | |
|---|---|
| Presidente: | Prof. Jhony Alceu Pereira - Orientador, FURB |
| Membro: | Prof. Mauro Marcelo Mattos, Doutor - FURB |
| Membro: | Prof. Paulo Fernando Da Silva, Mestre - FURB |

Blumenau, 05 de dezembro de 2013

Dedico este trabalho a minha família e todos os meus amigos, especialmente aqueles que me ajudaram diretamente na realização deste.

## AGRADECIMENTOS

A Deus, pelo seu imenso amor e graça.

À minha família, que me apoio e sempre esteve presente.

Aos meus amigos, pelos empurrões e cobranças.

Ao meu orientador, por ter acreditado na realização e conclusão deste trabalho.

A possibilidade de realizarmos um sonho é o que torna a vida interessante.

Paulo Coelho

# RESUMO

Este trabalho apresenta o desenvolvimento de um *middleware* com a finalidade de possibilitar as organizações trocarem informações importantes de forma segura através da rede, disponibilizando um serviço que possibilita a integração entre sistemas, no qual abstrai os detalhes da implementação das camadas de comunicação e os detalhes do *WS-Security* no qual é utilizado para realização da segurança dos dados na comunicação. Para isso foi utilizado a especificação JAX-WS como modelo de *web services* para desenvolver as funções na plataforma Java, utilizando a biblioteca Apache CXF para implementação de autenticação, criptografia e assinatura dos dados através de interceptadores do *framework spring*.

Palavras-chave: Comunicação de dados. Segurança. Sistemas distribuídos. Web services. WS-security.

# ABSTRACT

This paper presents the development of a middleware in order to enable organizations to exchange critical information securely across the network, providing a service that enables integration between systems, which abstracts the implementation details of communication layers. WS-Security is used to perform the security of data communication. For this we used the JAX-WS web services as a model to develop the functions on the Java platform, using Apache CXF library for implementing authentication, encryption and data signing using interceptors spring framework.



Keywords: Data communications. Security. Distributed systems. Web services. WS-security.

# LISTA DE ILUSTRAÇÕES

# LISTA DE SIGLAS

API - *Application Programming Interface*

DDL - *Data Definition Language*

DML - *Database Manipulation Language*

EA - *Enterprise Architect*

HTTP - *HyperText Transfer Protocol*

Jar - *Java Archive*

JAX-WS - *Java API for XML-Based Web Services*

JAXB - *Java Architecture for XML Binding*

JPA - *Java Persistence API*

JSF - *Java Server Faces*

MER - Modelo Entidade Relacionamento

MVC - *Model View Controller*

PKI - *Public-Key Infrastructure*

RPC - *Remote Procedure Calls*

SAML - *Security Assertion Markup Language*

SGBD - Sistema de gerenciamento de banco de dados

SLA - *Service level Agreement*

SOAP - *Simple Object Access Protocol*

SSL - *Secure Socket Layer*

TCP/IP - *Transmission Control Protocol/Internet Protocol*

TSL - *Transport Layer Security*

UML - *Unified Modeling Language*

URI - *Uniform Resource Identifier*

W3C - *World Wide Web Consortium*

WAE - *Web Application Extension*

WSDL - *Web Services Description Language*

WTLS - *Wireless Transport Layer Security*

XKMS - *Key Management Specification*

XML - *eXtensible Markup Language*

# SUMÁRIO

# 1 INTRODUÇÃO

A tecnologia das comunicações evolui de uma maneira muito rápida. O crescimento da Internet, a sua abrangência, flexibilidade e a redução significativa dos custos de acesso criam um ambiente favorável para o surgimento de novas formas de negociação. O modo tradicional, envolvendo presença física, documentos reais, averbação em cartório e burocracia em geral, está migrando para o universo digital, abrindo-se um novo paradigma (SILVA, 2004, p. 21).

Todos os dias pessoas e empresas executam milhares de transações eletrônicas, arquivos e informações confidenciais são compartilhados entre as organizações pela rede. À medida que a Internet torna-se cada vez mais presente na vida diária das organizações, a necessidade da segurança eletrônica torna-se ainda mais crítica. Qualquer empresa engajada em atividades *on-line* deve avaliar e gerenciar os riscos da segurança eletrônica associados a esta atividade (BURNETT; PAINE, 2002, p. 8).

A segurança dos dados das grandes corporações se faz cada vez mais necessário, principalmente quando se trata de informações que trafegam pela rede, como informações estratégicas, previsões de vendas, detalhes sobre os produtos, resultado de pesquisas, arquivos importantes e assim por diante. Contudo a maioria das empresas simplesmente querem ocultar as informações valiosas de pessoas desonestas. Para se ter um serviço de forma segura é preciso atender os principais aspectos da segurança, como confidencialidade, integridade, disponibilidade e controle de acesso quando se trata de troca de informações entre organizações (BURNETT; PAINE, 2002, p. 3).

A segurança dos dados é um fator fundamental para os serviços da web que trocam informações e dados de negócio. Pode haver consequências financeiras ou legais negativas se os dados forem interceptados por terceiros ou se dados fraudulentos forem aceitos como válidos. Sempre é possível projetar e implementar os procedimentos de segurança próprios de um aplicativo para serviços da web para qualquer tipo de troca de dados. Contudo, essa é uma abordagem arriscada, pois mesmo pequenas distrações podem resultar em vulnerabilidades graves. Um dos principais benefícios do *Simple Object Access Protocol* (SOAP), se comparada a outras formas de troca de dados, é que a mesma permite extensões modulares. Desde o lançamento da SOAP, as extensões têm focado muito na segurança, o que resultou na padronização da *Web Services Security (WS Security)* e das tecnologias relacionadas que permitem que a segurança seja configurada de forma apropriada para cada serviço.

Diante do exposto, este trabalho descreve um *middleware* que encapsula o *WS-Security* para a comunicação de dados entre organizações através de *web services* de forma segura e transparente para a aplicação, garantindo a confidencialidade, integridade e autenticidade dos dados.

Com o desenvolvimento do *middleware*, é possível ao usuário configurar regras de acesso que servirão de base para a comunicação de dados. Este será o responsável por implementar as questões de segurança e enviar os dados através de *web services* utilizando o *WS-Security* para a outra aplicação. Esta receberá os dados através do *middleware* que os entregará novamente a aplicação com base nas regras cadastradas, garantindo a segurança dos dados e a transparência na questão da implementação de todas as funcionalidades para suportar o *WS-Security* e demais camadas de rede.

## 1.1 OBJETIVOS DO TRABALHO

Este trabalho tem como objetivo disponibilizar um *middleware* para as aplicações, que abstrai o *WS-Security* na comunicação de dados entre as organizações utilizando *web services* de forma segura e transparente na questão da implementação de todas as funcionalidades para suportar o *WS-Security*, garantindo a confidencialidade, integridade e autenticidade dos dados.

Os objetivos específicos do trabalho podem ser detalhados da seguinte maneira:

a) disponibilizar uma funcionalidade que permite abstrair o *WS-Security*, *web service* e demais camadas de rede;
b) disponibilizar uma interface web onde será feito a manutenção das organizações e das regras de comunicação;
c) disponibilizar uma interface onde a aplicação solicitará dados com base em regras definidas para o *middleware*;
d) disponibilizar uma interface onde o *middleware* receberá os dados e retornará a aplicação destinatária.

## 1.2 ESTRUTURA DO TRABALHO

Este trabalho é composto de quatro capítulos. O segundo capítulo refere-se à fundamentação teórica necessária para a compreensão dos temas abordados na implementação.

O terceiro capítulo contempla a descrição do desenvolvimento, onde são descritos os diagramas de classes, casos de uso, atividades e sequência. Ainda, são apresentados os resultados e discussões obtidos com a implementação do projeto.

Ao final, o quarto capítulo apresenta as considerações finais e conclusões, finalizando com sugestões de extensões para projetos futuros.

# 2 FUNDAMENTAÇÃO TEÓRICA

Na seção 2.1 são apresentados os conceitos de sistemas distribuídos, *middleware* e segurança dos dados. Na seção 2.2 é apresentado o funcionamento do *WS-Security* utilizando *web services* com *Java API for XML-Based Web Services* (JAX-WS) e os benefícios da implementação de um *middleware* que abstrai o *WS-Security* e *web service* na questão da transparência da implementação em relação à camada da aplicação. Por fim, na seção 2.3 são descritos os trabalhos correlatos.

## 2.1 SISTEMAS DISTRIBUÍDOS E SEGURANÇA DOS DADOS

Com a evolução tecnológica entra-se na era do conhecimento, onde a informação é considerada um dos principais patrimônios e recurso estratégico das organizações (CUNHA et al., 2006, p. 1). Assim, pode-se afirmar que a segurança deve estar presente em todas as aplicações, independentemente do seu tipo, que pode ser sistema web, *desktop* ou móvel, pois torna-se imprescindível para o processo de gerência das informações.

Com a conscientização das empresas de que suas informações tem um valor fundamental, aumenta cada vez mais o número de tecnologias disponíveis para a implantação dos modelos de segurança dentro dos sistemas. Porém, muitas das técnicas utilizadas são fundamentadas e voltadas para as aplicações comerciais atuais, não visando o novo modelo de sistemas baseados na web, tornando estas aplicações mais vulneráveis no que diz respeito à segurança (RODRIGUES et al., 2007). Desta forma, torna-se cada vez mais necessário o conhecimento do funcionamento das tecnologias web para que se possa desenvolver novas técnicas de troca segura de informações no ambiente da internet.

Um dos principais modelos utilizados na troca de informações são os sistemas distribuídos. Atualmente muitos sistemas são implementados de forma distribuída, ou seja, operam utilizando a troca de mensagens via web. Um dos conceitos de sistemas distribuídos é que usuários separados geograficamente podem compartilhar, de forma dinâmica, recursos computacionais (DUARTE FILHO et al., 2008). O objetivo é criar uma ilusão de um computador virtual, de fácil acesso e com um grande poder computacional e dispositivos sendo compartilhados.

Os sistemas distribuídos removem as conexões fixas entre aplicações, servidores, bases de dados, máquinas, armazenamento, entre outros, tratando tudo como um serviço virtualizado. Assim, os recursos computacionais podem estar em um mesmo ambiente ou alocados em diferentes locais geograficamente distantes, operando apenas com a troca de informações via web (DUARTE FILHO et al., 2008).

Quando as organizações trabalham com a troca de informações via web utilizando serviços como *web services*, que na maioria das vezes são informações sigilosas e valiosas, deve-se analisar e ter estratégia de gerenciamento dos riscos que podem ser causados em relação ao vazamento de informações importantes.

Segundo Stallings (2005, p. 380), a aplicação desenvolvida deve levar em consideração alguns requisitos, como:

a) autenticidade: garantia de que o serviço é capaz de identificar o usuário;
b) privacidade: garante que o acesso à informação somente é obtido a quem é autorizado;
c) integridade: garantia de que a informação e o meio como é processada não estejam corrompidos;
d) disponibilidade: garantia de que as pessoas autorizadas a acessar determinada informação tenham acesso à ela sempre que necessário.

Kurose e Ross (2005, p. 514) dizem que a segurança nas aplicações envolve, além da proteção, a detecção de falhas e ataques nas comunicações e a reação a estas adversidades, fazendo desta forma com que os itens de autenticidade, privacidade, integridade e disponibilidade sejam considerados como componentes fundamentais na comunicação segura.

### 2.1.1 Middleware

Segundo Ikematu et al., (2003), "*middleware* é uma categoria de produtos ou módulos de software que são utilizados por aplicações cliente, para acessar aplicações servidoras, e tentam esconder da aplicação a rede, a comunicação e plataformas específicas", ou seja, os *middlewares* trabalham disponibilizando determinadas funções já desenvolvidas a uma aplicação cliente, abstraindo sua forma de funcionamento, plataformas e tecnologias utilizadas, fazendo com que estas funções não tenham que ser desenvolvidas novamente.

## 2.2 WS-SECURITY

*WS-Security* é um conjunto de especificações de segurança para serviços web, proposto em conjunto pela Microsoft Corporation, IBM Corporation e VeriSign, com o objetivo de que as empresas pudessem criar e construir aplicações de serviços web, utilizando *web services* com uma ampla interoperabilidade. O *WS-Security* é uma extensão do sistema de comunicação entre serviços baseados em contratos *Web Services Description Language* (WSDL). Esta extensão contém vários mecanismos que permitem a realização de comunicações seguras mesmo quando o protocolo de transporte não é seguro, garantindo

assim a confidencialidade para as mensagens sob qualquer sistema de transporte utilizado. As novas especificações de segurança definem um conjunto de padrões para extensões SOAP ou para cabeçalhos de mensagens, utilizados para oferecer maior integridade, confidencialidade e privacidade às aplicações com *web services* (ATKINSON et al., 2002).

O *WS-Security* possui um sistema rico para prover segurança, tanto em termos de confidencialidade, quanto em termos de autenticação/autorização. O sistema de autenticação funciona com diversos mecanismos, sendo os mais conhecidos: autenticação via *token Security Assertion Markup Language* (SAML), via *ticket Kerberos*, via fornecimento de usuário e senha (tando com senha em texto puro quanto com *hash*) e via cerificado *X.509*. O *WS-Security* possui também várias especificações/extensões que podem ser utilizadas de acordo com a necessidade de cada aplicação. Elas incluem:

a) *WS-Policy*: definição de recursos e restrições;
b) *WS-Trust*: definição de um modelo de confiança;
c) *WS-Privacy*: define de que forma os *web services* serão implementados;
d) *WS-Secure Conversation*: define como autenticar e gerenciar troca de mensagens;
e) *WS-Federation*: define o gerenciamento de relacionamentos em ambientes heterogêneos;
f) *WS-Authorization*: define a forma de administração dos dados pelos *web services*.

O *WS-Security* funciona em conjunto com as especificações *WS-Policy* e *WS-SecurityPolicy*. Estas duas especificações têm por objetivo, respectivamente, estabelecer políticas gerais a respeito de sugurança, *Service level Agreement* (SLA), qualidade de serviço, confiabilidade, etc.; e estabelecer, especificamente, quais são as políticas de segurança aplicáveis a um determinado serviço (SAUDATE, 2010).

A autenticação está relacionada à identificação do chamador. O *WS-Security* usa *tokens* de segurança para manter essas informações com um cabeçalho de segurança da mensagem SOAP. A integridade da mensagem é obtida com assinaturas digitais *eXtensible Markup Language* (XML). Isso garante que partes da mensagem não tenham sido adulteradas após a assinatura do originador. A confidencialidade da mensagem é baseada na especificação de criptografia XML e garante que partes correspondentes da mensagem só possam ser compreendidas pelo(s) destinatário(s) desejado(s) (ROSENBERG; REMY, 2004).

O *WS-Security* suporta, integra e unifica vários modelos, mecanismos e tecnologias de segurança em uso no mercado, permitindo que vários sistemas possam interoperar em plataformas e linguagens neutras.

Com os avanços tecnológicos, empresas operando de forma distribuída, informações importantes trafegando pela rede, sendo transportadas de aplicações para aplicações através da internet, foi desenvolvido um importante mecanismo de troca de mensagem SOAP chamado *web services* (WORLD WIDE WEB CONSORTIUM, 2007). SOAP é um protocolo projetado para invocar aplicações remotas através de *Remote Procedure Calls* (RPC) ou trocas de mensagens, em um ambiente independente de plataforma e linguagem de programação. SOAP se define usando XML, que proporciona com simplicidade e coerência uma maneira de uma aplicação enviar mensagem XML para outra. O SOAP é o que faz a integração entre aplicações ser possível, pois após a definição do conteúdo do XML, é o SOAP que transfere os dados de um lugar para outro pela rede. Permite enviar e receber documentos XML que suportam um protocolo comum de transferência de dados. Além disso, SOAP permite tratar mensagens XML retornadas de um serviço remoto e seu modelo possibilita de forma clara a separação entre os dados de processamento de infraestrutura e processamento de mensagens de aplicação (ROSENBERG; REMY, 2004).

O SOAP fornece um formato (Envelope SOAP) para a mensagem XML. Este formato é um *container* que protege os dados XML. O objetivo é criar um container uniforme para que mensagens SOAP possam ser transmitidas por qualquer protocolo de transporte. O protocolo evita que a aplicação conheça o protocolo de transporte e se mantém coerente com o envelope SOAP. O envelope SOAP é composto por duas partes: o cabeçalho que contém informações sobre a mensagem SOAP que são usadas para gerenciar ou prover segurança para o pacote. A outra parte é o corpo da mensagem que contém a mensagem SOAP propriamente dita. SOAP é, portanto, um padrão normalmente aceito para utilizar-se com *web services*. Desta forma pretende-se garantir a interoperabilidade e intercomunicação entre diferentes sistemas, através da utilização da linguagem XML e mecanismo de transporte *HyperText Transfer Protocol* (HTTP) padrões (ROSENBERG; REMY, 2004).

