PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
FACULDADE DE INFORMÁTICA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

DIEGO CASTANHEIRA PACHECO

# CONFIGSECURITYWS – UMA FERRAMENTA PARA CONFIGURAÇÃO DE MECANISMOS DE SEGURANÇA EM WEB SERVICES

Porto Alegre
2008

DIEGO CASTANHEIRA PACHECO

# CONFIGSECURITYWS – UMA FERRAMENTA PARA CONFIGURAÇÃO DE MECANISMOS DE SEGURANÇA EM WEB SERVICES

Trabalho apresentado como parte da avaliação na disciplina de Trabalho de Conclusão II, para o curso de Bacharelado em Ciência da Computação, Faculdade de Informática, Pontifícia Universidade Católica do Rio Grande do Sul.

**Orientador: Prof. Dr. Celso Maciel da Costa**

Porto Alegre

2008

## RESUMO

Este trabalho tem como objetivo o projeto e implementação de uma ferramenta que permite acrescentar mecanismos de segurança em aplicações de arquitetura Web Services baseadas no framework Axis2. Inicialmente é apresentado um estudo dos conceitos de segurança e das técnicas de ataques executados contra aplicações publicadas na Web. Após esse estudo, é apresentado o conceito de intermediários e as ações executadas pela ferramenta. Finalmente, são realizados testes com a finalidade de comprovar o funcionamento da ferramenta proposta.



Palavras-chave: Segurança, Web Services, Axis2.

## ABSTRACT

This work aims at the design and implementation of a tool that allows adding security mechanisms in applications of Web Services architecture based on the Axis2 framework. Initially it is presented a study of security concepts and techniques of attacks performed against applications published on the Web. After this study, it is presented the concept of handlers and the actions executed by the tool. Finally, tests are performed in order to demonstrate the operation of the proposed tool.



Keywords: Security, Web Services, Axis2.

## LISTA DE FIGURAS

**SUMÁRIO**

# 1 INTRODUÇÃO

## 1.1 MOTIVAÇÃO E JUSTIFICATIVA

A Internet hoje possui um grande número de usuários, tendo seu uso disseminado em diversos países e apresentando um notável crescimento nos últimos dez anos. Atualmente muitos serviços públicos e privados são oferecidos através da rede, exemplos disso são as lojas virtuais, sites de bancos, etc.

Uma forma de desenvolver e disponibilizar serviços e sistemas na Web é com o uso de Web Services: aplicações Web, modulares, que permitem a comunicação com outras aplicações através de uma especificação em linguagem XML.

Qualquer que seja a ferramenta utilizada no desenvolvimento das aplicações, um fator que compromete o funcionamento correto, adequado e esperado dessas aplicações são as falhas e ameaças de segurança que podem vir a ocorrer.

Aplicações Web, por estarem disponíveis em uma rede pública, estão expostas a ataques de usuários mal intencionados, que encontram nas falhas existentes das mesmas a oportunidade de obter acesso a dados restritos.

Este trabalho é dedicado ao estudo de segurança em Web Services.

Será feito um estudo sobre segurança na Web, descrevendo quais são os problemas de segurança e as técnicas de ataques utilizados e, por conseguinte, em Web Services, com o objetivo de definir, especificar e implementar mecanismos que aumentem a segurança das aplicações desenvolvidas com o uso de Web Services.

## 1.2 OBJETIVOS

O objetivo geral deste trabalho será a realização de um estudo de segurança em Web Services, tendo em vista a definição, especificação e implementação de uma ferramenta para acrescentar confiabilidade às aplicações desenvolvidas usando

Web Services. Os mecanismos definidos e especificados serão implementados, usando-se como plataforma de desenvolvimento o protocolo Axis2 (ws.apache.org/axis2).

Visando atingir o objetivo geral, foram identificados os seguintes objetivos específicos:

a. Estudo de falhas e ameaças de segurança em Web Services.

b. Definição, especificação e implementação de uma ferramenta que permitirá incluir mecanismos de segurança em uma arquitetura de Web Services, baseada no protocolo Axis2 e no *framework* de segurança Rampart (APACHE-RAMPART, 2008) para a implementação da solução concebida.

# 2 FUNDAMENTAÇÃO TEÓRICA

## 2.1 SEGURANÇA

A segurança é um fator de extrema importância em qualquer sistema. O software deve fornecer aos seus usuários a garantia de que o mesmo possuirá consistentemente as propriedades que foram requeridas. Entre essas propriedades, a segurança é a que permite o software exibir estas propriedades mesmo quando este software sofre um ataque. (IATAC, 2007)

Software seguro é aquele que está apto a resistir à maioria dos ataques, tolerar a maioria dos ataques que não puder resistir, e recuperar-se muito rapidamente com mínimos danos dos poucos ataques que não puder tolerar. Esses ataques ocorrem onde existem falhas ou fraquezas que podem ser exploradas tanto por invasores humanos como por códigos maliciosos. (IATAC, 2007)

Um site da Web pode ter suas vulnerabilidades identificadas, e uma vez que estas sejam identificadas, um invasor poderá utilizar pelo menos uma das muitas técnicas de ataques para ter acesso a informações restritas. Essas técnicas são classificadas de acordo com a forma que se aproveitam das vulnerabilidades. As ameaças de segurança em sites da Web podem ser classificadas em seis categorias: Autenticação, Autorização, Ataques ao Cliente, Execução de Comandos, Divulgação de Informação e Ataques Lógicos (WASC, 2004).

### 2.1.1 Autenticação

A Autenticação abrange os problemas de segurança em que o alvo do ataque é o método de identificação do usuário em um site da Web. A seguir serão apresentadas algumas das formas de ataque relativas à autenticação.

- Força Bruta: o ataque por força bruta se dá através do método de tentativa e erro, e tem por objetivo descobrir nomes de usuário, senhas, números de cartão de crédito ou chaves de criptografia. Exemplo: o invasor testa diversos nomes de usuários e senhas em um site com o objetivo de efetuar login utilizando a conta de um outro usuário.
- Autenticação Insuficiente: quando um site permite ao invasor acessar conteúdo e funcionalidades sem que tenha realizado uma autenticação apropriada. Exemplo: muitas aplicações na Web possuem um diretório onde se localizam as funcionalidades administrativas do sistema. O invasor ao acessar o site pela URL, acrescenta "/admin" e consegue o acesso a todas essas funcionalidades.
- Fraca Validação na Recuperação de Senha: ocorre quando um site da Web solicita cadastramento de informações para que o usuário possa recuperar uma senha, caso esta seja esquecida. Dessa forma é possível o invasor ilegalmente obter, alterar ou recuperar a senha do usuário que cadastrou essas informações. Exemplo: sites em que o usuário ao se cadastrar, informa frases/palavras que serão utilizadas futuramente para lembrança da senha.

### 2.1.2 Autorização

A Autorização é composta por ataques que tem como alvo o método de autorização que o site utiliza para determinar se um usuário, serviço ou aplicação possui as permissões necessárias para solicitar uma ação à aplicação. Algumas vulnerabilidades e as formas de ataques realizadas nessa categoria são:

- Predição de Credencial/Sessão: é um método de roubar ou se fazer passar por um usuário do site. Ocorre com a dedução ou adivinhação do valor único que identifica a sessão ou o usuário, permitindo que o invasor possa fazer requisições ao site ao identificar-se usando os dados do usuário vítima do ataque. Exemplo: para a geração do ID da sessão, são usados algoritmos que

realizam procedimentos complexos que determinam esse ID. Muitos sites da web mantêm o ID da sessão armazenado em “cookies”, campos de formulário ocultos ou na URL. Se o invasor puder determinar o algoritmo usado para geração do ID de sessão, poderá se conectar a aplicação usando esse ID.

- Autorização Insuficiente: quando um site permite que o invasor possa acessar conteúdos e funcionalidades que possuem restrições de acesso. Exemplo: muitos sites armazenam funcionalidades administrativas em diretórios ocultos como “/admin” ou “/logs”, se o invasor puder acessar esses diretórios, então ele passaria a ter a acesso a essas funcionalidades.
- Expiração de Sessão Insuficiente: quando um site permite ao invasor reutilizar IDs de sessões antigas para sua autorização. Exemplo: em um ambiente compartilhado (mais de uma pessoa tem acesso físico irrestrito a um computador) se a função de *logout* do sistema encaminha meramente a vítima à página inicial sem terminar sua sessão, um outro usuário poderia consultar as páginas do histórico e ver as páginas acessadas pela vítima. Caso o ID da sessão da vítima não tenha expirado, o invasor poderia ver a sessão da vítima sem ter requerido credenciais de autenticação.
- Fixação de Sessão: é uma técnica de ataque que força um ID de sessão de usuário com um valor explícito. Dependendo da funcionalidade do site alvo, um número variado de técnicas pode ser utilizado para “fixar” o valor do ID da sessão. Diferentemente do roubo de ID de sessão de um usuário após ele ter efetuado *login* no site, fixação de sessão fornece uma janela de oportunidades muito mais ampla, pois a parte ativa do ataque ocorre antes da autenticação do usuário. Exemplo: o ataque de fixação de sessão consiste geralmente em um processo de três passos:

    a) Configuração de sessão: o invasor configura uma “sessão armadilha” para o site alvo e obtém os IDs de sessão. Ou, o invasor pode selecionar um ID de sessão arbitrário usado no ataque.

    b) Fixação de sessão: o invasor introduz o valor da sessão armadilha no *browser* do usuário e fixa o ID da sessão do usuário.

c) Entrada na sessão: o invasor espera até que o usuário efetue *login* no site. Após isso, o ID de sessão fixado será usado e o invasor poderá assumir o controle da conta do usuário.

### 2.1.3 Ataques ao Cliente

A categoria Ataques ao Cliente compreende os ataques em que há abuso ou exploração de um usuário. Quando um usuário visita um site da Web, deve haver uma relação de confiança por ambas as partes, tanto tecnologicamente quanto psicologicamente. O usuário espera que os sites que ele visita apresentem conteúdos válidos, e que esses sites não realizem ataques ou tragam qualquer outra ameaça de segurança. Fingindo haver uma relação de confiança entre o usuário e o site que ele acessa, um invasor pode empregar algumas técnicas para explorar o usuário:

- Disfarce de Conteúdo: é uma técnica de ataque utilizado para enganar um usuário fazendo-o acreditar que o conteúdo do site que está acessando é legítimo e não de uma fonte externa. Exemplo: o invasor cria páginas da Web falsas, como formulários de *login*, *press releases*, etc, e o usuário que visita essas páginas acredita estar visualizando uma página de conteúdo legítimo, mas na verdade estará submetendo seus dados nessas páginas falsas.
- *Cross-site Scripting* (XSS): é uma técnica de ataque que força um site da Web a ecoar código executável fornecido pelo invasor, que é carregado no *browser* do usuário. O código é normalmente escrito em HTML/Java script, mas pode também ser estendido para VBScript, ActiveX, Java, Flash, ou qualquer outra tecnologia suportada em *browsers*. Alguns exemplos:

  a) Ataque Persistente: muitos sites da Web hospedam fóruns onde usuários registrados podem postar mensagens. Um usuário é normalmente identificado através de um *cookie* com ID de sessão autorizando a postagem de mensagens. Se um invasor conseguir postar uma mensagem contendo código JavaScript que permite roubar

*cookies*, o usuário pode ter seus *cookies* e sua conta comprometidos. Exemplo de código utilizado nessa técnica de ataque:

```
<SCRIPT>
document.location='http://attackerhost.example/cgi-
bin/cookiesteal.cgi?'+document.cookie
</SCRIPT>
```

b) Ataque Não Persistente: muitos portais na Web oferecem uma visualização personalizada de um site da Web e cumprimentam um usuário que efetuou *login* com a mensagem “Bem-vindo,”. Às vezes os dados referenciando esse usuário são armazenados dentro de uma *string* de uma URL e impressos na tela, na forma: “http://portal.example/index.php?sessionid=12312312&username=Joe“. Como se pode notar o nome de usuário “Joe” é armazenado na URL. A página da Web resultante exibe a mensagem “Welcome, Joe”. Se o invasor modificar o campo *username*, inserindo um código JavaScript para roubo de *cookie*, seria possível assumir o controle de uma conta de usuário.

### 2.1.4 Execução de Comandos

A categoria Execução de Comandos abrange os ataques projetados para executar comandos remotos em sites da Web. Todos os sites da Web utilizam valores de entrada fornecidos pelos usuários para preencher requisições. Esses dados fornecidos pelo usuário são usados frequentemente para comandos de construção que resultam em paginas Web dinâmicas. Se este processo for realizado sem segurança, um invasor poderia modificar execução desses comandos. Algumas das técnicas de ataques utilizadas nessa categoria:

- Estouro de Buffer: são ataques que consistem na alteração de algum código da aplicação sobrescrevendo partes da memória. É uma falha comum de software que resulta em condição de erro. Esse erro ocorre quando dados

são escritos na memória excedem o tamanho que foi alocado para o buffer. A maioria das vulnerabilidades de estouro de buffer ocorre geralmente em linguagens de programação como C e C++. Estouro de buffer pode ocorrer em um programa CGI ou quando uma página da Web acessa um programa escrito em C.

- Ataque por Formatação de String: ataque que consiste em alterar o fluxo de uma aplicação formatando características para acessar outros espaços de memória. Vulnerabilidades ocorrem quando dados fornecidos pelo usuário são usados como entrada de comandos de formatação de *strings* através de chamadas de funções das linguagens C/C++ (exemplo: fprintf, printf, sprintf, setproctitle, syslog, ...). Se o invasor utilizar parâmetros de conversão na chamada dessas funções (exemplo: “%f”, “%p”, “%n”) ele poderá: executar código malicioso no servidor, ler valores da pilha ou causar erros de falha de segmentação no sistema. Exemplo: uma aplicação web possui o parâmetro emailAdress, que é impresso na tela através da função printf(emailAdress). O invasor poderia chamar essa função utilizando parâmetros de conversão, e assim conseguiria ler outros valores da memória.
- Injeção de LDAP: é uma técnica usada para explorar sites da Web que constroem expressões LDAP de uma entrada fornecida pelo usuário. LDAP é um protocolo padrão para realização de consultas e manipulação de serviços de diretório X.500. O protocolo LDAP executa sobre protocolos de transporte da Internet, como o TCP. Aplicações web podem usar entradas fornecidas pelo usuário para criar expressões LDAP para requisições a páginas da Web dinâmicas. Quando uma aplicação falha em sanar entradas fornecidas por usuário, é possível um invasor alterar a construção de uma expressão LDAP. Dessa forma é possível que ele obtenha permissões para realizar consultas, modificar ou remover o que estiver dentro da árvore de diretórios LDAP.
- Comandos de SO: técnica para explorar sites da Web através da execução de comandos de sistema operacional em campos de entrada. Exemplo: a linguagem Perl permite conduzir dados de um processo em uma expressão aberta adicionando um caractere *pipe* “|” no final de um nome de arquivo. A seguir o código que retrata esse exemplo através da execução de um

comando UNIX que lista os arquivos de um diretório e coloca em um arquivo de saída:

```
# Execute "/bin/ls" and pipe the output to the open statement
open(FILE, "/bin/ls|")
```

- Injeção de SQL: é uma técnica usada para explorar sites da Web utilizando comandos SQL em campos de entrada fornecidas pelo usuário. Dessa forma é possível que o invasor assuma o controle do banco de dados utilizado pelo site. Por exemplo, uma aplicação Web utiliza o seguinte código SQL para a autenticação de um usuário:

```
SQLQuery = "SELECT Username FROM Users WHERE Username = '" & strUsername & "' AND Password = '" & strPassword & "'" strAuthCheck = GetQueryResult(SQLQuery)
```

Supor que um invasor submeta um nome de usuário e senha como é apresentado a seguir:
Login: ‘ ou ‘ ’=’
Password: ‘ ou ‘ ’=’
Isso resultará na seguinte consulta SQL:
SELECT Username FROM Users WHERE Username = '' OR ''='' AND Password = ' ' OR ' ' = ' '

Essa consulta ao invés de comparar o nome de usuário e senha fornecidos pelo usuário com os dados existentes na tabela de usuários cadastrados, ela retorna “true” para a comparação de ‘ ‘ (string vazia) com ‘ ‘ (string vazia), assim o invasor efetuará login como o primeiro usuário da tabela de usuários.

