**UNIVERSIDADE FEDERAL DE SERGIPE**

**CAMPUS ALBERTO CARVALHO**

**NÚCLEO DE SISTEMAS DE INFORMAÇÃO**

**ISRAEL MENESES SANTOS**

# ANÁLISE DE ESTRATÉGIAS PARA IMPLANTAÇÃO DE SEGURANÇA EM ARQUITETURAS ORIENTADAS A SERVIÇOS

**ITABAIANA**

**2010**

**UNIVERSIDADE FEDERAL DE SERGIPE**
**CAMPUS ALBERTO CARVALHO**
**NÚCLEO DE SISTEMAS DE INFORMAÇÃO**

**ISRAEL MENESES SANTOS**

# ANÁLISE DE ESTRATÉGIAS PARA IMPLANTAÇÃO DE SEGURANÇA EM ARQUITETURAS ORIENTADAS A SERVIÇOS

Trabalho de Conclusão de Curso submetido ao Núcleo de Sistemas de Informação da Universidade Federal de Sergipe como requisito parcial para a obtenção do título de Bacharel em Sistemas de Informação

Orientador: Marcos Barbosa Dósea

ITABAIANA
**2010**

Santos, Israel Meneses.

Análise de Estratégias para Implantação de Segurança em Arquiteturas Orientadas a Serviço / Israel Meneses Santos – Itabaiana: UFS, 2010.

70f. ; 99 cm (indica o tamanho)

Trabalho de Conclusão de Curso (graduação) – Universidade Federal de Sergipe, Curso de Sistemas de Informa, 2010.

1. Assunto. 2. Área de Concentração - TCC. 3. Curso. I. Título.

**Israel Meneses Santos**

# ANÁLISE DE ESTRATÉGIAS PARA SEGURANÇA EM ARQUITETURAS ORIENTADAS A SERVIÇOS

Trabalho de Conclusão de Curso submetido ao corpo docente do Núcleo de Sistemas de Informação da Universidade Federal de Sergipe (NSIITA/UFS) como parte dos requisitos para obtenção do grau de Bacharel em Sistemas de Informação.

Itabaiana, (20de dezembro e 2010 da aprovação).

**BANCA EXAMINADORA:**

_____

**Prof.(a) Marcos Barbosa Dósea, Msc**
**Orientador**
**Universidade Federal de Sergipe**

_____

**Prof.(a) Methanias Colaço Junior, Msc**
**Universidade Federal de Sergipe**

_____

**Prof.(a) Andrés Ignácio Martínez Menéndez, Msc**
**Universidade Federal de Sergipe**

**Análise de Estratégias para Implantação de Segurança em Arquiteturas Orientadas às Serviços**. 2010. Trabalho de Conclusão de Curso – Curso de Bacharelado Sistemas de Informação, Núcleo de Sistemas de Informação, Universidade Federal de Sergipe, Itabaiana, 2010.

## RESUMO

*O desenvolvimento de aplicações nas mais diversas tecnologias dentro das organizações criou a necessidade de definir mecanismos para a comunicação e integração destas aplicações de forma segura. A arquitetura Orientada a Serviços (SOA – Service Oriented Architecture) tem se destacado em relação às outras técnicas de integração de aplicações pelas suas características, como interoperabilidade, escalonamento e reutilização de software. A tecnologia mais utilizada atualmente para implantação de SOA são os Web Services, por sua facilidade de implementação, interoperabilidade e uso de padrões abertos. Eles são uma das principais escolhas para a troca de informações entre diferentes tipos de aplicações. Porém, o uso deles na troca de informações exige que a questão da segurança seja muito bem pensada antes da implementação das aplicações. Uma decisão equivocada pode comprometer a autenticidade, confidencialidade, integridade e disponibilidade dos dados transmitidos. Outro problema é que o excesso de mecanismos de segurança pode comprometer o desempenho e disponibilidade da aplicação. Esse trabalho tem como objetivo avaliar diversas abordagens que podem ser utilizadas para garantir a segurança de em aplicações desenvolvidas utilizando uma arquitetura orientada a serviços. Os resultados obtidos nas análises foram utilizados para definir uma solução de segurança para o Centro de Processamento de Dados da Universidade Federal de Sergipe. O objetivo é disponibilizar de forma segura serviços e dados atualizados para vários setores e departamentos da instituição.*

**Palavras-chave:** SOA *Web Services*, Segurança

**ABSTRACT**

*The development of applications in different technologies within organizations has created the need to establish mechanisms for communication and integration of these applications securely. Service Oriented Architecture (SOA - Service Oriented Architecture) has been outstanding in relation to other techniques for application integration because of their characteristics, such as interoperability, scalability and reuse of software. The most widely used technology for implementing SOA is Web Services, for its ease of implementation, interoperability and use of open standards. They are the top choice for exchanging information between different applications. But their use in the exchange of information requires that the safety issue is very well thought out before implementation of applications. A wrong decision could compromise the authenticity, confidentiality, integrity and availability of data transmitted. Another problem is that too much security mechanisms can impair performance and application availability. This study aims to evaluate various approaches that can be used to ensure the security of applications developed using a service-oriented architecture. The analysis results were used to define a security solution for the Data Processing Center, Federal University of Sergipe. The goal is to provide services safely and updated data for various sectors and departments of the institution.*

Keywords: SOA, Web Services, Security

**Keywords**: SOA, Web Services, Security

# Lista de Figuras

# SUMÁRIO

# **1-** INTRODUÇÃO

Integração de aplicações tem como objetivo possibilitar a troca de informações entre sistemas heterogêneos. EAI (*Enterprise Application Integration*) é uma referência aos meios computacionais e aos princípios de arquitetura de sistemas utilizados no processo de Integração de Aplicações Corporativas. EAI é a criação de uma nova estratégia de soluções de negócio por integrar a funcionalidade de aplicações existentes na empresa, aplicações de pacotes comerciais e novas aplicações usando um *middleware* comum. *Middlewares* são softwares independentes da aplicação que servem como mediadores entre as aplicações. [1]

Atualmente um dos mecanismos mais populares para integração de aplicações é a utilização da Arquitetura Orientada a Serviços (*Service Oriented Architecture* - SOA) que fornece uma maneira flexível para o desenvolvimento de soluções de integração entre sistemas. Além de algumas outras características, como interoperabilidade, escalonamento e reutilização de software [1]. SOA, é um estilo arquitetural onde as funcionalidades implementadas pelas aplicações devem ser disponibilizadas na forma de serviços [4].

Devido ao grande número de sistemas que são desenvolvidos dentro das organizações e à necessidade de integração entre os mesmos, é indispensável à definição de uma metodologia de desenvolvimento de software que permita integrá-los de maneira simples e flexível. A arquitetura SOA é uma das opções para a criação de soluções para integração de sistemas [3].

## 1.1 O Problema

Diversas organizações pelo mundo enfrentam hoje o desafio de integrar informações que foram registradas ao longo dos anos por diversos sistemas nas mais diferentes tecnologias. A arquitetura orientada a serviço tem se mostrado uma solução promissora e atualmente é uma das propostas mais utilizadas e difundidas para integração de aplicações.

Entretanto disponibilizar essas informações garantindo a segurança na manipulação desses dados é um grande desafio. Deve-se definir entre as diversas abordagens, aquela que melhor se adéqua ao contexto e às necessidades de segurança e desempenho da aplicação.

O Centro de Processamento de Dados da Universidade Federal de Sergipe (UFS) se enquadra nesse contexto por possuir uma constante demanda de acesso às informações disponíveis nos bancos de dados que administra pelos diversos departamentos da instituição.

Normalmente essas demandas objetivam a atualização de bancos de dados locais que são utilizados por aplicações desenvolvidas pelos próprios departamentos. Atualmente essas solicitações são atendidas, mas como a segurança e o controle dessas informações é um fator muito importante, as aplicações não possuem acesso direto a base dados, gerando uma constante demanda para os técnicos do Centro de Processamento de Dados da UFS, que precisam atender constantemente solicitações dos departamentos por dados mais atualizados.

## 1.2 Solução Proposta

Este trabalho tem como objetivo analisar diversas abordagens para implementação de segurança em arquiteturas orientadas a serviço com o uso de *Web Services.* A solução deve balancear as necessidades de segurança e disponibilidade dos serviços e utilizar primariamente ferramentas gratuitas. As principais técnicas analisadas se baseiam em mecanismos de segurança no nível de mensagem e segurança no nível de transporte**.**

Para essa análise, foi criada uma aplicação para testes dividida em cinco versões com características de segurança diferentes. A primeira versão caracteriza-se por não possuir nenhum mecanismo de segurança implementado; a segunda trás um mecanismo de segurança no nível de transporte; a terceira versão agrega uma forma de autenticação à segurança no transporte; a quarta versão tem a característica de segurança na mensagem; e a quinta mostra como garantir a autenticidade da mensagem. Baseado nessas análises, como estudo de caso inicial, é discutida a solução mais adequada para resolver o problema identificado no Centro de Processamento de Dados da Universidade Federal de Sergipe.

## 1.3 Contribuições

As principais contribuições desse trabalho são:

- A implementação de uma aplicação utilizando a arquitetura orientada a serviços dividida em cinco versões, cada uma usando mecanismos de segurança diferentes.
- Análise qualitativa dessas versões em relação às necessidades de segurança, desempenho e disponibilidade da solução.

- Utilização das análises obtidas para definição de uma solução para ser implantada no Centro de Processamento de Dados da Universidade Federal de Sergipe.

## 1.4 Organização desse Trabalho

O texto referente a este trabalho encontra-se dividido da seguinte forma*:*

- No Capítulo 2 é feito uma abordagem dos assuntos e de tecnologias referentes ao objetivo deste trabalho;
- No Capítulo 3 é mostrado o desenvolvimento de uma aplicação, devida em versões que demonstram diferentes mecanismos de segurança no uso de *Web Services.*
- No Capítulo 4 é feita analise de uma solução para o problema do CPD*;*
- No Capítulo 5 serão apresentadas todas as considerações e conclusões sobre este trabalho.

# 2. FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são abordados os conceitos fundamentais para o entendimento desse trabalho. Na Seção 2.1 é explicado o conceito de *Enterprise Application Integration* (EAI), necessários para o entendimento das questões relacionadas à integração de aplicações. Em seguida, na Seção 2.2 é introduzido o conceito de *Service Oriented Architecture* (SOA), uma das arquiteturas que podem ser utilizadas para integração de aplicações.Na Seção 2.3 são mostrados o conceito e funcionamento de *Web Services*, justificando porque a tecnologia é a mais utilizada para implementação de sistema na arquitetura SOA, por fim na Seção 2.4 são detalhados alguns conceitos em relação a segurança na Arquitetura SOA.

## 2.1- *Enterprise Application Integration* (EAI)

As organizações tem se esforçado por muito tempo para encontrar a melhor maneira para fazer com que suas aplicações trabalhem em conjunto. Assim elas desenvolviam suas próprias soluções para integração de aplicações. Mas muitas empresas criam produtos de EAI e sistemas para integração de aplicações.

*Enterprise Application Integration* (EAI) é a criação de novas soluções de negócios, combinando a funcionalidade das aplicações existentes de uma empresa comercial, pacotes de aplicativos, e um novo código utilizando um *middleware* comum. [5]

Para entender os principais mecanismos para integração de aplicação, a subseção 2.1.1 mostra os diferentes tipos de integração existentes e a subseção 2.1.2 explica a arquitetura usada na integração para a EAI.

### 2.1.1- Tipos de Integração

A principal meta do EAI é permitir que uma organização integre diversas aplicações de forma rápida, fácil e com redução de acoplamento.

Segundo Ruh (2001), a integração pode ocorrer em três pontos de uma aplicação: nas camadas de apresentação, funcional ou de dados.

A integração na apresentação permite a integração de um novo software através das interfaces de apresentações existentes do software legado. Isso normalmente é usado para criar uma nova interface, mas podem ser utilizadas para integrar com outras aplicações.

Um modelo de integração funcional permite a integração de software com a finalidade de invocar as funcionalidades existentes de outras aplicações novas ou existentes. A integração é feita através de interfaces para o software.

Um modelo de integração de dados permite a integração de software através do acesso aos dados que são criados, gerenciados e armazenados pelo software em geral para afim de reutilização ou sincronizar dados entre aplicativos.

### 2.1.2- Arquitetura de EAI

Uma arquitetura de EAI é uma combinação de tecnologias reunidas em uma maneira estruturada. Os blocos que formam a estrutura da arquitetura EAI são:

- Modelos de Comunicação
- Métodos de integração
- *Middleware*
- Serviços

Estes quatro blocos de construção devem existir e ser interligados em uma única arquitetura. Essa metodologia é necessária para ajudar a construir uma solução e identificar quais partes da arquitetura são mais adequadas para o problema [5]. As subseções a seguir explicam em detalhes o funcionamento de cada um desses componentes.

#### 2.1.2.1 - Modelos de Comunicação

As duas escolhas básicas para um modelo de comunicação são comunicação síncrona e comunicação assíncrona [5]. Na comunicação síncrona é necessário que o remetente espere por uma resposta da requisição para continuar com o processamento. Na comunicação assíncrona não existe essa espera após enviar uma mensagem o remetente continua com o processamento.

A comunicação síncrona requer que exista uma coordenação entre a troca de mensagens e o processamento nos remetentes e receptores. Esse sincronismo entre mensagem e processamento gera um forte acoplamento, no caso da comunicação entre dois sistemas isso que um sistema está dependente do outro, e terá que esperar uma resposta para suas mensagens antes de continuar com o processamento.

Esse tipo de comunicação é usado principalmente em sistemas onde o remetente necessita de uma informação ou notificação que processada por outros sistemas e com resposta imediata. A comunicação síncrona requer uma infra-estrutura confiável, pois um sistema pode ficar esperando por uma resposta que foi perdida, assim ele pode ficar enviando novas requisições e esperando a resposta, isso pode gerar uma queda de desempenho.

Existem algumas regras para a coordenação das mensagens, e elas irão depender do tipo de comunicação síncrona usada, mas existem três tipos que são os mais populares [5].

- Pedido-Resposta
- Sentido Único
- Verificação
- Pedido-Resposta

Pedido-Resposta é o padrão básico na comunicação síncrona, uma aplicação envia uma requisição para uma segunda aplicação e fica em modo de espera, a segunda aplicação processa a requisição da primeira aplicação gera uma resposta, essa resposta é enviada de volta à primeira aplicação, ao receber a resposta ela continua a executar suas funções normalmente.

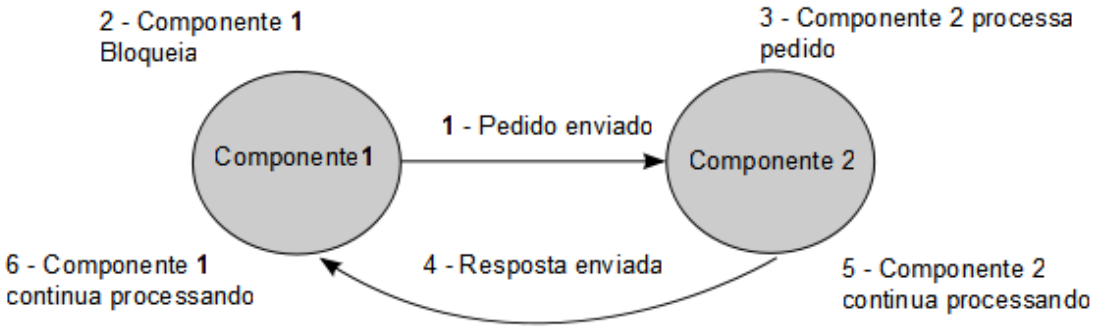


Figura 1 – Pedido-Resposta

A Figura 1 mostra o processo de comunicação usando o padrão Pedido-Resposta, no exemplo, no primeiro passo o componente 1 envia um pedido ao componente 2, no passo 2 o componente 1 fica bloqueado até receber a resposta do pedido, em seguida no terceiro passo da imagem, o componente 2 começa processar o pedido, após concluir o processamento do pedido o componente 2 inicia o quarto passo onde ele envia a mensagem de reposta para o componente 1, no quinto passo o componente 2 ao receber a mensagem de resposta do componente 1 continua a seu processamento normal ilustrado pelo sexto passo.

Esse tipo de regra de coordenação entre troca de mensagens é usado em situações onde a resposta trás informações necessárias para que o remetente continue com o seu processamento. Dependendo do tempo de resposta pode ocorrer uma queda no desempenho, pois o remetente deve fica parado enquanto espera a resposta.

**Sentido Único (*One-Way*)**

Esse tipo é um padrão simples de comunicação síncrona onde após enviar uma mensagem o remetente fica esperando uma confirmação de que o receptor recebeu a mensagem. A diferença entre o Sentido Único e o Pedido-Resposta é o fato de que nele o remetente não usa as informações da reposta, ele só precisa da confirmação de que a mensagem foi recebida para continuar.

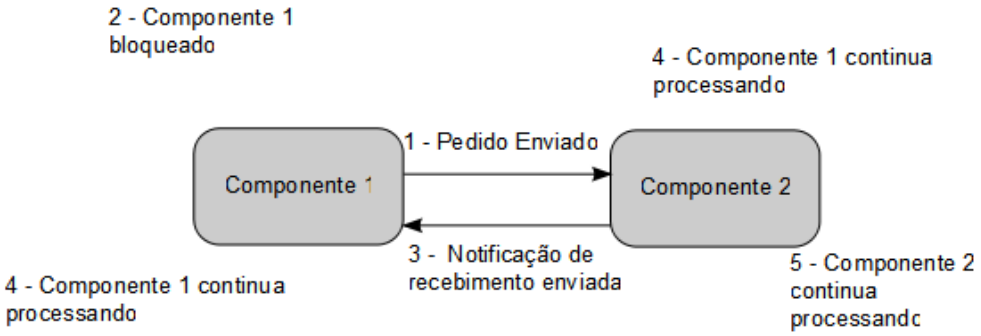


Figura 2 - Sentido Único

A Figura 2 mostra o processo de comunicação usando o padrão Sentido Único, no exemplo, no primeiro passo o componente 1 envia um pedido ao componente 2, o componente 1 fica bloqueado até receber a notificação de que a mensagem foi recebida, em seguida no terceiro passo o componente 2 envia uma notificação de que recebeu a mensagem, então no quarto passo os componentes 1 sai do estado bloqueado e continua processando normalmente enquanto o componente 2 fica processa o pedido enviado pelo componente 1, após concluir o processamento do pedido o componente 2 continua com seu processo normal, ilustrado pelo quinto passo.