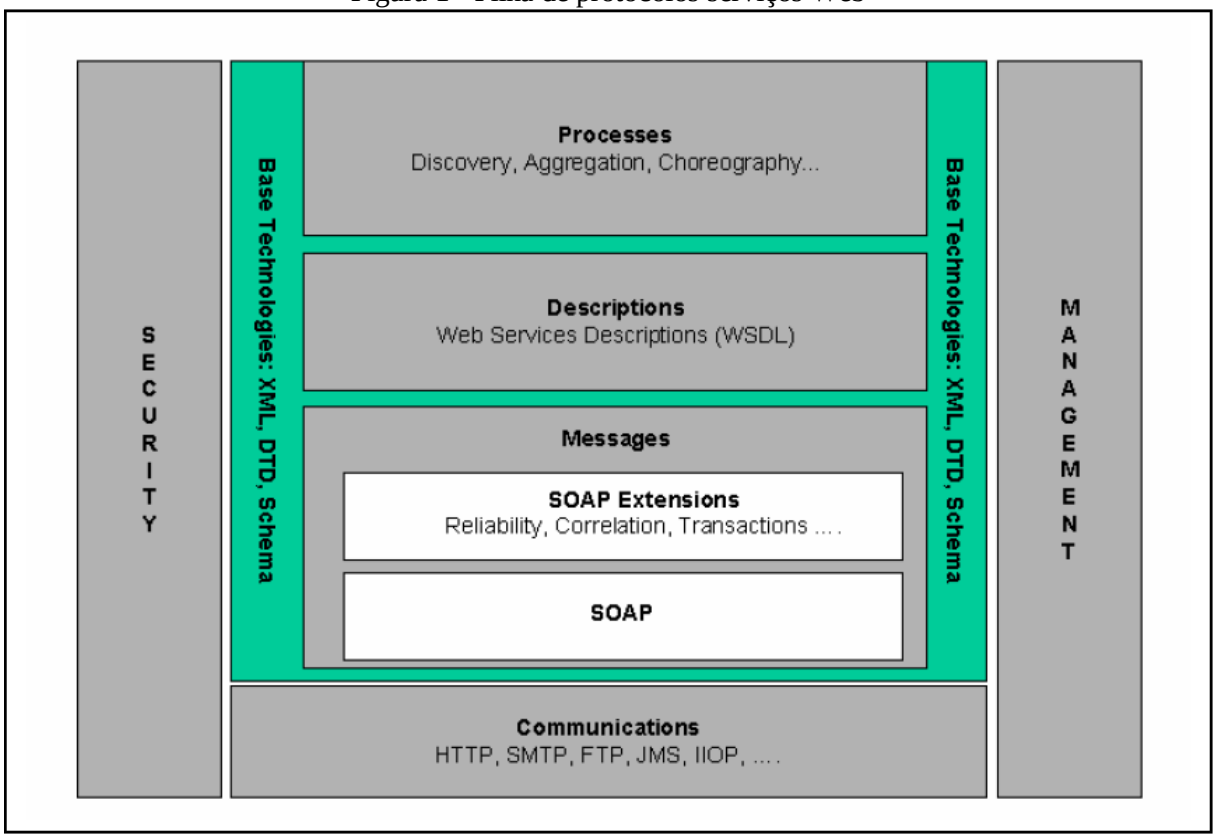
Os *web services* permitem a comunicação entre aplicações com linguagens heterogêneas, fornecendo funções e serviços. Sendo uma interface que descreve uma coleção de operações que são acessíveis pela rede através de mensagens padronizadas no formato XML (HANSEN; PINTO, 2003, p. 2).

Hansen e Pinto (2003, p. 2) afirmam ainda que um *web service* é “um componente de software independente de implementação e plataforma. É descrito utilizando uma linguagem de descrição de serviços, publicado em um registro e descoberto através de um mecanismo padrão. Pode também ser invocado a partir de uma API através da rede e ser composto juntamente com outros serviços.”.

Na construção de serviços web, os mesmos necessitam ser acessados em algum lugar na web por uma aplicação cliente. Uma forma de acessar um serviço web é fazer com que a aplicação cliente conheça a *Uniform Resource Identifier* (URI) do serviço, desta maneira é caracterizado o modo estático de localizar e acessar um serviço (CHAPPELL; JEWEL, 2002).

A figura abaixo ilustra a pilha de protocolos utilizados na construção de um serviço *web* (WORLD WIDE WEB CONSORTIUM, 2007).

Figura 1 - Pilha de protocolos serviços Web




Fonte: WORLD WIDE WEB CONSORTIUM (2007).

Na plataforma Java, há especificações que definem a implementação Java dos padrões estabelecidos pelo W3C. A especificação Java diretamente relacionada a *web services* que seguem os padrões da W3C é a API JAX-WS. A JAX-WS depende fortemente de outra especificação Java, a *Java Architecture for XML Binding* (JAXB). A ideia principal da JAXB é definir o mapeamento e transformação dos dados de uma aplicação Java para XML e vice versa. Com os recursos JAXB pode-se transformar uma árvore de objetos Java em texto XML ou vice versa. JAX-WS utiliza anotações, o que simplifica o desenvolvimento de serviços web e diminui o tempo de execução dos arquivos *Java Archive* (Jar) (WORLD WIDE WEB CONSORTIUM, 2007).

### 2.2.1 XML digital signature

O padrão *XML Digital Signature* especifica uma forma de criar assinaturas digitais para o uso em transações de XML. Este padrão define um esquema para pegar o resultado de uma operação sobre uma assinatura aplicada a dados no formato XML. Da mesma forma que as assinaturas digitais comuns, as assinaturas de documentos XML adicionam importantes características como a autenticação, a integridade dos dados e a sustentação do não-repúdio de dados previamente assinados (WATHIER et al., 2005).

Uma característica fundamental da assinatura de XML é a habilidade de assinar somente partes específicas da árvore de XML ao invés de assinar todo o documento original. Isto será relevante quando um único original de XML pode ter um longo histórico no qual os diferentes componentes são criados em momentos diferentes por pessoas diferentes, cada um assinando somente aqueles elementos relevantes para si. Esta flexibilidade também será crítica em situações nas quais é importante garantir a integridade de certas partes de um documento XML, enquanto deixa para outros a possibilidade de alterar outras partes do documento. Exemplo: um formulário XML assinado entregue a um usuário para ser preenchido. Se a assinatura for aplicada ao formulário XML completo, quaisquer alterações feitas pelo usuário sobre os valores padrão do formulário invalidarão a assinatura original do documento (WATHIER et al., 2005).

Uma assinatura XML pode assinar mais de um tipo de recurso. Por exemplo, uma única assinatura XML pode ser utilizada sobre dados de texto (um documento HTML), dados binários (uma imagem), dados no formato XML, assim como partes específicas de um documento XML (ROSENBERG; REMY, 2004).

A estrutura básica de uma assinatura XML, segundo (ROSENBERG; REMY, 2004) é listada na figura abaixo:

Figura 2 - Sintaxe do elemento <Signature> assinatura XML

```
<Signature>
  <SignedInfo>
    (CanonicalizationMethod)
    (SignatureMethod)
    (<Reference(URI=)?>
      (Transforms)?
      (DigestMethod)
      (DigestValue)
    </Reference>)+
  </SignedInfo>
  (SignatureValue)
  (KeyInfo)?
  (Object)*
</Signature>
```

Fonte: ROSENBERG; REMY, 2004

A informação assinada aparece dentro do elemento <SignedInfo>. O algoritmo usado no cálculo do elemento <SignatureValue> é mencionado dentro da seção assinada. O elemento <SignatureMethod> especifica o algoritmo usado para converter o <SignedInfo> canonizado (Algoritmo de verificação de inconsistência em conteúdos XML antes de extrair a representação em bits para posterior processamento de uma assinatura) no <SignatureValue>. Esta é uma combinação de um algoritmo que depende da chave e de um algoritmo de resumo. O elemento <KeyInfo> indica a chave que é usada para validar a assinatura, possíveis formas de identificação dos certificados são: nomes de chaves, algoritmos de aceitação de chaves e informação.

Cada recurso deve assinar seu próprio elemento <Reference>, identificado pelo atributo URI. O elemento <Transform> especifica uma lista ordenada de processos que são aplicados ao conteúdo do recurso especificado antes de ser aplicada a função *hash*. O <DigestValue> é o elemento que recebe o resultado da função *hash* aplicada ao recurso (WATHIER et al., 2005).

### 2.2.2 XML encryption

O padrão *XML Encryption* provê segurança no nível de privacidade por cifrar os dados evitando que terceiros vejam seu conteúdo. A segurança é realizada ponto a ponto para aplicações que requerem trocas seguras de dados estruturados. Criptografia baseada em XML é o caminho natural para segurança com troca de dados entre aplicações complexas.

O *Transport Layer Security* (TSL) é de fato o padrão de comunicação segura na internet, sendo um protocolo de segurança ponto a ponto que segue o *Secure Socket Layer* (SSL). O *XML Encryption* não pretende substituir o SSL/TSL, mas prover um mecanismo para requisitos de segurança que não é coberto pelo SSL como:

a) criptografar parte dos dados;

b) sessão segura entre mais de duas partes.

Com o *XML Encryption*, cada parte pode manter o estado de segurança ou não com qualquer das partes comunicantes. Ambos dados seguros ou não podem ser trocados no mesmo documento. O *XML Encryption* pode manusear dados no formato XML ou binário (WATHIER et al., 2005).

A tecnologia de criptografia poderá ser usada para criptografar um documento inteiro ou partes deste. Levando em consideração o consumo de tempo para o processo de criptografia e considerando a perspectiva de performance, é aconselhável que os dados não sejam cifrados a menos que comprometa a segurança (GALBRAITH, 2002, p. 14).

Expressado numa forma curta, o elemento `<EncryptedData>` tem a seguinte estrutura (onde "?" significa zero ou uma ocorrência; "+" significa uma ou mais ocorrências; "*" significa zero ou muitas ocorrências; e um elemento tag vazio (`<elemento/>`) significa que o elemento será vazio):

Figura 3 - Sintaxe do elemento `<EncryptedData>` criptografia XML

```
<EncryptedData Id? Type? MimeType? Encoding?>
    <EncryptionMethod/>?
    <ds:KeyInfo>
      <EncryptedKey>?
      <AgreementMethod>?
      <ds:KeyName>?
      <ds:RetrievalMethod>?
      <ds:*>?
    </ds:KeyInfo>?
    <CipherData>
      <CipherValue>?
      <CipherReference URI?>?
    </CipherData>
    <EncryptionProperties>?
  </EncryptedData>
```

Fonte: ROSENBERG; REMY, 2004

Os elementos mais representativos de `<EncryptedData>` são: `<EncryptionMethod>` e `<CipherData>`. `<EncryptionMethod>` aponta para algoritmo utilizado na criptografia.

<CipherData> contém os dados criptografados ou um ponteiro para as informações criptografadas (ROSENBERG; REMY, 2004).

## 2.3 TRABALHOS CORRELATOS

Existem disponíveis algumas ferramentas comerciais e acadêmicas que abordam a comunicação e segurança dos dados entre aplicações utilizando *web services* e *WS-Security*. No âmbito geral pode-se citar o trabalho de Hansen e Pinto (2003) que mostra um ambiente de educação baseada na web através de *web services* educacionais. Martins, Rocha e Henriques (2003), demonstram a segurança dos *web services* no comércio eletrônico móvel e Silva (2004), mostra a segurança em *web services*.

No trabalho desenvolvido por Hansen e Pinto (2003), os autores demonstram o funcionamento de *web services* e como eles podem ser utilizados em uma aplicação para ensino à distância, integrando os sistemas já existentes, fazendo com que consigam trocar informações e interoperar-se, facilitando a reutilização dos diversos serviços disponíveis pelos sistemas educacionais.

Foi desenvolvida uma arquitetura de integração, que utiliza a linguagem de composição chamada *GlueScript*, para permitir que recursos disponibilizados por sistemas educacionais possam ser reutilizados juntamente com serviços acessados através da arquitetura de *web services*. Foi utilizado o protocolo SOAP para o acesso aos serviços da web, pois este é encapsulado dentro do protocolo HTTP. Através do uso das *tags* da *GlueScript* foi possível localizar e acessar *web services* e recursos educacionais de forma dinâmica. O assistente da *GlueScript* permite a geração de código básico para acessar *web services* e recursos educacionais de forma que um desenvolvedor precisa apenas fazer as interfaces para uso dos mesmos.

Martins, Rocha e Henriques (2003) demonstram em seu estudo a necessidade de se aplicar segurança aos serviços disponíveis para transações na internet, voltado ao comércio eletrônico móvel *M-Commerce*, utilizando *web services* na comunicação dos dados, protocolos específicos como o *Wireless Transport Layer Security* (WTLS) e o *Secure Socket Layer* (SSL). Estes fornecem autenticação, integridade dos dados e privacidade de serviços dentro das limitações dos hardwares utilizadas na tecnologia *wireless*, como processamento, memória e largura de banda. Na criptografia dos dados foi verificado a necessidade de utilizar o *XML Encryption* e para assinatura digital o *XML Signature*. Verificou-se também a necessidade de se utilizar o *XML Key Management Specification* (XKMS), sendo que este fornece um protocolo baseado em XML/SOAP para distribuição e registro de chaves públicas.

Neste estudo os autores apresentam uma comparação entre os métodos abordados, onde demonstram a importância da segurança e a necessidade de um conjunto eficiente de especificações, como o *WS-Security*.

Silva (2004) apresenta várias formas de garantir a segurança em *web services*, apresentando um estudo sobre *web services* utilizando o protocolo SOAP em sua comunicação. Para tornar um *web service* seguro foi necessário encriptar a comunicação de duas maneiras, em nível de transporte e em nível de XML.

Para garantir a segurança em nível de transporte foi utilizando o protocolo SSL para encriptar a comunicação sobre o protocolo *Transmission Control Protocol/Internet Protocol* (TCP/IP). Com este modelo um cliente abre um *socket* seguro para um *web service* e então o utiliza para trocar mensagens SOAP via HTTPS, utilizando a *Public-Key Infrastructure* (PKI), que garante a segurança de comércio eletrônico e de comunicações na Internet através da autenticação, encriptação, não repúdio e o uso de certificado digitais. Para a segurança em nível XML que envolve a encriptação e decriptação dos documentos XML foi utilizado as especificações da W3C que envolve além da encriptação, também a assinatura digital e o gerenciamento de chaves. Para este processo foram utilizados as especificações *XKMS*, *WS-Security*, *XML Encryption* e *XML Signature*.

# 3 DESENVOLVIMENTO

Neste capítulo são abordadas as etapas envolvidas no desenvolvimento do *middleware* proposto. Na seção 3.1 são apresentados os requisitos principais do problema a ser trabalhado. A seção 3.2 descreve a especificação do *middleware*. Na seção 3.3 é apresentada a visão geral do sistema web. A seção 3.4 descreve a implementação e, por fim, na seção 3.5 são apresentados os resultados, discuções e sugestões para trabalhos futuros.

## 3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

Com o objetivo de aplicar a especificação do *WS-Security* no estudo de caso do *middleware*, o sistema atenderá os requisitos relacionados no Quadro 1 e Quadro 2. Além disto, cada RF está vinculado a um caso de uso, facilitando a identificação do propósito e justificando a sua existência.

Quadro 1 - Requisitos funcionais

| **REQUISITOS FUNCIONAIS** | **CASOS DE USO** |
|---|---|
| RF01: O sistema web deverá manter o registro de sistemas. | UC01 |
| RF02: O sistema web deverá manter o registro de regras. | UC02 |
| RF03: O sistema web deverá manter o registro de permissões. | UC03 |
| RF04: O sistema web deverá manter o registro de usuários. | UC04 |
| RF05: O sistema web deverá manter o registro de perfis. | UC05 |
| RF06: O sistema web deverá manter o registro de recursos. | UC06 |
| RF07: O *middleware* deverá integrar-se as aplicações de terceiros. | UC07 |
| RF08: O *middleware* deverá enviar requisições para outras aplicações. | UC08 |
| RF09: O *middleware* deverá receber requisições de outras aplicações. | UC09 |
| RF10: O *middleware* deverá enviar arquivos para outras aplicações. | UC10 |
| RF11: O *middleware* deverá receber arquivos de outras aplicações. | UC11 |
| RF12: O *web service* deverá disponibilizar o serviço ao *middleware*. | UC12 |
| RF13: O *web service* deverá autenticar usuários. | UC13 |
| RF14: O *web service* deverá identificar os sistemas na comunicação. | UC14 |
| RF15: O *web service* deverá verificar as regras de comunicação. | UC15 |
| RF16: O *web service* deverá verificar as permissões de cada usuário na comunicação. | UC16 |

Quadro 2 - Requisitos não funcionais

| **REQUISITOS NÃO FUNCIONAIS** |
|---|
| RNF01: O sistema deve ser implementado utilizando *web services* com a API JAX-WS. |
| RNF02: O sistema deve ser implementado utilizando a tecnologia Java. |
| RNF03: O sistema deve utilizar banco de dados Oracle 10G. |
| RNF04: O sistema deve ser implementado utilizando o ambiente de desenvolvimento NetBeans. |
| RNF05: O sistema deve ser implementado utilizando os conceitos definidos pelo conjunto de especificações *WS-Security*. |
| RNF06: O *middleware* deve garantir a confidencialidade, integridade e disponibilidade dos dados que serão utilizados na comunicação. |