- Injeção SSI (*Server-side Include*): é uma técnica de exploração utilizada em servidores que permite ao invasor enviar código para uma aplicação Web, que mais tarde será executado localmente pelo servidor Web. Antes de servir uma página HTML, um servidor Web pode fazer *parse* e executar comandos

*Server-side Include* antes de fornecê-las ao usuário. Em alguns casos (exemplos: fórum de mensagens ou sistemas de manutenção de conteúdo), uma aplicação Web insere dados no código-fonte de uma página Web. Se o invasor submeter comandos desse tipo, ele poderá executar comandos arbitrários de sistema operacional, ou incluir conteúdos de um arquivo restrito na próxima vez que a página for servida. Por exemplo, o seguinte SSI tag pode permitir ao invasor listar o conteúdo de um diretório raiz através de um comando UNIX:

```
< !--#exec cmd="/bin/ls /" -- >
```

- Injeção XPath: é uma técnica usada para esxplorar sites que constroem consultas XPath a partir de entradas fornecidas pelo usuário. XPath 1.0 é uma linguagem usada para referenciar partes de um documento XML. Pode ser usada diretamente por uma aplicação para consultar um documento XML, ou como parte de uma operação maior como aplicar uma transformação XSLT a um documento XML, ou aplicar uma consulta XQuery a um documento XML. Por exemplo, supor um documento XML que contém elementos nome de usuário, e cada um desses elementos contêm três subelementos – nome, senha e número da conta. A seguinte expressão XPath consulta o número da conta do usuário cujo nome é “jsmith” e cuja senha é “Demo1234”:

```
string(//user[name/text()='jsmith' and
password/text()='Demo1234']/account/text())
```

Se uma aplicação usa construção de consultas XPath em tempo de execução, encaixando entradas inseguras nessas consultas, é possível ao invasor injetar dados na consulta de forma que a nova consulta formada seja interpretada de uma forma diferente da intenção do programador.

### 2.1.5 Divulgação de Informação

A categoria Divulgação de Informação abrange os ataques realizados com o objetivo de adquirir informações específicas de sistema em um site da Web. Informações específicas de sistema incluem a distribuição do software, números de versão, e níveis de *patch*. Ou a informação pode conter a localização de arquivos de backup e arquivos temporários. Na maioria dos casos, a divulgação dessas informações não é necessária para atender as necessidades do usuário. A maioria dos sites revela certa quantidade de dados, mas o ideal é limitar essa quantidade de dados sempre que possível. Quanto mais informação sobre um site um invasor puder aprender, mais facilmente o sistema se tornará vulnerável a ataques. Algumas das técnicas de ataques utilizadas nessa categoria:

- Indexação de Diretório: a indexação automática de diretório é uma função de um servidor web que lista todos os arquivos dentro de um diretório requisitado se o arquivo base (index.html / home.html / default.htm) não está presente. Quando um servidor Web revela o conteúdo de um diretório, a listagem pode conter informações que não deveriam se tornar pública. Atualmente, scanners de vulnerabilidades, como o Nikto, podem dinamicamente adicionar diretórios ou arquivos que serão incluídos em sua busca baseado em dados obtidos previamente. Revisando o arquivo /robots.txt e/ou vendo o conteúdo da indexação por diretório, o scanner de vulnerabilidade pode agora interrogar o servidor Web com mais estes novos dados. Embora seja inofensiva, a indexação de diretórios pode permitir um escape de informação que fornece ao invasor as informações necessárias para que este lance um ataque contra o sistema. Por exemplo, as seguintes informações podem ser obtidas a partir da indexação de dados:

  a) Arquivos de Backup: com extensões como .bak, .old ou .orig.

  b) Arquivos temporários: arquivos normalmente removidos do servidor, mas que por alguma razão ainda estão disponíveis.

  c) Arquivos ocultos: arquivos cujo nome começa com um ”.” (ponto).

d) Convenções de nomenclatura: um invasor pode identificar a maneira utilizada pelo site para nomear arquivos ou diretórios. Exemplo: Admin versus admin, backup versus backup, etc.

e) Enumerar contas de usuário: contas pessoais de usuários em um servidor Web frequentemente têm diretórios “home” nomeados após o registro da conta.

f) Arquivos de configuração: esses arquivos podem conter controle de acesso aos dados. Possuem extensões como .conf, .cfg ou .config.

g) Scripts: a maioria dos servidores Web permite a execução de scripts especificando a localização desses scripts (por exemplo: “/cgi-bin”) ou configurando o servidor para testar e executar arquivos baseados em arquivos de permissões (por exemplo: o uso da diretriz do Apache XBitHack). Devido a essas opções, se a indexação de diretórios para conteúdo “cgi-bin” é permitida, é possível baixar ou revisar o código do script se as permissões estão incorretas.

- Escape de Informação: é quando um site da Web revela dados sensitivos, como comentários de um desenvolvedor ou mensagens de erro, que podem auxiliar um invasor a explorar as vulnerabilidades do sistema. Informação sensitiva pode estar presente dentro de comentários esquecidos no código-fonte, mensagens de erro, código-fonte, ou simplesmente esquecidas em alguma parte na interface da aplicação. Há muitas maneiras de um site ser persuadido ao revelar esse tipo de informação. Quando o escape de informação não representar necessariamente uma falha de segurança, fornece a um invasor a orientação útil para que ele realize uma exploração futura. O escape de informação pode possuir vários níveis de risco e deve ser limitado sempre for possível. Exemplo:

```
<TABLE border="0" cellPadding="0" cellSpacing="0"
height="59" width="591">
      <TBODY>
        <TR>
```

```
                <!--If the image files are missing,
        restart VADER -->
                <TD bgColor="#ffffff" colSpan="5"
        height="17" width="587"> </TD>
            </TR>
```

No exemplo anterior pode-se ver um comentário esquecido por algum desenvolvedor, indicando o que deve ser feito caso a imagem não seja mostrada. A brecha de segurança é a revelação do nome de *host* do servidor mencionado explicitamente no código: “VADER”.

- Path Traversal: técnica usada para forçar o acesso a arquivos, diretórios, e comandos que potencialmente residem fora do diretório raiz (denominado como "web document root" or "CGI root") da aplicação Web. Um invasor pode manipular uma URL de maneira que o site irá executar ou revelar os conteúdos de arquivos arbitrários em qualquer localização no servidor Web. Qualquer dispositivo que expõe uma interface baseada em HTTP é potencialmente vulnerável a ataques Path Traversal. Para acessar arquivos ou executar comandos em qualquer localização no sistema de arquivos, essa técnica utiliza seqüências de caracteres especiais para URLs (ex.: “../”). A menos que o servidor Web restrinja adequadamente tentativas de ataques Path Traversal no informadas no caminho da URL, a aplicação Web que está rodando poderá estar vulnerável a ataques desse tipo devido à manipulação imprópria de valores de entrada fornecidos pelo usuário. Exemplos:

    a) Ataques Path Traversal contra um servidor Web:
       Ataque: http://example/../../../../../some/file
       Ataque: http://example/..%255c..%255c..%255csome/file
       Ataque: http://example/..%u2216..%u2216some/file
    b) Ataques Path Traversal contra uma aplicação Web:
       Original: http://example/foo.cgi?home=index.htm
       Ataque: http://example/foo.cgi?home=foo.cgi

- Localização de Recurso Previsível: é uma técnica de ataque usada para descobrir conteúdo e funcionalidades ocultas de um site. Fazendo suposições, o ataque é uma busca for conteúdo que não está disponibilizado para o público. Arquivos temporários, arquivos de *backup*, arquivos de configuração são exemplos de potenciais arquivos a serem procurados por um invasor. Esses arquivos podem conter informações restritas como senhas, nome de máquinas, caminhos de arquivos ou possivelmente conter vulnerabilidades. Por exemplo, o invasor pode tentar acessar esses recursos das seguintes formas:

    a) Busca cega por arquivos comuns e diretórios:

        /admin/

        /backup/

        /logs/

        /vulnerable_file.cgi

    b) Inclusão de extensões para nomes de arquivos existentes: (/test.asp)

        /test.asp.bak

        /test.bak

        /test

### 2.1.6 Ataques Lógicos

A categoria Ataques Lógicos representa os ataques em que ocorre abuso ou exploração do fluxo lógico de uma aplicação Web. Recuperação de senha, registro de conta e compras eCommerce são exemplos de aplicações lógicas. Um site da Web pode solicitar a um usuário executar corretamente um processo que apresenta vários passos para completar uma ação particular. Um invasor pode estar apto a fazer mau uso ou driblar estas características para prejudicar o site e seus usuários. A seguir algumas das técnicas de ataques que estão relacionadas a essa categoria:

• Abuso de Funcionalidade: técnica de ataque que usa as próprias características e funcionalidades de um site para consumir, defraudar, ou driblar mecanismos de controle de acesso. Algumas funcionalidades de um site, possivelmente características de segurança, podem ser abusadas para causar um comportamento inesperado. As técnicas de abuso de funcionalidade são frequentemente combinadas com outras categorias de ataques às aplicações Web, como executar um ataque que introduz uma *string* em uma consulta que retorna uma pesquisa em um *proxy* remoto da Web. Ataques de abuso de funcionalidade são normalmente usados como multiplicador de força. Por exemplo, um invasor pode injetar um *snippet* (trecho de código adicionado por um disparador) do tipo *Cross-site Scripting* em uma site com *web chat* e então usar a função *broadcast* interna desse site para espalhar o código malicioso através do site. Em uma visão geral, todos os ataques eficazes contra sistemas computacionais envolvem o uso dessa técnica. Exemplos:

a) Uso da função de pesquisa de um site para acessar arquivos restritos fora de um diretório Web.

b) Submissão de um subsistema de *upload* de arquivos para substituir arquivos de configuração.

c) Executar um DoS (negação de serviço) inundando (*flooding*) um sistema de *login* com nomes de usuário e senhas existentes e senhas inválidas para bloquear usuários legítimos devido ao limite de tentativas de *login* ter sido atingido.

• Negação de Serviço: conhecida por DoS (*Denial of Service*) é uma técnica de ataque no qual a intenção é evitar que um site possa continuar com seu funcionamento normal e esperado. Ataques DoS são normalmente aplicados para a camada de rede, mas também podem ser usados na camada da aplicação. Esses ataques maliciosos fazem com que ocorra uma escassez de recursos críticos e seja possível a exploração de vulnerabilidades ou abuso de funcionalidades. Muitas vezes ataques DoS tentarão consumir todos os recursos disponíveis em um site da Web como: CPU, memória, espaço em

disco, etc. Quando algum desses recursos atinge sua utilização máxima, o site passa a ficar inacessível. Exemplo: supor um site de Health-Care que gera um relatório com histórico médico. Para cada requisição do relatório, o site consulta o banco de dados para buscar todos os registros que estão associados a um número de seguro. Dado que centenas de milhares de registros estão armazenados no banco de dados (para todos os usuários), o usuário irá esperar em torno de 3 minutos para poder visualizar o seu relatório de histórico médico. Durante esses três minutos, a CPU do servidor atinge 60% de utilização enquanto pesquisa pelos registros. Um ataque DoS passa a enviar 10 requisições simultâneas para a geração desse relatório médico. Isso fará com que o servidor de banco de dados atinja 100% de utilização, o que consequentemente torna o site inacessível para os outros usuários.

- Anti-Automação Insuficiente: é quando um site permite que um invasor possa automatizar um processo que deve apenas ser executado manualmente. Por exemplo, um robô (programa) pode executar centenas de requisições por minuto a um site, fazendo com que haja uma perda de desempenho, e consequentemente tornando-o inacessível.
- Validação Insuficiente de Processo: quando um site permite um invasor contornar o controle de fluxo da aplicação. Se o estado do usuário em relação ao processo não é verificado e reforçado, o site pode estar vulnerável a exploração de conteúdo ou fraudes.

## 2.2 WEB SERVICES

Web Services são aplicações distribuídas, modulares que podem ser publicadas, localizadas e invocadas através da Web. Uma vez que são implantadas, outras aplicações (e Web Services) podem descobrir e invocar os serviços publicados (RAVUTHULA, 2002), que são acessados por meio de protocolos de Internet padrão (HTTP). Uma vantagem no uso de Web Services para aplicações Web é que a aplicação cliente e o servidor não precisam rodar na mesma

plataforma, muito menos rodar a mesma aplicação. A seguir, é apresentado o esquema de funcionamento de Web Services.

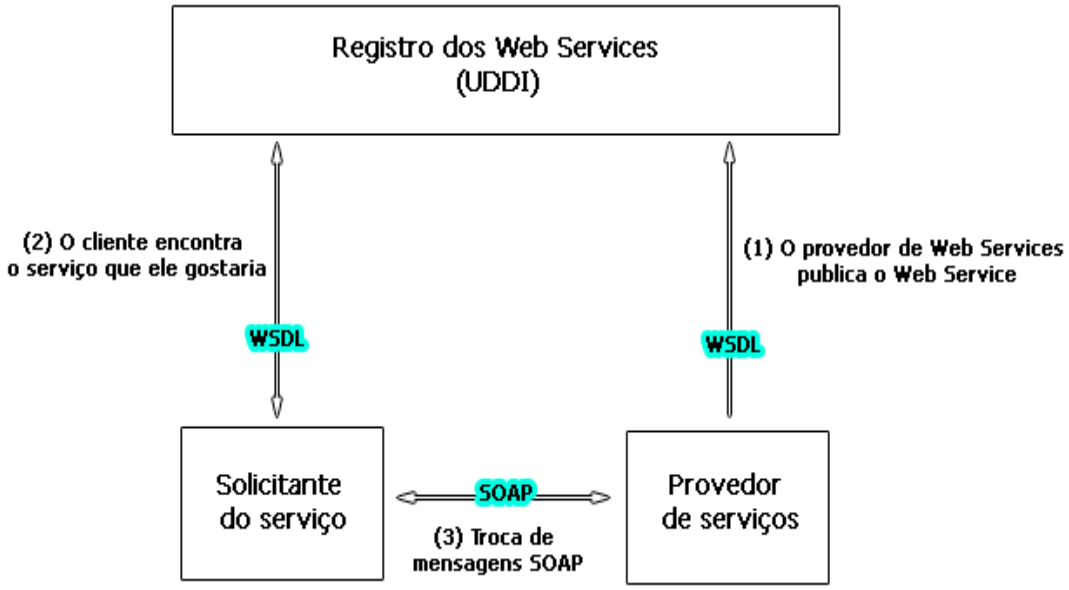


**Figura 2.1 - Esquema de Funcionamento dos Web Services**

Os componentes apresentados na figura acima são (APACHE, 2005):

- WSDL (*Web Services Description Language*): é a linguagem que descreve, em XML, o que um Web Service faz, ou seja, quais os métodos ou operações da aplicação que estarão visíveis aos clientes (SINGHAL, 2006). Um documento WSDL é composto por sete elementos XML que representam as partes abstrata e concreta. Na parte abstrata tem-se quatro elementos (DA CRUZ, 2005):

    1. types: fornece a definição dos tipos de dados para descrever as mensagens trocadas entre aplicações, normalmente representadas por um documento XSD (XML Schema Definition);
    2. message: representa a informação que será trocada através das definições dos tipos de dados;
    3. porttype: é um conjunto de operações suportadas por um ou mais *end points*, onde cada operação se refere a uma mensagem de entrada, saída ou erro;
    4. operation: descreve a ação suportada pelo serviço.

    Na parte concreta tem-se os outros três elementos:

1. binding: define uma especificação de protocolo e formato de dados para as mensagens definidas em um *port type*;
2. port: é um end point, representa a combinação de um *binding* e um endereço de rede;
3. service: é uma coleção de portas. Cada elemento *port* se relaciona com um elemento binding particular, indicando qual interface e qual protocolo de comunicação estão sendo utilizados nessa implementação.