**Verificação (*Polling*)**

Esse tipo de padrão permite que o remetente continue com seu processamento, mas de uma forma limitada, e fique sempre verificando se a resposta foi recebida, a Figura 3 demonstra o funcionamento desse padrão.

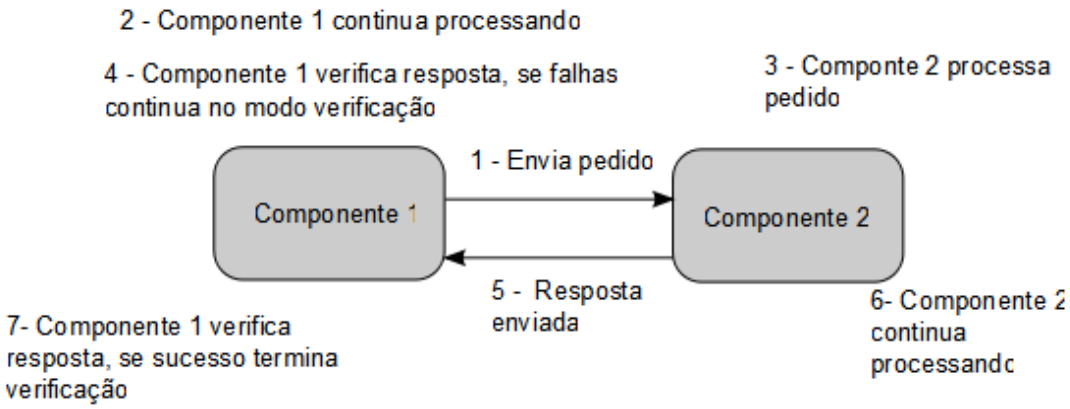


Figura 3 - Verificação (*Polling*)

O primeiro o passo é quando o pedido é enviado, mas diferente dos outros padrões, no segundo passo, o remetente não fica bloqueado à espera da resposta, ele continua com o processamento, em seguida, no terceiro passo, o receptor começa o processamento do pedido, em quanto o receptor não responde o remetente esta em um modo de verificação, isto é, ele fica testando se a resposta já foi recebida, ilustrado pelo quarto passo. O quinto passo se inicia quando o receptor termina o processamento, ele envia uma mensagem de reposta, no sexto passo, continua com seu processamento normal, então o remetente que esta em modo de verificação testa a chegada de resposta, ao ter a confirmação da chegada ele sai do modo de verificação no sétimo passo do processo de comunicação por verificação.

A comunicação assíncrona acontece quando a comunicação entre receptor e remetente acontece de uma maneira que permite que cada um deles continue seus processos de maneira independente. O receptor de uma mensagem não é obrigado a manipular a mensagem ou responder ao remetente. O remetente continua seus processos sem considerar como o receptor irá manipular seu pedido. Existem três tipos populares de comunicação assíncrona [5].

- Passagem de mensagem
- Publicar/subscrever
- Transmissão

Comunicação assíncrona é usada principalmente em contextos de transferência de informações. Aplicações onde uma alteração tem que ser informada a outras aplicações que possuem cópia dos dados alterados e esses dados precisaram ser atualizados. Outro exemplo é quando um evento dispara notificações para outras aplicações [5].

**Passagem de Mensagem**

Passagem de mensagem é o tipo mais simples de comunicação assíncrona, nela a mensagem é enviada sem que seja esperada uma resposta. Após enviar a mensagem o remetente não se preocupa como o pedido será processado ou se ele foi recebido. A desvantagem é que se o pedido for perdido durante a transmissão da mensagem ao receptor, o remetente não saberá que ocorreu esse problema. A Figura 4 mostra como funciona esse tipo de troca de mensagens.

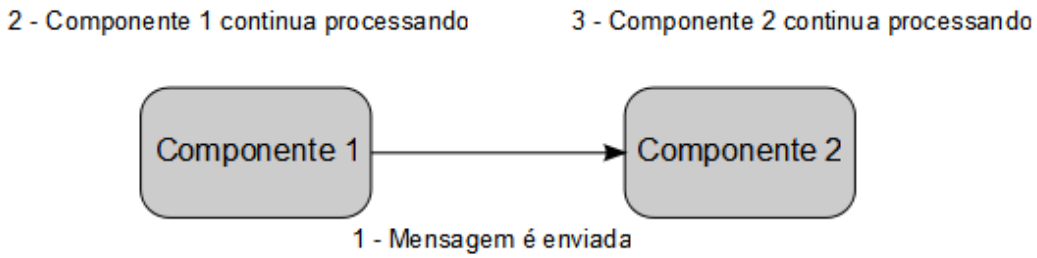


Figura 4 - Passagem de Mensagem

Como pode ser visto na Figura 4, o processo de troca de mensagens é muito simples. Na primeira etapa do processo, o remetente apenas envia a mensagem, no segundo passo ele continua processando, e na terceira etapa o receptor aceita a mensagem e continua processando, não há o envio de uma resposta.

**Publicar/subscrever**

Nesse tipo de comunicação assíncrona o endereçamento do pedido é baseado no interesse dos receptores pelo conteúdo das mensagens. A Figura 5 mostra o funcionamento desse tipo de comunicação assíncrona.

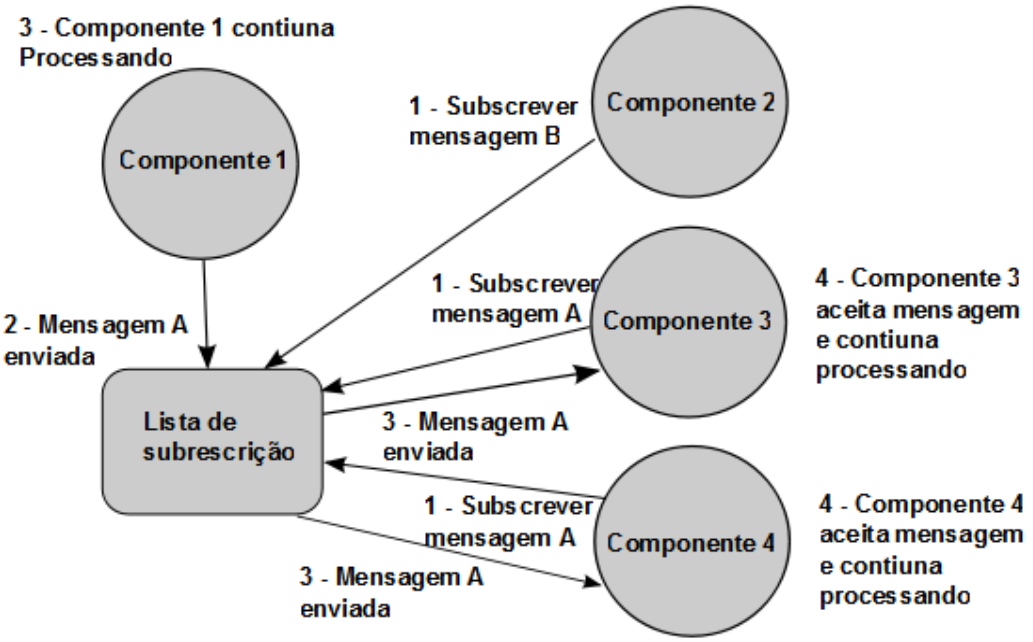


Figura 5 - publicar/subscrever

No primeiro passo desse padrão de comunicação os receptores declaram o tipo de mensagens que estão interessados, depois de definidos os interesses de cada receptor a comunicação a estrutura para a comunicação já esta pronta. Então na segunda etapa o remetente envia uma mensagem para a lista de sobrescrito, como base nos interesses declarados, na terceira etapa a mensagem será encaminhada para os receptores que tem interesse nessa mensagem. No quarto passo todos componentes que tem interesse na mensagem aceitam a mensagem e começam a processar a mensagem. Esse tipo de padrão de comunicação tem como desvantagem a complexidade para definir como serão determinados os receptores que receberão a mensagem.

**Transmissão**

Na comunicação assíncrona do tipo transmissão, a mensagem é enviada a todos os aplicativos interligados. Assim o receptor é que se encarrega de escolher se aceita ou não a requisição. Esse tipo de padrão trás como desvantagem a queda de desempenho dos sistemas, pois para cada mensagem enviada os possíveis receptores precisarão verificar se a mensagem é de seu interesse.

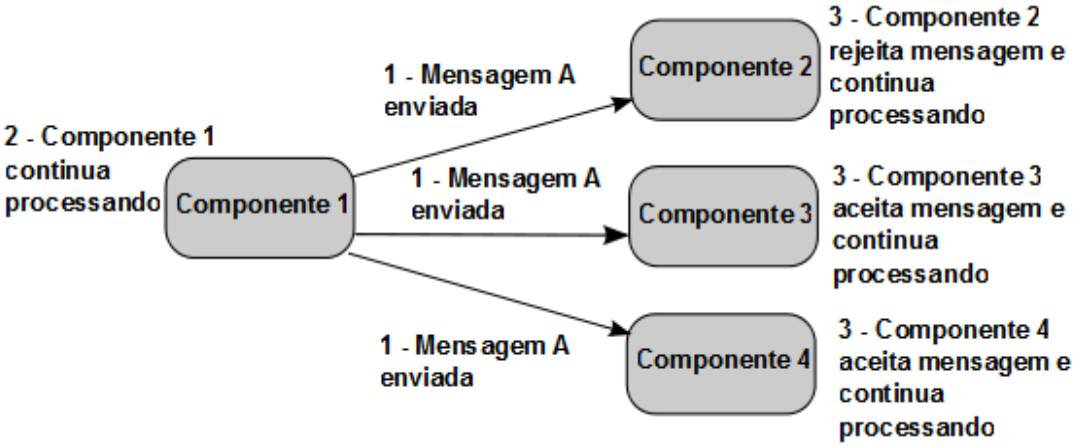


Figura 6 – Transmissão

Na Figura 6, no primeiro passo do processo de transmissão o componente 1 envia uma mensagem para todos os componentes disponíveis, e vai para a segunda etapa do processo, que é continuar processando, no terceiro passo os componentes que aceitaram mensagem começam a processá-la.

### 2.1.2.2 - Métodos de Integração

Um método de integração é a abordagem para construir um pedido de um remetente para um receptor. Existem dois métodos de integração [5]:

- Mensagem;
- Definições de interface.

Na abordagem com uso de mensagens, o cliente cria mensagens que contém as informações sobre as ações desejadas, e também os dados necessários para realização da ação. Assim, no início do projeto de integração é necessário o formato das mensagens seja definido.

Para que seja possível que aplicações possam se comunicar através de mensagens é necessário que remetente seja capaz de criar mensagens no formato apropriado de comunicação e que o receptor possa receber e de ler a mensagem e tomar a ação apropriada [5].

Comunicação com mensagem é bastante flexível por sua facilidade em manipular os dados, entretanto se as mensagens não forem bem documentadas torna difícil a comunicação entre as aplicações.

Na abordagem de interface, o remetente se comunica por meio de uma interface a qual define as ações que podem ser chamadas pela aplicação. Qualquer dado a ser processado é enviado através da interface. [5]

O processo para comunicação através de interfaces seque os seguintes passos:

No remetente:

- Criar uma camada a um componente;
- Executar a chamada.

No receptor:

- Receber a chamada e seus parâmetros;
- Executar as ações baseado no que é definido pela interface.

Nessa abordagem as definições das interfaces devem ser feitas com cuidado, pois as interfaces descrevem as ações que podem ser desempenhadas.

#### 2.1.2.3 - *Middleware*

*Middleware* refere-se à tecnologia que provê serviços independentes de aplicações e atua como o mediador entre os sistemas e aplicações. O termo também pode referenciar produtos de software que implementam esses serviços. Antes da invenção dos *middleware* de EAI as organizações freqüentemente tentavam implementar suas próprias soluções de integração, mas elas geralmente não tinham sucesso devido à complexidade e esforço para criar esses sistemas.

*Middleware* é um tipo de software que facilita a comunicação de pedidos entre componentes de software por meio de interfaces definidas ou mensagens. Provê o ambiente de execução para gerenciar os pedidos entre os componentes de software. Existem cinco tipos de *middlewares* no mercado hoje [5]:

- Chamadas de procedimentos remotos (*Remote Procedure Calls*);
- *Middleware* de acesso a banco de dados;
- *Middleware* orientado a mensagem;
- Tecnologia de objetos distribuídos;
- Monitores de processamento de transação.

RPC (*Remote Procedure Calls*) é um tipo de *middleware* que é baseado na noção de desenvolvimento de aplicações distribuídas que se integram no nível de procedimento. Esse tipo cria a habilidade para fazer chamadas a procedimentos pela rede.

Middleware de acesso a banco de dados é um tipo de *middleware* que é baseado na noção de acessar dados distribuídos em arquivos ou bancos de dados. Integra no nível de dados e permite consultas aos dados ou mudanças nos dados por meio da rede.

O chamado MOM (*Middleware* Orientado a Mensagem) provê a habilidade de integrar aplicações por meio do uso de mensagens. A mensagem é o conceito usado como método de integração. O MOM provê a habilidade para criar, manipular, armazenar e comunicar essas mensagens. Exemplos de Middleware desse tipo são ActiveMQ, MQSeries da IBM, MQIntegrator da IBM.

A chamada DOT (*Distributed Object Technology*), tecnologia de objetos distribuídos, é um tipo de *middleware* que estende o conceito da tecnologia orientada a objeto para processamento distribuído.

Interfaces são desenvolvidas para aplicações que fazem software parecer como objetos. Exemplos dessa tecnologia são: CORBA, COM e DCOM [12].

Os chamados TPMs (*Transaction Processing Monitors*) são um tipo de *middleware* que preservam a integridade da transação. Eles suportam características como *rollback*, refazer, armazenar histórico de erro e replicação para eliminar pontos de falha. TPMs asseguram que uma transação mantenha as propriedades ACID (Atomicidade, Consistência, Isolamento e Durabilidade). Exemplo desse tipo de *middleware* é o Tuxedo da BEA.

#### 2.1.2.4- Serviços

O modelo de comunicação, os métodos de integração e o *middleware* são núcleo de qualquer solução de EAI. O serviço é o um elemento muito importante na integração de sistemas. Eles não fazem parte do núcleo da EAI, mas eles ajudam significativamente na arquitetura e implementação de soluções de EAI. Um serviço é uma extensão funcional para comunicação básica ou capacidade de *middleware*. Alguns serviços básicos são considerados importantes para uma empresa baseada em EAI [5]:

Diretório: armazena todos os componentes e informações chaves sobre o sistema. É usado para automatizar a ação de busca de elementos da integração, também pode ser usado para gerenciar mensagens ou interfaces assim como metadados.

Ciclo de vida: ajuda o desenvolvedor a automatizar a criação de objetos ou mensagens e garantir que eles sejam gerenciados de forma apropriada.

Segurança: Esse serviço é freqüentemente esquecido. Geralmente é ignorado devido a sua complexidade.

Conversão e transformação: A representação de dados é um problema em toda solução de integração, pois existem várias formas para representar dados, é necessário ser capaz de converter e transformar os dados para um formato apropriado durante a integração.

Persistência: Assegurar que o estado das informações e dados esteja armazenado de forma segura. É muito importante garantir que informação não seja perdida.

Eventos: Habilidade para identificar quando um evento ou problema acontece.

Notificação: Uma vez que um evento é detectado informação aos componentes interessados que o ocorrido.

Fluxo de trabalho: gerenciar um conjunto de pedidos ou mensagens de vários componentes na ordem certa como uma ação simples.

Na próxima subseção começarão a ser explicados os conceitos de SOA. Essa arquitetura de software que tem muitos conceitos similares aos presentes na EAI. A base de sistema baseados em SOA são serviços, eles são o um dos principais conceitos da arquitetura, além disso, a troca de informações pode ser feita tanto com mensagens quanto através da definição de interfaces. Isso torna SOA uma arquitetura muito usada para a integração de sistema.

## 2.2- Arquitetura Orientada a Serviços (SOA)

Nessa seção são mostrados os fundamentos da arquitetura orientada a serviço, descrevendo suas características e os principais elementos que estão presentes em qualquer implementação de SOA.

SOA é uma abordagem arquitetural corporativa que permite a criação de serviços de negócio interoperáveis que podem facilmente ser reutilizados e compartilhados entre aplicações e empresas. A arquitetura define o uso desses serviços para apoiar processos e necessidades de usuários ou outras aplicações. O problema em definir SOA é que existem inúmeras maneiras para fazer isso. [3]

SOA provê flexibilidade para tratar processos de negócios e infra-estrutura de TI, como componentes que podem ser reutilizados e recombinados. Segundo Josuttis(2007), os principais conceitos em SOA são serviço, interoperabilidade, baixo acoplamento.

Um serviço, do ponto de vista da arquitetura SOA, é uma função de um sistema computacional que é disponibilizado para outro sistema. Um serviço deve funcionar de forma independente do estado de outros serviços, exceto nos casos de serviços compostos (*composite services*), e deve possuir uma interface bem definida. Normalmente, a comunicação entre o sistema cliente e aquele que disponibiliza o serviço é realizada através de *Web Services* [1].

SOA é focada em processos de negócios. Esses processos podem ser executados em diferentes etapas e em diferentes sistemas. O principal objetivo de um serviço é representar, de forma natural, uma etapa de uma funcionalidade de negócio. [3]

Basicamente tudo que tenha uma interface definida e faça parte de um processo de negócio pode ser um serviço. Um serviço pode ser definido como é uma interface que recebe mensagens, executa ações e/ou retorna alguma informação. . [3]

As implementações SOA dependem de uma rede de serviços. Cada serviço irá implementar uma ação, por exemplo, consultar a lista de professores de um departamento ou a lista de alunos matriculados em um curso. Se um serviço apresenta uma interface simples que abstrai a complexidade, os usuários podem acessar os serviços, sem conhecimento do serviço desenvolvido. Essa característica da arquitetura orientada a serviço é conhecida como interoperabilidade.

Interoperabilidade não é uma idéia nova. Antes de SOA, interoperabilidade já era um objetivo do EAI. Entretanto, em SOA as preocupações com interoperabilidade são no começo do projeto e não no final quando os sistemas já estão prontos e precisam ser integrados. Ela serve como base para o começo da implementação dos serviços e esses se espalhem por diversos sistemas distribuídos. [3]

A interoperabilidade entre diferentes sistemas e linguagens de programação fornece a base para a integração entre aplicações em diferentes plataformas, através de um protocolo de comunicação. Um exemplo dessa comunicação depende do conceito de mensagens. Usando mensagens, através de canais de mensagens definidos, diminui-se a complexidade da aplicação final, permitindo que o desenvolvedor do aplicativo se concentre na funcionalidade do aplicativo de verdade. O desejo é o de criar um conjunto de recursos a ser compartilhado. Para realizar isso é necessário que os serviços tenham o mínimo de dependência, isso significa a necessidade da criação de serviços com baixo acoplamento.