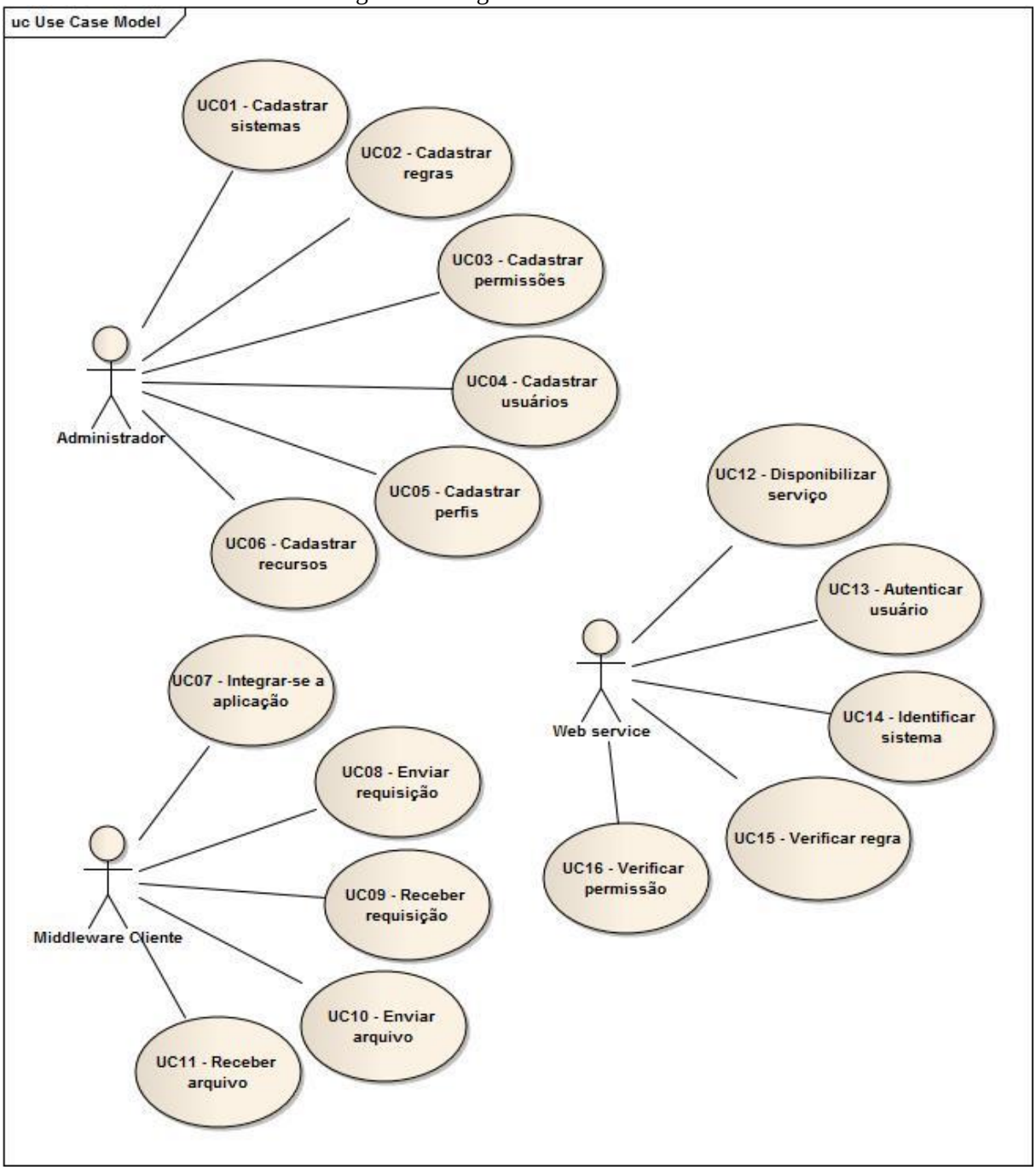
## 3.2 ESPECIFICAÇÃO

A especificação do *middleware* foi desenvolvida seguindo a análise orientada a objetos, utilizando a *Unified Modeling Language* (UML), com a ferramenta *Enterprise Architect* (EA) 7.5 onde foram especificados os diagramas de casos de uso, classes, atividades e sequência. Também foi especificado o Modelo Entidade Relacionamento (MER) utilizado na modelagem dos dados, com a ferramenta DB Designer Fork 2009.

### 3.2.1 Diagrama de casos de uso

Nesta seção são descritos os casos de uso com as funcionalidades do *middleware* que implementa o *WS-Security*. Foram identificados três atores: o `administrador`, que é responsável por cadastrar as regras de comunicação na interface web. O `middleware cliente`, sendo o responsável por implementar as funcionalidades do *WS-Secutiry* para enviar e receber dados e arquivos através de *web services*. O `web service`, sendo o responsável por disponibilizar o serviço e pelo controle de fluxo de dados do *middleware* verificando as regras de comunicação. Na figura 4 é apresentado o diagrama de casos de uso, sendo que, o detalhamento dos dois principais casos de uso encontra-se no Apêndice A.

Figura 4 - Diagrama de casos de uso




### 3.2.2 Diagrama de classes

Os diagramas de classes apresentados na Figura 5 exibem as principais classes do *middleware* e do *web service*, nas quais implementam as especificações do *WS-Security*.

Este diagrama mostra as principais classes, sendo elas:

a) `MiddlewareWSS`: é a estrutura principal do *middleware*. Classe na qual a aplicação irá se integrar para realização da comunicação de dados;

b) `Config`: armazena as credenciais do usuário;

c) `ClientPasswordCallback`: responsável pela validação das operações de *callback* do *WS-Security client*, sendo utilizado pelo arquivo XML de configuração;

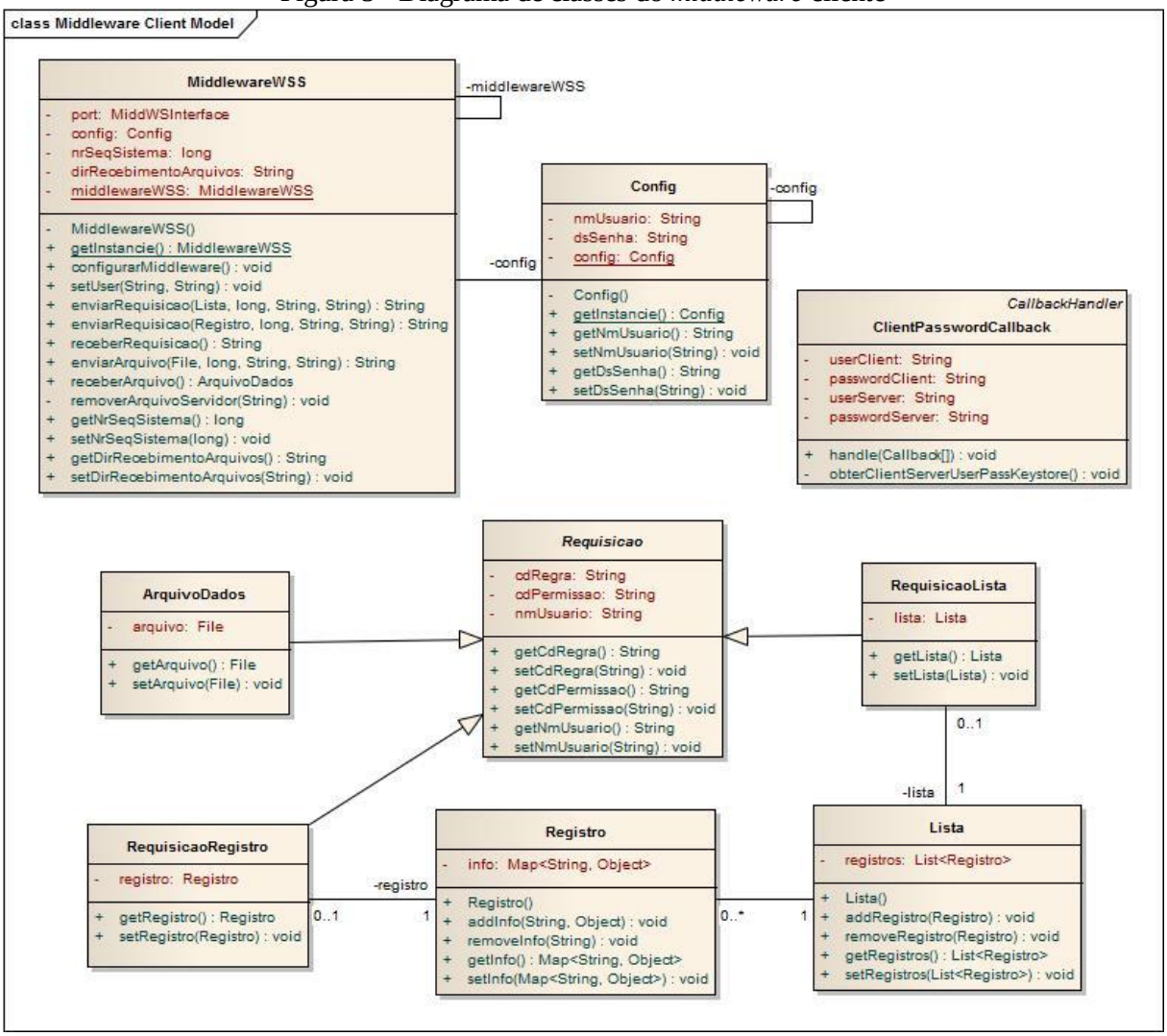
d) `Requisicao`: armazena informações de identificação da requisição, sendo que através dessas informações armazenadas nesta estrutura a aplicação destinatária irá se identificar;

e) `Registro` e `RequisicaoRegistro`: estrutura para armazenamento de dados da requisição;

f) `Lista` e `RequisicaoLista`: estrutura para armazenamento de uma lista de dados da requisição;

g) `ArquivoDados`: estrutura para armazenamento de arquivo.

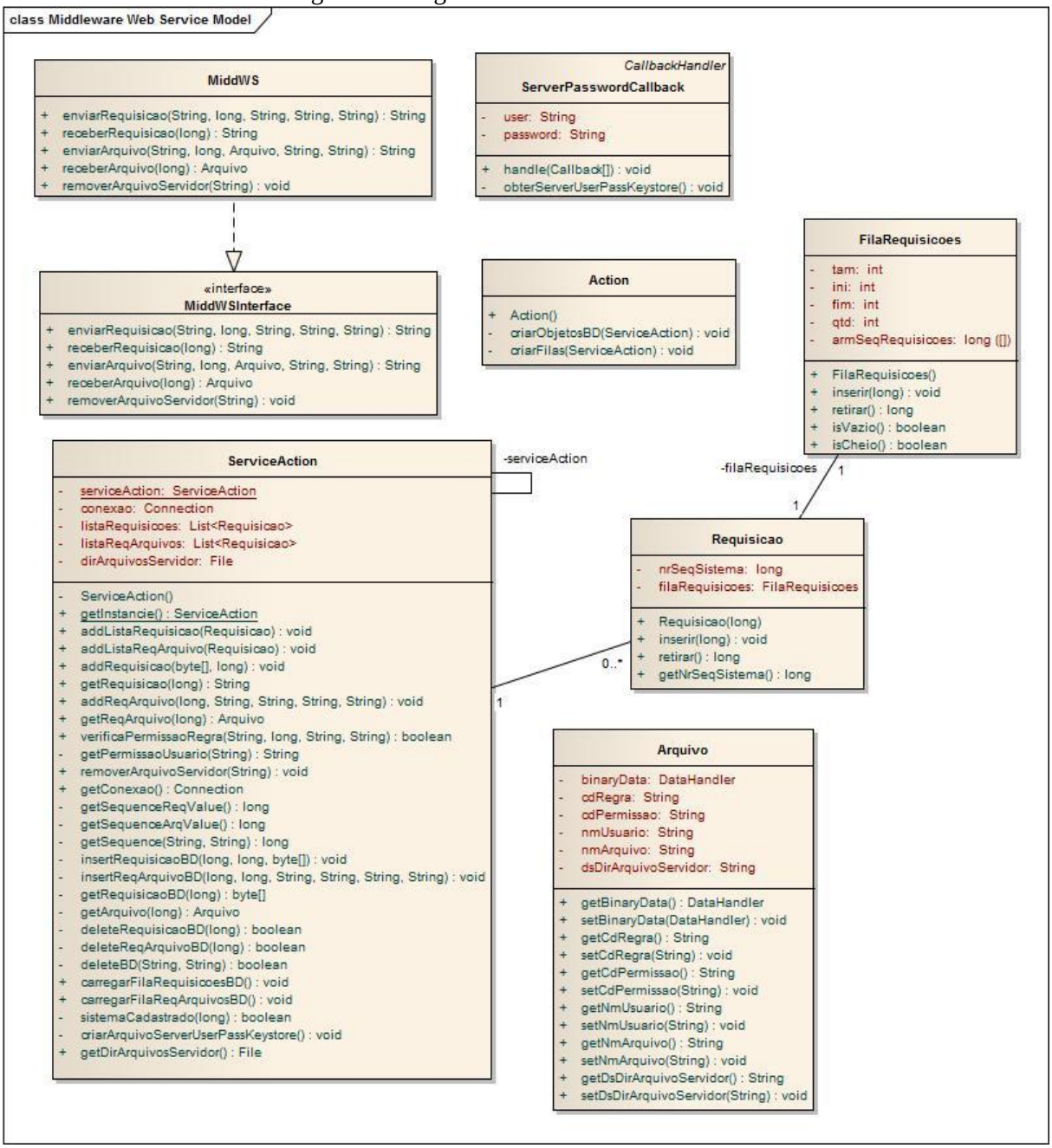
Figura 5 - Diagrama de classes do *middleware* cliente




Já na Figura 6 é apresentado o diagrama de classes do *web service*, sendo mostradas as principais classes, sendo elas:

a) `MiddWSInterface` e `MiddWS`: é a estrutura principal do *web service*, possui métodos para realização da comunicação de dados entre as aplicações;
b) `ServerPasswordCallback`: responsável pela validação das operações de *callback* do *WS-Security server*, sendo utilizado pelo arquivo XML de configuração;
c) `Action`: classe responsável por criar os objetos no banco de dados e criar as estruturas de dados utilizadas na comunicação, para que ocorra de forma assíncrona. Estes objetos são criados ao iniciar o serviço, sendo que esta classe também é utilizada pelo arquivo XML de configuração do *WS-Security*;
d) `ServiceAction`: classe responsável pelo controle de fluxo de dados utilizados na comunicação, esta classe possui métodos para verificação de regras e permissões. Possuí métodos para manipulação das filas que são utilizadas internamente para que a comunicação ocorra de forma assíncrona;
e) `Requisicao` e `FilaRequisicoes`: estrutura de dados utilizadas para implementação de filas de requisições, sendo que cada sistema possui uma fila;
f) `Arquivo`: estrutura para comunicação de dados utilizando arquivos.

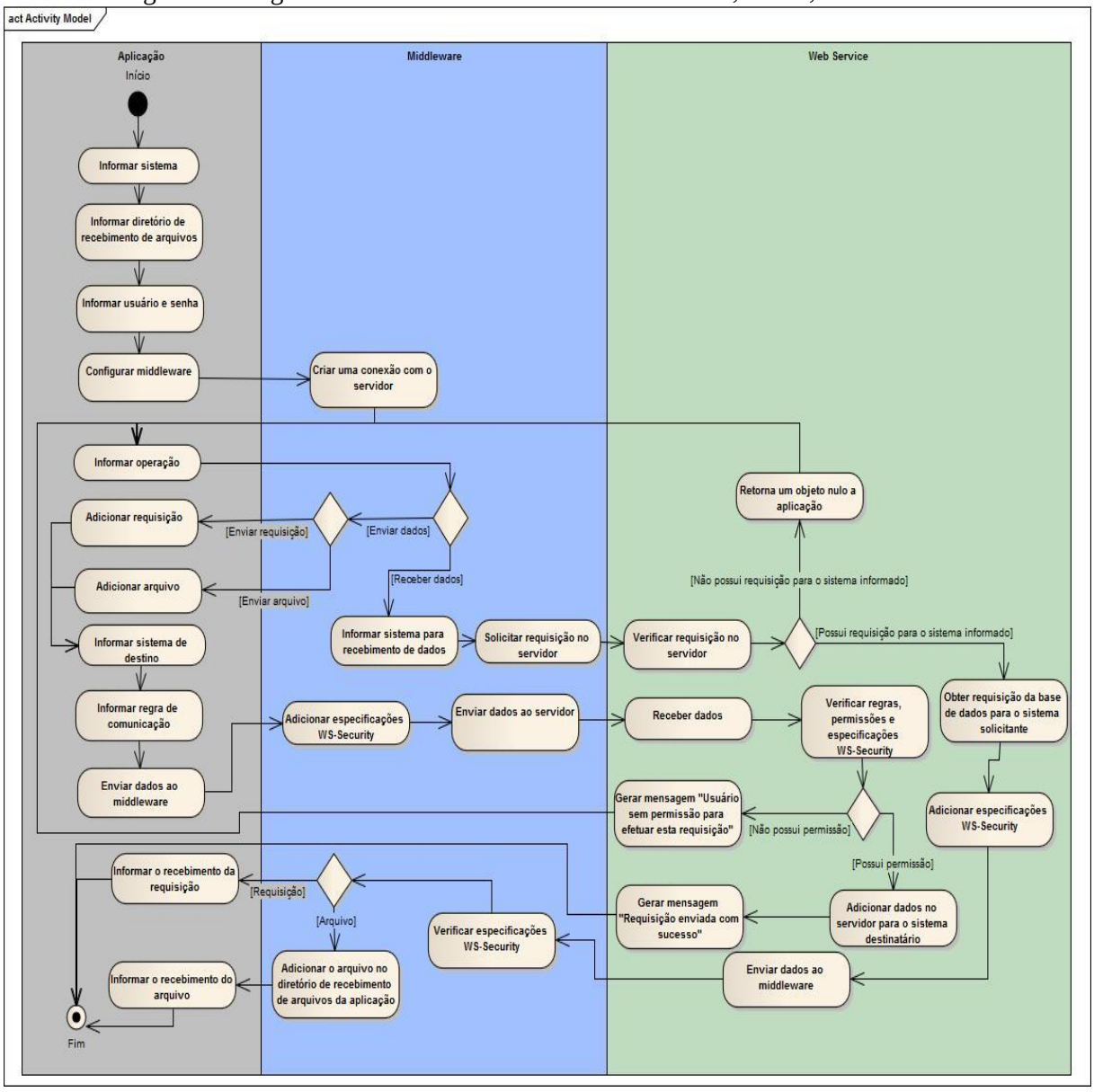
Figura 6 - Diagrama de classes do *web service*




### 3.2.3 Diagrama de atividades

O diagrama de atividades apresentado na Figura 7 exibe o fluxo de uma requisição na qual a aplicação solicita ao *middleware* o envio e recebimento de dados e arquivos de outra aplicação.