- UDDI (*Universal Description, Discovery and Integration*): protocolo para a publicação e descoberta das descrições dos Web Services. Através dele, os clientes podem ver quais Web Services que estão disponíveis para serem acessados.
- SOAP (*Simple Object Access Protocol*): protocolo de transporte que tem a função de enviar mensagens usando o protocolo HTTP. Essa mensagem é um documento XML e está dividida em três partes:

1. O elemento <Envelope>, é a raiz do documento XML e representa a mensagem propriamente dita;
2. O elemento <Body> que contém as operações e parâmetros a serem entregues ao destinatário da mensagem. Este elemento, de acordo com a especificação SOAP, pode conter um elemento <Fault>, que, quando presente, pode ser utilizado no processamento de falhas do serviço Web. O elemento <Fault> descreve erros de chamada de métodos remotos ou mantém informações acerca do tipo de erro. Portanto, ele conta com os elementos <FaultCode>, <FaultActor>, <FaultString> e <Detail>;
3. O elemento <Header> expande uma mensagem SOAP. Ele define algumas características opcionais e acordos negociáveis entre as partes; seu conteúdo deve ser aceito pelas aplicações que estiverem se comunicando.

No esquema anterior (figura 2.1), o passo (1) significa a criação da especificação em WSDL e a publicação do Web Service provedor (*service provider*) no UDDI. A etapa de publicação é opcional, pois caso o serviço seja publicado, ele

estará acessível por todos. O passo (2) representa a descoberta do serviço que o cliente deseja, que será requisitado pelo *service requestor*, através da definição da especificação do serviço em WSDL. No passo (3) está representada a comunicação através de trocas de mensagens entre o *service requestor* e o *service provider*.

## 2.3 AXIS

O Axis é uma implementação do protocolo SOAP. É utilizado para escrever o serviço no servidor (e disponibilizá-lo através de um servidor de aplicações, por exemplo, o Tomcat) como também para escrever o cliente que solicita os serviços (APACHE-AXIS2, 2008).

Atualmente, o Axis encontra em sua segunda versão, o Axis2. Pelo fato de o Axis2 ser a versão mais recente, a abordagem desse trabalho será focada nessa última versão.

O Axis2 é uma implementação baseada em Java de cliente e servidor Web Services. Fornece um completo modelo de objetos e uma arquitetura modular que facilita a adição de funcionalidades e suporte para novas especificações de Web Services. O Axis2 permite:

- Enviar mensagens SOAP.
- Receber e processar mensagens SOAP.
- Criar um Web Service fora de uma classe Java.
- Criar classes tanto do cliente como do servidor usando WSDL.
- Gerar o WSDL para um serviço.
- Enviar e receber mensagens SOAP com anexos.
- Criar ou utilizar um serviço baseado em REST.
- Criar ou utilizar serviços que seguem as recomendações WS-Security (OASIS, 2004), WS-ReliableMessaging, WS-Addressing, WS-Coordenation, e WS-Atomic Transaction.

- Adicionar suporte a novas recomendações WS- da W3C (organização que define padrões para a Web).

A figura 2.2 a seguir representa a estrutura de funcionamento do Axis2 tanto no lado do cliente (remetente) como no lado do servidor (destinatário):

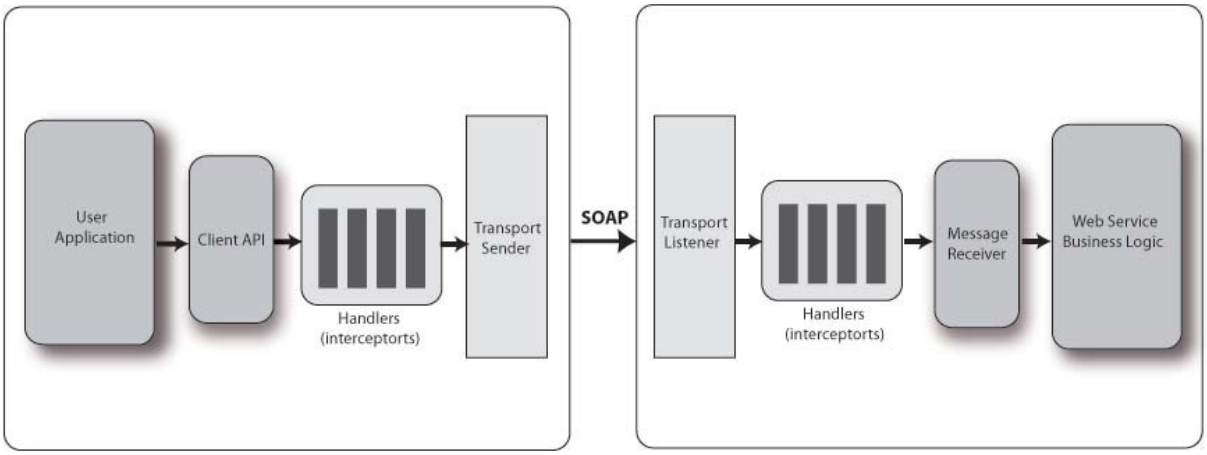


**Figura 2.2 - Esquema de Funcionamento do Axis2 (APACHE-AXIS2, 2005)**

Assumindo que Axis2 está rodando em ambos os lados, o processo de troca de mensagens consiste nos seguintes passos:

- O remetente cria uma mensagem SOAP.
- Os *handlers* do Axis executam todas as ações necessárias (ex. criptografia de mensagens que seguem especificações WS-Security).
- O cliente envia a mensagem.
- No destinatário o *listener* detecta a mensagem.
- O *listener* passa a mensagem para qualquer um dos handlers no lado receptor.
- Uma vez que a mensagem foi processada na fase *pre-dispatch*, ela é entregue pelos *dispatchers*, que a repassam para a aplicação apropriada.

No Axis2 essas ações estão divididas em fases, com diversas fases pré-definidas, como *pre-dispatch*, *dispatch* e processamento da mensagem. Cada fase é uma coleção de *handlers*. Axis2 permite ao desenvolvedor controlar que *handlers* entram em tais fases, bem como a ordem em que esses *handlers* são executados

dentro dessas fases. Também é possível criar e adicionar novas fases e *handlers.* Os *handlers* vêm de módulos que podem ser acoplados em um sistema que roda Axis2. Esses módulos como o Rampart, que fornece uma implementação do WS-Security, e Sandesha, que fornece uma implementação do WS-ReliableMessaging, são os principais mecanismos de extensibilidade no Axis2.

## 2.4 SEGURANÇA EM WEB SERVICES

Os problemas de segurança existentes em Web Services são os mesmos existentes na Web, mas considerando sua arquitetura e modo de comunicação, além destes, podem existir problemas de segurança na troca de mensagens entre essas aplicações. Pode-se então afirmar que os principais desafios para a construção de uma arquitetura de Web Services segura são:

- Integridade: garantir a que os dados transmitidos pelo protocolo SOAP não sejam alterados indevidamente;
- Confidencialidade: garantir que os dados transmitidos pelo protocolo SOAP não possam ser acessados durante seu transporte por entidades não autorizadas.
- Autenticação: garantir a identidade da fonte das informações.
- Autorização: garantir que os acessos aos métodos são realizados apenas com a devida permissão dos responsáveis.
- Análise de conteúdo: garantir que o conteúdo das mensagens trocadas esteja adequado e que não causará danos ao receptor.

A seguir, os tipos de ataques encontrados em Web Services que estão relacionados aos desafios citados anteriormente (WS-I, 2005):

- Alteração na Mensagem: a mensagem é alterada inserindo, removendo ou modificando informações criadas pelo remetente da informação, que é confundida pelo receptor como sendo a intenção do remetente.

- Falsificação de Mensagens: mensagens falsas são construídas e enviadas pelo invasor para um receptor, que acredita que estas vieram de um remetente confiável.
- *Man in the Middle*: um invasor se passa por um usuário para o remetente e ao receptor reais com o objetivo de enganar ambos os participantes.
- Disfarce (*Spoofing*): é enviada uma mensagem de forma que ela pareça ter sido enviada por um outro usuário existente. É uma variação da técnica usada em Falsificação de Mensagens.
- Reivindicações Forjadas: uma mensagem é enviada pelo invasor contendo diretivas de segurança forjadas, com o intuito de obter acesso a informações na qual não está autorizado. Exemplo: um *token* de segurança é colocado na mensagem, mesmo que este não tenha sido realmente fornecido pela autoridade especificada.
- Repetição de Partes de Mensagem: uma mensagem enviada pelo invasor contém partes de outra mensagem para a tentativa de obter acesso a alguma informação restrita ou para fazer com que o receptor tome alguma ação. Exemplo: é adicionado na mensagem um *token* de segurança.
- Repetição: uma mensagem é reenviada por inteiro pelo invasor.
- Negação de Serviço: o invasor realiza uma pequena quantidade de trabalho e força o sistema a realizar uma grande quantidade de trabalho com o objetivo de aumentar a utilização de recursos, o que faz com que o sistema se torne inacessível.

Atualmente, devido à larga adoção de Web Services, pode-se afirmar que os mesmos tem sido alvo de ataques maliciosos (visando o roubo de informações ou causando mau funcionamento do sistema). Esse fato demonstra que em Web Services existe a necessidade de implantar serviços de segurança para que seja possível proteger seus recursos (tanto informações como serviços oferecidos). Os serviços de segurança devem manter seguros a comunicação e os dados armazenados, bem como a execução apropriada e contínua dos Web Services (STOUPA, 2005).

## 2.5 INTERMEDIÁRIOS

Intermediários são entidades posicionadas entre um cliente e um fornecedor de serviço que visa prover funcionalidades adicionais e serviços de valor agregado. Intermediários baseados na Web estão sendo utilizados amplamente, oferecendo funções como customização, personalização, *caching*, filtragem e transcodificação, modificando e reforçando dados como eles fluem entre um cliente Web e um servidor. O funcionamento dos intermediários ocorre através da interceptação de mensagens, executando funções, e encaminhando a mensagem alterada para o último receptor. Podem existir vários intermediários entre um cliente e um serviço, de forma que as mensagens são encaminhadas de um intermediário para outro até atingir o último destino.

O uso de intermediários oferece uma forma não intrusiva de estender funcionalidades do cliente e do servidor. Eles oferecem flexibilidade também, desde que eles possam ser dinamicamente adicionados e removidos.

### 2.5.1 Intermediários SOAP

Protocolos de Web Services, como o SOAP, fornecem suporte claro, descentralizado e modular para intermediários. Portanto, intermediários podem ser implementados como serviços SOAP através do modelo de extensibilidade desse protocolo. A segurança da informação permanece contida dentro da mensagem SOAP e/ou anexo de mensagem SOAP que permite que a segurança da informação “viaje” junto com a mensagem ou o anexo da mensagem (RAHAMAN, MOHAMMAD, 2006).

Intermediários SOAP podem processar as mensagens destinadas a eles e, em seguida, encaminhar estas mensagens para o seu próximo destino. Estes intermediários podem estar ao longo do caminho que percorrem as mensagens SOAP entre o cliente e o servidor, como está representado na figura a seguir:

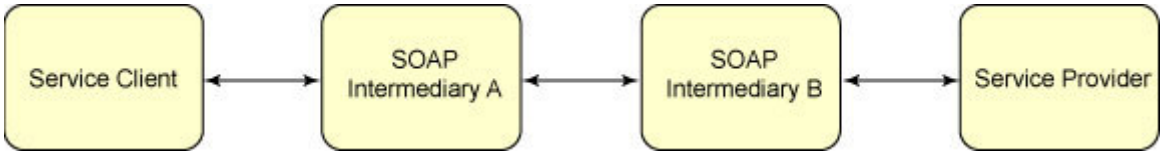


**Figura 2.3 - Caminho de uma mensagem SOAP usando intermediários (IBM, 2002)**

Intermediários SOAP (ou nodos de processamento SOAP) podem ser incluídos ao longo do caminho da que a mensagem percorre utilizando-se o cabeçalho SOAP da mesma. Os atributos de cabeçalho – *actor* e *mustUnderstand* – determinam quem deve processar o cabeçalho e se o processamento deve ser obrigatório ou não.

O atributo SOAP *actor* desempenha o papel principal na identificação de que parte do cabeçalho se destina para qual receptor. O receptor pode ser um intermediário ou o último destino da mensagem SOAP. Ambos os intermediários e o último destino são identificados por um URI (Universal Resource Identifier), que consiste em um endereço que identifica o caminho para os mesmos. Quando o atributo *actor* não está presente, então o processador SOAP para a entrada de cabeçalho é o último destino da mensagem SOAP.

O atributo SOAP *mustUnderstand* é utilizado para indicar se uma entrada de cabeçalho é obrigatória ou opcional. O valor desse atributo pode ser 1, indicando que a entrada de cabeçalho deve ser processada ou 0, que indica que esse processamento é opcional. Por padrão, quando esse atributo não é informado, é assumido 0 para o seu valor.

Os intermediários SOAP devem seguir o modelo de processamento de acordo com a especificação SOAP para processamento de entradas de cabeçalho. Esse modelo de processamento estabelece que:

- Um intermediário pode apenas processar a entrada de cabeçalho SOAP que for endereçada a ele.
- Antes de encaminhar uma mensagem ao próximo receptor, o intermediário deve remover a entrada de cabeçalho endereçada a ele.
- Um intermediário pode adicionar novos intermediários ao caminho da mensagem adicionando novas entradas para os atributos *actor* e *mustUnderstand.*

- Uma mensagem de falha deve ser gerada quando ocorrer uma falha em um intermediário. O elemento SOAP *Fault* gerado nessa mensagem deve conter um elemento *faultactor*, cujo valor indica o intermediário no qual ocorreu a falha.

Para que seja possível uma implementação de intermediários SOAP, será necessárias a implementação das seguintes funcionalidades:

- Roteamento
- Segurança
- Conhecimento de intermediários suportado pelo provedor de serviços

Roteamento é o processo de entregar uma mensagem SOAP através de intermediários SOAP. Por exemplo, considerar uma mensagem SOAP vai de um serviço cliente até um provedor de serviços através dos intermediários A e B. A mensagem SOAP enviada do serviço cliente direciona uma parte da mensagem para o intermediário A e outra parte para o intermediário B usando o atributo *actor*, mas este não contém a informação de roteamento para a mensagem, ou seja, ele não informa qual intermediário deve vir primeiro no caminho da mensagem.

A especificação WS-Routing define um modelo de caminho da mensagem que é totalmente compatível com o modelo de mensagem SOAP e que torna possível descrever a troca completa de uma mensagem SOAP do serviço cliente ao provedor de serviço. Essa especificação define uma entrada de cabeçalho SOAP, que descreve um caminho de ida, e opcionalmente um caminho de volta, para uma mensagem SOAP. Esse caminho de ida e volta da mensagem segue o modelo SOAP de requisição/resposta.