Segundo Josuttis (2007), baixo acoplamento é um conceito que é tipicamente usado em relação a requisitos de escalabilidade, flexibilidade e tolerância a falhas nos sistemas. O

objetivo do baixo acoplamento é minimizar a dependência entre sistemas, assim modificações e falhas em um sistema irão gerar poucas conseqüências em outros sistemas.

Segundo Josuttis (2007) existem alguns elementos que estão presentes em todas as implementações de projetos SOA. Ele refere-se a esses elementos com ingredientes da Arquitetura Orientada a Serviços, apresentados nas subseções a seguir.

Infraestrutura é a parte técnica de SOA que irá permitir a interoperabilidade. O termo que vem do conceito de EAI, e em SOA o *Enterprise Service Bus* (ESB) pode ser considerado parte da infra-estrutura. Um ESB é tipo de *middleware*, explicado na subseção 2.1.2.4, e ele trás algumas vantagens para implementação de SOA como: transformação de dados, roteamento inteligente, segurança, confiabilidade, gerenciamento de serviços, monitoramento e *logging*.

A arquitetura é necessária para criar manter o controle dos serviços, ferramentas e padrões usados em um projeto SOA. É necessário fazer a classificação dos diferentes tipos de serviços, o nível de desacoplamento necessário, ter um modelo de interfaces de dados bem definido, definir regras e políticas de segurança além de definir a infra-estrutura e padrões que serão utilizados.

Uma dificuldade em relação à definição e desenvolvimento de grandes sistemas são as diferentes pessoas e equipes envolvidas. Como conseqüência existe um grande caminho entre a idéia inicial do negócio até o sistema em uso na produção.

Para manter isso sobre controle é necessária a definição dos processos apropriados, esse definição inclui [3]:

Modelagem de Processos de Negócio (BPM – *Business Process Model*): É a tarefa de dividir processos de negócios em atividades e tarefas menores, que são os serviços no caso de SOA.

Ciclo de vida: Definir o ciclo de vida de um serviço envolve definir as diferentes etapas que deve ser seguidos ate que o serviço seja realizado.

Governança: é constituída por um conjunto de políticas que os prestadores e consumidores de serviços deverão seguir um conjunto de práticas de execução dessas políticas, e um conjunto de processos para garantir que as políticas são aplicadas corretamente. Existe normalmente uma estrutura organizacional local para definir e implementar políticas de governança e, freqüentemente, um repositório para automatizar e aplicá-las. [6]

SOA como já foi mostrado é uma arquitetura baseada no uso de serviço. Uma das principais tecnologias para implementação desses serviços são os *Web Service*, eles podem ser usados para criação e disponibilização de serviços entre sistemas.

## 2.3- *Web Services*

Nessa seção são apresentados os principais tecnologias necessárias para implementação de *Web Services.* As subseções descrevem as principais tecnologias usadas para implementação dos *Web Services.* Na Subseção 2.2.1 é explicado o que é um documento XML, a Subseção 2.2.2 descreve o protocolo SOAP, a Subseção 2.2.3 apresenta a linguagem para descrever *Web Services*, WSDL (*Web Services Description Language*) e na última subseção 2.2.4 mostra o padrão UDDI.

Como discutido na Seção 2.1, SOA não é uma arquitetura concreta, assim existem diversas maneira de implementá-la. Devido a sua fácil implementação e utilização, e o fato de serem independentes de linguagem de programação, os *Web Services* tornaram-se uma das principais maneiras utilizadas para implementar os serviços em SOA.

*Web Services* é um sistema de software que segue um conjunto de padrões abertos de interoperabilidade. Estes padrões permitem a interoperação mundial de computadores independentemente da plataforma de hardware, sistema operacional, infra-estrutura de rede ou linguagem de programação [3]. A grande utilização dos WS é baseada no fato de que ele usa protocolos abertos de comunicação na Internet e XML para transacionar o seu negócio. Um *Web Service* é, portanto, um sistema de software que pode agir a pedido de qualquer computador conectado à rede no mundo, que se comunica usando padrões XML. [1]

*Web Services* são uma solução utilizada na integração de sistemas e na comunicação entre aplicações diferentes. Com esta tecnologia é possível que novas aplicações possam interagir com aquelas que já existem e que sistemas desenvolvidos em plataformas diferentes sejam compatíveis. O único requisito é que os dois sistemas conheçam o formato em que as mensagens serão trocadas e que tenham suporta para manipulação de mensagens no formato XML.

Existem 4tecnologias usadas em *Web Services.*

- XML (*Extensible Markup Language*)
- WSDL (*Web Services Description Language*)
- Protocolo Simples de Acesso a Objetos (SOAP - *Simple Object Accesss Protocol*).

- UDDI (*Universal Description, Discovery and Integration*).

Cada uma das tecnologias mencionadas é explicada com mais detalhes nas próximas subseções.

### 2.3.1- XML

XML é uma linguagem de marcação usada para descrever dados. Ele consiste em tags que podem ser definidas livremente pelo usuário em seu documento XML, diferente do HTML que tem tags predefinidas.

Existem regras que devem ser seguidas na criação de documentos XML. As regras da gramática XML são muito mais rígidas que as regras encontradas na criação de arquivos HTML, mas isso não é uma coisa ruim, pois esse fato ajuda a assegurar que não existiram erros na hora do processamento dos dados do arquivo. [2] As regras básicas de criação são:

- XML é case sensitive, isso significa que as tags<aluno>, <Aluno> e <ALUNO> são diferentes.
- Na estrutura de um documento XML só deve existir uma tag raiz, que é a tag principal do arquivo.
- Toda tag aberta deve ser fechada, <aluno></aluno>.
- Tags de início e de término deve ser corretamente aninhadas.
- Valor dos atributos deve ser escrito entre aspas.

A Figura 7 mostra um exemplo de arquivo XML, nela pode-se ver toda a estrutura de um arquivo XML completo. A raiz do documento é a tag receita que possui os atributos nome, tempo_de_preparo e tempo_de_cozimento. Os nós filhos de receita são: ingredientes e instruções. Cada ingrediente tem como atributo quantidade e unidade. Instruções é composta por tags passo, cada passo descreve um passo que deve ser seguido na receita.