Figura 7 - Diagrama de atividades dos casos de uso UC08, UC09, UC10 e UC11




Na figura 7 pode-se observar que a primeira operação realizada pela aplicação é inicializar o *middleware*, informando o sistema para qual serão solicitadas as requisições do servidor. É informado o diretório de recebimento de arquivos e o usuário e senha utilizado na autenticação do *middleware* com o servidor *web service*. Após isso, o *middleware* cria uma conexão com o servidor.

Para o envio de dados ou arquivos para aplicação destinatária, a aplicação informa ao *middleware* o tipo de requisição o qual poderá ser um arquivo, ou uma requisição contendo dados para outra aplicação. É informado o sistema para o qual a requisição será enviada, sendo também informado a regra de comunicação para que a aplicação destinatária possa se identificar sobre o tipo de requisição enviada. Após isso, o *middleware* adiciona as especificações do *WS-Security* na requisição, informando um cabeçalho contendo dados de

autenticação, a mensagem é encriptada e assinada, sendo após enviada via *web service* ao servidor, o qual desencapsula a mensagem verificando as regras e permissões de comunicação, sendo também validado as questões de segurança, como a autenticidade, a confidencialidade e a integridade dos dados. Caso os itens de segurança estiverem em conformidade, os dados são armazenados no servidor para que quando a aplicação para onde a requisição foi destinada solicitar os dados, os mesmos são enviados ao *middleware* que efetuou a solicitação.

Para o recebimento de dados ou arquivos, o *middleware* informa ao servidor o sistema que deseja obter uma requisição e solicita o mesmo via *web service*. No servidor é verificado se há uma requisição para o sistema que efetuou a solicitação, caso possua uma requisição, o mesmo é obtido da base de dados, sendo adicionados também as especificações do *WS-Security* e enviado ao *middleware*. O *middleware* desencapsula a requisição validado as questões de segurança como a confidencialidade e a integridade dos dados. Caso os itens de segurança estiverem em conformidade, o *middleware* informa os dados a aplicação, caso for um arquivo, o mesmo é adicionado do diretório de recebimentos de arquivos.

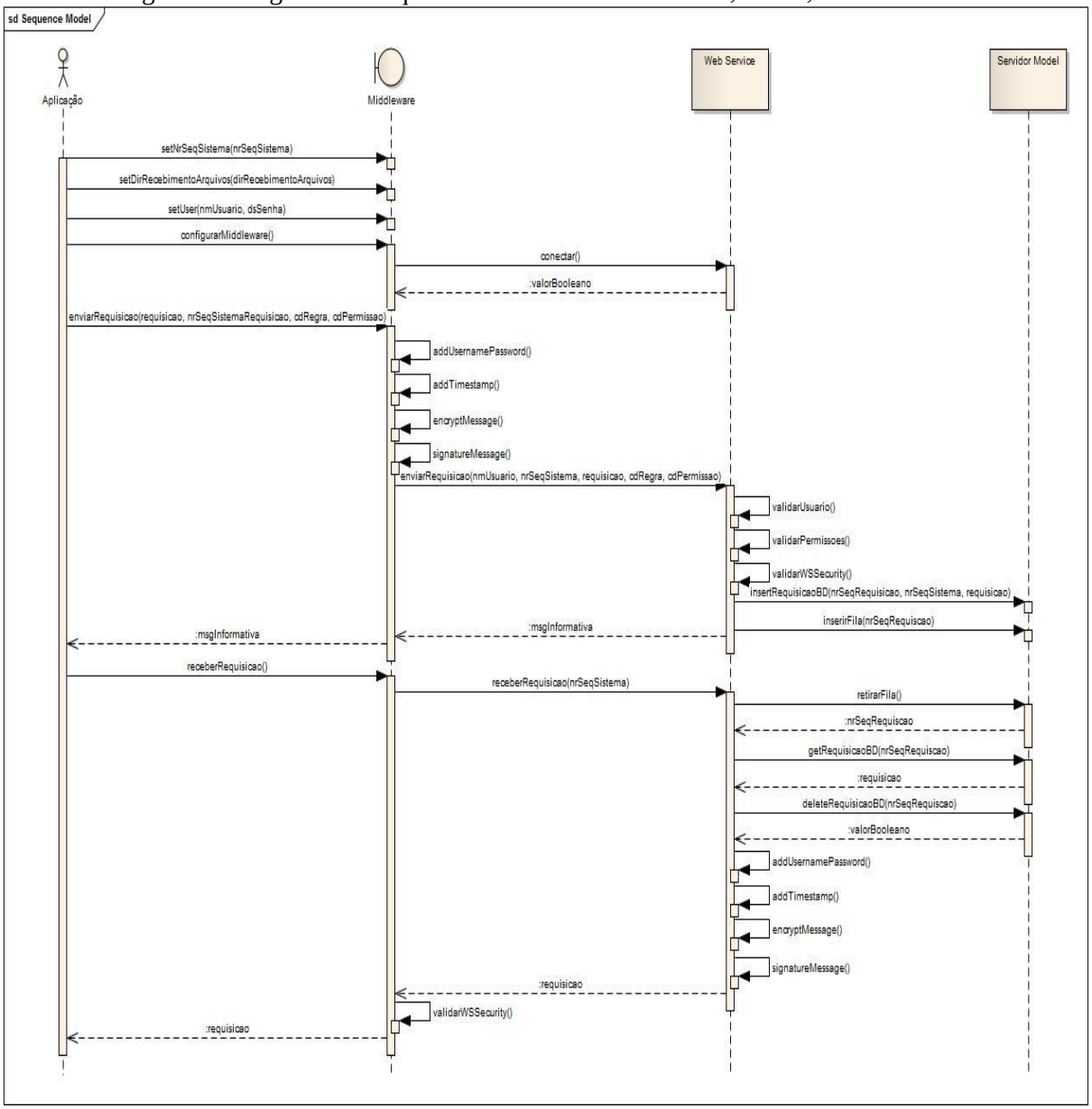
### 3.2.4 Diagrama de sequência

A figura 8 apresenta o diagrama de sequência dos casos de uso UC08 - Enviar requisição, UC09 - Receber requisição, UC10 - Enviar arquivo e UC11 - Receber arquivo. Inicialmente a aplicação configura o *middleware*, informando os dados para inicialização do mesmo o qual realizará uma conexão com o servidor.

Ao enviar uma requisição ao *middleware*, o mesmo adiciona as especificações do *ws-security* a requisição, adicionando as credenciais do usuário. Também é adicionado um *timestamp*, ou seja, um período onde a requisição é válida. Após isso, a requisição é encriptada e assinada, sendo após enviada ao servidor via *web service*. No servidor são verificadas as regras e permissões do usuário, em seguida a requisição é adicionada na base de dados para posteriormente ser consumido pelo *middleware* destinatário.

Para o recebimento da requisição o *middlerware* solicita a mesma via *web service*. O servidor verifica na base de dados, caso haja uma requisição a mesma é obtida, sendo adicionadas as especificações do *WS-Security* a mesma, e posteriormente enviada ao *middleware* o qual encaminha para aplicação novamente.

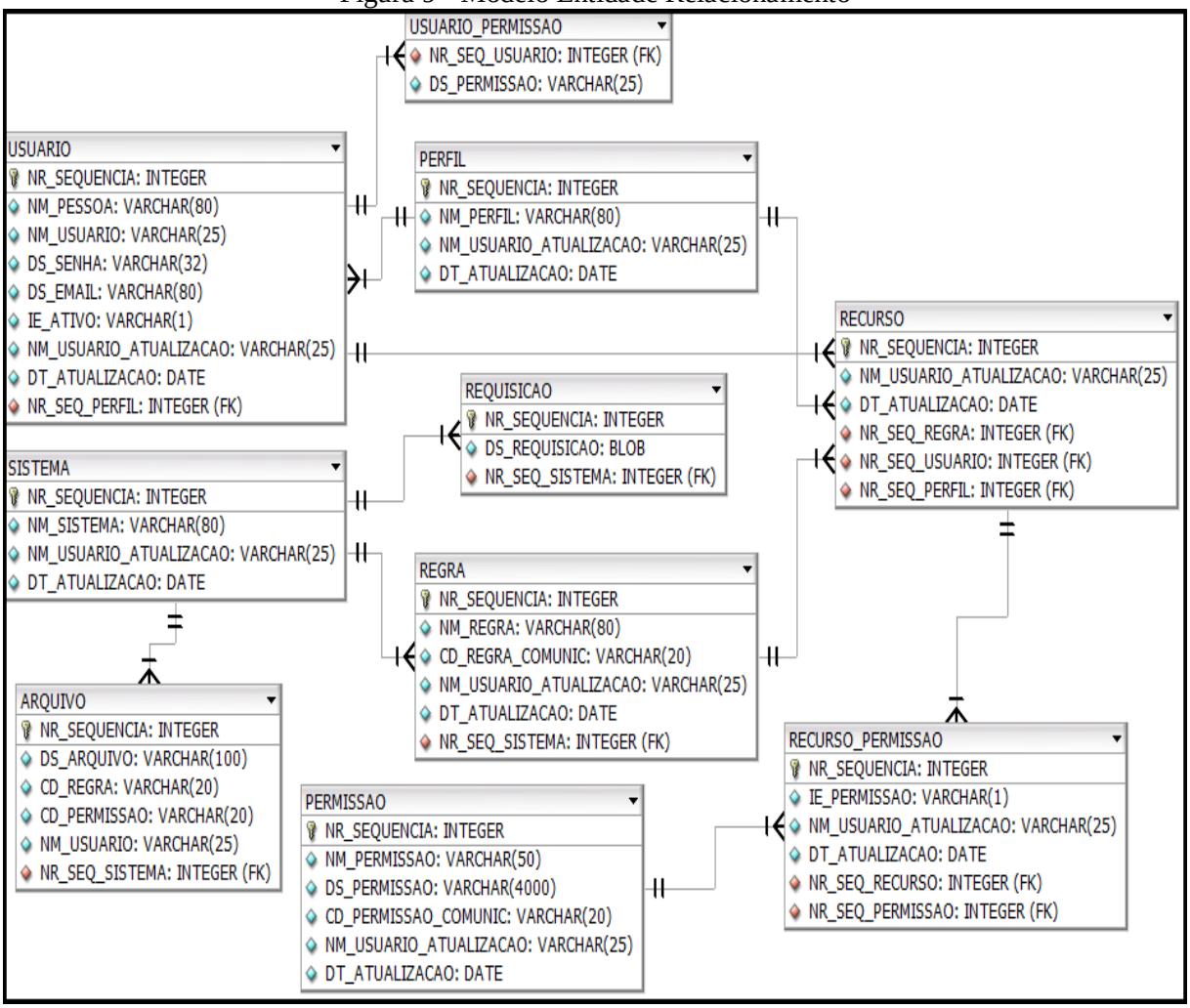
Figura 8 - Diagrama de sequência dos casos de uso UC08, UC09, UC10 e UC11




### 3.2.5 Diagrama de entidade relacionamento

A base de dados foi criada com base no diagrama apresentado na Figura 9. Nela podem ser vistos as tabelas, os tipos de dados das colunas e as chaves primárias e estrangeiras.

Figura 9 - Modelo Entidade Relacionamento




A partir da figura 9 pode-se observar que existem dez tabelas relacionadas ao funcionamento do sistema web e também para manutenção dos registros das requisições atendidas pelo *web service*. Segue uma breve descrição das entidades listadas na Figura 9:

a) `Usuario`: entidade reponsável por manter o cadastro de usuários;

b) `Usuario_Permissao`: entidade auxiliar que armazena as permissões de acesso dos usuários no sistema web;

c) `Perfil`: entidade responsável por manter o cadastro de perfis, sendo que, cada usuário poderá ter um perfil;

d) `Sistema`: entidade responsável por manter o cadastro de aplicações na qual utilizam o *middleware*;

e) `Regra`: entidade responsável por manter o cadastro de regras utilizadas na comunicação de dados entre as aplicações;

f) `Permissao`: entidade responsável por manter o cadastro de permissões utilizadas na comunicação de dados entre as aplicações;

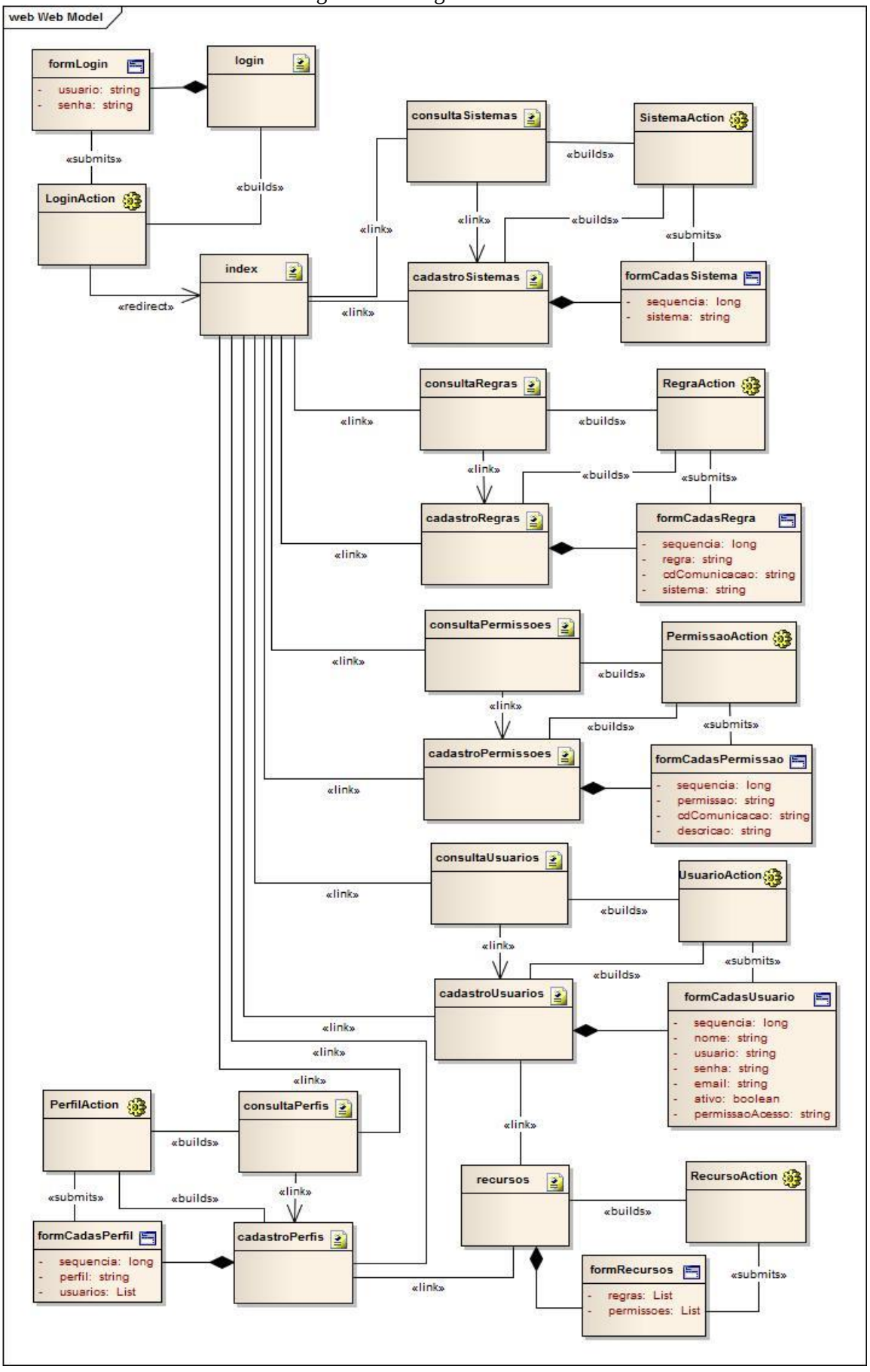
g) `Recurso`: entidade auxiliar utilizada para atribuir uma nova regra ao usuário ou perfil;
h) `Recurso_Permissao`: entidade utilizada para atribuir uma nova permissão ao recurso;
i) `Requisicao`: entidade responsável por armazenar as requisições atendidas pelo *web service*;
j) `Arquivo`: entidade responsável por armazenar os diretórios dos arquivos enviados pelo *middleware* ao servidor.