WS-Routing descreve o iniciador da mensagem SOAP, de onde a mensagem parte, como sendo o remetente inicial. A entrada de cabeçalho do WS-Routing indica o último receptor e zero ou mais intermediários no caminho de ida da mensagem. O último receptor é o destino final da mensagem enviada pelo remetente inicial. As regras para a troca das mensagens são as mesmas tanto para o caminho de ida como o de volta. A seguir as regras para a descrição do caminho da mensagem do remetente para o receptor:

- O remetente WS-Routing inicial deve gerar um cabeçalho *path* para indicar o caminho da mensagem. Os intermediários no caminho da mensagem devem ser indicados através do elemento *via*. Esse elemento deve estar presente como um subelemento do elemento *fwd* ou *rev.* O elemento fwd descreve a mensagem que percorrerá o caminho de ida, e *rev* descreve o caminho inverso da mensagem, sendo esse ultimo opcional. Se o elemento *rev* está presente, o caminho inverso da mensagem é construído dinamicamente quando a mensagem percorre o caminho de ida. O último destino da mensagem WS-Routing pode ser indicado usando um elemento *to.* Se o elemento *to* não está presente dentro da entrada *path* do cabeçalho, então o último destino é designado pelo último elemento *via.* Os elementos *to* e *via* identificam o receptor e os intermediários através de uma URI.
- O receptor WS-Routing deve inspecionar o elemento *path* após receber a mensagem, com o objetivo de verificar se ele é de fato o destinatário da mensagem. Se não há subelementos *via* no elemento *fwd*, ou não há elemento *fwd*, então o elemento *to* deve ser inspecionado para encontrar essa informação. Uma falha WS-Routing deve ser gerada quando o elemento *to* é vazio ou identifica um receptor diferente. Se um elemento *fwd* que contém um ou mais elementos *via* está presente, o receptor deve remover e inspecionar o primeiro elemento *via* de *fwd* e verificar que ele é o destinatário da mensagem. Caso não seja, o receptor deve gerar um erro. Esse receptor se tornará o último destino quando não há elementos *via* listados no elemento *fwd* após a remoção do primeiro elemento *via*, e não existe o elemento *to*.
- O intermediário WS-Routing deve inspecionar o elemento *fwd* pelos elementos *via*. Se eles estão presentes, o intermediário deve encaminhar a mensagem WS-Routing para o próximo intermediário no caminho da mensagem, obtendo a identidade deste através do elemento *via.* Se os elementos *via* não estão presentes, então a mensagem deve ser encaminhada para o último destino (identificado pelo elemento *to*). Se um elemento *rev* está presente, então o intermediário deve adicionar um elemento *via* como o primeiro subelemento desse elemento *rev*. O valor desse elemento *via* deve indicar o caminho reverso. Uma falha deve ser

gerada quando o processo de encaminhamento falhar ou a informação do caminho inverso não puder ser fornecida.

- Se o caminho de volta de uma mensagem WS-Routing é utilizada para o envio de uma mensagem, a lista ordenada de elementos no elemento *rev* é usada como a lista de elementos *via* no elemento *fwd.*
- Apenas um remetente inicial pode inserir um elemento *rev* para descrever o caminho inverso da mensagem.

A mensagem WS-Routing é transmitida da mesma forma que as mensagens SOAP são entregues. Ela é entregue para o primeiro receptor WS-Routing do caminho de ida da mensagem sobre o protocolo subjacente. Se este receptor é um intermediário, então a mensagem é enviada para o próximo receptor do caminho de ida da mensagem, podendo este receptor ser um intermediário ou o destino final.

Os códigos a seguir apresentam um exemplo de encaminhamento de mensagem através da especificação WS-Routing. O serviço cliente é o remetente inicial e o provedor de serviços é o ultimo receptor. A mensagem passa através de um intermediário WS-Routing.

Código 1: Uma mensagem do remetente inicial WS-Routing para o intermediário WS-Routing

```
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header>
    <mp:path xmlns:mp="http://schemas.xmlsoap.org/rp/"
      SOAP-ENV:actor="http://schemas.xmlsoap.org/soap/actor/next" SOAP-
ENV:mustUnderstand="1">
      <mp:action>http://www.ibm.com/quotes</mp:action>
      <mp:to>http://www.ibm.com/quotesservice</mp:to>
      <mp:fwd>
         <mp:via>http://www.compress.com/compress</mp:via>
      </mp:fwd>
      <mp:rev>
         <mp:via/>
      </mp:rev>
      <mp:from/>
         <mp:id>uuid:AS2324S4-CE2A-F9DC-BCB7-AS42B16F61E1</mp:id>
    </mp:path>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
        ...
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Código 2: Uma mensagem do intermediário WS-Routing para o provedor de serviços WS-Routing

```
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header>
    <mp:path xmlns:mp="http://schemas.xmlsoap.org/rp/"
       SOAP-ENV:actor="http://schemas.xmlsoap.org/soap/actor/next" SOAP-
ENV:mustUnderstand="1">
      <mp:action>http://www.ibm.com/quotes</mp:action>
      <mp:to>http://www.ibm.com/quotesservice</mp:to>
      <mp:fwd/>
      <mp:rev>
        <mp:via/>
      </mp:rev>
      <mp:from/>
      <mp:id>uuid:AS2324S4-CE2A-F9DC-BCB7-AS42B16F61E1</mp:id>
    </mp:path>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
        ...
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Outra funcionalidade que deve ser fornecida pelos intermediários é a segurança, cujo tema é o foco desse trabalho. Devido ao fato de as mensagens passarem da sua origem até o último destino, passando por intermediários, pode-se afirmar que esses intermediários, pelo fato de estarem intermediando a comunicação entre dois pontos, são vulneráveis a ataques *man-in-the-middle* (ou homem intermediário, como explicado anteriormente. É um ataque no qual o invasor simula um intermediário, fazendo com que remetente e destinatário acreditem estar se comunicando diretamente, tornando possível a interceptação da mensagem pelo invasor). Portanto, brechas de segurança em sistemas que utilizam intermediários, podem ocasionar em sérios problemas de segurança, como perda de informação e integridade da mensagem.

Atualmente, os protocolos SSL (Secure Socket Layer) e TSL (Transport Layer Security) são usados para fornecer segurança no nível de transporte para Web Services. Embora a segurança de mensagens SOAP que não passam por intermediários possa ser assegurada por SSL/TLS, brechas de segurança nos intermediários ocasionarão uma comunicação fim-a-fim insegura.

A especificação WS-Security, um modelo de segurança para Web Services proposto pela IBM e Microsoft, oferece suporte ao uso de mecanismos de segurança

como SSL/TLS juntamente com outras especificações WS para promover integridade e confidencialidade através de múltiplos transportes, intermediários e protocolos de transmissão.

Outra necessidade na implementação de intermediários é a de que o cliente deve saber sobre os intermediários suportados pelo provedor de serviços. É necessário que o cliente saiba quais intermediários estão processando a requisição no caminho da mensagem antes que a mesma atinja o seu último destino.

# 3 PROPOSTA DE UMA FERRAMENTA PARA INCLUSÃO DE SEGURANÇA EM UMA ARQUITETURA DE WEB SERVICES

A proposta desse trabalho consiste em definir e implementar uma ferramenta que acrescente mecanismos de segurança para os Web Services baseando-se no conceito do uso de intermediários.

A motivação para a construção dessa arquitetura é a automatização da tarefa de configurar a segurança do servidor como também das aplicações clientes.

Após a configuração de segurança realizada pela ferramenta, intermediários realizarão o controle de segurança nas mensagens SOAP, acrescentando assinaturas e criptografia a essas mensagens. O controle de segurança adicionado a cada mensagem transmitida será baseado no uso de padrões WS, como será explicado a seguir, para estabelecer uma comunicação fim-a-fim segura, garantindo integridade e confidencialidade às mensagens enviadas tanto do cliente para o servidor, como do servidor para o cliente.

Na figura a seguir, está representada quais os padrões de segurança serão utilizados pelo intermediário:

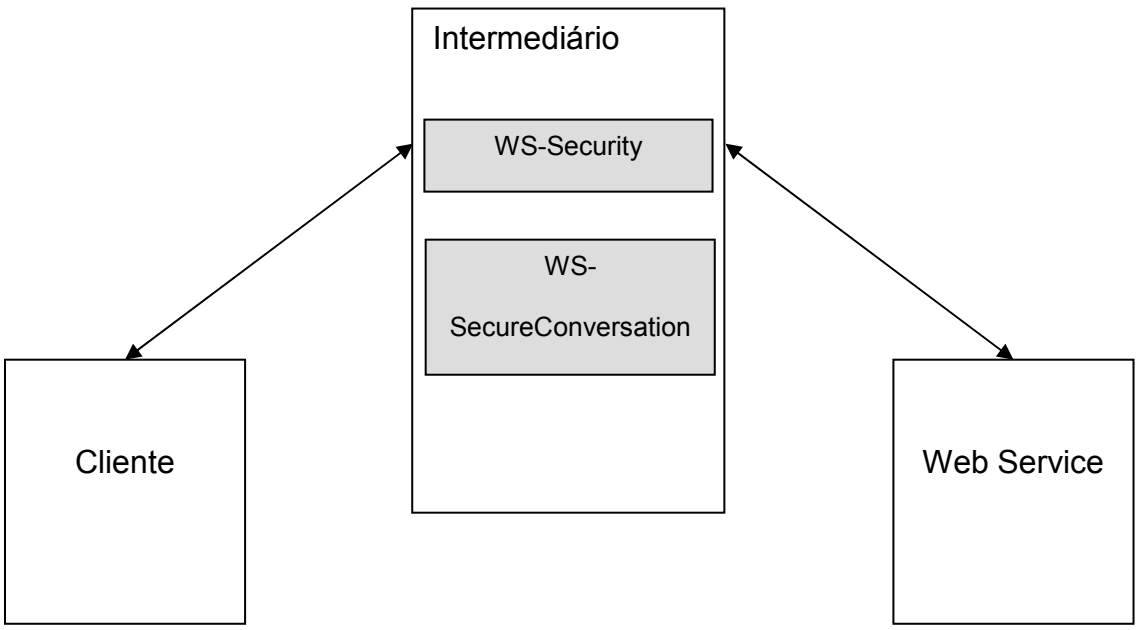


**Figura 3.1 – Padrões de segurança em Web Services utilizados pelo intermediário**

Como pode se observar na figura 3.1, o intermediário será responsável por acrescentar o controle de segurança nas mensagens SOAP utilizando os padrões WS-Security e WS-SecureConversation.

WS-Security descreve como anexar assinaturas e cabeçalhos para mensagens SOAP, bem como anexar *tokens* de segurança, incluindo *tokens* de segurança binários como certificados X.509 e tíquetes Kerberos. Com o objetivo de manter a integridade das mensagens, será utilizado o padrão XML Signature em conjunto com *tokens* de segurança. Esse padrão permite a associação de assinaturas aos cabeçalhos de uma mensagem, permitindo que alterações na mesma possam ser detectadas. Visando manter a confidencialidade da mensagem, adotar-se-á o padrão XML Encryption em conjunto com *tokens* de segurança. O padrão citado permite que partes de uma mensagem SOAP sejam criptografadas.

WS-SecureConversation descreve como gerenciar e autenticar trocas de mensagens entre as partes incluindo a troca do contexto de segurança e estabelecendo e derivando chaves de sessão.

A seguir será abordado como esses padrões serão utilizados nas mensagens SOAP para prover mecanismos de segurança.

## 3.1 WS-SECURITY

O padrão WS-Security é representado em uma mensagem SOAP através do bloco de cabeçalho <wsse:Security> que anexará informações de segurança para um receptor específico, indicado como um SOAP *actor*/*role.* Esse último pode ser tanto o último receptor da mensagem ou um intermediário. Um intermediário será responsável por adicionar um ou mais subelementos (cabeçalhos com assinatura, cabeçalhos para criptografia e cabeçalhos dos padrões WS-) para um bloco de cabeçalho <wsse:Security> existente se esses subelementos são endereçados para esse intermediário. Informações de segurança em uma mensagem endereçadas para diferentes destinos deverão aparecer em diferentes blocos de cabeçalho

<wsse:Security>. Isso é devido a potenciais problemas de ordem de processamento dos cabeçalhos.

Como subelementos são adicionados a um bloco de cabeçalho <wsse:Security>, eles deverão ser prefixados aos elementos existentes. Como tal, um bloco de cabeçalho <wsse:Security> representa os passos de assinatura e criptografia que o produtor da mensagem realizará para criar a mensagem. Essa regra de prefixação assegura que a aplicação receptora da mensagem pode processar os subelementos na ordem em que eles aparecem no bloco de cabeçalho <wsse:Security>, pois não haverá dependência à frente entre os subelementos.

### 3.1.1 Assinaturas Digitais

Para simplificar o processamento pelos intermediários e receptores da mensagem, um atributo em comum é definido para identificar um bloco de cabeçalho <wsse:Security>, o atributo wsu:ID. Esse atributo referencia uma assinatura digital definida no bloco de cabeçalho <ds:Signature>, conforme o padrão XML Signature. Esse cabeçalho utiliza algoritmos que terão a função de gerar assinaturas (valores gerados criptograficamente) que serão associados aos elementos de cabeçalho e corpo da mensagem.

### 3.1.2 Criptografia

Através dos elementos <xenc:ReferenceList> e <xenc:DataReference>, do padrão XML Encryption, será possível criptografar partes de uma mensagem SOAP. Quando um intermediário criptografar partes de uma mensagem SOAP usando XML Encryption, ele deverá prefixar um subelemento ao bloco de cabeçalho. Através do elemento <xenc:ReferenceList> é possível indicar quais partes da mensagem serão criptografadas. O elemento <xenc:DataReference> referencia a parte da mensagem

a ser criptografada. É necessário ainda, de acordo com o padrão XML Encryption, o elemento <xenc:EncryptedData>, onde estarão os elementos que contém as informações (valores dos objetos) que serão criptografadas.

## 3.2 WS-SECURECONVERSATION

Enquanto que a autenticação da mensagem é útil para envio de mensagens simples ou mensagens que possuem uma única direção, as partes que desejam trocar várias mensagens devem estabelecer um contexto que permite a troca de várias mensagens. Um contexto de segurança é compartilhado entre as partes que se comunicam durante o tempo de uma sessão. O padrão WS-SecureConversation (BOHREN, JEFF et al, 2005) define um contexto de segurança que é representado pelo elemento <wsc:SecurityContextToken>, que é um *token* de segurança. As referências para esse elemento dentro de uma mensagem SOAP devem ser feitas através do atributo wsu:Id ou uma referência do elemento <wsse:Reference> para o elemento <wsc:Identifier>. Uma vez que o contexto e o segredo (assinatura) foram estabelecidos (autenticados pela correspondência da assinatura), os mecanismos descritos nesse elemento podem ser usados para computar chaves derivadas para cada chave usada no contexto de segurança.

## 3.3 ACEITAÇÃO OU REJEIÇÃO DE UMA MENSAGEM

Uma mensagem que passa por um intermediário terá os mecanismos de segurança adicionados como os elementos citados neste capítulo. Antes de a mensagem chegar ao destino final, serão validados os *tokens* contidos nessa mensagem consultando o intermediário. No caso de ter sido gerada uma mensagem de falha, indicará que a mensagem enviada pelo remetente não possui as credenciais necessárias ou sofreu algum tipo de ataque durante o seu percurso,

caracterizando a não aceitação da mensagem. Caso contrário, a mensagem poderá ser entregue ao destino final.

## 3.4 ARQUITETURA PARA TROCA DE MENSAGENS UTILIZANDO INTERMEDIÁRIOS

Nesse capítulo será detalhado o funcionamento da arquitetura de intermediários a ser estabelecida pela ferramenta. A seguir serão descritos os passos envolvidos na troca de mensagens entre a aplicação cliente e o Web Service, no qual é realizada a validação das mensagens SOAP através de intermediários.