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<receita nome="pão" tempo_de_preparo="5 minutos" tempo_de_cozimento="1 hora">
  <titulo>Pão simples</titulo>
  <ingredientes>
    <ingrediente quantidade="3" unidade="xícaras">Farinha</ingrediente>
    <ingrediente quantidade="7" unidade="gramas">Fermento</ingrediente>
    <ingrediente quantidade="1.5" unidade="xícaras" estado="morna">Água</ingrediente>
    <ingrediente quantidade="1" unidade="colheres de chá">Sal</ingrediente>
  </ingredientes>
  <instrucoes>
    <passo>Misture todos os ingredientes, e dissolva bem.</passo>
    <passo>Cubra com um pano e deixe por uma hora em um local morno.</passo>
    <passo>Misture novamente, coloque numa bandeja e asse num forno.</passo>
  </instrucoes>
</receita>
```

Figura 7 - Exemplo documento XML que representa uma receita de pão

Como pode ser notada, na figura 7, a estrutura de um documento XML é bem simples e intuitiva. Sendo assim, mesmo tendo um conhecimento mínimo sobre XML, qualquer pessoa pode compreender as informações contidas nesse tipo de documento.

Devido às regras que garantem que os documentos XML não contêm erros de formação, tornando sua manipulação mais fácil, ele se tornou um meio padronizado para representação textual de dados. A maioria das linguagens de programação atuais implementam formas para processar dados em XML. Essa é uma característica que influenciou a adoção XML como formato padrão para a troca de mensagens em *Web Services* [2].

### 2.3.2- SOAP

SOAP (*Simple Object Accesss Protocol*), como o próprio nome diz, é um protocolo de acesso a objetos. OSOAP serve para a troca de informações e/ou dados através de objetos criados em diversas linguagens de programação como VB, Java, C++ etc. Com SOAP é possível criar aplicações para troca de dados entre empresas, o chamado B2B(*Business to Business*). O mais importante é que esse protocolo é baseado em XML, o que significa que as aplicações criadas, podem ser aceitas em qualquer plataforma e por qualquer linguagem que tenha suporte para a manipulação de arquivos XML.

SOAP é um protocolo para troca de informações entre aplicações utilizando HTTP, isso é uma características importante, pois permite a comunicação entre as aplicações usando a internet.

SOAP é a linguagem mais usada em *Web Services* (WS). Quando se diz que WS são baseados no XML, na verdade se quer dizer que são baseados em mensagens SOAP, que são

escritas em XML. O que torna SOAP especial e o difere do simples XML, é que cada mensagem SOAP segue um padrão que foi especificado pelas normas da W3C. [1]


Figura 8 – Modelo de Envelope de mensagem SOAP

A Figura 8 mostra um exemplo da estrutura de um arquivo SOAP. Cada mensagem SOAP inicia-se com uma tag SOAP-ENV:envelope . A tag envelope sinaliza a mensagem ao destinatário que ele está prestes a receber uma mensagem SOAP. Depois vem a tag SOAP-ENV:header Especifica informações especificas como informações sobre autenticação ,. Por fim há o corpo da mensagem SOAP que é definido dentro da tag SOAP-ENV:body que transporta os dados e instruções de operações exigidas pelo computador consumidor. Dentro do corpo da mensagem também pode ser descrido o tratamento de erros, essa definição é feita dentro da tag opcional <SOAP-ENV:fault>.

### 2.3.3- *Web Services* Description *Language* (WSDL)

WSDL é uma linguagem para descrição de *Web Services*. É um documento XML que contém as especificações de localização, as operações (métodos ou serviços) que fazem parte do *Web Services* .

Cada *Web Service* tem um documento WSDL que fornece ao potencial consumidor do serviço uma explicação de como o serviço funcionar e como acessá-lo. O WSDL descreve como criar uma solicitação SOAP que irá invocar um *Web Service* específico. Se um desenvolvedor de software pretende criar um programa do software consumidor de *Web Services*, tudo que ele necessita é este documento descritivo. O WSDL fornece todas as informações que o desenvolvedor necessita para criar um programa que requisite um *Web Service*. Isto significa que os desenvolvedores podem criar *Web Services* consumidores sem nunca ter conhecido ou falado com o desenvolvedor que criou o *Web Service* em questão [1].

Um documento WSDL descreve um serviço web usando esses elementos principais:

- <types> Tipo de dados usados pelo *Web Services*
- <message> A mensagem usada pelo *Web Services*
- <portType>As operações realizadas pelo *Web Services*
- <binding> Protocolo de comunicação usado pelo *Web Services*

### 2.3.4- Universal Description, Discovery and Integration (UDDI)

UDDI é uma especificação técnica para descrever, descobrir e integrar serviços na Web. É uma parte crítica da pilha de protocolos para *Web Services*, habilitando usuários dos serviços a publicarem e descobrirem *Web Services* .

A idéia original do UDDI é criar três papéis em relação aos *Web Services*: provedores forneceriam serviços, consumidores necessitam de serviços e *brokers* onde seriam reunidos e publicados os serviços [1].

O UDDI é uma especificação técnica que descreve como construir um diretório distribuído de negócios e *Web Services.* A informação UDDI é armazenada dentro de um formato específico em XML, definido por WSDL. A especificação inclui detalhes de uma API própria para buscar dados existentes e para publicação de novos dados.

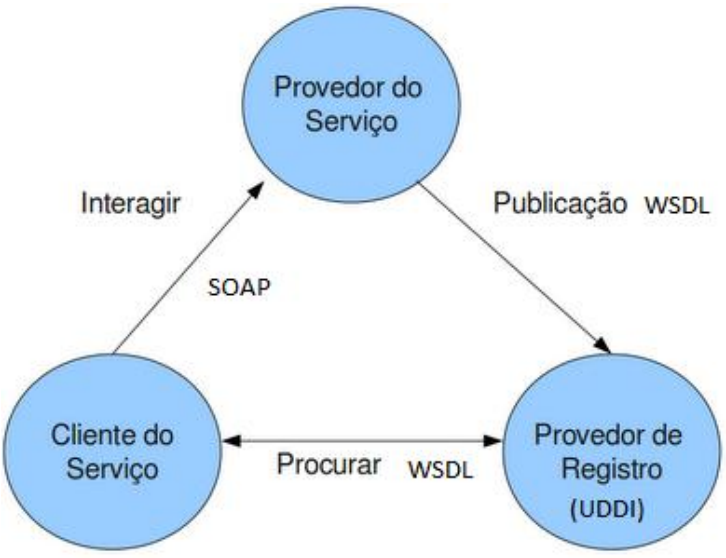


Figura 9 – Ciclo de um *Web Service*

A Figura 9 mostra o Ciclo de uma *Web Service*, onde um Provedor de Serviço publica a descrição do serviço (WSDL) em um Provedor de registro (UDDI). Assim que um cliente deseja usar um serviço ele procura a descrição do serviço (WSDL) no provedor de registro e depois ele com a descrição do serviço em mãos ele já sabe como interagir com o serviço através de mensagens (SOAP).

## 2.4- Segurança em Arquiteturas Orientadas a Serviço

Nessa seção são mostradas algumas características da segurança na arquitetura orientada a serviços (SOA). A subseção 2.4.1 aborda algumas das principais falhas de segurança de aplicações, a subseção 2.4.2 explica as características de segurança presentes na camada de transporte, na subseção 2.4.3 são explicadas as características da segurança em mensagens, a subseção 2.4.4 apresenta padrões de segurança em *Web Services* e XML, e por último na subseção 2.4.5 são apresentadas as normas de segurança para *Web Services* .

Segurança da informação é um conjunto de ações que devem ser executadas para garantir a segurança dos dados que possuem valor para indivíduos e organizações. Sempre que se fala sobre o assunto existem algumas termos comuns que também estão presentes quando se fala em segurança SOA. Os principais termos são:

Autenticação - é responsável por como verificar uma identidade. Uma identificação pode ser um usuário, um dispositivo físico ou uma requisição de um serviço externo. Em relação com SOA, isso pode significar identificar quem esta chamando o serviço.

Autorização – é responsável por verificar que uma identidade tem permissão de fazer. No caso de SOA, significa verificar que uma requisição tem permissão para chamar um serviço ou ver seu resultado.

Confidencialidade – Garantir que os dados permaneçam confidenciais durante a transmissão ou armazenamento é outro aspecto chave de segurança. Em relação a serviços, isso significa assegurar que ninguém além do que chamou o serviço possa ver os dados enquanto os dados são transmitidos entre o provedor e consumidor.

Integridade – A função chave na integridade é garantir que os dados não sejam alterados ou falsificados. Em relação a SOA, isso significa que ninguém possa conseguir a dados se autorização através do uso de dados falsos.

Disponibilidade - A função chave em garantir que os dados não sejam perdidos ou corrompidos, ou que o sistema fique indisponível.

Não Repúdio - A função chave é a garantia de segurança que impede uma entidade participante em uma operação de negar essa participação.

A segurança em aplicações SOA é um fator muito importante, pois como os serviços estão disponíveis em uma rede, é necessário garantir que só pessoas autorizadas possam ter acesso aos serviços.

Em aplicações tradicionais geralmente são criadas várias barreiras que servem para garantir a segurança. Na Figura 10 pode ser visto que para ter acesso às funções do sistema, primeiro é necessário passar por uma barreira de segurança que no caso é o Modulo de segurança (*Segurity Module*) da aplicação, a comunicação entre o cliente e ou servidor de aplicação é feita através de um canal seguro que usa protocolos que garantem a segurança no nível de transporte, no caso SSL/TLS [23].

Num contexto de aplicações centralizadas tradicionais essa configuração de segurança funciona muito bem, mas quando a aplicação está em um ambiente distribuído como SOA, esse tipo de estrutura de segurança não é o mais adequado.

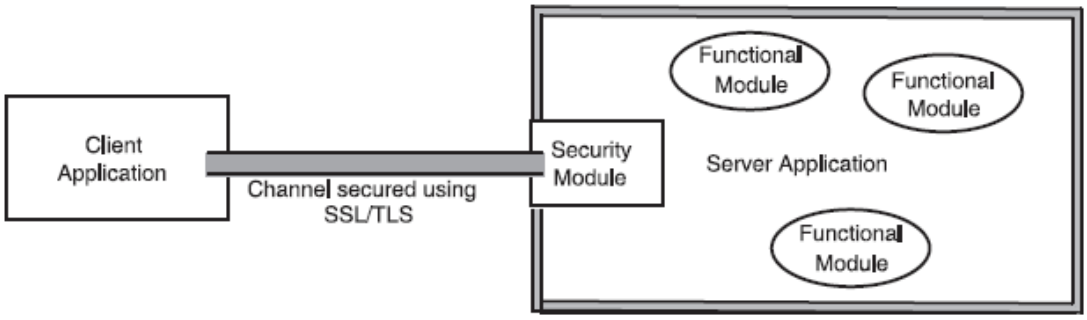


Figura 10 - Estrutura de segurança em sistema centralizado

SOA é uma arquitetura que pode trabalhar com processos de negócio distribuídos por diferentes sistemas. Podem existir diferentes tipos de mecanismos de segurança e políticas de segurança nesses sistemas. Assim é um desafio introduzir um conceito geral de segurança para os diferentes conceitos de segurança existentes nos sistemas [8].

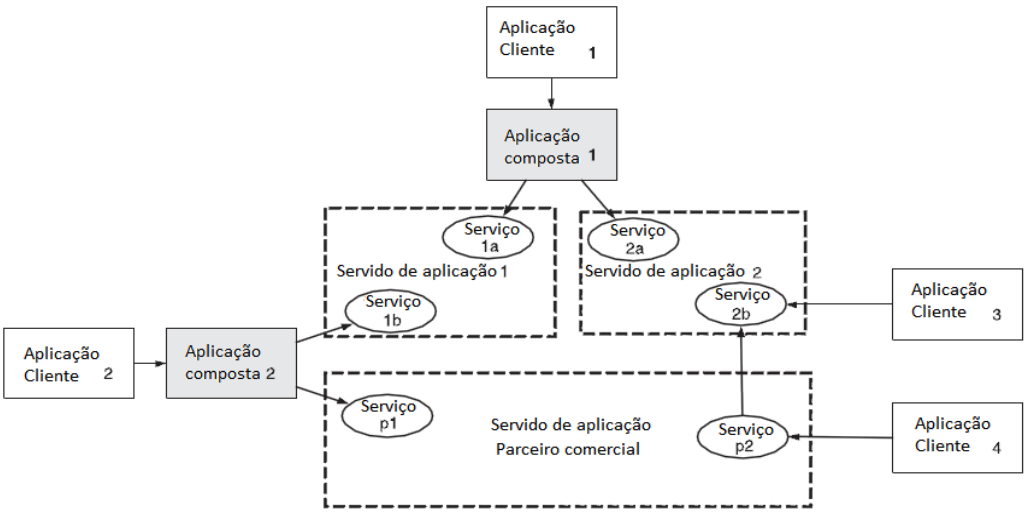


Figura 11 – Exemplo de Estrutura de sistema em SOA

A Figura 11 mostra um cenário do uso da arquitetura SOA onde facilmente pode-se ver a diferença da arquitetura centralizada. No cenário ilustrado no exemplo, existem três servidores de aplicações, incluído um parceiro, disponibilizando vários serviços que são usados por aplicações clientes.Além das aplicações clientes, existem aplicações compostas de serviços que estão divididos em servidores diferentes.Nesse contexto fica claro que o modelo de segurança não pode ser implementado de maneira centralizada dentro de uma aplicação.

Em relação à *Web Services* a segurança pode ser implementada de duas maneiras: através de mecanismos de segurança em nível de transporte ou em nível de mensagens. Esses mecanismos de segurança dispõem de padrões e protocolos específicos, e atendem diferentes requisitos de segurança da arquitetura SOA.

### 2.4.1- As principais Falhas de Segurança

O *Open Web Application Security Project* (OWASP) é uma organização mundial sem fins lucrativos focada na melhoria da segurança do software de aplicação.Em seu site é uma fonte muito rica informações contendo uma lista com os principais tipo de ataques [11]. Desses ataques alguns são muito comuns contra *Web Services:*

- **SQL Injection**
- **Ataque ao protocolo SOAP**
- **Negação de Serviço**
- **XML Injection**

Essas quatro vulnerabilidades possibilitam alguns tipos de ataques às aplicações, explicadas em detalhes a seguir.

**SQL Injection**

É um tipo de ameaça de segurança que se aproveita de falhas em sistemas que interagem com bases de dados via SQL. A injeção de SQL ocorre quando o atacante consegue inserir uma série de instruções SQL dentro de uma consulta através da manipulação das entradas de dados de uma aplicação.

Algumas dos problemas que podem permitir o SQL *Inject* são:

- A entrada de dados não validados podem ser usadas diretamente em uma consulta SQL e podem levar a um comprometimento da segurança.
- Quase todos os aplicativos de *Web Services* são construídos em cima de um banco de dados - todo mundo é um alvo potencial.
- XML fornece meios para ocultação de códigos maliciosos.

**Ataque ao protocolo SOAP**

O protocolo SOAP, que é o protocolo usado para a maioria dos *Web Services* não possui mecanismos de segurança embutidos. Segundo o OWASP as principais vulnerabilidades no protocolo SOAP são:

- Todas as chamadas de método são desprotegidas;
- API dos serviços é acessível ao público através dos arquivos WSDL;
- SOAP não tem estado (*stateless*);
- Fraquezas do Protocolo SOAP;
- Arquivos WSDL desprotegidos permitem fácil captura de informações sobre um sistema.

O protocolo é mal definido em algumas áreas e implementações diferentes. Cabeçalhos SOAP são processados de forma diferente em diferentes sistemas.

**Negação de Serviço**

Esse ataque é focado em fazer um recurso não disponível (site, aplicativo de servidor) à finalidade que foi projetado.Se um serviço recebe um número muito grande de pedidos, ele pode parar de fornecer o serviço aos usuários legítimos [11].

Analisar requisições SOAP muito grandes é pesado para o processador e a memória. XML precisa ser convertido e a validade do pedido deve ser verificada. Isso consome muitos recursos do servidor.

Métodos mal codificados podem aceitar parâmetros de entrada arbitrariamente grande e cabeçalhos SOAP podem ser adicionados à vontade, devido a um padrão flexível.

Um invasor pode facilmente enviar arquivos de texto múltiplos XML através de uma ligação HTTP, o que levará ao esgotamento dos recursos do servidor.

Todos esses exemplos podem ser usados de maneira que gerem pedidos que usem muito recurso de processamento, o que gera uma sobrecarga do sistema, fazendo com que os usuários que realmente necessitam dos serviços sejam prejudicados.

**XML Injection**

XML é a base para a implementação de *Web Services* ,ataques que obtêm sucessos contra *Web Services* geralmente significam elaborar um arquivo XML válido que é mal interpretado pelos sistemas.

*XML Injection* é baseado na inserção elementos XML como parte da entrada do usuário para um *parser* XML. Isto poderia ser usado para substituir dados inseridos por usuário no mesmo documento.

O analisador SAX [13] é conhecido por ser vulnerável, pois ele lê o documento com de forma seqüencial, assim o último elemento encontrado é o que será usado, então caso elemento novo seja adicionado por pessoas não autorizadas, após os elementos acionados pelos usuários, os dados originais são ignorados.

### 2.4.2- Segurança em Nível de Transporte

Atualmente os principais mecanismos responsáveis por garantir a segurança nas comunicações no nível de transporte são protocolos SSL (*Security Socker Layer*) e TLS (*Transport Layer Security*) [23]. Esses protocolos se tornaram padrão de fato do mercado e estão presente na grande maioria das transações seguras efetuadas atualmente na Internet.

Esses protocolos garantem um bom nível de segurança em comunicação ponto a ponto, garantindo que a comunicação entre um consumidor e um provedor de serviço seja

feita mantendo a integridade e o sigilo dos dados durante a transmissão. Como os protocolos só podem ser utilizados nas comunicações ponto a ponto, tornaria complicada a implementação numa arquitetura SOA, já que pode haver vários serviços intermediários para a execução de um determinado processo de negocio.

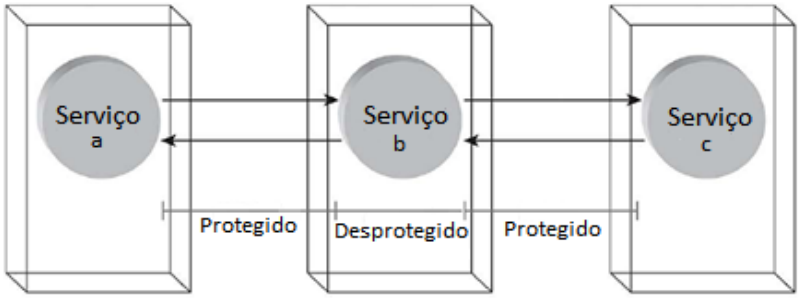


Figura 12 - Transporte de mensagens entre serviços, segurança nível de transporte

A Figura 12 mostra a comunicação entre três serviços usando métodos de segurança em nível de transporte, durante o transporte da mensagem, os mecanismos de segurança garantem que a mensagem esta segura, porém entre o serviço que faz a requisição e o serviço provedor existe um terceiro serviço intermediário por onde a mensagem tem que passar, dentro desse serviço a mensagem fica desprotegia e o serviço tem acesso a todo conteúdo da mensagem.

Um exemplo seria fazer compras em uma loja on-line, ao enviar informações sobre o cartão do cliente, durante o transporte entre o cliente e a loja e entre a loja e a operadora de cartão, os dados do cliente estariam seguros contra ataques externos, entretanto na hora que a mensagem chega ao sistema da loja, os dados do cartão do cliente estão disponíveis. Uma melhor abordagem seria se mesmo a loja não tivesse acessos a esses dados do cliente, bastando repassá-los a operadora de cartão e receber dela a resposta em relação à situação do cliente.

Como o uso de serviços intermédios são muito comuns na Arquitetura SOA, dificulta-se a implantação de mecanismos de segurança de transporte, pois esses não atendem completamente aos requisitos de segurança exigidos pela arquitetura SOA.

### 2.4.3- Segurança de Mensagem

A arquitetura SOA baseia-se na troca de mensagens entre serviços consumidores e provedores, essas mensagens usam o protocolos aberto XML como base, isso permite que possam ser utilizados mecanismos de segurança em mensagens.

Como explicado na seção 2.4.2, apesar dos serviços intermediários não integrarem o processo de negócio como um todo, nem toda informação presente nas trocas de mensagens será usada por eles e às vezes pode ser que eles nem devam ter acessos a determinadas informações. Assim,é importante que dentro da mensagem também existam mecanismos de segurança para garantir que só quem tem autorização possa ter acesso aos dados nas mensagens.

O fato de usar XML nas mensagens permite que possam ser usadas tecnologias de segurança para esse tipo de documento nas mensagens, como XML *Digital Signature* [14] e XML *encryption* [15], protocolos que garantem a segurança e sigilo das mensagens.

A Figura 13mostra a um o tráfego de uma mensagem entre dois serviços, como uso de um serviço intermediário, mas com o nível de segurança de mensagem. Mesmo dentro do serviço intermediário as informações da mensagem ainda estão protegidas, e o serviço só terá acessos a essas informações caso possua autorização.

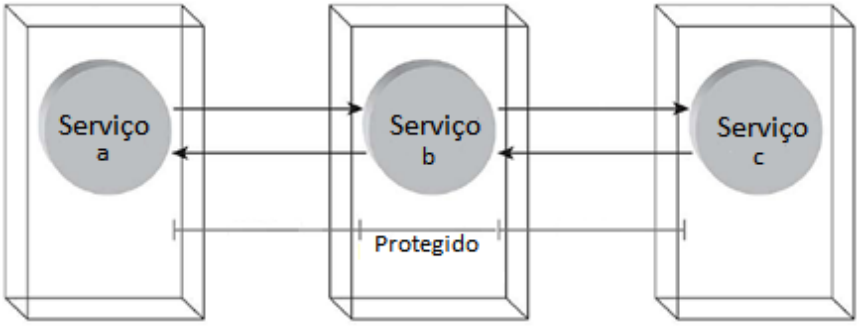


Figura 13 – Transporte de mensagens entre serviços, segurança nível de mensagem

### 2.4.4- Tecnologias de segurança em *Web Services* e XML

Segundo Josuttis (2007) Em princípio, segurança em *Web Services* e XML pode usar diferentes tipos de normas, incluindo as seguintes:

- Normas gerais de segurança;
- Normas de segurança para XML;
- Normas de segurança para *Web Services.*


Figura 14 - Pilha de tecnologias de Segurança em XML e *Web Services*

A Figura 14 mostra as normas gerais de segurança que incluem algoritmos de criptografia, bem como as normas básicas de segurança para criptografia e conversação segura, tais como SSL/TLS [23], Kerberos [18]. Existem também tecnologias especiais que lidam com documentos XML, como XML *Encryption* [15] e *XML Signature* [13], essas tecnologias de segurança são usadas para definir a criptografia dos arquivos XML. Finalmente, na parte superior do diagrama existem normas com aspectos especiais *Web Services*, como o WS-Security [3].

#### 2.4.4.1 Criptografia

Criptografia é a ciência da escrita secreta. É um ramo da matemática, que faz parte da criptologia [26]. Criptografia usa chaves para criptografar e descriptografar dados. A chave é um valor secreto, como uma senha. Um texto que for criptografado utilizando chaves diferentes resultará em mensagens cifradas diferente. Da mesma forma, a mensagem cifrada só pode ser descodificada para o texto original usando a chave correta. Existem dois tipos de chaves:simétricas e assimétricas. Chaves simétricas e assimétricas são usadas para criptografar dados em segurança no nível de transporte e no nível de mensagem

Criptografia com chaves simétricas usa a mesma chave para enviar e receber uma mensagem, como mostrado na figura 15, chaves simétricas são também chamados de chave privada [26].

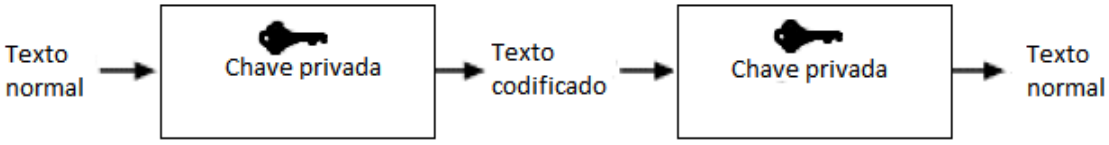


Figura 15 – Codificação Chaves simétrica

Como mostra a Figura 15, o texto é codificado usado uma chave privada e depois e decodificado usando uma chave privada igual.

Criptografia com chaves assimétricas usa uma chave para codificar a mensagem que será enviada e outra para decodificar mensagens recebidas [26]. Essas chaves são sempre geradas em par. Chaves assimétricas também são conhecidas como chaves públicas.

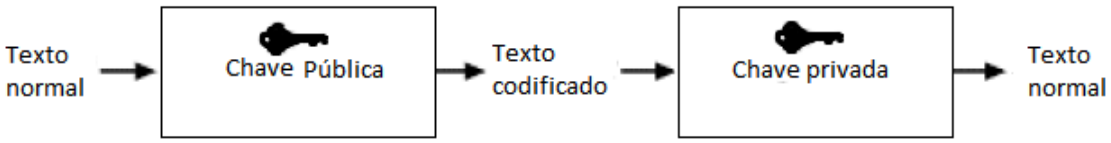


Figura 16 - Codificação com chaves assimétricas

Como mostra a Figura 16, o texto é codificado usado a chave pública do receptor e depois e decodificado usando uma chave privada do receptor.

Agora surge um problema no uso de chaves públicas, como provar que a chave pública realmente pertence ao destinatário correto. Para resolver esse problema existem as assinaturas e os certificados digitais.

Uma assinatura digital serve para garantir a integridade da mensagem. Uma assinatura é um resumo da mensagem (*message digest*). Esse resumo da mensagem é um número especial calculado a partir de um conjunto de dados de entrada como mostra a Figura 17.


Figura 17 – Criando resumo de mensagem

Para garantir que a mensagem realmente não foi alterada, basta que o resumo da mensagem seja enviado junto com a mensagem. Tendo o resumo e a mensagem em mãos basta que o receptor use o mesmo algoritmo para resumir mensagem recebida e comparar o

resultado com o resumo recebido junto com a mensagem, se os dois forem iguais então significa que a mensagem não foi alterada.

Um certificado é essencialmente, uma chave pública assinada por uma pessoa ou entidade de confiança. Essa entidade cria o certificado, colocando algumas informações sobre ela, informações sobre o dono da chave pública, e então armazena a chave pública em um arquivo. Ela em seguida, assina o arquivo com sua própria chave privada.

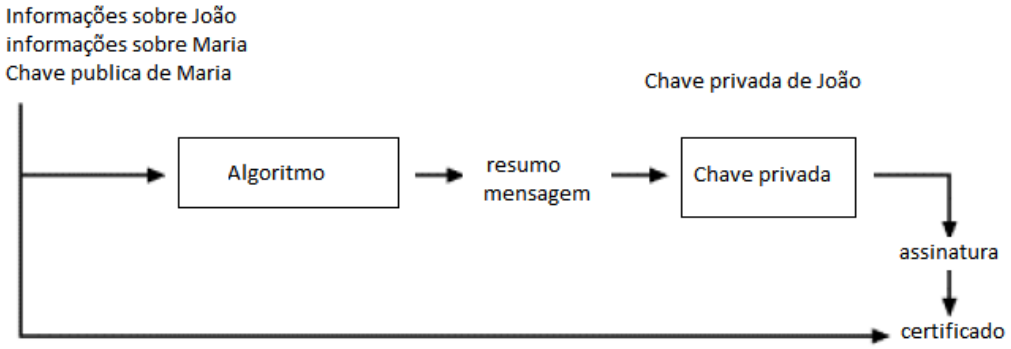


Figura 18 – Criação de certificado

Como a Figura 18 mostra, as informações sobre João e Maria, assim como a chave publica de Maria, são colocadas diretamente no certificado. Estas mesmas informações são assinadas por João. O resultado da assinatura é colocado no certificado com o resto dos dados. Agora para verificar a autenticidade do certificado basta que:

1. Calcular uma mensagem digerir para o conteúdo certificado.
2. Decodificar a assinatura usando a chave de João. O resultado é um resumo da mensagem.
3. Comparar o resumo mensagem decifrada e o resumo da mensagem calculado. Se forem iguais, o certificado é válido e a chave publica de Maria no certificado já pode ser usada.

#### 2.4.4.2 Normas de segurança para XML

XML *Signature* é um método para associar uma chave a dados referenciados (octetos), normativamente não especifica se as chaves serão associadas a pessoas ou instituições, nem o significado dos dados está sendo referenciado e assinado [9]. XML *encryption*, é uma especificação que define como deve ser feita a criptografia dos dados em documentos XML. Através da especificação XML *encryption* pode ser feita a criptografia de

qualquer tipo de dados que seja definido como elementos XML. XML *Signature* [14] permite que os dados sejam assinados garantindo que a integridade dos dados.

XML *Encryption* foi desenvolvida pela organização *World Wide Web Consortium* (W3C) [15] formado por um grupo de trabalho composto por varias empresas como, Microsoft, Motorola, IBM, Sun Microsystems, VeriSign , BEA Systems, entre outros. XML *Encryption* proporciona regras e sintaxes para criar e representar criptografia para garantir o sigilo em transações baseadas em XML.

*XML Key Management Specification* (XKMS) usa a estrutura de serviços web para tornar mais fácil para os desenvolvedores à intercomunicação de aplicações utilizando infra-estrutura de chave pública (PKI) [16]. XKMS é um protocolo desenvolvido pelo W3C que descreve a distribuição e registro de chaves públicas. Os serviços podem acessar um servidor XKMS compatível para receber informações atualizadas chave para a criptografia e autenticação.O XKMS é formado por duas partes [16]:

XML *Key Information Service Specification* (XKISS)

XML *Key Registration Service Specification* (XRSS)

A especificação do serviço XKISS tem como principal objetivo a gestão das chaves públicas. Enquanto o XKRSS é uma especificação que tem como objetivo a gestão das chaves privadas. Em ambos os casos o objetivo de XKMS é permitir que toda a complexidade das implementações tradicionais PKI possa ser transferida do cliente para um serviço externo.

### ***Security Assertion Markup Language* (SAML)**

SAML é um padrão baseado em XML para troca de dados sobre autenticação e autorização entre domínios de segurança, ou seja, entre um provedor de identidade (*identity provider*) e um fornecedor de serviços (*service provider*) [19]. O problema mais importante que o SAML está tentando resolver é fazer com que usuário só precise se autenticar uma única vez e possa usar outros sistema sem a necessidade de se autenticar novamente em cada um deles.

O SAML tenta solucionar um importante problema para as aplicações distribuídas, que é a possibilidade de transportar seus direitos de um usuário ou sistema entre diferentes *Web Services*. Isto é importante para aplicações que tencionam integrar um número de *Web Services* para formar uma aplicação unificada.

A especificação da SAML é organizada com uma abordagem modular e baseia-se em quatro blocos de construção.

Asserções descrevem as instruções sobre um usuário ou sistema, que é chamado de principal. As asserções são geralmente criadas por um provedor de identidade (*identity provider* ou IDP) e contêm instruções que provedores de serviço usam para tomar decisões de controle de acesso. Três tipos de instruções são fornecidos pela SAML:

- Instruções de autenticação;
- Instruções de atributo;
- Instruções de decisão de autorização.

As instruções de autenticação declaram ao provedor de serviço que o principal se autenticou no provedor de identidade em um dado momento, usando um método bem definido de autenticação. Outras informações sobre o método de autenticação usado pelo principal (chamado contexto de autenticação) podem ser incluídas em uma instrução de autenticação.

Uma instrução de atributo é simplesmente um par nome-valor. Partes dependentes usam atributos para tomar decisões de controle de acesso de baixa granularidade.

Uma instrução de decisão de autorização declara que um usuário ou sistema está autorizado a realizar a ação “A” no recurso “R” dada o atributo “E”.

Os protocolos descrevem como as asserções são empacotadas na solicitação SAML e elementos de resposta. Correspondendo aos três tipos de instrução, há três tipos de consulta SAML:

- Consulta de autenticação;
- Consulta de atributo;
- Consulta de decisão de autorização.

Ligações são mapeamentos de uma mensagem de protocolo SAML em formatos de sistema de mensagens padrão e/ou protocolos de comunicação. Por exemplo, a ligação SOAP SAML especifica como uma mensagem SAML é encapsulada em um envelope SOAP, sendo ele mesmo ligado a uma mensagem HTTP.

Perfis descrevem em detalhes como as asserções, protocolos e ligações SAML se combinam.

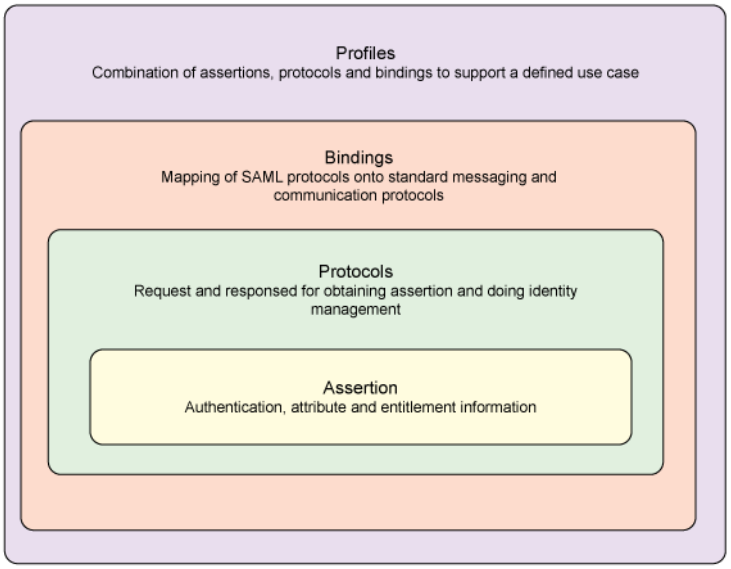


Figura 19 Estrutura do SAML

Como mostra a Figura 19. Cada bloco tem uma hierarquia onde as Asserções são os itens mais básicos, os Protocolos usam a asserções para descrever com será cria a requisição SAML, as Ligações descrevem como os Protocolos devem ser usados para os tipos de mensagens e por fim todos esses blocos combinados formam um Perfil.

**XACML (*eXtensible Markup Language Access Control*).**

É uma linguagem declarativa para políticas de controle de acesso implementadas em XML e um modelo de processamento, descrevendo a forma de interpretar as políticas [21].

As políticas são definidas com um conjunto de regras. Ambas as regras e os pedidos usam assuntos, recursos e ações,onde:

- Um assunto é um elemento que requisita acesso. Um assunto tem um ou mais atributos.
- Um recurso é um dado, serviço ou componente de sistema. Um recurso tem um único atributo.
- Uma ação define tipos de acesso requisitado a um recurso. Uma ação pode ter um ou mais atributos.

Na próxima subseção serão apresentadas as normas de segurança para *Web Services*, essas normas usam como base para usa descrição as normas de segurança para XML apresentadas.

### 2.4.4.3 - Tecnologias de segurança para *Web Services*

O WS-Security define como aplicar diferentes técnicas para autorização, integridade em um *Web Services* privado [26]. Descreve um meio padrão para embutir informações sobre segurança com *tokens*, criptografia, assinaturas digitais, SAML, *tokens* Kerberos em um header de uma mensagem SOAP.