## 3.3 VISÃO GERAL DO SISTEMA WEB

O sistema web desenvolvido tem por finalidade manter os registros dos usuários, perfis, definições de regras, cadastramento das aplicações e permissões de acesso aos dados sendo utilizados pelo *web service* na comunicação de dados entre as aplicações através do *middleware*.

A visão geral do sistema web é representada pelo diagrama *Web Application Extension* (WAE), exposto na Figura 10, o qual é uma extensão da notação UML com semânticas adicionais e restrições, a fim de permitir a modelagem arquitetural de elementos web. Além disto, com esta extensão é possível realizar a captura, desenvolvimento e a análise de como será a execução das regras de negócio nas páginas web do sistema.

Figura 10 - Diagrama de WAE



## 3.4 IMPLEMENTAÇÃO

Nesta seção serão apresentadas informações sobre técnicas e ferramentas utilizadas para a implementação, bem como a própria implementação e cada etapa de desenvolvimento. Ao final, será descrita a operacionalidade do sistema web desenvolvido.

### 3.4.1 Técnicas e ferramentas utilizadas

O sistema foi desenvolvido utilizando técnicas de orientação a objetos e o ambiente de desenvolvimento NetBeans IDE 7.2. Para persistência dos dados foi utilizado o banco de dados Oracle 10g e o ambiente SQL Developer na versão 3.2.2 para testes e execução dos comandos *Data Definition Language* (DLL) e *Database Manipulation Language* (DML). O servidor container do *web service* e sistema web utilizado foi o Apache Tomcat 7.0.27.

A estrutura para a disponibilização do serviço de segurança web foi separada em três módulos, sendo que a linguagem de programação utilizada para o desenvolvimento dos módulos foi o Java na versão 7. O módulo que compõe o sistema web, sendo o responsável pela administração do serviço foi desenvolvido utilizando as seguintes bibliotecas:

a) JDBC Oracle 10g;
b) JPA Hibernate 4.2.2;
c) Spring Security 3.2.0;
d) JSF 2.1.22;
e) JSTL 1.2.1;
f) Spring Security Facelets TagLib;
g) PrimeFaces 3.5;
h) PrimeFaces Theme Sam 1.0.10.

Já para o módulo servidor *web service*, sendo o responsável por disponibilizar o serviço e pelo controle de fluxo de dados do *middleware* verificando as regras de comunicação, foram utilizadas as seguintes bibliotecas no desenvolvimento:

a) Apache CXF 2.7.6;
b) JDBC Oracle 10g.

Para o módulo *middleware* cliente, responsável por integrar-se as aplicações foram utilizadas as seguintes bibliotecas:

a) Apache CXF 2.7.6;
b) XStream 1.4.4.

As subseções a seguir demonstram a implementação dos módulos, bem como suas principais características e amostras de código fonte.

#### 3.4.1.1 Implementação do sistema web

Para o desenvolvimento do sistema web foi utilizado a especificação Java Server Faces (JSF) em conjunto com a implementação PrimeFaces, possuindo componentes visuais, permitindo a criação de páginas web sem se preocupar com o JavaScript ou HTML. Outra característica importante na arquitetura JSF é a separação entre as camadas de apresentação e de aplicação, sendo utilizado o modelo *Model View Controller* (MVC). O JSF possui uma camada de visualização que, por padrão, se baseia nos *Facelets* e um conjunto de classes conhecidas como *Managed Beans*.

Uma das principais vantagens de utilizar esta API é que a mesma permite a criação de *templates*, ou seja, uma página padrão, sendo utilizada por todas as outras páginas do sistema, com isso as páginas do sistema herdam da *master page* a aparência visual. O PrimeFaces também possibilita a utilização de temas para a aparência visual do sistema, sendo que no desenvolvimento foi utilizado o tema `sam` para o *layout* das páginas.

Como o JSF segue o padrão arquitetural MVC é necessário a configuração de *servlets* no arquivo `web.xml` conforme apresentado entre as linhas 8 e 17 do Quadro 3, que representa o *controller* dentro do modelo MVC. O *servlet* é o responsável em receber as requisições e delegá-las ao *core* JSF.

Para prover a segurança do sistema web foi utilizado o *Spring Security*, este também é configurado no arquivo `web.xml`, conforme apresentado entre as linhas 34 e 53 do Quadro 3.

Quadro 3 - Configuração do arquivo `web.xml`

```
1 <?xml version="1.0" encoding="ISO-8859-1"?>
2
3 <web-app xmlns="http://java.sun.com/xml/ns/javaee"
4       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5       xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
6       version="3.0">
7   <display-name>MiddServidorSegurancaWS</display-name>
8   <servlet>
9      <display-name>FacesServlet</display-name>
10     <servlet-name>FacesServlet</servlet-name>
11     <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
12     <load-on-startup>1</load-on-startup>
13  </servlet>
14  <servlet-mapping>
15     <servlet-name>FacesServlet</servlet-name>
16     <url-pattern>*.jsf</url-pattern>
17  </servlet-mapping>
18  <context-param>
19     <param-name>javax.faces.PROJECT_STAGE</param-name>
20     <param-value>Development</param-value>
21  </context-param>
22  <welcome-file-list>
23     <welcome-file>/pages/principal/login.jsf</welcome-file>
24  </welcome-file-list>
25  <session-config>
26     <session-timeout>30</session-timeout>
27  </session-config>
28  <resource-ref>
29     <description>DataSource ServSecurityDB</description>
30     <res-ref-name>jdbc/ServSecurityDB</res-ref-name>
31     <res-type>javax.sql.DataSource</res-type>
32     <res-auth>Container</res-auth>
33  </resource-ref>
34  <context-param>
35     <param-name>contextConfigLocation</param-name>
36     <param-value>
37        /WEB-INF/applicationContext.xml
38        /WEB-INF/applicationContext-security.xml
39     </param-value>
40  </context-param>
41  <filter>
42     <filter-name>springSecurityFilterChain</filter-name>
43     <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
44  </filter>
45  <filter-mapping>
46     <filter-name>springSecurityFilterChain</filter-name>
47     <url-pattern>/*</url-pattern>
48     <dispatcher>FORWARD</dispatcher>
49     <dispatcher>REQUEST</dispatcher>
50  </filter-mapping>
51  <listener>
52     <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
53  </listener>
54  <context-param>
55     <param-name>primefaces.THEME</param-name>
56     <param-value>sam</param-value>
57  </context-param>
58 </web-app>
```

O *Spring Security* é configurado em dois arquivos xml, sendo que o arquivo `applicationContext-security.xml` possui as configurações de segurança da aplicação, possuindo permissões de acesso aos usuários a determinadas páginas do sistema web através de *roles*, conforme as linhas 10 e 11 do Quadro 4. Sendo que na linha 10 que contém a `ROLE_ADMINISTRADOR` indica que somente usuários administradores tem acesso as páginas da pasta admin, que possui somente páginas de cadastro. Já na linha 11 que contém a `ROLE_USUARIO` e `ROLE_ADMINISTRADOR` indica que para as páginas de consulta, administradores e usuários em geral podem acessar. Caso um usuário não tenha acesso a uma determinada página do sistema será apresentado uma página de acesso negado conforme linha 9 do Quadro 4. Nas linhas 12 a 16 é configurado a página de login do sistema. A autenticação é realizada através das linhas 18 a 34, para a senha do usuário é calculado um *hash* `md5`.

Quadro 4 - Configuração do arquivo `applicationContext-security.xml`

```
1 <?xml version="1.0" encoding="ISO-8859-1"?>
2 <b:beans xmlns="http://www.springframework.org/schema/security"
3     xmlns:b="http://www.springframework.org/schema/beans"
4     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5     xsi:schemaLocation="http://www.springframework.org/schema/beans
6                 http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
7                 http://www.springframework.org/schema/security
8                 http://www.springframework.org/schema/security/spring-security-3.2.xsd">
9   <http access-denied-page="/pages/principal/acessoNegado.jsf">
10    <intercept-url pattern="/pages/admin/**" access="ROLE_ADMINISTRADOR"/>
11    <intercept-url pattern="/pages/consulta/**" access="ROLE_ADMINISTRADOR,ROLE_USUARIO"/>
12    <form-login login-page="/pages/principal/login.jsf"
13          always-use-default-target="true"
14          default-target-url="/pages/principal/index.jsf"
15          authentication-failure-url="/pages/principal/login.jsf?login_error=1"/>
16    <logout/>
17  </http>
18  <authentication-manager>
19    <authentication-provider>
20      <password-encoder hash="md5"/>
21      <jdbc-user-service data-source-ref="ServSecurityDataSource"
22                authorities-by-username-query="select   u.nm_usuario,
23                                                p.ds_permissao
24                                        from    usuario u,
25                                                usuario_permissao p
26                                        where   u.nr_sequencia = p.nr_seq_usuario
27                                        and     u.nm_usuario = ?"
28                users-by-username-query="select  nm_usuario,
29                                         ds_senha,
30                                         ie_ativo
31                                    from   usuario
32                                    where  nm_usuario = ?"/>
33    </authentication-provider>
34  </authentication-manager>
35 </b:beans>
```

Já o arquivo `applicationContext.xml` contém a configuração de conexão do *spring* com o banco de dados Oracle, utilizando também o arquivo `context.xml` que contém um *data source* de conexão. No Quadro 5 e 6 é apresentado os dois arquivos de conexão que são utilizados pelo *spring security* na autenticação do usuário no sistema web.

Quadro 5 - Configuração do arquivo `applicationContext.xml`

```
1 <?xml version="1.0" encoding="ISO-8859-1"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xsi:schemaLocation="http://www.springframework.org/schema/beans
5                 http://www.springframework.org/schema/beans/spring-beans-3.2.xsd">
6   <bean id="ServSecurityDataSource" class="org.springframework.jndi.JndiObjectFactoryBean">
7     <property name="jndiName">
8       <value>java:comp/env/jdbc/ServSecurityDB</value>
9     </property>
10  </bean>
11 </beans>
```

Quadro 6 - Configuração do arquivo `context.xml`

```
1 <?xml version="1.0" encoding="ISO-8859-1"?>
2 <Context path="/MiddServidorSegurancaWS" reloadable="true">
3   <Resource name="jdbc/ServSecurityDB"
4         auth="Container"
5         type="javax.sql.DataSource"
6         maxActive="100"
7         maxIdle="30"
8         maxWait="10000"
9         username="systemmidd"
10        password="key50100"
11        driverClassName="oracle.jdbc.OracleDriver"
12        url="jdbc:oracle:thin:@localhost:1521:XE"/>
13 </Context>
```

Para a persistência dos dados no bando de dados foi utilizado o *Java Persistence API* (JPA) com a biblioteca *Hibernate*, pois o mesmo permite mapear o objeto e persistir no banco de dados. Esta tecnologia foi utilizado pois o sistema web possui somente cadastros que são utilizados no controle de fluxo de dados do *middleware* para a comunicação de dados entre as aplicações. No Quadro 7 é apresentado o arquivo `persistence.xml` contendo as configurações do JPA. Nas linhas 8 a 14 possuem as classes que são persistidas no banco de dados. O arquivo `persistence.xml` também utiliza o *data source* configurado no arquivo `context.xml` conforme demontrado na linha 17 do Quadro 7. As entidades do banco de dados são criados pelo JPA ao iniciar o sistema web através da linha 21.

Quadro 7 - Configuração do arquivo `persistence.xml`

```
1 <?xml version="1.0" encoding="ISO-8859-1"?>
2 <persistence version="2.0" xmlns="http://java.sun.com/xml/ns/persistence"
3                 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4                 xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
5                                 http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">
6   <persistence-unit name="MiddServidorSegurancaWSPU" transaction-type="RESOURCE_LOCAL">
7     <provider>org.hibernate.ejb.HibernatePersistence</provider>
8     <class>br.server.model.entities.Sistema</class>
9     <class>br.server.model.entities.Permissao</class>
10    <class>br.server.model.entities.Regra</class>
11    <class>br.server.model.entities.RecursoPermissao</class>
12    <class>br.server.model.entities.Recurso</class>
13    <class>br.server.model.entities.Usuario</class>
14    <class>br.server.model.entities.Perfil</class>
15    <exclude-unlisted-classes>false</exclude-unlisted-classes>
16    <properties>
17      <property name="hibernate.connection.datasource" value="java:comp/env/jdbc/ServSecurityDB"/>
18      <property name="hibernate.dialect" value="org.hibernate.dialect.Oracle10gDialect"/>
19      <property name="hibernate.transaction.factory_class" value="org.hibernate.transaction.JDBCTransactionFactory"/>
20      <property name="hibernate.cache.provider_class" value="org.hibernate.cache.NoCacheProvider"/>
21      <property name="hibernate.hbm2ddl.auto" value="update"/>
22    </properties>
23  </persistence-unit>
24 </persistence>
```

Na criação das páginas web foram utilizados *templates*, conforme apresentado no Quadro 8. O arquivo `principal.xhtml` é o arquivo na qual todas as outras páginas do sistema herdam, sendo que substituem apenas o conteúdo da tag `<ui:insert>` do arquivo template pelo `<ui:define>` definido nas demais páginas do sistema, com isso todas as páginas possuem o mesmo layout.

Quadro 8 - Arquivo *template* `principal.xhtml`

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!--
Document   : principal.xhtml
Created on : 13/08/2013, 19:22:32
Author     : Caio
-->
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:p="http://primefaces.org/ui"
      xmlns:f="http://java.sun.com/jsf/core">
    <h:head>
        <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1"/>
        <title><ui:insert name="tituloPg">Servidor WS-Security</ui:insert></title>
        <h:outputStylesheet library="css" name="principal.css"/>
        <h:outputScript library="js" name="principal.js"/>
    </h:head>
    <h:body>

        <p:panel styleClass="pnContainer">

            <div id="titulo">
                <ui:insert name="titulo">Servidor WS-Security</ui:insert>
            </div>

            <div id="imgLogoTipo">
                <h:graphicImage library="imagens" name="LogoTipo.png"/>
            </div>

            <p:panel styleClass="pnInterno" id="pnInternoID">
                <f:facet name="header">
                    <h:outputText value="Usuário: #{request.remoteUser}"/>
                    <h:outputText value=" " id="espacamento"/>
                    <a href="#{request.contextPath}/j_spring_security_logout">Sair</a>
                </f:facet>

                <p:panel styleClass="pnCorpo" id="pnCorpoID">
                    <div id="menu">
                        <h:form id="formMenu" acceptcharset="ISO-8859-1">
                            <p:menu>
                                <p:submenu label="Principal">
                                    <p:menuitem value="Início" outcome="/pages/principal/index" icon="ui-icon-arrowrefresh-1-w"/>
                                </p:submenu>
                                <p:submenu label="Sistemas">
                                    <p:menuitem value="Consulta" outcome="/pages/consulta/consSistema" icon="ui-icon-document"/>
                                    <p:menuitem value="Cadastro" outcome="/pages/admin/cadasSistema" icon="ui-icon-disk"/>
                                </p:submenu>
                                <p:submenu label="Permissões">
                                    <p:menuitem value="Consulta" outcome="/pages/consulta/consPermissao" icon="ui-icon-document"/>
                                    <p:menuitem value="Cadastro" outcome="/pages/admin/cadasPermissao" icon="ui-icon-disk"/>
                                </p:submenu>
                                <p:submenu label="Regras">
                                    <p:menuitem value="Consulta" outcome="/pages/consulta/consRegra" icon="ui-icon-document"/>
                                    <p:menuitem value="Cadastro" outcome="/pages/admin/cadasRegra" icon="ui-icon-disk"/>
                                </p:submenu>
                                <p:submenu label="Usuários">
                                    <p:menuitem value="Consulta" outcome="/pages/consulta/consUsuario" icon="ui-icon-document"/>
                                    <p:menuitem value="Cadastro" outcome="/pages/admin/cadasUsuario" icon="ui-icon-disk"/>
                                </p:submenu>
```

Quadro 8 - Continuação arquivo *template* `principal.xhtml`

```
61                    <p:submenu label="Perfil">
62                      <p:menuitem value="Consulta" outcome="/pages/consulta/consPerfil" icon="ui-icon-document"/>
63                      <p:menuitem value="Cadastro" outcome="/pages/admin/cadasPerfil" icon="ui-icon-disk"/>
64                    </p:submenu>
65                  </p:menu>
66                </h:form>
67              </div>
68              <div id="corpo">
69                <ui:insert name="corpo"/>
70              </div>
71            </p:panel>
72
73            <f:facet name="footer">
74              <h:outputText value="Middleware WS Server Security 1.0"/>
75            </f:facet>
76          </p:panel>
77
78        </p:panel>
79
80    </h:body>
81 </html>
```

### 3.4.1.2 Implementação do servidor *web service*

Para o desenvolvimento do servidor *web service* que implementa as especificações *WS-Security* foi utilizado a API Apache CXF o qual é uma implementação da especificação JAX-WS. O Apache CXF permite configurar o *WS-Security* através de interceptadores xml via *Spring*. A API proporciona as implementações padrões do *WS-Security* como *UsernameToken*, *Timestamp*, *Signature* e *Encrypt*.

Para realização da comunicação de dados entre as aplicações foi criado um serviço web seguro que permite as aplicações trocarem dados e arquivos confidenciais de forma segura. No Quadro 9 são apresentados os métodos da classe `MiddWS`, sendo a classe responsável por implementar o serviço web. Esta classe possui cinco métodos, sendo que o método `enviarRequisicao` é o responsável por enviar uma requisição ao servidor. Esta requisição é armazenada no banco de dados para que quando a aplicação para qual a requisição foi destinada solicitar a mesma através do método `receberRequisicao` o mesmo é obtido da base de dados e retornado a aplicação. Os métodos `enviarArquivo` e `receberArquivo` possuem as mesmas funções, porém são utilizados para troca de arquivos, sendo que na base de dados é armazenado somente o diretório do arquivo que foi salvo em uma pasta no servidor. Quando o mesmo é consumido pelo *middleware*, o *middleware* realiza uma chamada ao método `removerArquivoServidor` para remoção do arquivo físico no servidor.

Quadro 9 - Métodos utilizados na comunicação de dados pelo *middleware*