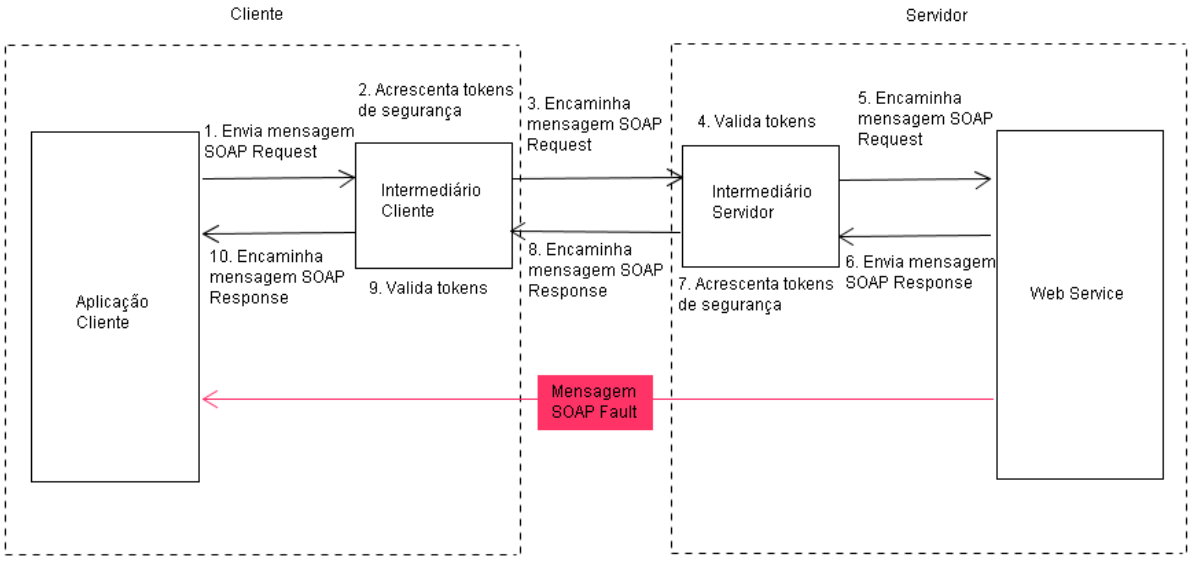


**Figura 3.2 - Arquitetura para troca de mensagens SOAP utilizando Intermediários**

1. A aplicação cliente envia uma mensagem SOAP Request que invoca um determinado método do Web Service.

2. A mensagem é processada pelo Intermediário Cliente 1. Esse processamento consiste em acrescentar os tokens de segurança, também acrescentando uma assinatura e criptografando o corpo da mensagem SOAP Request.

3. O Intermediário Cliente 1 encaminha a mensagem SOAP Request para o próximo intermediário que está no lado Servidor.

4. A mensagem SOAP Request é validada pelo Intermediário Servidor 2. Se a mensagem não possuir algum dos elementos de segurança considerados obrigatórios, então será retornada uma mensagem SOAP com o elemento Fault para a aplicação cliente quando esta mensagem for recebida pelo Web Service.

5. O Intermediário Servidor 2 encaminha a mensagem para o Web Service.

6. O Web Service processa a mensagem SOAP Request e envia de volta uma mensagem SOAP Response com o resultado desse processamento.

7. O Intermediário Servidor 1 acrescenta os tokens de segurança, também acrescentando uma assinatura e criptografando o corpo da mensagem SOAP Response.

8. A mensagem então é encaminhada para o lado Cliente.

9. O Intermediário Cliente 2 valida a mensagem SOAP Response. Se a mensagem não possuir algum dos elementos de segurança considerados obrigatórios, então será retornada uma mensagem SOAP com o elemento Fault para a aplicação cliente.

10. Por fim a mensagem SOAP Response é encaminhada para a aplicação cliente.

## 3.5 CONFIGSECURITYWS

Neste capitulo será apresentada o ConfigSecurityWS, a ferramenta desenvolvida com o objetivo de configurar as aplicações clientes e o Web Service (servidor), acrescentando o controle de segurança para a troca de mensagens SOAP.

### 3.5.1 Motivação

A seguir é apresentado o cenário no qual se percebe a necessidade de automatizar a configuração de segurança em uma arquitetura do tipo cliente-servidor.

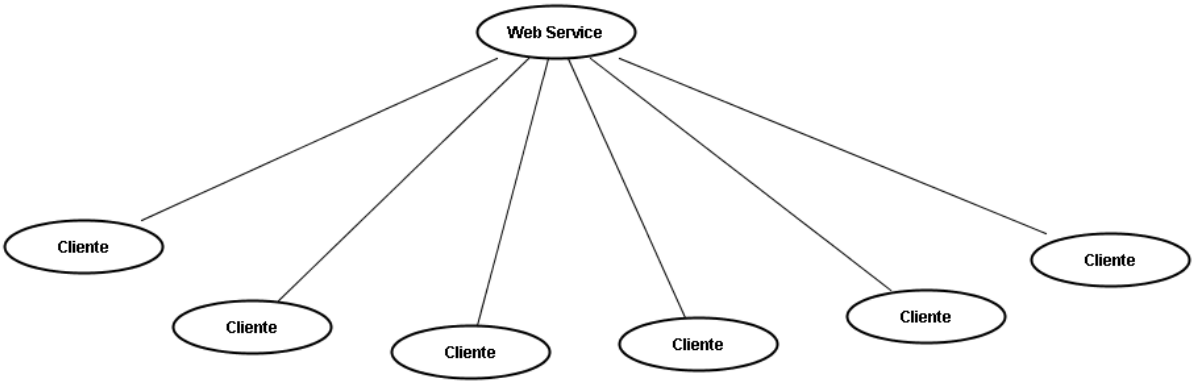


**Figura 3.3 – Arquitetura cliente-servidor com mais de um cliente**

A figura 3.3 demonstra que é possível existir vários clientes que utilizam os serviços oferecidos pelo Web Service. Configurar cada cliente manualmente para permitir uma comunicação segura demandaria muito tempo. Sabendo que atualmente o mercado de TI exige um projeto com o menor custo, no menor prazo e com a qualidade máxima, esta ferramenta permitirá automatizar o processo de configuração das aplicações clientes e do Web Service, reduzindo o tempo necessário para essa tarefa.

### 3.5.2 Cenário para a atuação da ferramenta

O objetivo da ferramenta consiste em configurar os arquivos .java e .xml, para que seja possível a inclusão de segurança nas mensagens SOAP em uma arquitetura de aplicação Axis2.

Na figura 3.4 é apresentado um diagrama de componentes que demonstra como exemplo uma arquitetura cliente-servidor que não possui segurança para as trocas de mensagem SOAP.

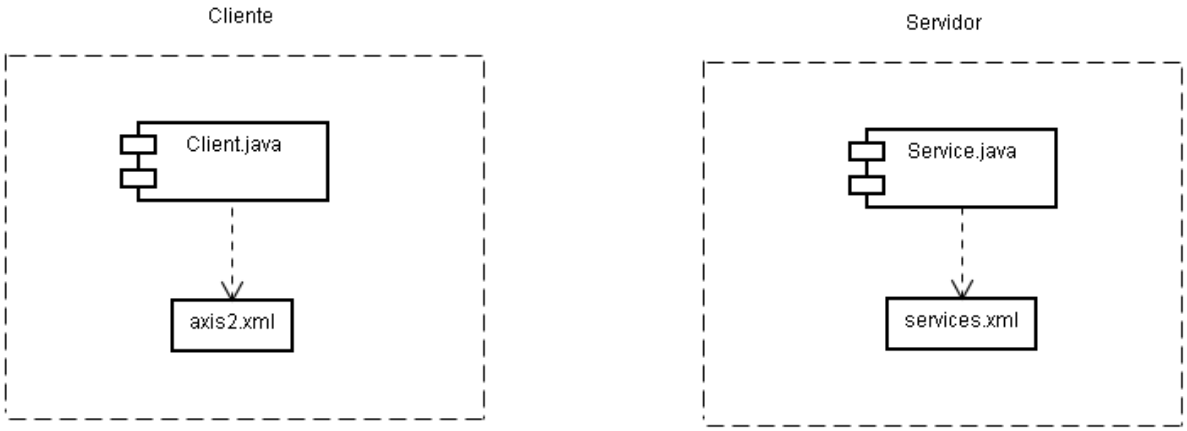


**Figura 3.4 – Diagrama de Componentes – Arquitetura cliente-servidor sem controle de segurança**

O arquivo axis2.xml contém as configurações necessárias para inclusão de módulos para realização de tarefas que também podem ser reutilizadas em outras aplicações, por exemplo, módulo de segurança e módulo para geração de *logs*. Não é obrigatória a sua existência no lado cliente, desde que não haja a necessidade de acrescentar algum módulo.

A figura 3.5 representa a arquitetura apresentada na figura 3.4 após a ferramenta ter realizado as alterações necessárias para acrescentar segurança na arquitetura. Na cor cinza estão representados os arquivos que foram alterados.

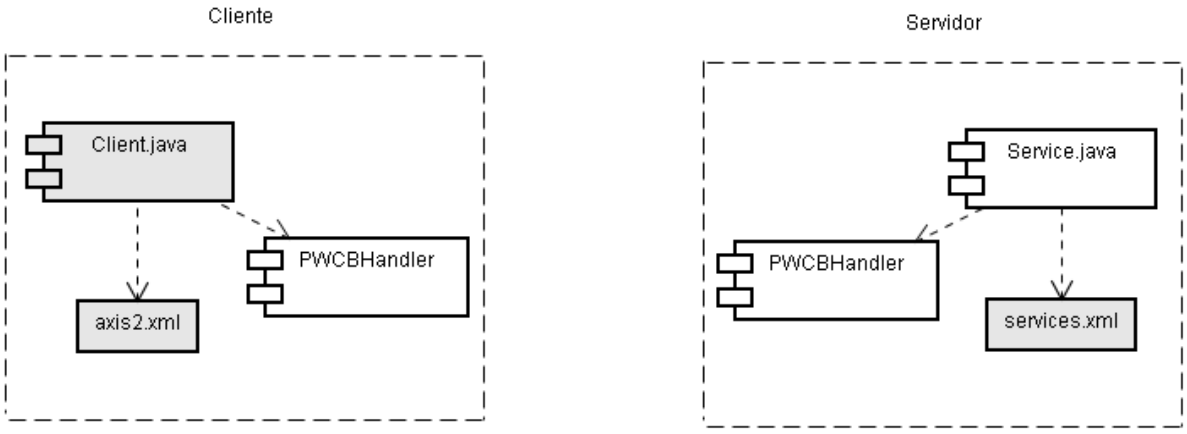


**Figura 3.5 – Diagrama de Componentes – Arquitetura cliente-servidor com controle de segurança**

### 3.5.3 Casos de Uso

Para a implementação do ConfigSecurityWS foram identificadas duas funcionalidades: Incluir segurança no lado Cliente e Incluir segurança no lado Servidor. A seguir é apresentado o diagrama de casos de uso.

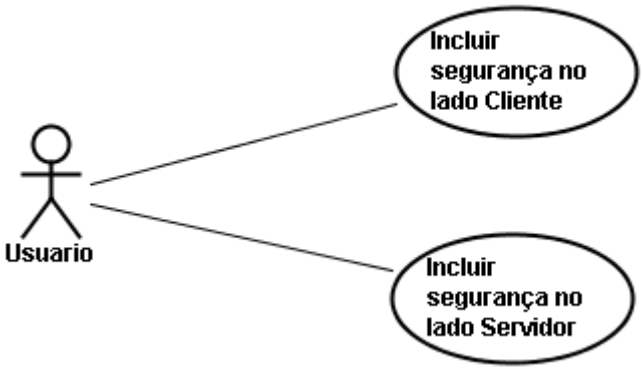


**Figura 3.6 – Diagrama de Casos de Uso**

Nas seções a seguir serão detalhados os casos de uso em questão, apresentando os diagramas de atividades.

### 3.5.4 Ações executadas no lado Cliente

A ferramenta analisará a estrutura de diretórios da aplicação com o objetivo de incluir segurança, realizando as seguintes tarefas:

- Inclusão da chave de segurança;
- Inclusão do intermediário que realizará a autenticação das mensagens, a classe PWCBHandler;
- Alteração dos arquivos axis2.xml e dos clientes que estão invocando os serviços, sendo possível que o usuário determine quais classes clientes possuirão controle de segurança.

Todas essas alterações não serão feitas diretamente no código da aplicação do usuário. A ferramenta cria uma cópia completa da estrutura de diretórios e arquivos da aplicação – representada através do subdiretório “\saida” – permitindo que as configurações de segurança a serem realizadas não afetem diretamente o código da aplicação.

O diagrama de atividades a seguir demonstra quais serão as ações executadas pela ferramenta no lado cliente.

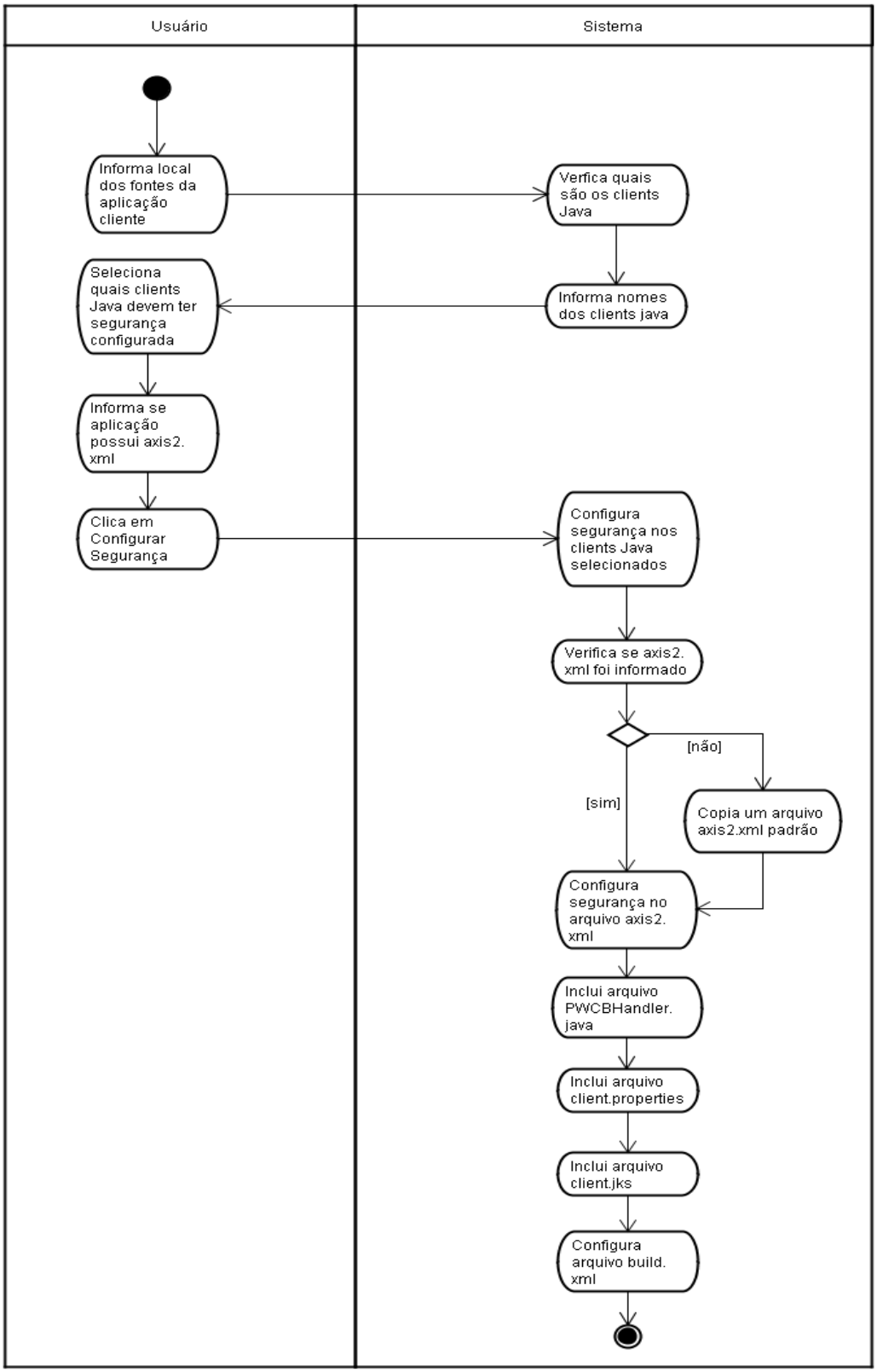


**Figura 3.7 – Diagrama de Atividades - Incluir segurança no lado Cliente**

Alterações nas classes Java

Nas classes do cliente que invocam métodos do Web Service será incluída a chamada de um método que especifica a utilização do axis2.xml por essas classes. Esse método permite a criação de um contexto para a mensagem, nesse caso um contexto de segurança.