```
<SOAP-ENV:Header>
 <wsse:Security  xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
        <wsse:UsernameToken>
            <wsse:Username>usuario</wsse:Username>
            <wsse:Password>senha</wsse:Password>
        </wsse:UsernameToken>
 </wsse:Security>
</SOAP-ENV:Header>
```

Figura 20 - Exemplo de cabeçalho SOAP com WS-security

A Figura 20 mostra um cabeçalho SOAP onde foi incluído um novo conjunto de tags. Essa tags são especificados pelo WS-Security, no exemplo foi incluído um token do tipo Username indicado pela tag wsse:UsernameToken . Este token carrega dois itens, no caso nome do usuário e senha, indicados pelas tags wsse:Username e wsse:Password respectivamente.

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

O WS-Security oferece mecanismos de segurança necessários para realizar a troca segura de mensagens certificadas em um ambiente de *Web Services.* Ele pode ser usado, para fornecer um mecanismo de autenticação no qual é possível passar diretamente credenciais e tokens entre clientes SOAP.

O WS-Security define como garantir a integridade, a confidencialidade e a autenticação das mensagens com o sistema de transmissão de mensagens SOAP, através do uso de tokens, mantendo essas informações de segurança no cabeçalho da mensagem SOAP. A integridade da mensagem é obtida com Assinaturas digitais XML.

Isso garante que partes da mensagem não tenham sido adulteradas após a assinatura feita na origem. A confidencialidade da mensagem é baseada na especificação de criptografia XML, usando XML-Encryption e XML-Signature explicados na subseção 2.4.4.2, assim garante que as mensagens só possam ser compreendidas pelo destinatário desejado.

O padrão WS-Security define como utilizar tecnologias existente para poderem ser usadas na construção de aplicação segurança, mas existem muitas tecnologias que pode ser usadas junto com o WS-Segurity, assim foi criada uma norma para especificar os tipos de tecnologias para segurança são exigidos por uma aplicação.

O WS-SecurityPolicy oferece mecanismos para representar as capacidades e necessidades de *Web Services* como políticas. Um provedor de serviço pode definir que, por exemplo, aceita X.509 [17] e Kerberos [18] e exige alguns assinatura específica.

O usando WS-Security nas mensagens há um aumento no numero de informações durante a troca de mensagem, esse fato em sistema onde existe um grande trafego de mensagem pode fazer com que haja uma queda no desempenho. Por isso motivo foi criada a norma WS-SecureConversation, que ajuda diminuir o volume de informações nas mensagens

A principal motivação para a criação WS-SecureConversation é melhorar o desempenho reduzindo o número de informações de sobre segurança que trafegam nas mensagens, para isso ele estabelece um contexto de segurança compartilhada entre trocas de mensagens múltiplas, assim não é necessário o envio de todas as informações sobre segurança em cada mensagem.

Com a criação de WS-Security, WS-SecurityPolicy e WS-SecureConversation, já pode criar um contexto de segurança das trocas das mensagens, mas para gerenciar esse essas informações e garantir que a validade das informações foi criada a norma *WS-Trust*.

*WS-Trust* é uma especificação que possibilita extensões para o WS-Security, que tratam especificamente com a emissão, renovação e validação de tokens de segurança, bem como as formas de estabelecer, avaliar a presença e a confiança entre os participantes em uma troca de mensagens seguras.

Com uso de WS-Security, WS-SecurityPolicy e WS-SecureConversation e *WS-Trust* podemos configurar um domínio de segurança onde o usuário pode acessar com segurança os sistemas disponíveis dentro desse domínio. Entretanto em uma organização pode ser necessário que um usuário use sistemas que estejam em outro domínio de segurança com regras e métodos de autenticação diferentes. Então surge o problema de como criar um mecanismo de interação entre esses domínios, para isso foi criada a norma WS-Federation.

Uma federação é um conjunto de domínios de segurança (*realms*) que estabelecem relações de compartilhamento de recursos de forma segura. Um provedor de recursos em um domínio pode fornecer acesso autorizado a um recurso que gere com base em afirmações sobre um atributo (como identidade) de um principal, que são invocados por um provedor de identidade.

A especificação *WS-Federation* define mecanismos que permitem integração entre federações de domínios de segurança e confiança diferentes, permitindo exista uma intermediação de identidades, atributos e autenticação entre esses domínios [22].

# 3. ANÁLISE DE ABORDAGENS PARA SEGURANÇA EM SERVIÇO

Neste capítulo são discutidas as abordagens implementadas para análise do nível de segurança que pode ser aplicada em aplicações orientadas a serviços. O objetivo dessa análise é determinar a solução mais adequada para ser implantada no Centro de Processamento de Dados (CPD) da Universidade Federal de Sergipe (UFS). Para essa análise foi construída um protótipo de uma aplicação que utiliza diferentes abordagens e combinações de mecanismos de segurança. Para cada uma das versões construídas é feita uma análise de custos e benefícios da solução. A Seção 3.1 mostra a primeira versão que não possui mecanismos de segurança no nível de transporte e no nível de mensagem.Na Seção 3.2 é incluído um mecanismo para segurança de transporte, na seção 3.3 é mostrado como acrescentar um método para autenticação usando recursos do próprio servidor e também é feita uma analise sobre a segurança no nível de transporte implementados nas versões das seções 3.1 e 3.2. Na Seção 3.3, são feitas alterações para que a segurança passe a ser feita no nível de mensagens e na Seção 3.4 é mostrado com configurar os serviços para garantir a autenticidade das mensagens e logo após é realizada uma análise sobre as técnicas de segurança em mensagens apresentada nas seções 3.3 e 3.4.

Foi usada a plataforma Java para o desenvolvimento da aplicação protótipo, utilizando o JDK (*Java Development Kit*) 1.6 [28]. Para servidor de aplicação foi utilizado o JBoss-5.1.0.GA [27]. As ferramentas IDE que auxilia o desenvolvimento da aplicações foram o Eclipse [29] e Netbeans 6.8 [30] rodando no sistema operacional Windows 7 [31].

A implementação dos *Web Services* foi realizada usando a IDE Eclipse em quanto o a interface gráfica da aplicação foi desenvolvida no Netbeans.

Todos os códigos da aplicação de teste esta disponível na URL http://code.google.com/p/soaclient/source/checkout.

## 3.1- Sem Segurança

Para realização da análise dos diferentes mecanismos de segurança que poderiam ser adotados no CPD da UFS foi desenvolvida uma pequena aplicação que simula uma aplicação de vendas de produtos. Foram desenvolvidas também funcionalidades para manutenção de dados clientes e produtos usados na aplicação. Todas as operações em relação à persistência de dados são feitas atrás do uso de *Web Services*.

Em relação à parte de persistência, todas as operações no servidor usam o framework Hibernate [24], que é uma das principais ferramentas Java para fazer a persistência de dados. A utilização desse framework já resolve o problema de um dos ataques citados na seção 2.4.1, o *SQL Injection*, já que nas operações são utilizados métodos disponibilizados pelo framework que validam os dados, evitando o que sejam inseridas instruções SQL dentro dos valores dos objetos.

Outra característica da versão é que nenhum serviço requer qualquer tipo de identificação para uso, eles são totalmente públicos e para ter acessos aos dados basta que qualquer aplicações implemente as chamadas aos *Web Services* disponibilizados.

Para facilitar a explicação das técnicas utilizadas, é utilizada somente uma das telas do sistema que contém os serviços relacionados com a manutenção dos dados de produtos.

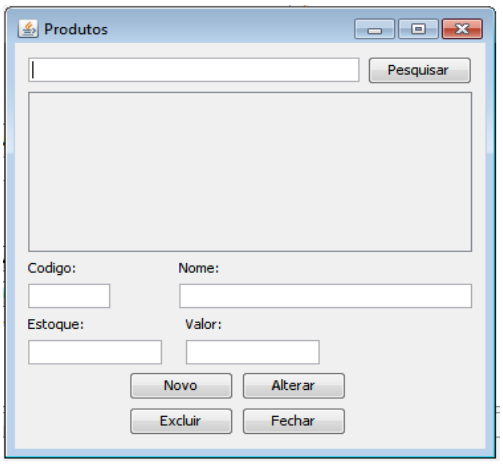


Figura 21 - Tela para manutenção de dados sobre produtos

A Figura 21 mostra a tela da aplicação cliente que gerencia os dados sobre produtos, nessa versão como já foi explicado, toda comunicação entre essa tela e o *Web Service* que mantém os dados dos produtos é feita sem qualquer tipo de mecanismos de segurança, tanto no nível de transporte quanto no nível de mensagem.

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:web="http://webservice.tcc.ufs/">
   <soapenv:Header/>
   <soapenv:Body>
      <web:SalvarProduto>
         <produto>
            <nomeProduto>Lápis</nomeProduto>  (1)
            <unidadesEstoque>15</unidadesEstoque> (2)
            <valorUnitario>2.5</valorUnitario> (3)
         </produto>
      </web:SalvarProduto>
   </soapenv:Body>
</soapenv:Envelope>
```

Figura 22 - Mensagem SOAP que solicita uma lista dos produtos cadastrados

A Figura 22 mostra a mensagem que é enviada ao *Web Services* para salvar um registro de produtos. Como pode ser visto nos pontos 1, 2 e 3, todas a informações sobre produtos estão expostas,a mensagem é envidada sem nenhum tipo de proteção podendo ser interceptado e alterada durante o envio.

Nessa primeira versão da aplicação como mostrado, não foi incluído nenhum mecanismo para garantir segurança na troca de mensagens. Esse tipo de implementação, mesmo sem garantias de segurança, pode ser usado dependendo nível de importância da informação trocada na mensagem. Um exemplo para a utilização de serviço que poderia ser usado sem implementar segurança seria algo como uma consulta de CEP, pois esse tipo de informação não expõem nenhum tipo de informação importante para a organização e não compensaria o custo.

Nas próximas seções serão mostras técnicas para garantir a segurança das informações nos níveis de transporte e mensagem.

## 3.2- Segurança no Nível de Transporte

Essa seção mostra as alterações que foram realizadas no projeto dos *Web Services* para que seja incluído um mecanismo de segurança no nível de transporte, explicada na Seção 2.4.2.Após as alterações realizadas, os *Web Services* utilizados na aplicação conseguem confidencialidade e integridade garantidas durante a comunicação.

Nessa versão, para implementar a segurança no nível de transporte foram utilizados os recursos de servidor de aplicação, no caso o servidor JBOSS, para criar uma conexão segura usando SSL/TLS. Como explicado na Seção 2.4.2 essa solução permite que as mensagens não sejam alteradas durante o transporte, garantindo dessa forma a integridade, além da confidencialidade das informações.

Para implementar essa versão foi necessário configurar o servidor para permitir o uso de conexões SSL/ TLS, já que esse tipo de conexão não vem configurada por padrão.

O primeiro passo e desabilitar o comentário do seguinte trecho de código no arquivo server.xml que fica na pasta de instalação do JBoss/deploy/JBossweb.sar

```
<Connector port="8443"
  ...
  scheme="https"          (1)
  secure="true"           (2)
  clientAuth="false"      (3)
  keystoreFile="${jboss.server.home.dir}/conf/server.keystore" (4)
  keystorePass="serverpass"      (5)
  sslProtocol = "TLS" />      (6)
```

Figura 23 – Trecho de Código do arquivo server.xml

A Figura 23 mostra a configuração realizada no arquivo server.xml, Após desabilitar o comentário pode-se deixar os valores padrão, mas é geralmente aconselhável alterar o atributo port que configura a porta do serviço de 8443 para 443 que é a porta padrão para comunicação SSL. Agora é necessário configurar a chave para o servidor. O atributo scheme (1) define o protocolo que esta sendo usado , no caso HTTPS, o atributo secure (2) informa ao JBoss que esta sendo configurado um conector de seguro.Definir o atributo (4) ClientAuthcomo false faça com que o usuário não precise ser autenticar com certificados .Os atributos keystoreFile (4) e keystorePass (5) definem a localização do arquivo de keystore contendo o certificado do servidor e a senha para o arquivo keystore, respectivamente.

Para criar a chave para o teste foi usada a ferramenta keytool [25] que já vem disponível no JDK. O seguinte comando pode ser usado para criar a chave:

keytool-genkey-keystore chap8.keystore -storepass senha -keypass senha -keyalg RSA -alias server -validity3650–dname “cn=nsi, ou=ufs,c=itabaiana, s=se, c=br"

Onde:

**keytool**: comando, nome do aplicativo

**-genkey**: parâmetro para criação das chaves

**-keystore**: indica o caminho/nome do keystore (*chap8.keystore*)

**-storepass**: indica a senha do keystore (*senha*)

**-keypass**: indica a senha da chave (*senha*)

**-keyalg**: indica o algoritmo de criptografia utilizado (*RSA*)

**-alias**: indica o nome da chave a ser criada dentro do keystore (*server*)

**-validity**: Validade do certificado em dias (*3650*)

**-dname "[...]"**: Informações do certificado (CN = nome comum, OU = unidade organizacional (departamento, divisão), O = nome da organização, L = nome da localidade (cidade), S = estado, C = código do país)

O próximo passo é mover a chave criada para a pasta indicada na configuração do conector SSL, keystoreFile= "$ {JBoss.server.home.dir} /conf/". Como essas configurações o servidor já pode fornecer o serviço de conexões através do SSL/TLS.