```
1
2   public String enviarRequisicao(String nmUsuario, long nrSeqSistema, String requisicao, String cdRegra, String cdPermissao) throws Exception {
3       ServiceAction serviceAction = ServiceAction.getInstancie();
4       if (serviceAction.verificaPermissaoRegra(nmUsuario, nrSeqSistema, cdRegra, cdPermissao)) {
5           serviceAction.addRequisicao(requisicao.getBytes(), nrSeqSistema);
6           return "Requisição enviada com sucesso";
7       }
8       throw new Exception("Usuário sem permissão para efetuar esta requisição");
9   }
10
11  public String receberRequisicao(long nrSeqSistema) throws Exception {
12      ServiceAction serviceAction = ServiceAction.getInstancie();
13      return serviceAction.getRequisicao(nrSeqSistema);
14  }
15
16  public String enviarArquivo(String nmUsuario, long nrSeqSistema, Arquivo arquivo, String cdRegra, String cdPermissao) throws Exception {
17      ServiceAction serviceAction = ServiceAction.getInstancie();
18      if (serviceAction.verificaPermissaoRegra(nmUsuario, nrSeqSistema, cdRegra, cdPermissao)) {
19
20          DataHandler dataHandler = arquivo.getBinaryData();
21
22          String nmArquivo = "SIS" + nrSeqSistema + "_" + arquivo.getNmArquivo();
23
24          String nome = nmArquivo.substring(0, nmArquivo.lastIndexOf("."));
25          String extencao = nmArquivo.substring(nmArquivo.lastIndexOf("."), nmArquivo.length());
26          File novoArquivo = new File(serviceAction.getDirArquivosServidor().getPath() + "\\", nmArquivo);
27          int cont = 1;
28          while (novoArquivo.exists()) {
29              novoArquivo = new File(serviceAction.getDirArquivosServidor().getPath() + "\\", nome + "_" + cont + extencao);
30              cont++;
31          }
32
33          FileOutputStream out = new FileOutputStream(novoArquivo);
34          dataHandler.writeTo(out);
35          out.flush();
36          out.close();
37
38          serviceAction.addReqArquivo(nrSeqSistema, cdRegra, cdPermissao, novoArquivo.getPath(), nmUsuario);
39
40          return "Arquivo enviado com sucesso";
41      }
42      throw new Exception("Usuário sem permissão para efetuar esta requisição");
43  }
44
45  public Arquivo receberArquivo(long nrSeqSistema) throws Exception {
46      ServiceAction serviceAction = ServiceAction.getInstancie();
47      return serviceAction.getReqArquivo(nrSeqSistema);
48  }
49
50  public void removerArquivoServidor(String dsArquivo) throws Exception {
51      ServiceAction serviceAction = ServiceAction.getInstancie();
52      serviceAction.removerArquivoServidor(dsArquivo);
53  }
```

Para implementação da especificação do *WS-Security* no servidor foi criado o arquivo `cxf-servlet.xml` que contém a publicação do serviço web e realiza a segurança do serviço via interceptadores S*pring* como apresentado no Quadro 10.

Quadro 10 - Configuração do arquivo `cxf-servlet.xml`

```
1 <?xml version="1.0" encoding="ISO-8859-1"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xmlns:jaxws="http://cxf.apache.org/jaxws"
5     xsi:schemaLocation="http://www.springframework.org/schema/beans
6                 http://www.springframework.org/schema/beans/spring-beans.xsd
7                 http://cxf.apache.org/jaxws
8                 http://cxf.apache.org/schemas/jaxws.xsd">
9
10   <bean id="servico" class="br.server.webservice.MiddWS"/>
11
12   <bean id="initAction" class="br.server.action.Action"/>
13
14   <jaxws:endpoint id="middWSService"
15             implementor="#servico"
16             address="/servico">
17
18     <jaxws:properties>
19       <entry key="mtom-enabled" value="true"/>
20     </jaxws:properties>
21
22     <jaxws:outInterceptors>
23       <ref bean="UsernameTokenTimestampSignEncrypt_Response"/>
24     </jaxws:outInterceptors>
25     <jaxws:inInterceptors>
26       <ref bean="UsernameTokenTimestampSignEncrypt_Request"/>
27       <bean class="org.apache.cxf.ws.security.wss4j.DefaultCryptoCoverageChecker"/>
28     </jaxws:inInterceptors>
29
30   </jaxws:endpoint>
31
32   <bean id="UsernameTokenTimestampSignEncrypt_Request" class="org.apache.cxf.ws.security.wss4j.WSS4JInInterceptor">
33     <constructor-arg>
34       <map>
35         <entry key="action" value="UsernameToken Timestamp Signature Encrypt"/>
36         <entry key="passwordType" value="PasswordDigest"/>
37         <entry key="passwordCallbackClass" value="br.server.callback.ServerPasswordCallback"/>
38         <entry key="signaturePropFile" value="br\server\properties\servidor.properties"/>
39         <entry key="decryptionPropFile" value="br\server\properties\servidor.properties"/>
40         <entry key="encryptionKeyTransportAlgorithm" value="http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p"/>
41         <entry key="signatureAlgorithm" value="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
42       </map>
43     </constructor-arg>
44   </bean>
45
46   <bean id="UsernameTokenTimestampSignEncrypt_Response" class="org.apache.cxf.ws.security.wss4j.WSS4JOutInterceptor">
47     <constructor-arg>
48       <map>
49         <entry key="action" value="UsernameToken Timestamp Signature Encrypt"/>
50         <entry key="user" value="sysserver"/>
51         <entry key="passwordType" value="PasswordDigest"/>
52         <entry key="passwordCallbackClass" value="br.server.callback.ServerPasswordCallback"/>
53         <entry key="signaturePropFile" value="br\server\properties\servidor.properties"/>
54         <entry key="encryptionPropFile" value="br\server\properties\servidor.properties"/>
55         <entry key="encryptionUser" value="useReqSigCert"/>
56         <entry key="signatureUser" value="sysserver"/>
57         <entry key="signatureParts" value="{Element}{http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd}
58                   Timestamp;{Element}{http://schemas.xmlsoap.org/soap/envelope/}Body"/>
59         <entry key="encryptionParts" value="{Element}{http://www.w3.org/2000/09/xmldsig#}Signature;{Content}
60                   {http://schemas.xmlsoap.org/soap/envelope/}Body"/>
61         <entry key="encryptionSymAlgorithm" value="http://www.w3.org/2001/04/xmlenc#tripledes-cbc"/>
62         <entry key="encryptionKeyTransportAlgorithm" value="http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p"/>
63         <entry key="signatureAlgorithm" value="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
64       </map>
65     </constructor-arg>
66   </bean>
67
68 </beans>
```

A configuração do *WS-Security* é realizada entre as linhas 32 e 66, sendo implementados todas as funcionalidades do *WS-Security* no servidor. Nas linhas 32 a 44 é configurada a operação de *request*, já nas linhas 46 a 66 é configurada a operação de *response*. As funcionalidades implementadas do *WS-Security* no Quadro 10 são:

a) *UsernameToken*: utiliza um token para autenticação;
b) *Timestamp*: utilizado para estipular uma janela de tempo durante o transporte da requisição, o mesmo define um tempo em que a requisição é válida;
c) *Signature*: utilizado para a confidêncialidade dos dados;
d) *Encrypt*: utilizado para integridade dos dados.

Para assinatura e criptografia da mensagem SOAP é utilizado o arquivo `servidor.properties` conforme apresentado no Quadro 11. Este arquivo apresenta informações do *KeyStore* utilizado para o armazenamento de chaves no servidor.

Quadro 11 - Configuração do arquivo `servidor.properties`

```
1 org.apache.ws.security.crypto.provider=org.apache.ws.security.components.crypto.Merlin
2 org.apache.ws.security.crypto.merlin.keystore.type=jks
3 org.apache.ws.security.crypto.merlin.keystore.password=spserv500
4 org.apache.ws.security.crypto.merlin.keystore.alias=sysserver
5 org.apache.ws.security.crypto.merlin.keystore.file=C:/KeyStore/serverKeystore.jks
```

O arquivo *KeyStore* é utilizado tanto no servidor quanto no *middleware*, sendo que no *middleware* também é criado um *KeyStore* para criptografia e assinatura dos dados. No Quadro 12 é apresentada a criação dos arquivos `serverKeystore.jks` e `clientKeystore.jks` via comandos.

Quadro 12 - Configuração do *KeyStore*

```
1   keytool -genkey -keyalg RSA -sigalg SHA1withRSA -validity 730 -alias sysserver -keypass serv50200
2      -storepass spserv500 -keystore serverKeystore.jks -dname "cn=localhost"
3
4   keytool -genkey -keyalg RSA -sigalg SHA1withRSA -validity 730 -alias sysuser -keypass clie50200
5      -storepass spclient250 -keystore clientKeystore.jks -dname "cn=clientuser"
6
7   keytool -export -rfc -keystore clientKeystore.jks -storepass spclient250 -alias sysuser -file crtClient.cer
8
9   keytool -import -trustcacerts -keystore serverKeystore.jks -storepass spserv500 -alias sysuser -file crtClient.cer -noprompt
10
11   keytool -export -rfc -keystore serverKeystore.jks -storepass spserv500 -alias sysserver -file crtServ.cer
12
13   keytool -import -trustcacerts -keystore clientKeystore.jks -storepass spclient250 -alias sysserver -file crtServ.cer -noprompt
```

Para as operações de *callback* do *WS-Security* foi utilizado a classe `ServerPasswordCallback` conforme apresentado no Quadro 13. Esta classe realiza a autenticação do usuário no servidor, adiciona a chave do *KeyStore* obtida do arquivo `serverUserPassKeystore.key` no *WS-Security* para validação da assinatura e criptografia. As operações de *callback* do *WS-Security* são realizados tanto nas operações de *request* quanto nas operações de *response* do *web service*.

Quadro 13 - Classe `ServerPasswordCallback.java`

```
1 import br.server.action.ServiceAction;
2 import java.io.BufferedReader;
3 import java.io.File;
4 import java.io.FileReader;
5 import java.io.IOException;
6 import java.sql.ResultSet;
7 import javax.security.auth.callback.Callback;
8 import javax.security.auth.callback.CallbackHandler;
9 import javax.security.auth.callback.UnsupportedCallbackException;
10 import org.apache.ws.security.WSPasswordCallback;
11
12 public class ServerPasswordCallback implements CallbackHandler {
13
14    private String user;
15    private String password;
16
17    @Override
18    public void handle(Callback[] callbacks) throws IOException, UnsupportedCallbackException {
19       ServiceAction serviceAction;
20       try {
21          serviceAction = ServiceAction.getInstancie();
22       } catch (Exception ex) {
23          throw new SecurityException("Não foi possível estabelecer conexão com o servidor");
24       }
25       ResultSet rs = null;
26       boolean passwordInformado = false;
27       try {
28          obterServerUserPassKeystore();
29          WSPasswordCallback pc;
30          for (Callback callback : callbacks) {
31             if (callback instanceof WSPasswordCallback) {
32                pc = (WSPasswordCallback) callback;
33                if (pc.getUsage() == WSPasswordCallback.USERNAME_TOKEN) {
34                   if (user.equals(pc.getIdentifier())) {
35                      pc.setPassword(password);
36                      passwordInformado = true;
37                   } else {
38                      rs = serviceAction.getConexao().prepareStatement("SELECT NM_USUARIO, DS_SENHA, IE_ATIVO FROM USUARIO "
39                            + "WHERE NM_USUARIO = '" + pc.getIdentifier() + "'").executeQuery();
40                      if (rs.next()
41                            && rs.getString("NM_USUARIO").equals(pc.getIdentifier())
42                            && rs.getBoolean("IE_ATIVO") == true) {
43                         pc.setPassword(rs.getString("DS_SENHA"));
44                         passwordInformado = true;
45                      }
46                   }
47                } else if (pc.getUsage() == WSPasswordCallback.DECRYPT
48                      || pc.getUsage() == WSPasswordCallback.SIGNATURE) {
49                   if (user.equals(pc.getIdentifier())) {
50                      pc.setPassword(password);
51                      passwordInformado = true;
52                   }
53                }
54             }
55          }
56       } catch (Exception ex) {
57          throw new SecurityException("Não foi possível autenticar o usuário no servidor");
58       } finally {
59          try {
60             if (rs != null) {
61                rs.close();
62             }
63          } catch (Exception ex) {}
64          if (!passwordInformado) {
65             throw new SecurityException("Usuário/Senha inválido");
66          }
67       }
68    }
69
70    private void obterServerUserPassKeystore() throws Exception {
71       BufferedReader in = new BufferedReader(new FileReader(new File(System.getProperty("user.home") + "\\", "serverUserPassKeystore.key")));
72       user = in.readLine();
73       user = user.substring(user.indexOf(":") + 1, user.length());
74       password = in.readLine();
75       password = password.substring(password.indexOf(":") + 1, password.length());
76       in.close();
77    }
78 }
```

No servidor *web service* foi implementado uma fila de requisições para atender as solicitações dos *middlewares* para que a comunicação ocorra de forma assíncrona. Desta forma o sistema que envia uma requisição não precisa esperar o outro sistema receber a requisição. Na inicialização do *web service* é executada a classe `Action` conforme apresentado na linha 12 do Quadro 10, sendo criado uma fila para cada sistema cadastrado no servidor. Na medida em que novos sistemas enviem requisições ao servidor, para estes sistemas são criadas suas filas em tempo de execução. Nesta fila é armazenada apenas a sequência da requisição armazenada no banco. Quando o *middleware* destinatário solicitar uma requisição para um sistema o mesmo é verificado na fila. Caso possua uma sequência para o sistema informado, a mesma é retirada e obtida da base de dados, sendo enviada a aplicação na qual foi destinada.

#### 3.4.1.3 Implementação do *middleware*

Para a implementação do *middleware* foi utilizado a API Apache CXF. Sendo que no Quadro 14 é apresentada a configuração do arquivo `cxf-client.xml`, utilizado para implementação das especificação do *WS-Security client* através de interceptadores S*pring*, sendo o consumidor do serviço web disponibilizado.

No *middleware* consumidor do serviço também são implementadas as especificações *UsernameToken*, *Timestamp*, *Signature* e *Encrypt* padrão do *WS-Security*.

Quadro 14 - Configuração do arquivo `cxf-client.xml`

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:jaxws="http://cxf.apache.org/jaxws"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
                http://www.springframework.org/schema/beans/spring-beans.xsd
                http://cxf.apache.org/jaxws
                http://cxf.apache.org/schemas/jaxws.xsd">

  <import resource="classpath:META-INF/cxf/cxf.xml"/>
  <import resource="classpath*:META-INF/cxf/cxf-extension-*.xml"/>

  <jaxws:client id="middWSClient"
          serviceClass="ws.MiddWSInterface"
          address="http://localhost:8084/MiddServicoWS/ws/servico">

    <jaxws:properties>
      <entry key="mtom-enabled" value="true"/>
    </jaxws:properties>

    <jaxws:inInterceptors>
      <ref bean="UsernameTokenTimestampSignEncrypt_Response"/>
      <bean class="org.apache.cxf.ws.security.wss4j.DefaultCryptoCoverageChecker"/>
    </jaxws:inInterceptors>
    <jaxws:outInterceptors>
      <ref bean="UsernameTokenTimestampSignEncrypt_Request"/>
    </jaxws:outInterceptors>

  </jaxws:client>

  <bean id="UsernameTokenTimestampSignEncrypt_Request" class="org.apache.cxf.ws.security.wss4j.WSS4JOutInterceptor">
    <constructor-arg>
      <map>
        <entry key="action" value="UsernameToken Timestamp Signature Encrypt"/>
        <entry key="user" value="sysuser"/>
        <entry key="passwordType" value="PasswordDigest"/>
        <entry key="passwordCallbackClass" value="br.client.middleware.callback.ClientPasswordCallback"/>
        <entry key="signaturePropFile" value="br\client\middleware\config\properties\cliente.properties"/>
        <entry key="encryptionPropFile" value="br\client\middleware\config\properties\cliente.properties"/>
        <entry key="encryptionUser" value="sysserver"/>
        <entry key="signatureUser" value="sysuser"/>
        <entry key="signatureParts" value="{Element}{http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd}
                Timestamp;{Element}{http://schemas.xmlsoap.org/soap/envelope/}Body"/>
        <entry key="encryptionParts" value="{Element}{http://www.w3.org/2000/09/xmldsig#}Signature;{Content}
                {http://schemas.xmlsoap.org/soap/envelope/}Body"/>
        <entry key="encryptionSymAlgorithm" value="http://www.w3.org/2001/04/xmlenc#tripledes-cbc"/>
        <entry key="encryptionKeyTransportAlgorithm" value="http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p"/>
        <entry key="signatureAlgorithm" value="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
      </map>
    </constructor-arg>
  </bean>

  <bean id="UsernameTokenTimestampSignEncrypt_Response" class="org.apache.cxf.ws.security.wss4j.WSS4JInInterceptor">
    <constructor-arg>
      <map>
        <entry key="action" value="UsernameToken Timestamp Signature Encrypt"/>
        <entry key="passwordType" value="PasswordDigest"/>
        <entry key="passwordCallbackClass" value="br.client.middleware.callback.ClientPasswordCallback"/>
        <entry key="signaturePropFile" value="br\client\middleware\config\properties\cliente.properties"/>
        <entry key="decryptionPropFile" value="br\client\middleware\config\properties\cliente.properties"/>
        <entry key="encryptionKeyTransportAlgorithm" value="http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p"/>
        <entry key="signatureAlgorithm" value="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
      </map>
    </constructor-arg>
  </bean>

</beans>
```

Para assinatura e criptografia da mensagem SOAP no cliente também é utilizado um arquivo *properties*. O arquivo `cliente.properties` conforme apresentado no Quadro 15, contém informações do *KeyStore* utilizado para o armazenamento de chaves no cliente.