A seguir é apresentado um exemplo de um cliente Java que ainda não possui a configuração de segurança, a classe FuncionarioClient, explicada em maiores detalhes no capítulo 5.

```
 1  package exemplo.rh.rpcclient;
 2
 3  import javax.xml.namespace.QName;
 4
 5  import org.apache.axis2.AxisFault;
 6  import org.apache.axis2.addressing.EndpointReference;
 7  import org.apache.axis2.client.Options;
 8  import org.apache.axis2.rpc.client.RPCServiceClient;
 9
10  import exemplo.rh.persistencia.ContaBancaria;
11
12  public class ContaBancariaClient {
13
14      public static void main(String[] args1) throws AxisFault {
15
16          RPCServiceClient serviceClient = new RPCServiceClient();
17
18          Options options = serviceClient.getOptions();
19
20          EndpointReference targetEPR = new EndpointReference(
21                  "http://127.0.0.1:8080/axis2/services/RHService");
22          options.setTo(targetEPR);
23
24          // ////////////////////////////////////////////////////////////////////
25
26          /*
27           * Cria uma ContaBancaria e armazena no cadastro de RH.
28           */
29
30          // QName do metodo destino
31          QName opIncluirContaBancaria = new QName("http://service.rh.exemplo",
    "incluirContaBancaria");
32
33          /*
34           * Construindo nova conta
```

```
35          */
36          ContaBancaria conta1 = new ContaBancaria();
37
38          conta1.setContaCorrente(123456789);
39          conta1.setAgencia(123);
40          conta1.setNomeBanco("Banco do Brasil");
41
42
43          // Construção do array de argumentos para a chamada do metodo
44          Object[] opIncluirContaBancariaArgs = new Object[] { conta1 };
45
46          // Chamando o metodo
47          options.setAction("urn:incluirContaBancaria");
48          serviceClient.invokeRobust(opIncluirContaBancaria, opIncluirContaBancariaArgs);
49
50          ////////////////////////////////////////////////////////////////////////
51
52      }
53  }
```

Código 3: Cliente Java sem segurança configurada

Após executar a ferramenta, serão realizadas as alterações como mostrado a seguir.

```
 1  package exemplo.rh.rpcclient;
 2
 3  import javax.xml.namespace.QName;
 4  import org.apache.axis2.context.ConfigurationContext;
 5  import org.apache.axis2.context.ConfigurationContextFactory;
 6
 7  import org.apache.axis2.AxisFault;
 8  import org.apache.axis2.addressing.EndpointReference;
 9  import org.apache.axis2.client.Options;
10  import org.apache.axis2.rpc.client.RPCServiceClient;
11
12  import exemplo.rh.persistencia.ContaBancaria;
13
14  public class ContaBancariaClient {
15
16      public static void main(String[] args1) throws AxisFault {
17  ConfigurationContext ctx =
    ConfigurationContextFactory.createConfigurationContextFromFileSystem(args1[1], args1[1] +
    "/conf/axis2.xml");
18
19          RPCServiceClient serviceClient = new RPCServiceClient(ctx, null);
20
21          Options options = serviceClient.getOptions();
22
```

São adicionados os imports necessários

Cria o ConfigurationContext

O ConfigurationContext é passado como parâmetro ao instanciar o cliente

```
23        EndpointReference targetEPR = new EndpointReference(
24              "http://127.0.0.1:8080/axis2/services/RHService");
25        options.setTo(targetEPR);
26
27        // ////////////////////////////////////////////////////////////////////
28
29        /*
30         * Cria uma ContaBancaria e armazena no cadastro de RH.
31         */
32
33        // QName do metodo destino
34        QName opIncluirContaBancaria = new QName("http://service.rh.exemplo",
    "incluirContaBancaria");
35
36        /*
37         * Construindo nova conta
38         */
39        ContaBancaria conta1 = new ContaBancaria();
40
41        conta1.setContaCorrente(123456789);
42        conta1.setAgencia(123);
43        conta1.setNomeBanco("Banco do Brasil");
44
45
46        // Construção do array de argumentos para a chamada do metodo
47        Object[] opIncluirContaBancariaArgs = new Object[] { conta1 };
48
49        // Chamando o metodo
50        options.setAction("urn:incluirContaBancaria");
51        serviceClient.invokeRobust(opIncluirContaBancaria, opIncluirContaBancariaArgs);
52
53        ////////////////////////////////////////////////////////////////////
54
55    }
56 }
```

Código 4: Cliente Java com segurança configurada

No código 4 foram apresentadas as seguintes alterações (texto em negrito):

- Imports são adicionados no inicio do código (linhas 4 e 5). Estes são necessários para incluir as referências às classes ConfigurationContext e ConfigurationContextFactory;
- Criação de um ConfigurationContext (linha 17). Essa classe permite que a mensagem utilize um contexto de segurança, adicionando na mensagem SOAP os itens de segurança especificados no axis2.xml, localizado no subdiretório “conf”;

- A instância do ConfigurationContext é passada como parâmetro ao instanciar o cliente, nesse caso uma instância de RPCServiceClient (linha 19). Dessa forma, todas as mensagens enviadas pelo cliente possuirão o contexto de segurança que foi definido.

Alterações no arquivo axis2.xml

No arquivo axis2.xml serão realizadas as seguintes ações: a anexação do módulo de segurança Rampart (APACHE-RAMPART, 2008) através do arquivo rampart-1.4.mar que deverá constar na máquina do cliente e do servidor, do intermediário (classe PWCBHandler que fará a validação da mensagem autenticando-a pelo id e senha do usuário) e dos itens de segurança que deverão estar representados na mensagem como o *timestamp* e criptografia.

As alterações no arquivo axis2.xml indicarão quais os itens de segurança que serão incluídos nas mensagens SOAP enviadas pelas classes clientes. O arquivo axis2.xml será então copiado para o subdiretório “conf”, de modo que a classe cliente possa localizá-lo ao criar o contexto de segurança. A seguir é apresentado o arquivo axis2.xml após a ferramenta ter adicionado os itens de segurança.

```
  1 <?xml version="1.0" encoding="iso-8859-1"?>
  2 <axisconfig name="AxisJava2.0">
  3     <!-- ================================================= -->
  4     <!-- Parameters -->
  5     <!-- ================================================= -->
  6     <parameter name="hotdeployment">true</parameter>
  7     <parameter name="hotupdate">false</parameter>
  8     <parameter name="enableMTOM">false</parameter>
  9     <parameter name="enableSwA">false</parameter>
 10     <!--Uncomment if you want to enable file caching for attachments -->
 11     <!--parameter name="cacheAttachments">true</parameter>
 12     <parameter name="attachmentDIR"></parameter>
 13     <parameter name="sizeThreshold">4000</parameter-->
       ...
       ...
       ...
305     <!-- ================================================= -->
306     <!-- Phases  -->
```

```
307     <!-- ================================================= -->
308     <phaseOrder type="InFlow">
309         <!--  System predefined phases       -->
310         <phase name="Transport">
311             <handler
312                 class="org.apache.axis2.dispatchers.RequestURIBasedDispatcher"
    name="RequestURIBasedDispatcher">
313                 <order phase="Transport"/>
314             </handler>
315             <handler
316                 class="org.apache.axis2.dispatchers.SOAPActionBasedDispatcher"
    name="SOAPActionBasedDispatcher">
317                 <order phase="Transport"/>


318             </handler>
319         </phase>
320         <phase name="Addressing">
321             <handler
322                 class="org.apache.axis2.dispatchers.AddressingBasedDispatcher"
    name="AddressingBasedDispatcher">
323                 <order phase="Addressing"/>
324             </handler>
325         </phase>
326         <phase name="Security"/>
327         <phase name="PreDispatch"/>
328         <phase class="org.apache.axis2.engine.DispatchPhase" name="Dispatch">
329             <handler
330                 class="org.apache.axis2.dispatchers.RequestURIBasedDispatcher"
    name="RequestURIBasedDispatcher"/>
331             <handler
332                 class="org.apache.axis2.dispatchers.SOAPActionBasedDispatcher"
    name="SOAPActionBasedDispatcher"/>
333             <handler
334                 class="org.apache.axis2.dispatchers.RequestURIOperationDispatcher"
    name="RequestURIOperationDispatcher"/>
335             <handler
336                 class="org.apache.axis2.dispatchers.SOAPMessageBodyBasedDispatcher"
    name="SOAPMessageBodyBasedDispatcher"/>
337             <handler
338                 class="org.apache.axis2.dispatchers.HTTPLocationBasedDispatcher"
    name="HTTPLocationBasedDispatcher"/>
339         </phase>
340         <phase name="RMPhase"/>
341         <!--  System predefined phases       -->
342         <!--   After Postdispatch phase module author or service author can add any phase he
    want      -->
343         <phase name="OperationInPhase"/>
344         <phase name="soapmonitorPhase"/>
345     </phaseOrder>
346     <phaseOrder type="OutFlow">
347         <!--      user can add his own phases to this area  -->
348         <phase name="soapmonitorPhase"/>
349         <phase name="OperationOutPhase"/>
```

```
350         <!--system predefined phase-->
351         <!--these phase will run irrespective of the service-->
352         <phase name="RMPhase"/>
353         <phase name="PolicyDetermination"/>
354         <phase name="MessageOut"/>
355         <phase name="Security"/>
356     </phaseOrder>
357     <phaseOrder type="InFaultFlow">
358         <phase name="Addressing">
359             <handler
360                 class="org.apache.axis2.dispatchers.AddressingBasedDispatcher"
    name="AddressingBasedDispatcher">
361                 <order phase="Addressing"/>
362             </handler>
363         </phase>
364         <phase name="Security"/>
365         <phase name="PreDispatch"/>
366         <phase class="org.apache.axis2.engine.DispatchPhase" name="Dispatch">
367             <handler
368                 class="org.apache.axis2.dispatchers.RequestURIBasedDispatcher"
    name="RequestURIBasedDispatcher"/>
369             <handler
370                 class="org.apache.axis2.dispatchers.SOAPActionBasedDispatcher"
    name="SOAPActionBasedDispatcher"/>
371             <handler
372                 class="org.apache.axis2.dispatchers.RequestURIOperationDispatcher"
    name="RequestURIOperationDispatcher"/>
373             <handler
374                 class="org.apache.axis2.dispatchers.SOAPMessageBodyBasedDispatcher"
    name="SOAPMessageBodyBasedDispatcher"/>
375             <handler
376                 class="org.apache.axis2.dispatchers.HTTPLocationBasedDispatcher"
    name="HTTPLocationBasedDispatcher"/>
377         </phase>
378         <phase name="RMPhase"/>
379         <!--      user can add his own phases to this area  -->
380         <phase name="OperationInFaultPhase"/>
381         <phase name="soapmonitorPhase"/>
382     </phaseOrder>
383     <phaseOrder type="OutFaultFlow">
384         <!--      user can add his own phases to this area  -->
385         <phase name="soapmonitorPhase"/>
386         <phase name="OperationOutFaultPhase"/>
387         <phase name="RMPhase"/>
388         <phase name="PolicyDetermination"/>
389         <phase name="MessageOut"/>
390         <phase name="Security"/>
391     </phaseOrder>
392     <module ref="rampart"/>
393     <parameter name="OutflowSecurity">
394         <action>
395             <items>Timestamp Signature Encrypt</items>
396             <user>client</user>
```

```
397
    <passwordCallbackClass>exemplo.rh.rpcclient.PWCBHandler</passwordCallbackClass>
398         <signaturePropFile>client.properties</signaturePropFile>
399         <signatureKeyIdentifier>DirectReference</signatureKeyIdentifier>
400         <encryptionKeyIdentifier>DirectReference</encryptionKeyIdentifier>
401         <encryptionUser>service</encryptionUser>
402       </action>
403   </parameter>
404   <parameter name="InflowSecurity">
405       <action>
406           <items>Timestamp Signature Encrypt</items>
407
    <passwordCallbackClass>exemplo.rh.rpcclient.PWCBHandler</passwordCallbackClass>
408         <signaturePropFile>client.properties</signaturePropFile>
409       </action>
410   </parameter>
411 </axisconfig>
```

Código 5: Arquivo axis2.xml com segurança configurada

Como mostrado no código 5, serão adicionados ao final do arquivo os itens de segurança (linhas 392 a 410) para a mensagem SOAP. Primeiramente serão incluídos os itens da fase OutflowSecurity, e por último os itens de InflowSecurity, seguindo a ordem de execução Axis2, onde o transporte da mensagem está divido por fases (APACHE-AXIS2-CONFIG, 2007), primeiro o cliente envia uma requisição e após o processamento pelo servidor ele receberá uma resposta. Portanto os itens incluídos sob a *tag* OutflowSecurity indicam os itens de segurança que serão acrescentados pelo handler de envio de mensagem do Rampart, o Outflow security handler, e os itens localizados sob InflowSecurity indicam os itens de segurança necessários para a validação da mensagem pelo handler de recebimento de mensagem do Rampart, o Inflow security handler, antes de a mensagem ser recebida pelo cliente:

- Timestamp Signature Encrypt (linhas 395 e 406): indica a ordem da configuração de segurança a ser realizada na mensagem enviada (no caso do parameter Outflow), significa a ordem a ser seguida: primeiro será incluído um *timestamp* (data e hora da mensagem), uma assinatura (identificador único) e após esses passos será criptografado o corpo da mensagem;

- user (linha 396): indica o usuário que está enviando a mensagem. Nesse caso foi definido o usuário “client”. Esse usuário será validado pela classe PWCBHandler, responsável pela autenticação dos usuários;
- passwordCallbackClass (linhas 397 e 407): indica qual classe realizará a autenticação do usuário. A classe indicada, PWCBHandler, será usada para fornecer a senha necessária para autenticar a mensagem;
- signaturePropFile (linhas 398 e 408): indica o arquivo usado para montar os parâmetros da assinatura: a classe responsável pela criptografia da mensagem, o *keystore* (arquivo que armazena a chave privada de segurança) e a mesma senha que será validada na classe PWCBHandler. Nesse caso, o arquivo definido será o client.properties, que será inserido no lado cliente. A seguir é apresentada a representação desse arquivo:

```
1 org.apache.ws.security.crypto.provider=org.apache.ws.security.components.crypto.Merlin
2 org.apache.ws.security.crypto.merlin.keystore.type=jks
3 org.apache.ws.security.crypto.merlin.keystore.password=apache
4 org.apache.ws.security.crypto.merlin.file=client.jks
```

Código 6: Arquivo client.properties

- signatureKeyIdentifier (linha 399): identificador a ser usado para referenciar a chave na assinatura da mensagem. Nesse caso será utilizado DirectReference, o que indica que o Rampart deverá adicionar o certificado de chave pública do cliente na mensagem (cabeçalho de security) como um elemento BinarySecurityToken e adicionar uma referência para este elemento na assinatura;
- encryptionKeyIdentifier (linha 400): identificador a ser usado para referenciar a chave de criptografia. Nesse caso, é utilizado DirectReference, para identificar qual o certificado de criptografia utilizado;
- encryptionUser (linha 401): elemento que especifica o *alias* do certificado (chave) de criptografia. Nesse caso o certificado de chave publica estará disponível na keystore sob o alias “service”, ou seja, no lado do servidor.

Inclusão da classe PWCBHandler

Para que seja possível a validação das mensagens enviadas e recebidas pelo cliente, será incluída a classe PWCBHandler, presente no diretório “config” da ferramenta. Após a ferramenta copia-la para o diretório das classes do cliente, será incluído no início da classe o caminho da *package* do cliente, como é mostrado no código 7, linha 1.

A classe PWCBHandler analisa a mensagem enviada ou recebida verificando se possui o id correto e então define a senha que será validada, nesse caso a senha será “apache”. Esta senha definida estará criptografada na mensagem enviada para o servidor, então o servidor irá descriptografar a senha da mensagem usando a sua chave de segurança. Após realizar esses passos a classe PWCBHandler do lado servidor validará a mensagem.