O próximo passo é configurar os serviços para utilizarem a segurança no nível de transporte disponibilizada pelo servidor. Os *Web Services* usados no momento estão configurados par o uso de conexões HTTP comuns. Para alterar as configurações para o uso de conexões HTTPS, que usam SSL/TSL, basta fazer algumas alterações no arquivo web.config do serviço.

```
<security-constraint>
    <web-resource-collection>
        <web-resource-name>WebSoaServicesSeguro</web-resource-name> (1)
        <url-pattern>/Produto</url-pattern> (2)
        <http-method>POST</http-method>  (3)
    </web-resource-collection>
    <user-data-constraint>
            <transport-guarantee>CONFIDENTIAL</transport-guarantee> (4)
    </user-data-constraint>
</security-constraint>
```

Figura 24 - Configuração do serviço para HTTPS

A Figura 24 mostra o código que tem que ser adicionado ao web.xml para que a comunicação seja feita com o uso de protocolos seguros.O ponto (1) representa o nome do recurso, a tag (2) indica a URL do recurso a ser protegido, a tag (3) especifica tipo de requisição usada. A tag trasnport-guarantee (4) especifica que o serviço vai usar conexões seguras, no caso o conector HTTPS configurando no servidor.

Com essa configuração o serviço já pode garantir a integridade e a confidencialidade das trocas de mensagens, mas não apresenta nenhum tipo de mecanismos de autenticação.

No cliente, para que a aplicação possa se conectar ao servidor através do SSL foi necessário incluir o seguinte código.

```
ProdutoWSService soaService = new ProdutoWSService();
produtoWS = soaService.getProdutoWSPort();
BindingProvider bp = (BindingProvider) produtoWS;
Map<String, Object> rc = bp.getRequestContext(); (1)
rc.put(BindingProvider.SOAPACTION_URI_PROPERTY,
                "https://localhost/WebSoaServices/Produto"); (2)
return produtoWS;
```

Figura 25 – Código que retorno um ProdutoWSService

A Figura 25 mostra o código usado para a criação de um ProdutoWSService que será usado na aplicação cliente para chamar os métodos do *Web Service* Produtos, no ponto (1) é criando uma variável Map onde é armazenado o endereço do serviço, ponto (2).

### 3.3- Segurança no nível de Transporte com Autenticação no Servidor

Nessa seção será mostrado como alterar a versão apresentada na Seção 4.2 que já possui mecanismos de segurança no transporte garantindo a integridade e confidencialidade, para que possa ser incluído um mecanismo de autenticação.

Para implementar a autenticação são utilizados recursos do próprio servidor de aplicações, JBoss.O fato de o próprio servidor fornecer formas para incluir autenticação é algo extremamente útil, pois o desenvolvedor tem a opção de delegar essa função para o próprio servidor evitando a preocupação com a criação de módulos de segurança que tratem da autenticação de usuário.

O JBoss pode ser configurado para exigir algumas formas de autenticação com usuário e senha, entre outros. No servidor também é possível criar perfis e usuários que serão utilizados para liberar o acesso aos serviços que estão sendo executados. Esses usuários podem ser armazenados de diversas formas, como banco de dados, por exemplo, mas para simplificar no protótipo construído foram utilizados arquivos texto.

Para configurar a autenticação no servidor é necessário que sejam feitas algumas alterações no arquivo login-config.xml que fica localizado na pasta de instalação do JBoss dentro da pasta /conf/.

```
<application-policy name="JBossWS">
  <authentication>
    <login-module code="org.jboss.security.auth.spi.UsersRolesLoginModule"
      flag="required">
      <module-option name="usersProperties">./users.properties</module-option>
      <module-option name="rolesProperties">./roles.properties</module-option>
      <module-option name="unauthenticatedIdentity">anonymous</module-option>
    </login-module>
  </authentication>
</application-policy>
```

Figura 26 - Trecho do arquivo login-config.xml alterado

A Figura 26 mostra a configuração do arquivo login-config.xml. Na *tags* modulo-option, como atributo *name* igual à *usersProperties* é indicado o local onde esta o arquivo com as informações sobre contas dos usuários, na outra tag com atributo name

*rolesProperties* é indicada a localização do arquivo que registra informações sobre os perfis que são associados aos usuários.

No arquivo que guarda informações sobre usuários, *users.properties*, o armazenamento é feito de maneira em que cada linha exista um registro no formato "usuário=senha", enquanto no arquivo que armazena os registros de perfis (*Roles*), *roles.properties*, o formato é "usuário=perfil".

Esse tipo de gerenciamento para usuário foi escolhido por ser uma forma mais simples para fazer a configuração de autenticação no servidor, ela atende as necessidades da aplicação de testes que foi criada, mas com já foi mencionado existem outros formas para armazenar essas informações, tudo vai depender da necessidade da aplicação.

```
usuario1=vendedor
usuario2=vendedor
usuario3=gerente
```

Figura 27 – Exemplo arquivo *roles.properties*

A Figura 27 mostra, como é configurado o arquivo *roles.properties*, em cada linha é colocada um nome de usuário e um perfil que será associado a ele

```
usuario1=senha1
usuario2=senha2
usuario3=senha3
```

Figura 28 -Exemplo arquivo *users.properties*

A Figura 28 mostra, como é configurado o arquivo *users.properties*, em cada linha é colocada um nome de usuário e sua senha.

O uso desse tipo de gerencia de usuários usando arquivos de configuração é bem simples e não em muito eficiente para sistemas com muitos usuários, mas ele pode ser substituído por armazenamento em banco de dados.

Feitas as inclusões de perfis e usuários que serão utilizados, tudo que precisa ser feito são pequenas alterações no arquivo web.xml no projeto dos *Web Services* .

```
<security-constraint>
    <web-resource-collection>
        <web-resource-name>WebSoaServicesSeguro</web-resource-name>
        <url-pattern>/produto</url-pattern>
        <http-method>POST</http-method>
    </web-resource-collection>
    <auth-constraint>
        <role-name>Vendedor</role-name> (1)
    </auth-constraint>
    <user-data-constraint>
            <transport-guarantee>CONFIDENTIAL</transport-guarantee>
    </user-data-constraint>
</security-constraint>
<login-config>
    <auth-method>BASIC</auth-method>     (2)
    <realm-name>JBossWS</realm-name>     (3)
</login-config>
<security-role>
    <role-name>vendedor</role-name>     (4)
</security-role>
```

Figura 29 - Trecho do arquivo web.xml com a configuração da autenticação

A Figura 29 mostra um trecho do código que sofreu as alterações para incluir a autenticação. O ponto (1) mostrar que foi incluída uma restrição de acesso para o perfil “vendedor”, que foi adicionado no ponto (4), também foi incluída a tag login-config, essa tag indica qual elemento do arquivo login-config.xml no servidor será responsável por tratar os dados referentes à autenticação (2) e qual o tipo de método de autenticação será utilizado (3), sobre os métodos de autenticação existem outros:

FORM – Esse método é semelhante à autenticação básica, mas uma página HTML com um formulário de login é enviado para o navegador para o usuário fazer login.

DIGEST – Esse mecanismo de autenticação usa uma caixa de dialogo igual ao método BASIC, a diferença é que o hash, que gera um valor para a senha antes de ser enviada para o servidor. Como esse método é necessário que a senha seja armazenada já com o valor hash.  No JBoss já existe uma função a gerar os valores de hash das senha.

```
java -classpath jbosssx-server.jar
  org.jboss.security.auth.spi.RFC2617Digest usuario realm senha
```

Figura 30 – Código para gerar o valor de hash de uma senha

A figura 30 mostra como gerar um valor hash para uma senha, basta informar o nome do usuário, uma palavra que será usada para gerar o valor e por ultimo a senha do usuário, o arquivo jar jbosssx-server.jar se encontra na pasta common/lib dentro da pasta de instalação do JBoss.

CLIENT-CERT – Testa se o cliente tem algum certificado Digital.

Com essas pequenas modificações a autenticação já esta habilitada direto no servidor para o *Web Service* da aplicação.Na aplicação cliente para que a aplicação pudesse se conectar ao servidor através do SSL foi necessário incluir o seguinte código.

```
ProdutoWSService soaService = new ProdutoWSService();
produtoWS = soaService.getProdutoWSPort();
BindingProvider bp = (BindingProvider) produtoWS;
Map<String, Object> rc = bp.getRequestContext();
rc.put(BindingProvider.SOAPACTION_URI_PROPERTY,
                "https://localhost/WebSoaServices/Produto");
rc.put(BindingProvider.USERNAME_PROPERTY, "Usuario");        (1)
rc.put(BindingProvider.PASSWORD_PROPERTY, "Senha");          (2)
```

Figura 31 – Inclusão de usuário e senha para cliente *Web Service* Produtos

Na Figura 31 mostra que a única alteração sofrida no cliente foi a inclusão das duas ultimas linhas, pontos (1) e (2), onde são passados respectivamente usuário e senha, que serão validados no servidor, caso sejam validos a conexão será realizada.

O uso de segurança no nível de transporte é mostrado nas seções 3.1 e 3.2 garante a integridade e confidencialidade e ainda pode garantir a autenticidade como no exemplo da seção 3.2. Segurança no nível de transporte é uma técnica utilizada há muito tempo e tem um desempenho bom. Entretanto à medida que o numero de serviços e a quantidade de servidores onde esses serviços rodam vai aumentando, gerenciar a configuração de todos esses serviços em cada servidor pode se tornar uma tarefa complicada dependendo do número de servidores usados.

Esse tipo de mecanismo geralmente é aconselhável ser usado com *Web Services* onde a conexão é feita direto entre cliente e serviço, sem uso de serviços intermediário, já que teria que ser criada uma conexão entre o cliente e o serviço intermediário e entre o serviço intermediário e o serviço que irá tratar a mensagem do cliente.

Na próxima Seção começa a ser implementada a segurança no nível de mensagens.

### 3.4- Segurança no Nível de Mensagem com WS-Security

Como discutido na Seção 2.4.2, a segurança no nível de transporte tem algumas limitações em relação a flexibilidade com uso de múltiplos serviços, o que torna necessário em alguns casos que sejam implementados mecanismos de segurança no nível de mensagem.

Nesta seção é abordada a implementação da segurança em nível de mensagem explicando a implementação realizada no protótipo desenvolvido para realização da avaliação.

O primeiro requisito para implementação do mecanismo usado é possuir chaves pública e privada e os certificados digital, explicados na Seção 2.4.4.1, que são necessários para a criptografia das mensagens. Para criação das chaves foi utilizada novamente a ferramenta KeyTool, explicada na Seção 3.2.

Os seguintes comandos foram utilizados para criar os certificados de chave. É necessário guardar as senhas das chaves criadas através da execução do comando para serem utilizados na configuração das aplicações.

keytool -genkey -alias server -keyalg RSA -keystoreserver.keystore - Gerar o repositório com chave do servidor com nome server.keystore

keytool -genkey -alias client -keyalg RSA -keystoreclient.keystore - Gera repositório com chave do cliente com nome client.keystore

keytool -export -alias server -keystoreserver.keystore -file server_pub.key - Exporta a chave do servidor do repositório server.keystore para o arquivo server_pub.key

keytool -export -alias client -keystoreclient.keystore -file client_pub.key - Exporta a chave do cliente do repositório client.keystore para o arquivo client_pub.key

keytool -import -alias client -keystoreserver.keystore -file client_pub.key - Importa a chave pública client_pub.key para o repositório server.keystore

keytool -import -alias server -keystoreclient.keystore -file server_pub.key - Importa a chave pública server_pub.key para o repositório client.keystore

keytool -import -alias client -keystoreclient.truststore -file client_pub.key - Importa a chave pública client_pub.key para o repositório client.trustore

keytool -import -alias server -keystoreserver.truststore -file server_pub.key - Importa a chave pública server_pub.key para o repositório server.trustore

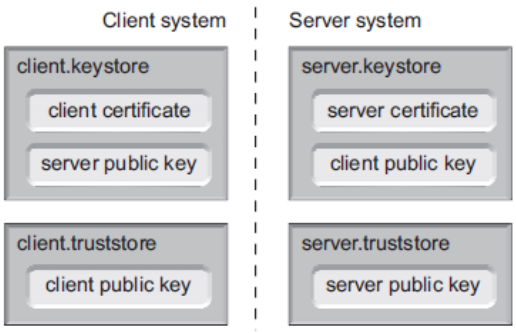


Figura 32 - Configuração das chaves e certificados do cliente e do servidor

A Figura 32 mostra como serão distribuídas as chaves e os certificados na comunicação.Os arquivos”.trustore" e”.keystore" são usados para armazenar de maneira segura chaves e certificados gerados, no arquivo client.keystore na aplicação do cliente ficam armazenados os certificado do cliente e a chave pública do servidor.Na aplicação do servidor, dentro do arquivo server.keystore, ficam o certificado do cliente e a chave pública do cliente. Por último nos arquivos server.truststore e client.truststore são armazenados chave pública do servidor e chave pública do cliente respectivamente.

Depois de criados certificados e chaves necessários, já se pode dar inicio às alterações nas aplicações. O primeiro passo para alterar os *Web Services* é a inclusão do arquivo jboss-wsse-server.xml e dos arquivos server.keystore e server.truststore dentro da pasta dentro da pasta WEB-INF no projeto.

```
<?xml version="1.0" encoding="UTF-8"?>
<jboss-ws-security>
  <key-store-file>WEB-INF/server.keystore</key-store-file> 1
  <key-store-password>SenhadoArquivoKeystore</key-store-password> 2
  <key-store-type>jks</key-store-type> 3
  <trust-store-file>WEB-INF/server.truststore</trust-store-file> 4
  <trust-store-password>SenhadoArquivoTruststore</trust-store-password> 5
  <trust-store-type>jks</trust-store-type> 6
  <key-passwords>
    <key-password alias="server" password="serverpwd" /> 7
  </key-passwords>
  <config>
    <encrypt type="x509v3" alias="client" /> 8
    <requires>
      <encryption /> 9
    </requires>
  </config>
</jboss-ws-security>
```

Figura 33- Conteúdo do arquivo jboss-wsse-server.xml

A Figura 33 mostra o arquivo jboss-wsse-server.xml, que é usado para a configuração da criptografia das mensagens.Na linha marcada com 1 mostra o local do arquivo keystore e na linha 4 mostra do arquivo truststore são relativos ao diretório base do arquivo WAR. O local onde indica as senhas dos arquivos keystore e truststore são as linhas marcadas com 2 e 5 respectivamente. As tags<key-store-type> marcadas no ponto 3 e as tags<trust-store-type> no ponto 5 indicam o padrão utilizado para criação dos arquivos keystore e truststore. A senha das chaves do servidor é indicada no ponto 7 pela tag<key-passwords>.Essa senha é usada para acessar o certificado de servidor no arquivo keystore. A tag<encryption/>, linha 9 pede que a mensagem seja criptografada utilizando o alias fornecido pelo tag<encrypt>na linha 8, assim a chave pública do cliente é usada para criptografar a

mensagem no servidor e é decifrado no cliente usando a chave privada do cliente que esta no arquivo keystore na aplicação cliente.

Depois de incluído esse arquivo, agora é necessário usar uma anotação na classe do *Web Service* para indicar que ele que espera que seja usado o padrão *WS-Security*, explicado na Seção 2.4.5.

```
@WebService()
@EndpointConfig(configName = "Standard WSSecurity Endpoint")
public class ProdutoWS {
        @WebMethod(operationName = "getProdutos")
        public List<Produto> getProdutos() {
            Session session = HibernateUtil.getSessionFactory().openSession();
            session.beginTransaction();
            HibernateDAO<Produto> clienteDao = new HibernateDAO<Produto>(Produto.class, session);
            return clienteDao.getBeans();
        }
```

Figura 34 - Trecho do código da classe do Web Service ProdutoWS

A Figura 34 mostra a alteração realizada na classe ProdutoWS, essa classe representa um *Web Service* que disponibiliza algumas operações com relação aos produtos usados pela aplicação de teste. Na classe foi incluída a anotação @EndpointConfig (configName = “Standard WSSecurityEndPoint”) que indica o nome da configuração no arquivo standard-jaxws-endpoint-config.xml, que será usada para manipular a troca de mensagens pelo servidor.

O arquivo standard-jaxws-endpoint-config.xml localizado na pasta de instalação do JBoss na subpasta \deployers\jbossws.deployer\META-INF, é quem configura no servidor qual classe irá manipular toda a parte referente a criptografia das mensagens do servidor. A tag 1 na figura indica o nome da configuração utilizado na anotação na classe do Web Service. A tag 2 indica o nome da classes que irá cuidar da parte de criptografia das mensagens.

```
<jaxws-config ...>
 ...
<endpoint-config>
<config-name>Standard WSSecurity Endpoint</config-name> (1)
 <post-handler-chains>
  <javaee:handler-chain>
  <javaee:protocol-bindings>##SOAP11_HTTP</javaee:protocol-bindings>
   <javaee:handler>
    <javaee:handler-name>WSSecurity Handler</javaee:handler-name>
    <javaee:handler-class>
 ➥org.jboss.ws.extensions.security.jaxws. (2)
 ➥WSSecurityHandlerServer
    </javaee:handler-class>
   </javaee:handler>
  </javaee:handler-chain>
 </post-handler-chains>
</endpoint-config>
</jaxws-config>
```

Figura 35 - Trecho do arquivo standard-jaxws-endpoint-config.xml

Agora é necessário fazer a configuração do lado cliente, para isso é necessário fazer alterações muito parecidas com as realizadas no servidor.

O primeiro passo para alterar o projeto cliente é a inclusão do arquivo jboss-wsse-client.xml e dos arquivos client.keystore e client.truststore dentro da pasta META-INF no projeto.

```
<jboss-ws-security xmlns="http://www.jboss.com/ws-security/config"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.jboss.com/ws-security/config
  http://www.jboss.com/ws-security/schema/jboss-ws-security_1_0.xsd">
  <key-store-file>META-INF/client.keystore</key-store-file>
  <key-store-password>SenhaArquivoKeystore</key-store-password>
  <key-store-type>jks</key-store-type>
  <trust-store-file>META-INF/client.truststore</trust-store-file>
  <trust-store-password>SenhaArquivoTruststore</trust-store-password>
  <trust-store-type>jks</trust-store-type>
  <key-passwords>
    <key-password alias="client" password="clientpwd" />
  </key-passwords>
  <config>
    <encrypt type="x509v3" alias="server" />
    <requires>
      <encryption />
    </requires>
  </config>
</jboss-ws-security>
```

Figura 36 - Conteúdo do arquivo jboss-wsse-client.xml

A Figura 36 mostra o arquivo jboss-wsse-client.xml, a descrição das tags são iguais às mostras na Figura 33. Nele será necessário indicar o local (tag key-store-file) e senha (tag key-store-password) do arquivo client.keystore, também será indicado o local (tag trust-store-file) e senha (tag trust-store-password) do arquivo client.trustore e a senha do certificado (tag key-password).