Quadro 15 - Configuração do arquivo `cliente.properties`

```
1 org.apache.ws.security.crypto.provider=org.apache.ws.security.components.crypto.Merlin
2 org.apache.ws.security.crypto.merlin.keystore.type=jks
3 org.apache.ws.security.crypto.merlin.keystore.password=spclient250
4 org.apache.ws.security.crypto.merlin.keystore.alias=sysuser
5 org.apache.ws.security.crypto.merlin.keystore.file=clientKeystore.jks
```

Para as operações de *callback* do *WS-Security client* foi utilizado a classe `ClientPasswordCallback` conforme apresentado no Quadro 16. Esta classe adiciona as credenciais do usuário à requisição para autenticação no servidor e adiciona as chaves do *KeyStore* obtida do arquivo `clientUserPassKeystore.key` e `serverUserPassKeystore.key` no *WS-Security* para validação da assinatura e criptografia no cliente. As operações de *callback* do *WS-Security* são realizados tanto nas operações de r*equest* quanto nas operações de *response* conforme solicitação realizada ao *web service*.

Quadro 16 - Classe `ClientPasswordCallback.java`

```
1 import br.client.middleware.config.Config;
2 import java.io.BufferedReader;
3 import java.io.File;
4 import java.io.FileReader;
5 import java.io.IOException;
6 import javax.security.auth.callback.Callback;
7 import javax.security.auth.callback.CallbackHandler;
8 import javax.security.auth.callback.UnsupportedCallbackException;
9 import org.apache.ws.security.WSPasswordCallback;
10
11 public class ClientPasswordCallback implements CallbackHandler {
12
13    private String userClient;
14    private String passwordClient;
15    private String userServer;
16    private String passwordServer;
17
18    @Override
19    public void handle(Callback[] callbacks) throws IOException, UnsupportedCallbackException {
20       try {
21          obterClientServerUserPassKeystore();
22       } catch (Exception ex) {
23          throw new SecurityException("Não foi possível realizar a autenticação do usuário");
24       }
25       boolean passwordInformado = false;
26       WSPasswordCallback pc;
27       for (Callback callback : callbacks) {
28          if (callback instanceof WSPasswordCallback) {
29             pc = (WSPasswordCallback) callback;
30             if (pc.getUsage() == WSPasswordCallback.USERNAME_TOKEN) {
31                if (userServer.equals(pc.getIdentifier())) {
32                   pc.setPassword(passwordServer);
```

Quadro 16 - Continuação classe `ClientPasswordCallback.java`

```
33                  passwordInformado = true;
34              } else {
35                  Config config = Config.getInstancie();
36                  pc.setIdentifier(config.getNmUsuario());
37                  pc.setPassword(config.getDsSenha());
38                  passwordInformado = true;
39              }
40          } else if (pc.getUsage() == WSPasswordCallback.DECRYPT
41                  || pc.getUsage() == WSPasswordCallback.SIGNATURE) {
42              if (userClient.equals(pc.getIdentifier())) {
43                  pc.setPassword(passwordClient);
44                  passwordInformado = true;
45              }
46          }
47      }
48    }
49    if (!passwordInformado) {
50      throw new SecurityException("Usuário/Senha inválido");
51    }
52  }
53
54  private void obterClientServerUserPassKeystore() throws Exception {
55    BufferedReader in = new BufferedReader(new FileReader(new File("clientUserPassKeystore.key")));
56    userClient = in.readLine();
57    userClient = userClient.substring(userClient.indexOf(":") + 1, userClient.length());
58    passwordClient = in.readLine();
59    passwordClient = passwordClient.substring(passwordClient.indexOf(":") + 1, passwordClient.length());
60    in.close();
61
62    in = new BufferedReader(new FileReader(new File("serverUserPassKeystore.key")));
63    userServer = in.readLine();
64    userServer = userServer.substring(userServer.indexOf(":") + 1, userServer.length());
65    passwordServer = in.readLine();
66    passwordServer = passwordServer.substring(passwordServer.indexOf(":") + 1, passwordServer.length());
67    in.close();
68  }
69 }
```

Para integração da aplicação com o *middleware* cliente para realização da comunicação de dados entre as aplicações foi criada a classe `MiddlewareWSS`, sendo disponibilizados métodos para o envio e recebimento de requisições. No Quadro 17 são apresentados os métodos da classe `MiddlewareWSS`, sendo que a aplicação poderá ter somente uma instância desta classe durante sua execução. A conexão do *middleware* com o serviço web é realizada através do método `configurarMiddleware()`. Para transformação de objetos em XML foi utilizado a biblioteca *XStream*.

Quadro 17 – Métodos da classe `MiddlewareWSS.java`

```
1   private MiddlewareWSS() {
2      this.config = Config.getInstancie();
3   }
4
5   public static MiddlewareWSS getInstancie() {
6      if (middlewareWSS == null) {
7         middlewareWSS = new MiddlewareWSS();
8      }
9      return middlewareWSS;
10  }
11
12  public void configurarMiddleware() {
13     ClassPathXmlApplicationContext context = new ClassPathXmlApplicationContext(new String[]{"br/client/middleware/config/xml/cxf-client.xml"});
14     this.port = (MiddWSInterface) context.getBean("middWSClient");
15     if (!this.dirRecebimentoArquivos.endsWith("\\")) {
16        this.dirRecebimentoArquivos += "\\";
17     }
18  }
19
20  public void setUser(String nmUsuario, String dsSenha) {
21     this.config.setNmUsuario(nmUsuario);
22     this.config.setDsSenha(DigestUtils.md5DigestAsHex(dsSenha.getBytes()));
23  }
24
25  public String enviarRequisicao(Lista registros, long nrSeqSistemaRequisicao, String cdRegra, String cdPermissao) throws Exception {
26
27     RequisicaoLista requisicao = new RequisicaoLista();
28     requisicao.setCdRegra(cdRegra);
29     requisicao.setCdPermissao(cdPermissao);
30     requisicao.setNmUsuario(config.getNmUsuario());
31     requisicao.setLista(registros);
32
33     XStream xStream = new XStream();
34     xStream.alias("requisicao", RequisicaoLista.class);
35     xStream.alias("lista", Lista.class);
36     xStream.alias("registro", Registro.class);
37
38     String xml = xStream.toXML(requisicao);
39
40     return port.enviarRequisicao(config.getNmUsuario(), nrSeqSistemaRequisicao, xml, cdRegra, cdPermissao);
41  }
42
43  public String enviarRequisicao(Registro registro, long nrSeqSistemaRequisicao, String cdRegra, String cdPermissao) throws Exception {
44
45     RequisicaoRegistro requisicao = new RequisicaoRegistro();
46     requisicao.setCdRegra(cdRegra);
47     requisicao.setCdPermissao(cdPermissao);
48     requisicao.setNmUsuario(config.getNmUsuario());
49     requisicao.setRegistro(registro);
50
51     XStream xStream = new XStream();
52     xStream.alias("requisicao", RequisicaoRegistro.class);
53     xStream.alias("registro", Registro.class);
54
55     String xml = xStream.toXML(requisicao);
56
57     return port.enviarRequisicao(config.getNmUsuario(), nrSeqSistemaRequisicao, xml, cdRegra, cdPermissao);
58  }
59
60  public String receberRequisicao() throws Exception_Exception {
61     return port.receberRequisicao(nrSeqSistema);
62  }
63
64  public String enviarArquivo(File arquivo, long nrSeqSistemaRequisicao, String cdRegra, String cdPermissao) throws Exception {
65
66     Arquivo arq = new Arquivo();
```

Quadro 17 – Continuação métodos da classe `MiddlewareWSS.java`

```
67      arq.setCdRegra(cdRegra);
68      arq.setCdPermissao(cdPermissao);
69      arq.setNmUsuario(config.getNmUsuario());
70      arq.setNmArquivo(arquivo.getName());
71
72      DataHandler dataHandler = new DataHandler(new FileDataSource(arquivo));
73      arq.setBinaryData(dataHandler);
74
75      return port.enviarArquivo(config.getNmUsuario(), nrSeqSistemaRequisicao, arq, cdRegra, cdPermissao);
76    }
77
78    public ArquivoDados receberArquivo() throws Exception {
79      Arquivo arq = port.receberArquivo(nrSeqSistema);
80
81      ArquivoDados arquivoDados = new ArquivoDados();
82      arquivoDados.setNmUsuario(arq.getNmUsuario());
83      arquivoDados.setCdRegra(arq.getCdRegra());
84      arquivoDados.setCdPermissao(arq.getCdPermissao());
85
86      DataHandler dataHandler = arq.getBinaryData();
87
88      String nmArquivo = arq.getNmArquivo();
89
90      String nome = nmArquivo.substring(0, nmArquivo.lastIndexOf("."));
91      String extencao = nmArquivo.substring(nmArquivo.lastIndexOf("."), nmArquivo.length());
92      File novoArquivo = new File(dirRecebimentoArquivos, nmArquivo);
93      int cont = 1;
94      while (novoArquivo.exists()) {
95        novoArquivo = new File(dirRecebimentoArquivos, nome + "_" + cont + extencao);
96        cont++;
97      }
98
99      FileOutputStream out = new FileOutputStream(novoArquivo);
100     dataHandler.writeTo(out);
101     out.flush();
102     out.close();
103
104     arquivoDados.setArquivo(novoArquivo);
105
106     removerArquivoServidor(arq.getDsDirArquivoServidor());
107
108     return arquivoDados;
109   }
110
111   private void removerArquivoServidor(String dsArquivo) {
112     try {
113       port.removerArquivoServidor(dsArquivo);
114     } catch (Exception ex) {}
115   }
116
117   public long getNrSeqSistema() {
118     return nrSeqSistema;
119   }
120
121   public void setNrSeqSistema(long nrSeqSistema) {
122     this.nrSeqSistema = nrSeqSistema;
123   }
124
125   public String getDirRecebimentoArquivos() {
126     return dirRecebimentoArquivos;
127   }
128
129   public void setDirRecebimentoArquivos(String dirRecebimentoArquivos) {
130     this.dirRecebimentoArquivos = dirRecebimentoArquivos;
131   }
```

### 3.4.2 Operacionalidade da implementação

Esta seção apresenta a operacionalidade do *middleware* desenvolvido. Para realizar a demonstração do *middleware* foram desenvolvidas duas aplicações de testes onde demonstra-se o funcionamento do *middleware* e elencam-se os passos que devem ser seguidos para utilizá-lo.

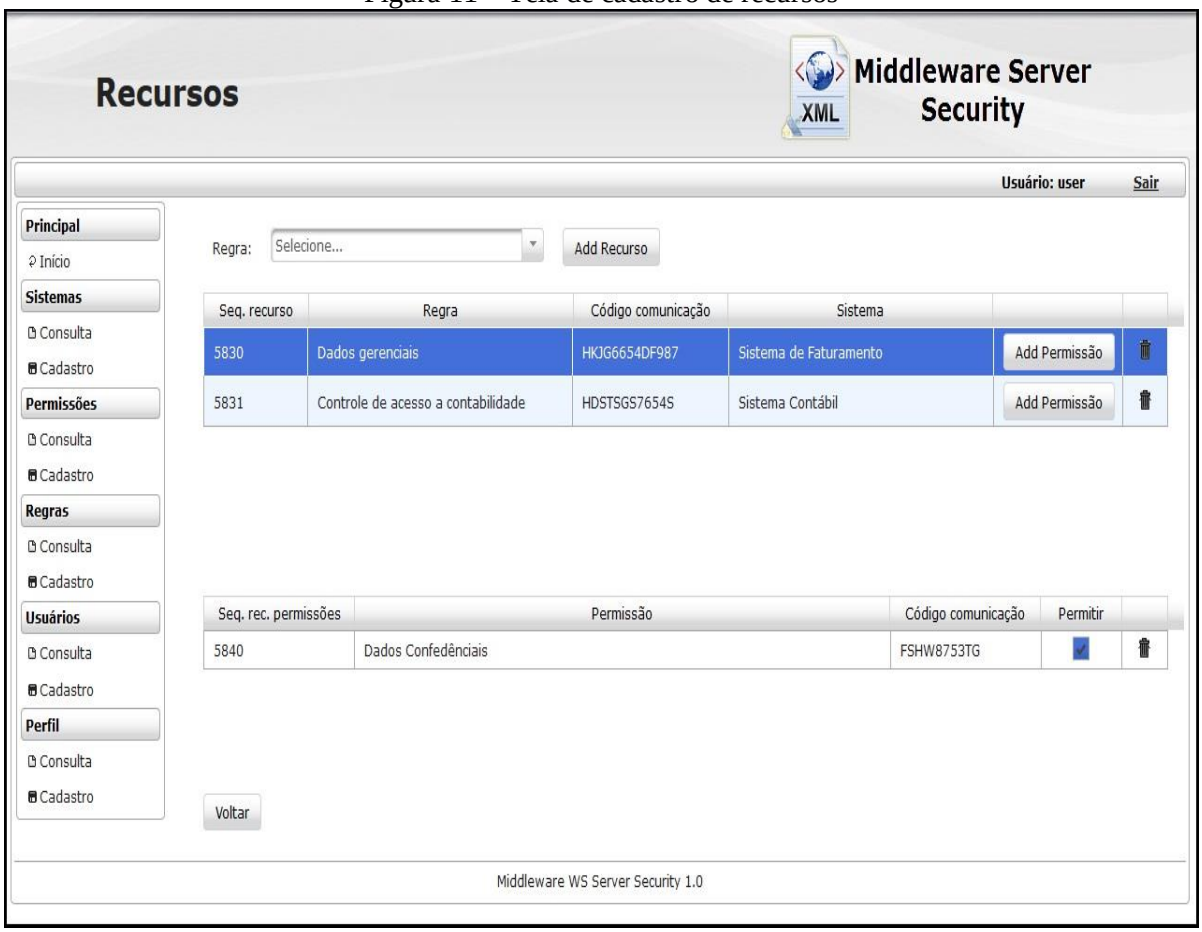
Considerando um cenário hipotético tem-se um sistema distribuído e um servidor separado o qual possui o serviço web configurado. Sendo que, o módulo de faturamento deseja enviar informações e arquivos confidênciais para o módulo contábil de forma segura.

Primeiramente é necessário cadastrar os usuários dos módulos faturamento e contábil que realizarão a troca de informações no servidor web. Apenas administradores com a permissão ‘administrador’ cadastrados também no sistema web podem realizar os cadastros em geral. Os usuários que possuem a permissão de ‘usuário’ possuem acesso somente as páginas de consultas. Estes podem alterar somente sua senha através da tela de consulta. No servidor web também é necessário realizar o cadastramento das aplicações, regras e permissões de acesso. Na tela de cadastro de regras é necessário informar um código de comunicação. Este código é informado na comunicação de dados entre as aplicações, pois quando a aplicação destinatária receber uma requisição, a mesma possa identificar através deste código qual foi a aplicação que enviou a requisição. No cadastro de permissões também é necessário cadastrar um código de comunicação, pois este código indica qual o objetivo da requisição, se é enviar arquivo, consultar dados, exportar informações, etc.

As regras de comunicação são aplicadas a usuários e/ou perfis, sendo que, o sistema permite a criação de perfis. Cada perfil poderá ter vários usuários vinculados. Quando uma regra é criada para um perfil são atribuídos permissões a esta regra. Todos os usuários vinculados ao perfil poderão realizar somente este tipo de comunicação definido. Se para um usuário em específico é necessário que haja uma regra separada, neste caso, deverá ser criada esta regra contendo permissões específicas e vinculado somente para este usuário.

Para vincular as regras aos usuários ou perfis, existe um botão em suas devidas telas de cadastro chamado recurso onde essas regras poderão ser atribuídas. A Figura 11 contém a vinculação de uma regra a um perfil, onde, através do botão ‘add recurso’ é possível adicionar novas regras. O grid regras contém o botão ‘add permissão’ onde é possível adicionar permissões às regras. Já no grid permissões há um *check* onde é possível autorizar, ou seja permitir que o usuário realize esta operação. Todas as regras definidas neste cadastro são consistidos no servidor web disponibilizado.

Figura 11 – Tela de cadastro de recursos




Para realizar a comunicação de dados entre as aplicações é necessário que o *web service* esteja ativo em um servidor Tomcat. Neste servidor deverá ser criado uma pasta chamada `KeyStore` no diretório C da máquina, Nesta pasta deve ser adicionado manualmente o arquivo `serverKeystore.jks`, utilizado pelo *WS-Security*.