A seguir é apresentado o código da classe, após a ferramenta incluir o caminho da package:

```
 1  package exemplo.rh.rpcclient;
 2  import org.apache.ws.security.WSPasswordCallback;
 3
 4  import javax.security.auth.callback.Callback;
 5  import javax.security.auth.callback.CallbackHandler;
 6  import javax.security.auth.callback.UnsupportedCallbackException;
 7
 8  import java.io.IOException;
 9
10  public class PWCBHandler implements CallbackHandler {
11
12      public void handle(Callback[] callbacks) throws IOException,
13              UnsupportedCallbackException {
14          for (int i = 0; i < callbacks.length; i++) {
15              WSPasswordCallback pwcb = (WSPasswordCallback)callbacks[i];
16              String id = pwcb.getIdentifer();
17              if("client".equals(id)) {
18                  pwcb.setPassword("apache");
19              } else if("service".equals(id)) {
20                  pwcb.setPassword("apache");
21              }
22          }
23      }
24
25  }
```

Código 7: Arquivo PWCBHandler.java

Inclusão da chave de segurança

Como foi apresentado anteriormente, no arquivo client.properties sob o subdiretório “config” da ferramenta estará indicado qual a keystore a ser usada, nesse caso o arquivo client.jks. Esses arquivos serão copiados pela ferramenta para o subdiretório “keys” Do diretório “saida\cliente”, permitindo que as mensagens enviadas pelo cliente possuam o corpo criptografado.

### 3.5.5 Ações executadas no lado Servidor

No lado servidor será realizada a configuração no arquivo services.xml, sendo acrescentado os mesmo itens de segurança que foram acrescentados no axis2.xml, também incluindo a classe PWCBHandler na arquitetura do servidor. Os arquivos .java do servidor não precisarão ser alterados, pois a configuração consistirá nas seguintes ações: indicação de inclusão do módulo Rampart e inclusão dos parâmetros de segurança no arquivo services.xml, inclusão dos arquivos relativos à chave de segurança, neste caso os arquivos service.properties e service.jks.

Como explicado anteriormente, as configurações de segurança para o lado servidor também serão realizadas em uma cópia da estrutura da aplicação do usuário, gerada sob o subdiretório “saida”.

O diagrama de atividades a seguir demonstra quais serão as ações a serem executadas no lado servidor.

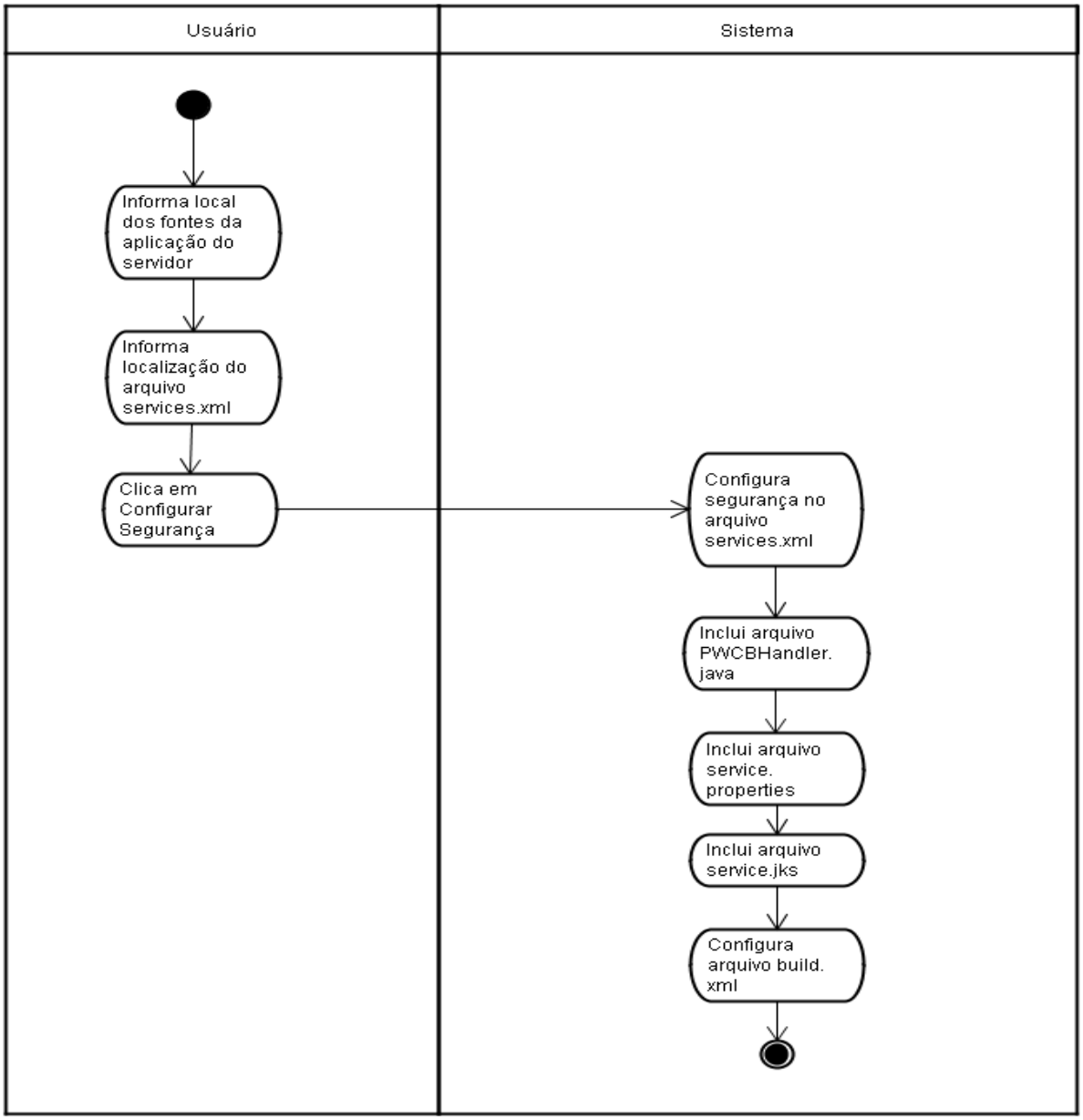


**Figura 3.8 – Diagrama de Atividades – Incluir segurança no lado Servidor**

Alterações no arquivo services.xml

No arquivo services.xml serão realizadas pela ferramenta alterações semelhantes às que foram explicadas anteriormente para o lado cliente. A diferença será na ordem de inclusão dos itens de segurança e nas referências de destino da mensagem, que no caso do servidor será o cliente.

O código 8 exemplifica essas alterações, e a seguir é explicado em detalhes as alterações realizadas.

```
 1 <?xml version="1.0" encoding="iso-8859-1"?>
 2 <service name="RHService" scope="application">
 3    <description>
 4       POJO: RH Service
 5    </description>
 6    <operation name="incluirFuncionario">
 7       <messageReceiver
 8          class="org.apache.axis2.rpc.receivers.RPCInOnlyMessageReceiver"
   mep="http://www.w3.org/2004/08/wsdl/in-only"/>
 9       <actionMapping>urn:incluirFuncionario</actionMapping>
10    </operation>
11    <operation name="pesquisarFuncionario">
12       <messageReceiver
13          class="org.apache.axis2.rpc.receivers.RPCMessageReceiver"
   mep="http://www.w3.org/2004/08/wsdl/in-out"/>
14       <actionMapping>urn:pesquisarFuncionario</actionMapping>
15    </operation>
16    <operation name="incluirEmpresa">
17       <messageReceiver
18          class="org.apache.axis2.rpc.receivers.RPCInOnlyMessageReceiver"
   mep="http://www.w3.org/2004/08/wsdl/in-only"/>
19       <actionMapping>urn:incluirEmpresa</actionMapping>
20    </operation>
21    <operation name="incluirContaBancaria">
22       <messageReceiver
23          class="org.apache.axis2.rpc.receivers.RPCInOnlyMessageReceiver"
   mep="http://www.w3.org/2004/08/wsdl/in-only"/>
24       <actionMapping>urn:incluirContaBancaria</actionMapping>
25    </operation>
26    <parameter name="ServiceClass">exemplo.rh.service.RHService</parameter>
27    <module ref="rampart"/>
28    <parameter name="InflowSecurity">
29       <action>
30          <items>Timestamp Signature Encrypt</items>
31
   <passwordCallbackClass>exemplo.rh.service.PWCBHandler</passwordCallbackClass>
32          <signaturePropFile>service.properties</signaturePropFile>
33       </action>
34    </parameter>
35    <parameter name="OutflowSecurity">
36       <action>
37          <items>Timestamp Signature Encrypt</items>
38          <user>service</user>
39
   <passwordCallbackClass>exemplo.rh.service.PWCBHandler</passwordCallbackClass>
40          <signaturePropFile>service.properties</signaturePropFile>
41          <signatureKeyIdentifier>DirectReference</signatureKeyIdentifier>
42          <encryptionKeyIdentifier>DirectReference</encryptionKeyIdentifier>
43          <encryptionUser>useReqSigCert</encryptionUser>
```

```
44          </action>
45      </parameter>
46  </service>
```

Código 8: Arquivo services.xml

Como é demonstrado no código 8, a ordem de inclusão dos itens será primeiramente aqueles que pertencem ao parâmetro de InflowSecurity (linhas 28 a 34) e, por último, do OutflowSecurity (linhas 35 a 45). Os itens de segurança adicionados são os mesmos explicados na seção anterior para o lado cliente.

Inclusão da classe PWCBHandler

Para a validação das mensagens enviadas e recebidas pelo servidor, a ferramenta adicionará no diretório das classes do servidor a classe PWCBHandler, da mesma forma como explicado na seção 3.5.5.

Inclusão da chave de segurança

No lado servidor será incluído o arquivo service.properties, e o arquivo da chave de segurança do servidor, nesse caso o arquivo service.jks. Estes arquivos serão copiados para o subdiretório “keys”, localizado no diretório “saida\servidor”. A seguir a representação do arquivo service.properties.

```
1  org.apache.ws.security.crypto.provider=org.apache.ws.security.components.crypto.Merlin
2  org.apache.ws.security.crypto.merlin.keystore.type=jks
3  org.apache.ws.security.crypto.merlin.keystore.password=apache
4  org.apache.ws.security.crypto.merlin.file=service.jks
```

Código 9: Arquivo service.properties

### 3.5.6 Ações executadas no arquivo build.xml

Devido à necessidade de inclusão de chaves de segurança às aplicações cliente e servidor, será também de responsabilidade da ferramenta a configuração do arquivo build.xml, para que seja possível compilar e executar tais aplicações com estas fazendo uso das chaves de segurança.

Para realizar essas alterações, a ferramenta faz um *parsing* do arquivo build.xml utilizando-se dos métodos especificados pela API JDOM (JDOM, 2007). Após o parsing é gerado um novo build.xml com as alterações explicadas nas seções a seguir.

Inclusão das tags <property>

As tags <property> definem os nome dos diretórios e os módulos a serem utilizados pelo build.xml. Os passos a seguir serão executados para as tags <property> necessárias no build.xml.

- Inclusão da tag <property name=“temp”.dir value="build/temp" /> para indicar a criação do diretório “temp” a ser usado dentro do diretório “build”.
- Inclusão da tag <property name="keys.dir" value="keys" /> indicando que o diretório de localização das chaves de segurança será “keys”.
- Inclusão da tag <property name="conf.dir" value="conf" /> indicando o diretório “conf”, onde estará o arquivo axis2.xml configurado.
- Inclusão da tag <property name="build.conf.dir" value="build/${conf.dir}" /> indicando a criação do diretório “conf” dentro do diretório “build”.
- Inclusão da tag <property name="build.modules.dir" value="build/modules" /> indicando a criação do diretório “modules” dentro do diretório “build”. Nesse diretório estarão localizados os módulos descritos nos próximos itens.

- Inclusão da tag <property name="rampart.mar" value="rampart-1.4.mar" /> indicando que será incluído o módulo de segurança rampart-1.4.mar .
- Inclusão da tag <property name="addressing.mar" value="addressing-1.4.mar" /> indicando que será incluído o módulo addressing-1.4.mar que permite endereçar as mensagens trocadas por Web Services.

Alterações nos subelementos das tags <target> de execução

Para a execução das classes do cliente com o uso de chave de segurança, a ferramenta realiza algumas alterações nos subelementos da tag <target> de execução. Essa alteração segue os seguintes passos executados:

1. Percorre o arquivo build.xml até encontrar uma tag <target>, responsável pela execução das classes no lado cliente. Por exemplo, a tag <target name="rpc.client.run.funcionario">. A condição de busca verifica se o atributo *name* contém a palavra “client”.
2. Pesquisa pelo subelemento <java>. Se for encontrado, é pesquisado o subelemento <classpath>.
3. Caso o subelemento <classpath> seja encontrado a ferramenta inclui os subelementos necessários para que seja possível o uso de chaves de segurança durante a execução da classe cliente (linhas 2 a 9). O elemento <dirset dir="${temp.dir}" /> é incluído como subelemento de <classpath> (linha 18). Dessa forma ao executar a classe cliente, é informada a localização das chaves de segurança para a máquina virtual Java.

No trecho de código apresentado a seguir são mostradas as alterações realizadas no passo 3.

```
1   <target name="rpc.client.run.funcionario">
2     <mkdir dir="${temp.dir}" />
3     <copy file="${keys.dir}/client.jks" tofile="${temp.dir}/client.jks" overwrite="true" />
4     <copy file="${keys.dir}/client.properties" tofile="${temp.dir}/client.properties" overwrite="true" />
5     <copy file="${conf.dir}/axis2.xml" tofile="${build.conf.dir}/axis2.xml" overwrite="true" />
6     <property name="modules.dir" value="${env.AXIS2_HOME}/repository/modules/" />
7     <mkdir dir="${build.modules.dir}" />
8     <copy file="${modules.dir}/${addressing.mar}" tofile="${build.modules.dir}/${addressing.mar}"
    overwrite="true" />
9     <copy file="${modules.dir}/${rampart.mar}" tofile="${build.modules.dir}/${rampart.mar}"
    overwrite="true" />
10    <java classname="exemplo.rh.rpcclient.FuncionarioClient">
11      <classpath>


12        <fileset dir="${env.AXIS2_HOME}/lib">
13          <include name="*.jar" />
14        </fileset>
15        <fileset dir="${dest.dir.lib}">
16          <include name="*.jar" />
17        </fileset>
18        <dirset dir="${temp.dir}" />
19      </classpath>
20      <arg value="${services.url}" />
21      <arg value="build" />
22    </java>
23  </target>
```

Código 10: Configuração do <target> de execução do build.xml para uso de segurança no lado Cliente

Alterações nos subelementos da tag <target> de compilação

Visando incluir as chaves de segurança também no lado servidor, a ferramenta também executa alterações na compilação das classes a serem publicadas no servidor. Nesse caso, os seguintes passos são executados:

1. Percorre o arquivo build.xml até encontrar uma tag <target> usada para o lado servidor. A condição de busca pela ferramenta verifica se o atributo *name* contém a palavra “service”.
2. Pesquisa pelo subelemento <javac>. Se for encontrado, é pesquisado o subelemento <classpath>.
3. Caso o subelemento <classpath> seja encontrado, a ferramenta inclui os subelementos <copy> (linhas 2 e 3) para que os arquivos relativos à chave de segurança, arquivos service.properties e service.jks, sejam incluídos no pacote de distribuição a ser publicado no servidor.

O trecho de código a seguir apresenta essas alterações no arquivo build.xml

```
1  <target name="generate.service" depends="clean,prepare">
2      <copy file="${keys.dir}/service.jks" tofile="${dest.dir.classes}/service.jks"
   overwrite="true" />
3      <copy file="${keys.dir}/service.properties" tofile="${dest.dir.classes}/service.properties"
   overwrite="true" />
4      <copy file="src/META-INF/services.xml" tofile="${dest.dir.classes}/META-
   INF/services.xml" overwrite="true" />
5      <javac srcdir="src" destdir="${dest.dir.classes}"
   includes="exemplo/rh/service/**,exemplo/rh/persistencia/**">
6          <classpath>
7              <fileset dir="${env.AXIS2_HOME}/lib">
8                  <include name="*.jar" />
9              </fileset>
10         </classpath>
11     </javac>
12     <jar basedir="${dest.dir.classes}" destfile="${dest.dir}/RHService.aar" />
13     <copy file="${dest.dir}/RHService.aar" tofile="${repository.path}/RHService.aar"
   overwrite="true" />
14 </target>
```

Código 10: Configuração do <target> de compilação do build.xml para uso de segurança no lado Servidor

### 3.5.7 Diagrama de Classes

Com o objetivo de representar as relações entre todas as classes implementadas na ferramenta ConfigSecurityWS, identificando atributos e métodos, elaborou-se um diagrama de classes. O software JUDE Community 5.2.1 foi utilizado para a modelagem do diagrama. A seguir é apresentado o diagrama de classes e as responsabilidades de cada uma das classes.