É necessário incluir o arquivo standard-jaxws-client-config.xml dentro da pasta META-INF/ do projeto cliente. Esse arquivo é uma cópia do arquivo standard-jaxws-client-config.xml encontrado dentro da pasta de instalação do JBoss na subpasta \deployers\jbossws.deployer\META-INF.

```
<?xml version="1.0" encoding="UTF-8"?>
<jaxws-config xmlns="urn:jboss:jaxws-config:2.0"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:javaee="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="urn:jboss:jaxws-config:2.0 jaxws-config_2_0.xsd"
>
<client-config>
 <config-name>Standard WSSecurity Client</config-name>
 <post-handler-chains>
  <javaee:handler-chain>
  <javaee:protocol-bindings>##SOAP11_HTTP</javaee:protocol-bindings>
   <javaee:handler>
<javaee:handler-name>WSSecurityHandlerOutbound</javaee:handler-name>
    <javaee:handler-class>
➥org.jboss.ws.extensions.security.jaxws.      ❶
➥WSSecurityHandlerClient
    </javaee:handler-class>
   </javaee:handler>
  </javaee:handler-chain>
 </post-handler-chains>
</client-config>
</jaxws-config>
```

Figura 37 -Trecho do arquivo standard-jaxws-client-config.xml

A Figura 37 mostra o arquivo standard-jaxws-client-config.xml é quem indica a configuração no servidor qual classe irá manipular toda a parte referente a criptografia das mensagens no vindas do cliente. O tag1 na figura indica o nome da classe que irá cuidar da parte de criptografia das mensagens. Poder ser nesse arquivo podem ser definidas outras classe para manipular as mensagem bastando que classe escolhida estenda a classe org.jboss.ws.core.jaxws.handler.GenericSOAPHandler.

Com essas alterações, toda a troca de mensagens entre cliente e servidor será criptografada usando as chaves públicas e privadas, do cliente e do servidor, essa técnica garante que confidencialidade e integridade das mensagens, a desvantagem, é que o tamanho das mensagens aumenta. Outro ponto relevante é o custo de processamento das mensagens também elevado devido à criptografia das mensagens. Logo é necessário fazer uma analise das reais necessidades de segurança das aplicações antes de incluir mecanismos de criptografia, pois isso pode implicar em queda no desempenho da aplicação.

### 3.5- Segurança no Nível de Mensagem com Assinatura Digital

Nessa seção será mostrado umas das maneiras disponíveis no padrão WS-Security para garantir a identidade do usuário a de uma maneira para usar recursos do JBoss para a autorização dos usuários ao uso dos serviços.

WS-Security fornece um mecanismo para assinar uma mensagem usando *XML Signature*, explicado na seção 2.4.4, proporcionando um meio alternativo de autenticar o usuário.Para assinar uma mensagem, o remetente usa sua chave privada, e o receptor usa chave pública do remetente para verificar a identidade do remetente. Na implementação da aplicação de teste será usada a autenticação do usuário usando certificados digitais.

A implementação com certificados foi escolhida por ser bastante simples de ser feita e também por permitir reaproveitar os as chaves criadas na Seção 4.4. A primeira alteração que será realizada será a importação das chaves públicas para os arquivos truststore do cliente e *Web Service.* Os seguintes comandos fazem essa importação.

keytool -import -alias server -keystoreclient.truststore  -file server_pub.key

Importa a chave publica do servidor para o arquivo client.truststore

keytool -import -alias client -keystoreserver.truststore  -file client_pub.key

Importa a chave publica do cliente para o arquivo server.truststore

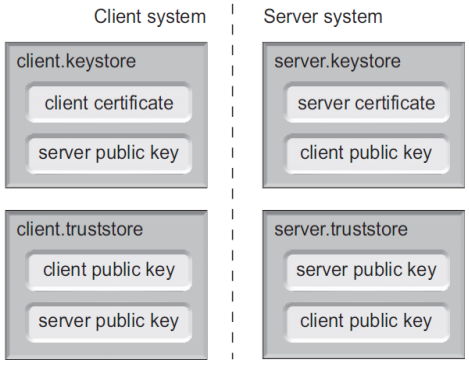


Figura 38 Nova configuração das chaves e certificados do cliente e do servidor

Como pode ser visto na Figura 38, a única diferença na configuração das chaves e certificados em relação à Figura 32, foi a inclusão da chave pública do servidor dentro da arquivo client.truststore e da chave pública do cliente dentro do arquivo server.truststore no projeto do *Web Service.*

Depois de armazenar as chaves públicas, é necessário fazer um pequena alteração no arquivo jboss-wsse-server.xml.

```
<jboss-ws-security>
      .
      .
      .
  <config>
    <sign type="x509v3" alias="server" />
    <encrypt type="x509v3" alias="client" />
    <requires>
      <signature />
      <encryption />
    </requires>
  </config>

</jboss-ws-security>
```

Figura 39 - Alterações no arquivo jboss-wsse-server.xml

A Figura 39 mostra a alteração que deve ser feita para incluir a assinatura da mensagem. Dentro da tag<config> na tag<requires>, é incluída a tag<signature/> que indica que é necessário que a mensagem seja assinada para que seja valida, na tag<sign> é usada para informar que as mensagens enviadas pela *Web Services*serão assinadas usando a chave "server" do serviço.

Agora no lado cliente é só fazer uma alteração parecida com a feita no servidor. É necessário alterar o arquivo jboss-wsse-client.xml.

```
<jboss-ws-security>
      .
      .
      .
  <config>
    <sign type="x509v3" alias="client" />
    <encrypt type="x509v3" alias="server" />
    <requires>
      <signature />
      <encryption />
    </requires>
  </config>

</jboss-ws-security>
```

Figura 40 - Alteração do arquivo jboss-wsse-client.xml no projeto cliente

A Figura 40 mostra a alteração feita no arquivo jboss-wsse-client.xml, assim como no arquivo jboss-wsse-client.xml, no qual foram incluídas as tags<signature/> para definir que a mensagem recebida deve ser assinada e a tag<sign> indicando que deve ser usada a chave client para assinar a chave.

Depois dessas alterações, tanto o Web Service quanto a aplicação cliente já estão configurados para exigirem a assinatura das mensagens durante a comunicação o que garante a identidade de que esta enviando as mensagens. A implementação desse mecanismo de segurança garante a integridade e confidencialidade, e o não repúdio das mensagens.

Uma solução simples para o uso de permissão seria o uso da técnica explicada na Seção 4.3, usando os recursos do próprio JBoss para implementar um sistema simples de controle de acesso a recursos do servidor.

Um problema que deve ser muito bem analisado antes do uso de técnicas para segurança no nível de mensagem, é o aumento no tamanho das mensagens. Uma mensagem simples quando criptografada aumenta muito de tamanho isso poder prejudicar muito o desempenho das aplicações.

```
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
   <env:Header/>
   <env:Body>
      <ns2:getProdutosResponse xmlns:ns2="http://webservice.tcc.ufs/">
         <return>
            <idProduto>13</idProduto>
            <nomeProduto>caneta</nomeProduto>
            <unidadesEstoque>120</unidadesEstoque>
            <valorUnitario>3.0</valorUnitario>
         </return>
      </ns2:getProdutosResponse>
   </env:Body>
</env:Envelope>
```

Figura 41 – Mensagem com resultado de uma pesquisa por produto

A Figura 41 mostra a mensagem SOAP que trás o resultado não criptografado de uma pesquisa por um produto.

```
< env:Header>
 <wsse:Security env:mustUnderstand='1' xmlns:ds='http://www.w3.org/2000/09/xmldsig#'
  xmlns:wsse='http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd'
  xmlns:wsu='http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd'>
   <wsse:BinarySecurityToken
   EncodingType='http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#Base64Binary'
   ValueType='http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3'
   wsu:Id='token-3-1291211260668-28347194'>
   MIICaDCCAdECBEcEAQ0wDQYJKoZIhvcNAQEEBQAwezELMAkGA1UEBhMCWFgxDjAMBgNVBAgTBUVh
cnRoMREwDwYDVQQHEwhBbnl3aGVyZTEYMBYGA1UEChMPSkJvc3MgaW4gQW90aW9uRzEwDQYDVQQL
ExJKQm9zcyBXEWIgU2VydmljZXMxEjAQBgNVBAMTCVNhbGVzIFRheDAeFw0wNzEwMDMyMDU0MTFa
Fw0xMTExMTEyMDU0MTFaMHsxCzAJBgNVBAYTAlhYMD4wDAYDVQQIEwVFYXJ0aDERMA8GA1UEBxMI
QW55d2hlcmUxGDAWBgNVBAoTD0pCb3NzIGluIEFjdGlvbjEbMBkGA1UECxMSSkJvc3MgV2ViIFN1
cnZpY2VzMRIwEAYDVQQDEwlTYWxlcyBUYXgwgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAIdT
xV6TI4I8xj1kXjMKVchAUaM2gbLndi4F6LL/RanpMSKU7egEMPLhSaz/C2KPQz5Y94EmOXZfY7nU
YqFbfLTU3XA6SaHFuGlSlhQaBkDL/qOO6FWGhDJ8PRIkvh7QUzzAOrgx7dWY78m6ODLv8N2ol/do
wLXCHOC9dJQTAHiXAgMBAAEwDQYJKoZIhvcNAQEEBQADgYEAHlYHxSA+YGt44UNr9RRED3JboDKw
CfE92CdrwtaFF1ZbN+GRVCFoGa5ydzWyR8nAb9iXLA5/Kd9HhSTKidtfjeaM8zCvOc5nxveas/pM
VH/LfEMDQ5rCmiVUwWOMQpVJSeem96whnOZS/nM9/tYbEECXDO+QnYxOXtT5U20Wm2A=</wsse:BinarySecurityToken>
   <xenc:EncryptedKey xmlns:xenc='http://www.w3.org/2001/04/xmlenc#'>
    <xenc:EncryptionMethod Algorithm='http://www.w3.org/2001/04/xmlenc#rsa-1_5'
    xmlns:xenc='http://www.w3.org/2001/04/xmlenc#'/>
    <ds:KeyInfo>
     <wsse:SecurityTokenReference wsu:Id='reference-4-1291211260670-25467657'>
      <wsse:Reference URI='#token-3-1291211260668-28347194'
      ValueType='http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3'/>
     </wsse:SecurityTokenReference>
    </ds:KeyInfo>
    <xenc:CipherData xmlns:xenc='http://www.w3.org/2001/04/xmlenc#'>
     <xenc:CipherValue xmlns:xenc='http://www.w3.org/2001/04/xmlenc#'>
     MXRaX3RExSplmD6ochmXaX78FDT02WTxXNOh1BK5x25Pjlse3juuJRHfX/WUv4H2b1QxcNC4hi/6
    0zTiPTaujeRdzFpWswuBwdfaK8bVxVuPXTcaQgulxXGL4CfXBdcoKXpMsA/rLfv3JjLsyzO0/9dT
    qt0K17hdzD6eOGATCTk=
    </xenc:CipherValue>
    </xenc:CipherData>
    <xenc:ReferenceList xmlns:xenc='http://www.w3.org/2001/04/xmlenc#'>
     <xenc:DataReference URI='#encrypted-2-1291211260630-22851870' xmlns:xenc='http://www.w3.org/2001/04/xmlenc#'
    </xenc:ReferenceList>
   </xenc:EncryptedKey>
  </wsse:Security>
 </env:Header>
```

Figura 42 - Cabeçalho uma mensagem SOAP com criptografia

A Figura 42 representa o cabeçalho da mensagem SOAP da Figura 41 depois de ser criptografado, como pode ser visto a um grande aumento no tamanho.

```
<env:Body wsu:Id='element-1-1291211260630-22972579'
xmlns:wsu='http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd'>
 <xenc:EncryptedData Id='encrypted-2-1291211260630-22851870' Type='http://www.w3.org/2001/04/xmlenc#Content'
 xmlns:xenc='http://www.w3.org/2001/04/xmlenc#'>
  <xenc:EncryptionMethod Algorithm='http://www.w3.org/2001/04/xmlenc#aes128-cbc'
  xmlns:xenc='http://www.w3.org/2001/04/xmlenc#'/>
  <xenc:CipherData xmlns:xenc='http://www.w3.org/2001/04/xmlenc#'>
   <xenc:CipherValue xmlns:xenc='http://www.w3.org/2001/04/xmlenc#'>
   u7M2YFF3oloFpj2qPTRCJ+kUIDZzy69/tUQzsA9w10gOBlXtjM/UimPt/dL+jqOucyQxpEw3JI0C
   kA3hTckGIrfopXrG/CDxlRbBRpnwNE6mv0OfI57j5ZQf4d5lKctbhNxnB9Sqx4u99Z5suO+nDOPW
   XyxiDPeEkV8t/k9NvL1+feVD92ciQEFeWOOjT/vqs4rkkOdLyyeIzghGqb91a1aUxYp/hBJ1CIkv
   I0zeQ+yqUX41YjeRDtCMff8Hxe3ZVv5Nym2B8JsyPzKqdkS6PpyFaAOrAHbYfPW/nYrA1Vevgrba
   nJn6uS3FW0JahFmBxWUHXkR/u35DqpmDXW9j/lpGL0v+zNS6ATQSmp6pLOIvyStj9/Vd2KaXj51O
   blgjOKFQVgOObZgdPfX0Rgm4NnKyb0MqYaLRUGVt0IYtOBdZBaVjfcs3VeQwb7llpDHIxk2SAaLb
   EQ5lbztYTSnq1QUNm85DzjvZbC5CBG7onmZdjsiCthAP3DZL/zKjVDFwNWTapu0oBpq7G/Egzhet
   GntSbJiQLMZTMoyJV8z1aWc=
   </xenc:CipherValue>
  </xenc:CipherData>
 </xenc:EncryptedData>
</env:Body>
```

Figura 43- Corpo da mensagem criptografado

A Figura 43 mostra respectivamente, o cabeçalho e o corpo da mensagem da Figura 40, depois de ser criptografados. Como pode ser observado na figura, houve um aumento muito grande no tamanho da mensagem. Com certeza uma aplicação com o fluxo muito grande de troca de mensagens desse tipo terá uma grande queda no desempenho. As conexões SSL garantem a segurança na comunicação usando muito menos recursos, por isso deve existir uma a avaliação da melhor forma de segurança para a aplicação.

Como pode ser visto nas seções 3.4 e 3.5 a configuração de serviços para uso de mecanismos de segurança no nível de mensagem é bem mais simples que no nível de transporte, pois quase toda a configuração é feita diretamente na aplicação. Entretanto a criptografia das mensagens exigem processamento extra e aumentam muito o tamanho das mensagens.

O WS-Security é um padrão para segurança no nível muito bom, ele padroniza o uso de varias tecnologias de segurança. A implementação de mecanismos de segurança usando o WS-Security é muito simples, pois existem muitas bibliotecas e frameworks que facilitam muito a criação de serviços seguros usando o WS-Security.

A principal desvantagem do uso do WS-Security em relação ao SSL/TLS esta não aumento do tamanho das mensagens e o uso extra de processamento necessário na criptografia das mensagens. A vantagem desse mecanismo de segurança esta nos recursos extras que são conseguidos como a autenticidade .Ao usar o WS-Security e a XML signature para assinar uma mensagem garantem que ela pode ser verificada como autêntico no

momento de recebimento e processamento, além de que pode ser armazenada para propósitos de auditoria com a garantia de autenticidade intacta.

Em relação à autenticidade o que pode ser feito com o uso de SSL é pedir alguma forma de identificação com usuário e senha, ou um certificado digital, mas isso é uma garantia mais fraca de autenticidade do que a assinatura digital do arquivo XML. Além disso, é mais complicado guardar a os dados trocados entre aplicações, isso dificulta os processos de auditoria.

# 4 - ANÁLISE DAS ABORDAGENS DE SEGURANÇA PARA O CENTRO DE PROCESSAMENTO DE DADOS DA UNIVERSIDADE FEDERAL DE SERGIPE

O cenário do CPD é formado aplicações internas da universidade, que necessitam acessar a informações no banco de dados da universidade, geralmente esse acesso é de consultas simples como lista de alunos e professores de um curso. Atualmente essas informações não são liberadas diretamente as aplicações dos departamentos.

Com base nas informações sobre as necessidades de acesso a dados que são solicitados ao CPD da UFS, onde geralmente as solicitações são consultas simples de acesso ao banco de dados no usando o padrão de comunicação Pedido-Resposta, explicado na seção 2.3.3, pode-se resolver esse problema de acesso criando *Web Services* que forneçam as informações que os departamentos necessitam.

A criação dos *Web Services* resolvem o problema de como as aplicações desenvolvidas fora do CPD terão acesso a informações no banco de dados, agora a única preocupação é a escolha de uma mecanismos de segurança que garanta a integridade e confidencialidade dos dados que serão transmitidos entre banco e aplicações.

Uma aplicação sem nenhum mecanismo de segurança, como discutido na Seção 3.1, traria como vantagens o desempenho, pois as mensagens não são criptografadas, isso faz com que o tamanho das mensagens seja muito menor além da facilidade de implementação da abordagem. A desvantagem é o fato de que se as mensagens forem interceptadas, suas informações estão totalmente expostas e que as mensagens interceptadas podem ser alteradas. Para o CPD essa solução não poderia ser utilizada devido à falta de segurança, pois as informações requisitas ao CPD nem sempre são informações que podem ser disponibilizadas ao público em geral, como dados de alunos e professores, além disso, essas informações devem ser confiáveis, o que não pode ser garantido por uma aplicação sem mecanismos para garantir a integridade das informações.

O uso de mecanismo de segurança no nível de mensagem como mostrado na Seção 3.2, traria como vantagem a confidencialidade e integridade no transporte das mensagens, isso garante que as mensagens recebidas não foram alteradas e mesmo que elas sejam interceptadas, seu conteúdo não pode ser entendido, garantindo dessa forma a integridade e confidencialidade dos dados das mensagens. A desvantagem desse tipo de mecanismo é que à medida que o número de serviços e servidores usados aumenta, o controle configuração de

servidores vai ficando mais complexo. Para o CPD a solução apresentada é viável no caso onde só seja necessário garantir a integridade dados, pois essa implementação não tem mecanismos para autenticação.

A versão implementada na versão 3.3 também usa mecanismos de segurança em transporte por isso ela tem a mesmas vantagens e desvantagens apresentadas na versão implementada na Seção 3.2, mas nessa versão é incluído um recurso do servidor para permitir autenticação de usuários. Para o CPD esse solução é viável, pois garante integridade e confidencialidade. A versão garante que só usuários autenticados podem acessar determinados serviços.