Para utilização do *middleware*, o mesmo deve ser adicionado como uma biblioteca ao projeto contendo suas dependências da pasta lib. Também devem ser adicionados ao projeto os arquivos `clientKeystore.jks`, `clientUserPassKeystore.key` e `serverUserPassKeystore.key` sendo utilizados pelo *WS-Security Client*. Todos os arquivos deverão estar juntos ao projeto da aplicação principal.

Para efetuar a comunicação de dados entre a aplicação faturamento e contabilidade, a chamada aos métodos do *middleware* devem ocorrer conforme apresentado no Quadro 18. No exemplo, a aplicação faturamento enviará dados com base nas regras cadastradas para a aplicação contabilidade. Sendo que, com a utilização do *middleware* a aplicação não presisa se preocupar com os detalhes da implementação do *WS-Security*.

Quadro 18 – Configuração da aplicação de teste faturamento

```
1
2 public class Faturamento {
3
4   public static void main(String[] args) {
5     try {
6
7       //Criação de uma única instância do middleware durante toda execução
8       MiddlewareWSS middlewareWSS = MiddlewareWSS.getInstancie();
9
10      //Configurações iniciais
11      middlewareWSS.setNrSeqSistema(5027);//Sequência da aplicação Faturamento cadastrada
12      middlewareWSS.configurarMiddleware();//Cria uma conexão com o web service
13
14      //Credenciais do usuário que irá realizar a comunicação
15      middlewareWSS.setUser("crhobus", "12345678");
16
17      //Criação de uma lista contendo 2 registros
18
19      Lista lista = new Lista();
20
21      Registro registro = new Registro();
22      registro.addInfo("NR_SEQUENCIA", 1);
23      registro.addInfo("VL_TOTAL", 1300.87);
24      lista.addRegistro(registro);
25
26      registro = new Registro();
27      registro.addInfo("NR_SEQUENCIA", 2);
28      registro.addInfo("VL_TOTAL", 7654.7);
29      lista.addRegistro(registro);
30
31      //Envia a requisição - informando o sistema destinatário, o código da regra e o código da permissão
32      System.out.println(middlewareWSS.enviarRequisicao(lista, 5028, "HKJG6654DF987", "FSHW8753TG"));
33
34      System.out.println("");
35
36      //Envia um arquivo - informando o sistema destinatário, o código da regra e o código da permissão
37      System.out.println(middlewareWSS.enviarArquivo(new File("C:\\Arquivos\\Análise de contas.pdf"), 5028, "JHCDD7E34GI", "SIS6353GDJ8"));
38
39      System.out.println("");
40
41    } catch (Exception ex) {
42      ex.printStackTrace();
43    }
44  }
45 }
46
```

Inicialmente a aplicação faturamento cria uma única instância do *middleware* que será utilizada durante toda a sua execução. Após é informado o código do sistema cadastrado na interface web. A aplicação realiza uma chamada ao método `configurarMiddleware()`, onde o *middleware* cria uma conexão com o servidor. É informado o usuário com a senha no qual deseja realizar a comunicação. Após a aplicação cria uma requisição ou um arquivo sendo o mesmo informado ao *middleware* juntamente com o código do sistema destinatário, sendo também informado o código da regra e a permissão utilizada na comunicação, para que o *middleware* possa enviar a requisição a outra aplicação.

Para o recebimento dos dados pela aplicação contabilidade, a mesma deve ser configurada conforme o Quadro 19.

Inicialmente a aplicação contabilidade cria uma única instância do *middleware* que será utilizada durante toda a sua execução. Após é informado o código do sistema cadastrado na interface web e também é informado o diretório para recebimento de arquivos. A aplicação realiza uma chamada ao método `configurarMiddleware()`, onde o *middleware* cria uma conexão com o servidor. É informado o usuário com a senha no qual deseja realizar a comunicação. Após a aplicação solicita ao *middleware* através do método `receberRequisicao()` ou `receberArquivo()` para o recebimento dos dados no qual foram enviados por outras aplicações.

Quadro 19 – Configuração da aplicação de teste contabilidade

```
1
2 public class Contabilidade {
3
4   public static void main(String[] args) {
5     try {
6
7       //Criação de uma única instância do middleware durante toda execução
8       MiddlewareWSS middlewareWSS = MiddlewareWSS.getInstancie();
9
10      //Configurações iniciais
11      middlewareWSS.setNrSeqSistema(5028);//Sequência da aplicação Contabilidade cadastrada
12      middlewareWSS.setDirRecebimentoArquivos("C:\\Arquivos\\Contabilidade\\");//Diretório para recebimento de arquivos
13      middlewareWSS.configurarMiddleware();//Cria uma conexão com o web service
14
15      //Credenciais do usuário que irá realizar a comunicação
16      middlewareWSS.setUser("usertest", "653683455");
17
18      //Recebe uma requisição do servidor no formato xml
19      System.out.println(middlewareWSS.receberRequisicao());
20
21      //Recebe um arquivo do servidor
22      ArquivoDados arq = middlewareWSS.receberArquivo();
23      System.out.println(arq.getNmUsuario());
24      System.out.println(arq.getCdRegra());
25      System.out.println(arq.getCdPermissao());
26      System.out.println(arq.getArquivo().getPath());
27
28    } catch (Exception ex) {
29      ex.printStackTrace();
30    }
31  }
32 }
33
```

## 3.5 RESULTADOS E DISCUSSÃO

Os resultados obtidos confirmam os objetivos propostos em relação a segurança dos dados entre sistemas distribuídos utilizando as especificações do *WS-Security*. Foi evidenciado que com a utilização da especificação do *WS-Security* é possível as aplicações

trocarem informações e dados confidênciais através de *web services* garantindo a confidencialidade, integridade e autencidade dos dados.

Através do serviço de segurança disponibilizado, qualquer aplicação já desenvolvida pode incorporar funções de criptografia, autenticação, assinatura digital e permissões de acesso aos dados sem a necessidade de maiores implementações, uma vez que a implementação é disponibilizada pelos serviços de segurança, reforçando o conceito inicial de ser um *middleware* e uma parte de um sistema distribuído.

A partir do módulo web é possível registrar as aplicações que utilizarão o *middleware*, criar regras de comunicação e permissões de acesso aos dados, podendo ser criado para um perfil contendo um grupo de usuários ou ser criado para um usuário em específico. Todas as regras defnidas no sistema web é utilizado no controle de fluxo de dados do *web service*.

Como resultado final, foi criado o *middeware* para integração com outras aplicações, com a finalidade de disponibilizar serviços de segurança à aplicações que não possuem estas funcionalidades. A maior vantagem deste *middleware* é fazer com que os desenvolvedores não precisem desenvolver rotinas de segurança em suas aplicações, podendo utilizar as rotinas já definidas.

Com relação aos trabalhos correlatos, os mesmos apresentam soluções baseadas em *web services*, as quais foram estudadas para verificar o desenvolvimento do *middleware* de segurança. Os trabalhos de Hansen e Pinto (2003), Martins, Rocha e Henriques (2003) e Silva (2004) apresentam ainda algumas soluções para segurança em sistemas distribuídos, sendo que, diferentemente do trabalho desenvolvido, limitam-se apenas ao estudo, sem o desenvolvimento ou utilização das técnicas descritas. A Figura 12 demonstra as diferenças entre os trabalhos correlatos e o trabalho desenvolvido, apresentando como principal diferencial a utilização tanto da especificação de *web services* como de especificações de segurança aplicando regras na comunicação.

Figura 12 - Diferenças entre trabalhos correlatos e projeto desenvolvido

| | Trabalho 1 | Trabalho 2 | Trabalho 3 | Projeto |
|---|---|---|---|---|
| Web services | X | X | X | X |
| Segurança | | X | X | X |
| Regras de comunicação | | | | X |

# 4 CONCLUSÕES

Cada vez mais as empresas estão operando de forma distribuída, trocando informações pela rede, ou seja, um único sistema sendo operado por vários usuários, utilizando computadores diferentes na rede. Estes podem estar em um mesmo ambiente ou separados geograficamente, apenas trocando informações pela rede e compartilhando recursos como hardware, software e dados de forma transparente para o usuário. Com isso torna-se cada vez mais necessário a implementação de um mecanismo de segurança para que os dados e informações importantes que trafegam na rede não sejam interceptados por terceiros.

Este trabalho apresentou o desenvolvimento da especificação do *WS-Security* na comunicação de dados entre as aplicações, utilizando regras e permissões. O *framework* Apache CXF utilizado na implementação do projeto se demonstrou muito flexível e extensível para diversas situações e necessidades. O projeto possui uma limitação, sendo que na comunicação de dados utilizando arquivos é possível apenas utilizar um arquivo por vez na transfêrencia de dados de uma aplicação a outra. Esta limitação foi implementada devido a performace do *middleware* e do trafego na rede.

A API Apache CXF em Java foi escolhida como base para implementação da especificação do *WS-Security*, se demonstrou bem intuitiva. Comparada com outras implementações, a API Apache CXF é bem mais simples e leve, principalmente porque utiliza interceptadores *spring* em sua implementação, e por estes motivos foi escolhida como base para o desenvolvimento do projeto.

Em relação aos trabalhos correlatos apresentados pode-se afirmar que a diferencia deste trabalho com os demais é que este poderá integrar-se a qualquer aplicação para efetuar a comunicação de dados e informações via web de forma segura e de modo transparente em relação à camada de aplicação, pois a aplicação irá precisar apenas enviar e receber os dados do *middleware* com base nas regras registradas, não se preocupando da forma como estão sendo implementadas as questões de segurança e camadas posteriores de rede.

## 4.1 EXTENSÕES

Durante a especificação e implementação do sistema web surgiram algumas sugestões que podem ser agregadas ou melhoradas ao desenvolvido deste trabalho.

Na implementação de criptografia e assinatura do *WS-Security client*, parte do *middleware*, foi utilizado um único *KeyStore* para o armazenamento de chaves de todos os *middlewares* utilizado por todas as aplicações envolvidas, sendo proposto para extensões a implementação de um *KeyStore* para cada *middleware* separadamente.

Outra sugestão é implementar no projeto as extensões do *WS-Security*, sendo elas:

a) *WS-Policy*: definição de recursos e restrições;
b) *WS-Trust*: definição de um modelo de confiança;
c) *WS-Privacy*: define de que forma os *web services* serão implementados;
d) *WS-Secure Conversation*: define como autenticar e gerenciar troca de mensagens;
e) *WS-Federation*: define o gerenciamento de relacionamentos em ambientes heterogêneos;
f) *WS-Authorization*: define a forma de administração dos dados pelos web services.

# REFERÊNCIAS BIBLIOGRÁFICAS


ATKINSON, Bob. et al. **Web services security.** [S.l.], 2002. Disponível em: <http://www.verisign.com/wss/wss.pdf>. Acesso em: 24 set. 2013.

BURNETT, Steve; PAINE, Stephen. **Criptografia e segurança:** o guia oficial RSA. Rio de Janeiro: Elsevier, 2002.

CHAPPELL, David; JEWEL, Tyler. **Java web services.** [S.l.], 2002. Disponível em: <http://oreilly.com/catalog/javawebserv/chapter/ch06.html>. Acesso em: 09 out. 2013.

CUNHA, Daniel P. et al. **Detecção de intrusão.** Criciúma, [2006]. Universidade do Extremo Sul Catarinense - UNESC.

DUARTE FILHO, Nemésio F. et al. **Implantação do middleware Globus Toolkit 4 para aplicações em ambientes de Grid.** Lavras, 2008. Disponível em: <http://www.bcc.ufla.br/monografias/2008/Implantacao_do_middleware_Globus_Toolkit_4_para_aplicacoes_em_ambientes_de_grid.pdf>. Acesso em: 18 set. 2012.

GALBRAITH, Ben et al. **Professional web services security.** Birmingham: Wrox Press, 2002.

HANSEN, Roseli; PINTO, Sérgio C. **Construindo ambientes de educação baseada na web através de web wervices educacionais.** Canoas, 2003. Disponível em: <http://www.nce.ufrj.br/sbie2003/publicacoes/paper07.pdf>. Acesso em: 17 set. 2012.

IKEMATU, Ricardo S. et al. **Middleware:** a market overview. [S.l.], 2003. Disponível em: <http://www.pr.gov.br/batebyte/edicoes/1996/bb54/seminar.htm>. Acesso em: 04 nov. 2012.

KUROSE, James; ROSS, Keith. **Redes de computadores e a internet.** 3. ed. São Paulo: Pearson, 2005.

MARTINS, Ricardo; ROCHA, Jorge; HENRIQUES, Pedro. **Segurança dos web services no comércio eletrônico móvel.** Braga, 2003. Disponível em: <http://www.di.fc.ul.pt/~paa/projects/conferences/coopmedia2003/10.pdf>. Aceso em: 04 nov. 2012.

RODRIGUES, Gustavo V. et al. **Disponibilização de serviços de segurança para sistemas distribuídos através de Web Services.** Blumenau, 2007. Disponível em: <http://campeche.inf.furb.br/tccs/2007-II/TCC2007-2-17-VF-GustavoVRodrigues.pdf>. Acesso em: 15 set. 2012.

ROSENBERG, Jothy; REMY, David. **Securing web services with WS-Security.** [S.l.], 2004. Disponível em: <http://www.abebooks.com/book-search/author/remy-david-rosenberg-jothy/>. Acesso em: 23 out. 2012.

SAUDATE, Alexandre. **SOA aplicado:** integrando com web services e além. [S.l.], 2010. Disponível em: <http://www.casadocodigo.com.br/products/livro-soa-webservices>. Acesso em: 20 out. 2013.

SILVA, Jandson A. et al. **Segurança em web services.** [S. l.], 2004. Disponível em: <http://araticum.infonet.com.br/andres/apresentacoes/artigos/SegurancaWebServices%20-%20JandsonAlmeidaSilva.pdf>. Acesso em: 04 nov. 2012.

SILVA, Lino S. **Public Key Infrastructure - PKI:** conheça a infra-estrutura de chaves públicas e a certificação digital. São Paulo: Novatec, 2004.

STALLINGS, William. **Redes e sistemas de comunicação de dados.** 5. ed. Rio de Janeiro: Campus, 2005.

WATHIER, Adair J. et al. **Segurança em web services com WS-Security.** Porto Alegre, 2005. Disponível em:<http://saloon.inf.ufrgs.br/twikidata/Docs/OnlineDoc20051217200943/SegurancaWebService_WSSecurity.pdf>. Acesso em: 5 out. 2013.

WORLD WIDE WEB CONSORTIUM. **Web services activity.** [S.l.], 2007. Disponível em: <http://www.w3.org/2002/ws/>. Acesso em: 17 set. 2012.

# APÊNDICE A – Descrição dos principais casos de uso

Neste apêndice são apresentados em detalhes os dois principais casos de uso presentes no diagrama da Figura 4, sendo: UC08 - enviar requisição e UC09 - receber requisição.

O caso de uso `UC08`, descrito em detalhes no Quadro 20, representa o envio de dados da aplicação ao *middleware*, na qual encaminhará a aplicação destinatária.

Quadro 20 - Caso de uso `UC08` em detalhes

| UC08 - Enviar requisição | |
|---|---|
| Descrição | O *middleware* enviará os dados para aplicação destinatária com base nas regras cadastradas. |
| Pré condição | O servidor deverá estar ativo com o serviço web disponível. |
| Cenário principal | 1. A aplicação cria uma requisição;<br>2. A aplicação informa ao *middleware* o usuário, o sistema destinatário e a regra de comunicação;<br>3. A aplicação envia a requisição ao *middleware*;<br>4. O *middleware* implementa as funcionalidades do *WS-Security* na requisição;<br>5. O *middleware* envia a requisição ao servidor;<br>6. O servidor processa a requisição verificando as regras de comunicação;<br>7. O servidor envia um retorno ao *middleware* informando o envio da requição;<br>8. O *middleware* retorna a informação a aplicação. |
| Exceção | Na etapa 6, caso o usuário e senha forem inválidos ou o usuário não tem permissão para efetuar a comunicação o servidor lança uma exceção informando-o sobre o mesmo. |
| Pós condição | A requisição foi enviada a outra aplicação com base nas regras cadastradas. |

O caso de uso UC09, descrito em detalhes no quadro 21, representa o rebebimento de uma requisição através do *middleware* encaminhando o mesmo a aplicação novamente.

Quadro 21 - Caso de uso UC09 em detalhes

| UC09 – Receber requisição | |
|---|---|
| Descrição | O *middleware* irá receber os dados do servidor e retornar a aplicação. |
| Pré condição | O servidor deverá estar ativo com o serviço web disponível. |
| Cenário principal | 1. A aplicação solicita ao *middleware* a requisição enviada por outra aplicação ao servidor;<br>2. A aplicação informa ao *middleware* o usuário que deseja realizar a comunicação;<br>3. O *middleware* solicita a requisição ao servidor;<br>4. O servidor implementa as funcionalidades do *WS-Security* na requisição;<br>5. O servidor envia a requisição ao *middleware* que realizou a solicitação;<br>6. O *middleware* retorna a requisição a aplicação. |
| Exceção | Na etapa 4, caso o usuário e senha forem inválidos o servidor lança uma exceção informando-o sobre o mesmo. |
| Pós condição | A aplicação recebe a requisição que foi enviada por outra aplicação com base em regras, sendo que a aplicação que recebeu a requisição irá se idêntificar através destas regras. |