A versão da aplicação discutida na Seção 3.4 implementa segurança no nível de mensagem e tem como vantagem garantir que a integridade e confidencialidade das mensagem, além de que não é necessário que sejam feitas configurações no servidor, como é feito nas versões com segurança no nível de transporte. A desvantagem dessa versão é o aumento do tamanho da mensagem devido a criptografia, o que pode diminuir o desempenho das aplicações. Para o CPD essa solução seria viável em uma situação onde o volume de dados nas mensagens seja pequeno, em casos onde o seja solicitado um volume muito grande de informações, como uma consulta de informações sobre todos os alunos de um curso ou matriculados em uma disciplina, Essa solução não é a melhor, pois irá gerar uma quantidade enorme de dados criptografados.

Finalmente a versão 3.5 também implementa segurança nível de mensagem, só que com o diferencial que as mensagens também são assinadas com certificados digitais dos usuários e servidores dos serviços. Essa versão trás as mesmas vantagens e desvantagens da versão anterior, porém o uso de assinatura digital garante a autenticidade das mensagens o que é um nível a mais de segurança. Essa versão também seria viável para uso no CPD em um caso onde o volume de dados seja pequeno e onde seja necessário garantir a autenticidade das mensagens. Porém não seria indicada em situações onde seja requisitada grande quantidade de dados.

Como foi mostrado existem principalmente duas maneiras pra garantir a segurança dos dados, segurança no nível de transporte e nível de mensagem. Segurança no nível de mensagem, mostrado nas Seções 3.4 e 3.5, tem a desvantagem do aumento do tamanho das mensagens, esse fato é não é interessante no cenário descrito, pois provavelmente o volume de informações consultadas será grande, assim com a criptografia das mensagens irá gerar um volume de dados muito grande o que pode prejudicar o desempenho das aplicações.

A segurança no nível de transporte é uma técnica muito madura e tem um bom desempenho, em um cenário simples como o que foi descrito, onde não existiriam serviços intermediários e a comunicação será feita diretamente ao serviço que disponibilizará os dados, criar conexões SSL diretas com o serviço é uma forma muito simples e eficiente para garantir a segurança no transporte dos dados. Assim para garantir que os dados trafeguem de forma segura, basta que os *Web Services* sejam implementados com o uso de conexões SSL, como descrito na Seção 3.3, que além da integridade e confidencialidade também tem um mecanismo para autenticação de usuários. A seguir serão mostrados testes realizados com os dois níveis de segurança mostra.

Threads: 10 Strategy Thread Start Threads 1 End Threads 10

| Test Step | min | max | avg | last | cnt | tps | bytes | bps | err |
|---|---|---|---|---|---|---|---|---|---|
| Normal | 4 | 260 | 21,93 | 28 | 23298 | 438,57 | 11346477 | 3424834 | 0 |
| SSL | 6 | 210 | 34,74 | 34 | 17386 | 279,42 | 6395571 | 2182044 | 0 |
| WS | 8 | 392 | 48,17 | 34 | 11291 | 199,69 | 8996238 | 2755356 | 0 |

Figura 44 – Teste de desempenho com SOAPUI com 50 itens

Threads: 10 Strategy Thread Start Threads 1 End Threads 10

| Test Step | min | max | avg | last | cnt | tps | bytes | bps | err |
|---|---|---|---|---|---|---|---|---|---|
| Normal | 4 | 213 | 15,83 | 10 | 27363 | 603,4 | 1675240 | 582287 | 0 |
| SSL | 5 | 1177 | 22,1 | 16 | 22558 | 436,25 | 1228445 | 420988 | 0 |
| WS | 6 | 385 | 26,1 | 18 | 18978 | 370,03 | 5377822 | 1682146 | 0 |

Figura 45Teste de desempenho com SOAPUI com 5 itens

As figura 44 e 45 mostram os resultados de testes realizando com os web services usando três nível de segurança, sem criptografia, segurança no nível de transporte e segurança em mensagem. Os testes foram realizados com a ferramenta SOAP UI [32]. O teste consiste a disparar varias requisições para os servidores a, a ferramenta se encarrega de gerar os dados dos resultados e também de realizar as requisições. O web service que foi testado devolve como retorno uma lista com 50 itens e depois uma com 5 itens, esse itens são uma resposta padrão e é qual para os três testes (normal, SSLSSL, WS). As requisições são realizadas por um grupo de 10 threads e o teste tem duração de 60 segundos.

Os dados retornados nos testes são:

- min – tempo mínimo para realização de uma requisição
- max - tempo maximo para realização de uma requisição
- avg - tempo médio para realização de uma requisição
- last – resultado do ultimo tempo registrado para realizar uma requisição.
- cnt – número de requisições realizadas
- tps – requisições por segundo

- bytes – quantidade de bytes
- bps – bytes por segundo
- err – quantidade de erros

Com o resultado do teste pode ser notado o impacto que cada tipo de mecanismo de segurança tem sobre o desempenho da aplicação, a analise do resultado tem como base de comparação os valores retornados com a execução das requisições para o web service sem mecanismos de segurança (normal).

Como pode ser visto na figura 44, o teste com a configuração Normal, sem segurança, em num intervalo de 60 segundos conseguiu realizar 23.298 requisições (cnt), a configuração SSL, usando segurança no nível de transporte, consegui realizar 17.386 e a configuração WS, usando segurança nível de mensagem implementado com WS-Security, realizou 11.291 requisições. Como pode ser visto na figura 44, o valor tps (279.42) da configuração com SSL é bem maior que o tps (199.69) da configuração com WS-Security. Isso significa que o servidor configurado com SSL conseguiu responder as requisições muito mais rápido do que quando esta configurado para criptografar as mensagens.

Na figura 45, o teste é feito com menos dados, o web service tem como returno um lista com apenas 5 itens. Nesse teste pode ser notado que a diferença de desempenho diminue, já é o volume de dado é menor.  A figura 46 mostra o gráfico dos resultados do testes para medir o desempenho.

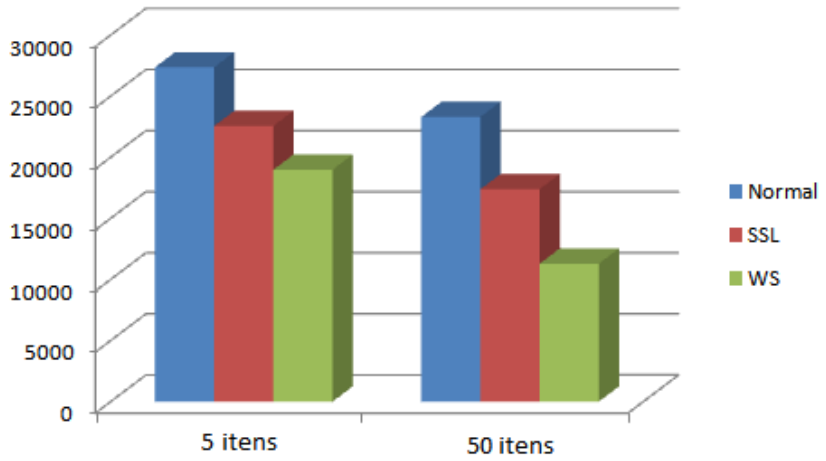


Figura 46- Grafico de desempenho

As requisições usando segurança em nível de transporte foi mais rápida que a que usa criptografia da mensagem, já que consegui um resultados cnt e tps mais próximo aos números atingido pela configuração normal .

Threads: 10 Strategy Simple Test Delay 0 Random 0.5

| Test Step | min | max | avg | last | cnt | tps | bytes | bps | err |
|---|---|---|---|---|---|---|---|---|---|
| Normal | 5 | 59 | 17,23 | 5 | 1000 | 130,22 | 7809000 | 1016929 | 0 |
| SSL | 6 | 95 | 31,56 | 6 | 1000 | 130,22 | 7809000 | 1016929 | 0 |
| WS | 9 | 83 | 24,52 | 9 | 1000 | 130,22 | 13798077 | 1796858 | 0 |
| TestCase: | 20 | 237 | 73,31 | 20 | 1000 | 130,22 | 29416077 | 3830717 | 0 |

Figura 47- Segundo teste usando SOAPUI

Outro fato que pode ser observado em teste é o volume de dados transportados nas requisições, a figura 47 mostra a quantidade de bytes que uma aplicação cliente receberia pela rede ao realizar requisições aos tipos de web services. Nesse teste foram feitas 1000 requisições para cada web services, como pode ser visto na coluna bytes a configuração normal e SSL apresentam valores iguais já que não há alteração na mensagem, entretanto o volume de dados recebidos no teste como a configuração WS tem um aumento de aproximadamente 77% no volume de dados das mensagens. Nesse teste o valor tps não é significante pois o software não leva em consideração tempo, o teste é realizando com base em número de requisições fixado no inicio do teste. A figura 48 mostra um gráfico com a diferença no volume de dados entre o uso de SSL e WS.

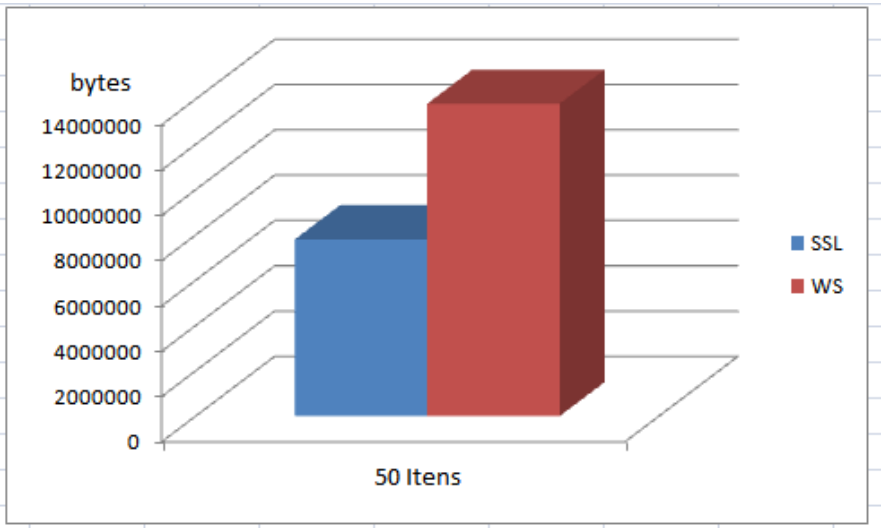


Figura 48-Gráfico volume de dados

Como os valores dos testes indicam o custo da WS-Security pode sem muito alto dependendo do tipo de aplicação, e em muitas situações o uso da SSL se torna uma melhor solução para segurança.

# 5 – CONCLUSÃO

Este trabalho tem como objetivo analisar diversas abordagens para implementação de segurança em arquiteturas orientadas a serviço com o uso de *Web Services.*

Para fazer essa análise foi implementada uma aplicação utilizando a arquitetura orientada a serviços divida em cinco versões, cada uma usando mecanismos de segurança diferentes. Essa aplicação foi implementada com sucesso e serviu como base para estudo de técnicas de segurança para *Web Services.*

Após a conclusão da implementação de todas a versões aplicação foi realizada a analise dos resultados em relação à necessidades de segurança e desempenho que cada versão proporciona, isso foi necessário para poder ser feito um melhor julgamento do que seria melhor para o problema apresentado pelo CPD.

Depois de feita a analise dos resultados alcançados com cada uma das versões foi definida uma possível solução para o CPD.

A abordagem escolhida foi o uso de mecanismo de segurança no nível de transporte com autenticação. Esse tipo de modelo de segurança mostra-se melhor para o caso do CPD, pois os tipos de serviços que devem ser utilizados para departamentos serão voltados para consultas, que podem retornar grandes volumes de dados. Isso faz com que a utilização de segurança em nível de mensagem, não seja tão eficiente, pois ela aumenta muito o volume dos dados devido à criptografia da mensagem, esse fato pode gerar uma queda de desempenho das aplicações, essa característica ajudou a escolher o uso de conexões SSL/TLS com a melhor opção.

## 5.1 - Trabalhos Futuros

Em relação a trabalhos futuros são feitas algumas sugestões para continuidade no estudo dos desafios dos aspectos de segurança na adoção Arquitetura Orientada a Serviços.

- Avaliação de desempenho entre mecanismo de segurança no nível de transporte e de mensagem
    - Existe uma diferença boa diferença do nível de desempenho entre SSL/TLS e aplicações que usam WS-Security, um estudo mais detalhado sobre essa diferença ajudaria a julgar melhor a viabilidade dessas abordagens e situações mais críticas.

- Estudo de mecanismo para uma autenticação única entre diferentes serviços disponível na UFS.
    - Em uma situação onde possam existir muitos serviços que necessitem de autenticação, seria interessante o uso estratégias que permitissem que uma fez que o usuário tenha se autenticado em para um sistema, esses dados sobre a identidade se propagassem de forma transparente para outros serviços disponível.

# 6 - REFERENCIA BIBLIOGRAFICA


1. PULIER, H. T. E. **Understanding Enterprise SOA**. 1. ed. Greenwich, CT, USA: Manning, 2006. 280 p.

2. JOSHI, Bipin. - **Beginning XML with C# 2008 From Novice to Professional**.1.ed. Estados Unidos da América:Apress, 2008

3. JOSUTTIS, Nicolai. SOA **in Practice**.1.ed.Estados Unidos da América:O'Reilly., 2007

4. DIRK Krafzig, KARL Banke, DIRK Slama. **Enterprise SOA: Service-Oriented Architecture Best Practices**. 1.ed. Estados Unidos da América: Prentice Hall, 2004

5. Ruh, William A.; Maginnis, Francis X.; Brown, William J. – **Enterprise Application Integration**: A Wiley tech brief .1. ed. Estados Unidos da América:John Wiley & Sons. Inc, 2001.

6. ROSEN, M. et al. Applied **SOA: service-oriented architecture and design strategies**.Indianapolis: Wiley Publishing, Inc., 2008.

7. HUNTER, D. et al. Beginning XML. Indianápolis: Wrox,2007

8. KANNEGANTI Ramarao; CHODAVARAPU Prasad**. SOA Security**. Estados Unidos da América:Manning,2008

9. W3C, XML Signature Syntax and Processing (Second Edition), W3C Recommendation, 10 June 2008. Disponivel em: <http://www.w3.org/TR/xmldsig-core/>. Acessado em 14/10/2010 .

10. IBM. Federação de identidade utilizando a SAML e o software WebSphere.Disponivel em: <http://www.ibm.com/developerworks/br/websphere/library/ws-SAMLWAS/index.html>. Acessado em 18/11/2010

11. COMMONS, C.OWASP. categoty:attack, 2010. Disponível em:<http://www.owasp.org/index.php/Category:Attack>. Acesso em 17/11/2010

12. GOULART, Ricardo.Um estudo comparativo sobre os modelos de objetos distribuídos CORBA, DCOM e RMI. Disponível em: <http://www.inf.ufrgs.br/gppd/disc/cmp167/trabalhos/mp2000-1/RicardoGoulart/index.html>. Acessado em: 25/11/2010

13. MEGGINSON, David. SAX 2.0. Disponível em:<http://www.saxproject.org/>.Acessado em: 25/11/2010

14. W3C, XML-Signature XPath Filter 2.0, W3C Recommendation, 08 November 2002. Disponível em: <http://www.w3.org/TR/xmldsig-filter2/>. Acessado em: 14/10/2010

15. W3C, XML Encryption Syntax and Processing, W3C Recommendation, 10 December 2002, Disponível em: <http://www.w3.org/TR/xmlenc-core/>. Acessado em: 14/10/2010

16. W3C. XML Key Management Specification (XKMS 2.0) Version 2.0. W3C Recommendation 28 June 2005. Disponível em: <http://www.w3.org/TR/xkms2/>. Acessado em: 14/10/2010

17. ADAMS, et al. Internet X.509 Public Key Infrastructure Certificate Management Protocols. Disponível em: <http://tools.ietf.org/html/rfc4210>.Acessado em: 15/10/2010

18. Massachusetts Institute of Technology. Kerberos: The Network Authentication Protocol. Disponível em:<http://web.mit.edu/kerberos/>. Acessado em: 20/8/2010

19. GOODNER, Marc et al. Understanding WS-Federation. Disponível em: <http://msdn.microsoft.com/en-us/library/bb498017.aspx>. Acessado em 20/8/2010

20. OASIS Standard, SAML V2.0. Disponível em: <http://saml.xml.org/saml-specifications#samlv20>. Acessado em 15/10/2010

21. OASIS Standard, A Brief Introduction to XACML. Disponível em: <http://www.oasis-open.org/committees/download.php/2713/Brief_Introduction_to_XACML.html>. Acessado em 16/10/2010

22. OASIS Standard. *Web Services* Federation Language (WS-Federation) Version 1.2, 22 May 2009. Disponível em:<http://docs.oasis-open.org/wsfed/federation/v1.2/ws-federation.html>. Acessado em 16/10/2010

23. RFC. The Transport Layer Security (TLS) Protocol Version 1.2. Disponível em:<http://datatracker.ietf.org/doc/rfc5246/>. Acessado em 15/10/2010

24. JBoss.org. HIBERNATE - Relational Persistence for Idiomatic Java. Disponível em <http://docs.jboss.org/hibernate/core/3.6/reference/en-US/html_single/> acessado em 10/12/2010

25. Oracle. keytool - Key and Certificate Management Tool . Disponível em: <http://download.oracle.com/javase/1.3/docs/tooldocs/win32/keytool.html>.Acessado em 10/12/2010

26. ROSENBERG, J. & Remy, D., Securing *Web Services* with WS-Security: Demystifying WS-Security, WS-Policy, SAML, XML Signature ano XML Encryption. SAMS,1.ed. Estados Unidos da América: Pearson Higher Education, 2004.

27. JBoss.org. JBoss application server. 2010. Disponível em: <http://www.jboss.org/>. Acessado em 12/12/2010.

28. Oracle. JDK (*Java Development Kit*) 1.6. Disponível em <http://www.oracle.com/technetwork/java/javase/downloads/index.html>. Acessado em 12/12/2010.

29. The Eclipse Foundation. Eclipse IDE for Java EE Developers. Disponível em: <http://www.eclipse.org/downloads/packages/eclipse-ide-java-ee-developers/heliossr1>.Acessado em 12/12/2010

30. Oracle.Netbeans. Disponível em: <http://netbeans.org/index.html>. Acessado em 12/12/2010

31. Microsoft. Windows7. 2010. Disponível em <http://windows.microsoft.com/pt-BR/windows7/products/home>. Acessado em 12/12/2010

32. SoapUI. Disponível em http://www.soapui.org/. acessado em 12/12/2010