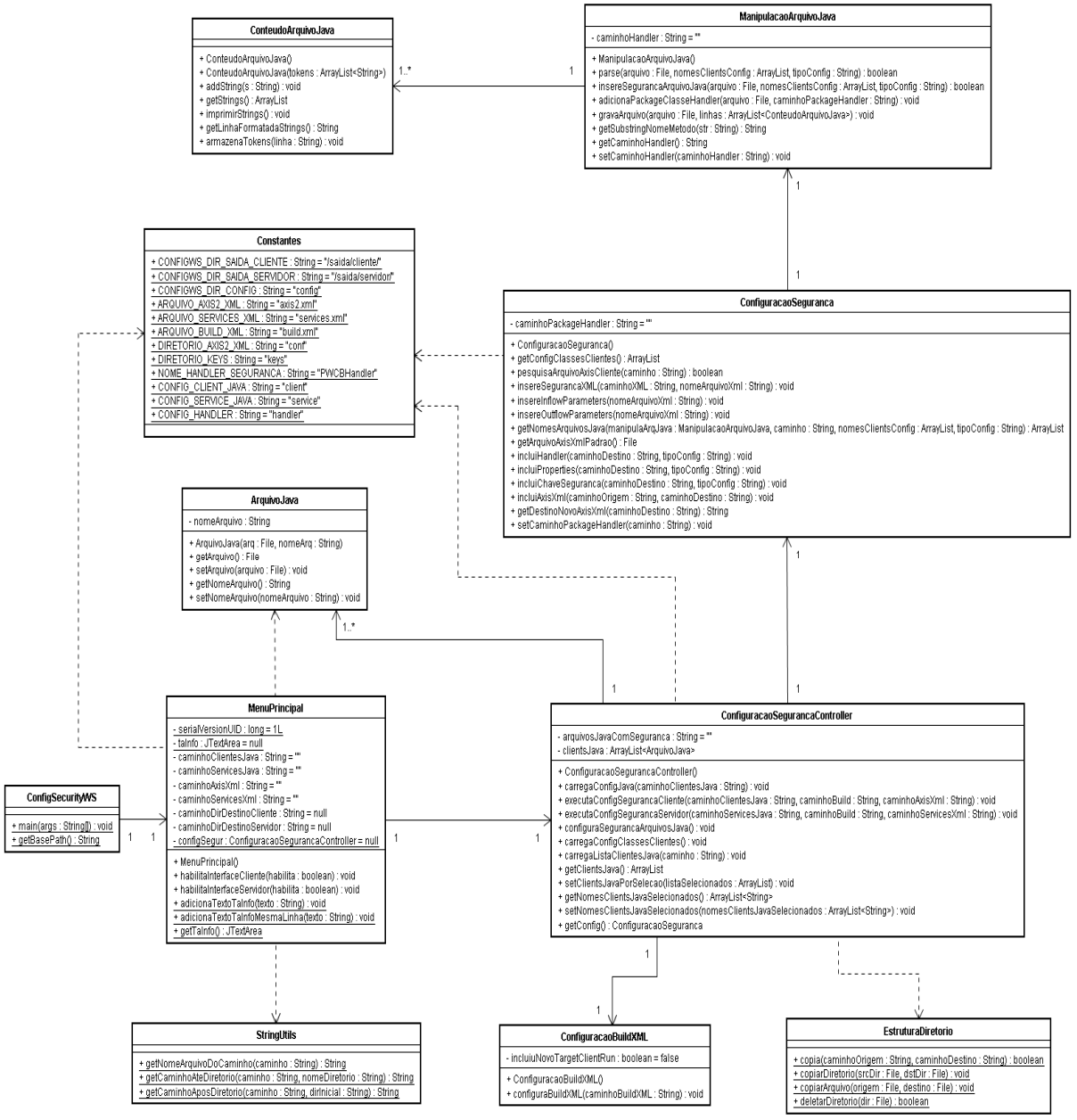


**Figura 3.9 – Diagrama de Classes**

As classes representadas no diagramas são explicadas em maiores detalhes a seguir:

- ConfigSecurityWS: classe principal da ferramenta. Possui o método main() que permite executar o ConfigSecurity-WS.
- MenuPrincipal: classe que implementa a interface gráfica. É a janela exibida durante a execução da ferramenta.
- Constantes: classe no qual estão representados os valores estáticos a serem utilizados pela ferramenta. Esses valores estáticos são mantidos em uma única classe evitando a necessidade de declarar esses valores em cada uma das classes facilitando a manutenção do código.
- ConfiguracaoSegurancaController: classe controladora utilizada para responder aos eventos disparados pela classe MenuPrincipal. Esses eventos por sua vez chamam os métodos implementados na classe ConfiguracaoSeguranca.
- ConfiguracaoSeguranca: é a classe responsável por executar a configuração de segurança. Realiza a inclusão dos itens de segurança nos arquivos .xml e chama o método configuraSegurancaArquivoJava da classe ManipulacaoArquivoJava para a configuração de segurança nos arquivos Java do lado cliente. Os métodos principais são esses:
    - insereSegurancaXML: faz um *parsing* do arquivo axis2.xml ou services.xml e inclui os parâmetros de segurança ao final do arquivo.
    - incluiHandler: inclui no diretório do cliente e do servidor a classe PWCBHandler, intermediário responsável pela autenticação das mensagens.
    - incluiProperties: inclui no diretório do cliente e do servidor os arquivos .properties, necessários para indicar a senha das chaves de criptografia.
    - incluiChaveSeguranca: inclui as chaves de criptografia no diretório do cliente e do servidor.

    - incluiAxisXML: inclui o arquivo axis2.xml padrão que acompanha a ferramenta. Esse método é chamado caso o cliente não possua este arquivo.
- ConfiguracaoBuildXML: classe responsável por configurar o arquivo build.xml, permitindo a execução com o uso das chaves de segurança.
- ManipulacaoArquivoJava: classe responsável pela configuração de segurança nos arquivos .java do cliente. Realiza um “parsing” do arquivo .java, incluindo os imports necessários e a instância da classe ConfigurationContext, que habilita o uso do contexto de segurança.
- ArquivoJava: classe que representa um arquivo Java. Possui os seguintes atributos:
    - nomeArquivo: nome do arquivo .java;
    - arquivo: atributo do tipo File. Determina o caminho completo do arquivo.
- ConteudoArquivoJava: classe que possui todo o conteúdo de um arquivo Java. Armazena o conteúdo cada linha lida. Possui o seguinte atributo:
    - strings: lista contendo todos os *tokens* existentes em uma linha.
- EstruturaDiretorio: classe que implementa métodos úteis para alterações em estruturas de diretórios. Possui os seguintes métodos:
    - copia: copia um arquivo/diretório de um determinado caminho para outro.
    - copiarDiretorio: copia um subdiretório de um diretório para outro diretório.
    - copiarArquivo: copia um arquivo de um diretório para outro.
    - deletarDiretorio: deleta um diretório.
- StringUtils: classe que possui métodos para manipulação dos caminhos de localização dos arquivos.

# 4 DESCRIÇÃO DA INTERFACE DO CONFIGSECURITY-WS

Nesse capítulo será apresentada a interface de utilização do ConfigSecurity-WS, mostrando a seqüência de passos necessária para a inclusão de mecanismos de segurança em uma arquitetura de Web Services.

## 4.1 CONFIGURAÇÕES NECESSÁRIAS

Para que seja possível a execução da ferramenta serão necessários que os seguintes itens estejam instalados e/ou configurados na maquina cliente e no servidor:

- Axis2: API Java para implementação dos Web Services;
- Rampart 1.4: módulo do Axis2 que permite o uso do padrão WS-Security nas mensagens SOAP.

## 4.2 DEMONSTRAÇÃO DA INTERFACE

O exemplo utilizado para a demonstração da interface é apresentado no capítulo 5, onde está detalhada a responsabilidade de cada uma das classes.

O usuário poderá executar a ferramenta através do arquivo ConfigSecurityWS.bat. Ao executar a ferramenta, a janela inicial será a apresentada a seguir.

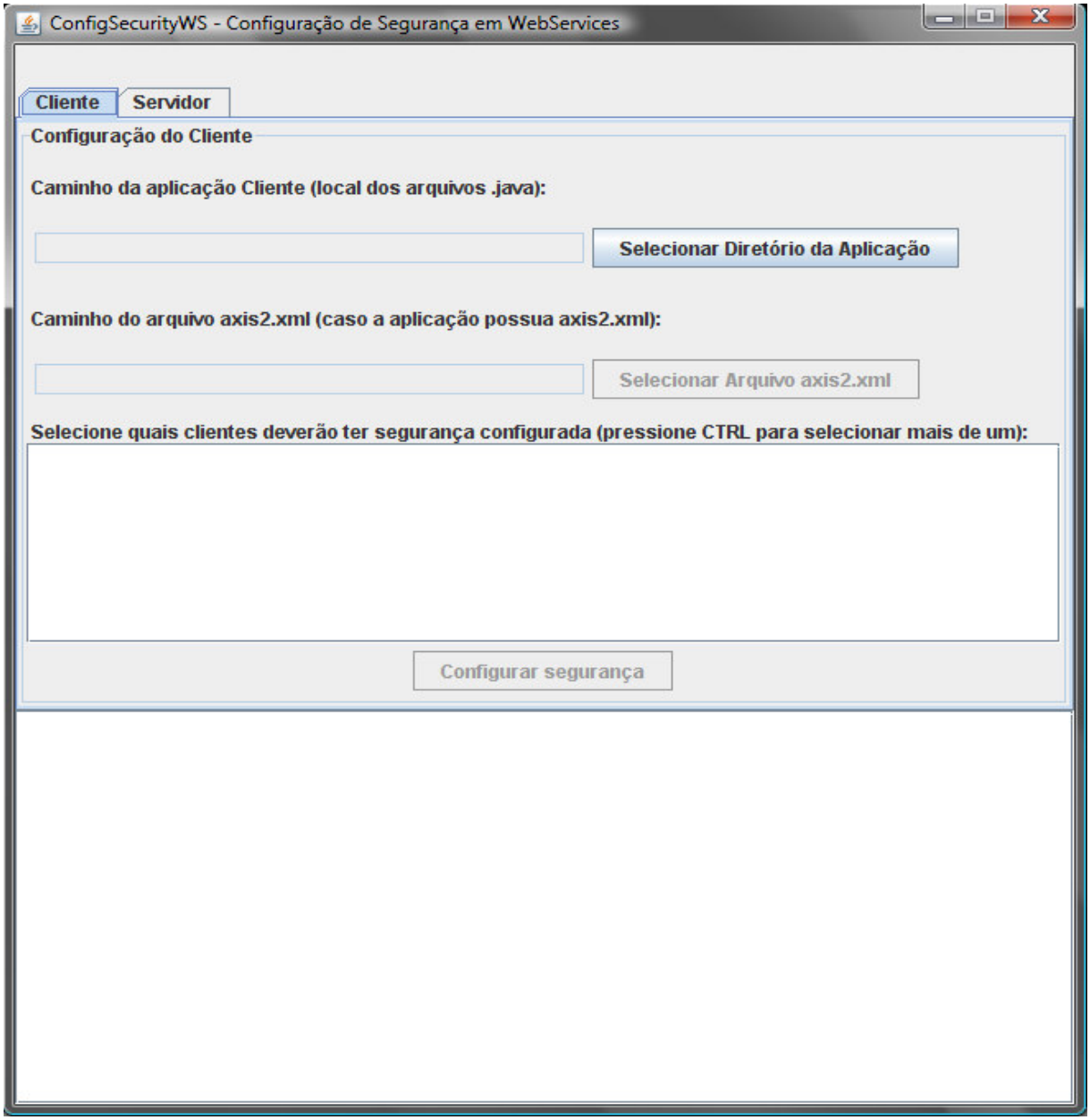


**Figura 4.1 – Janela exibindo interface de configuração do lado Cliente**

Como pode se observar na figura 4.1, ao executar a ferramenta, a aba selecionada por padrão será “Cliente”, que apresenta a interface de configuração para o lado cliente. Opcionalmente, o usuário poderá selecionar a aba “Servidor“ caso deseje iniciar a configuração no lado servidor.

## 4.3 INTERFACE DE CONFIGURAÇÃO DO LADO CLIENTE

A partir do botão “Selecionar Diretório da Aplicação” o usuário poderá selecionar o diretório no qual se encontram as classes Java em que ele deseja incluir a configuração de segurança. Ao clicar no botão a seguinte janela será exibida.

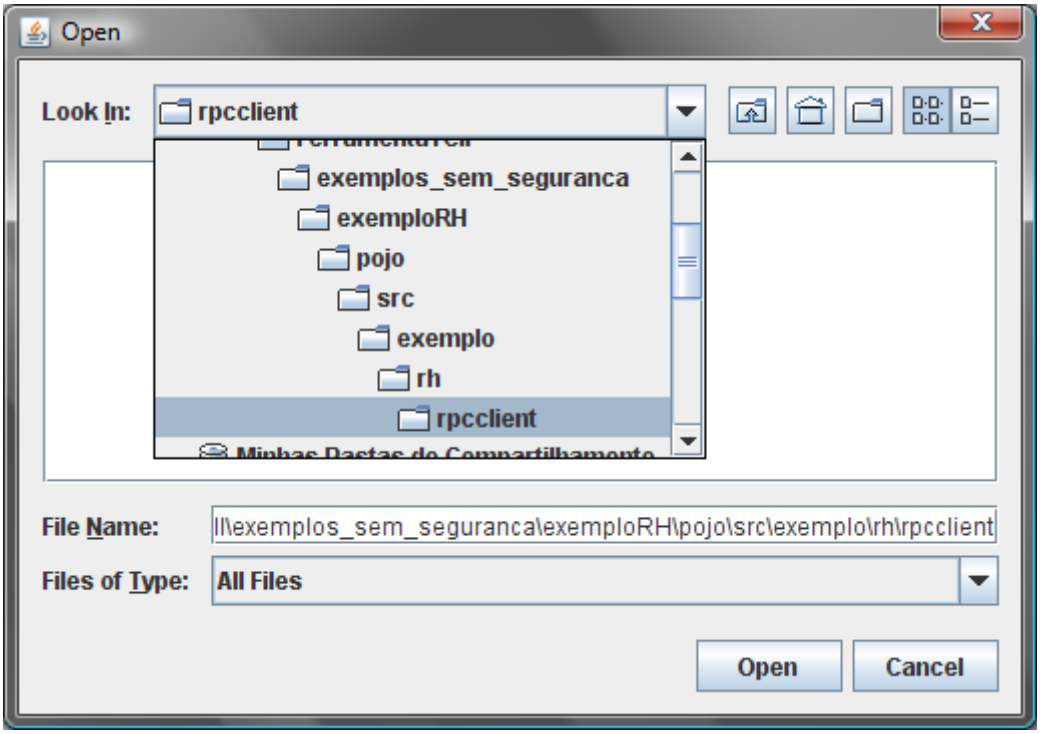


**Figura 4.2 – Janela de seleção do diretório das classes no lado Cliente**

Na figura 4.2 é apresentado, como exemplo, a seleção do diretório “rpcclient”, onde se encontra as classes clientes Java. Para confirmar a seleção desse diretório o usuário deverá clicar em “Open”.

Após a seleção do diretório, o caminho do diretório é exibido na janela principal, são exibidos os nomes das classes clientes Java encontradas nesse diretório, como representado na figura 4.3.

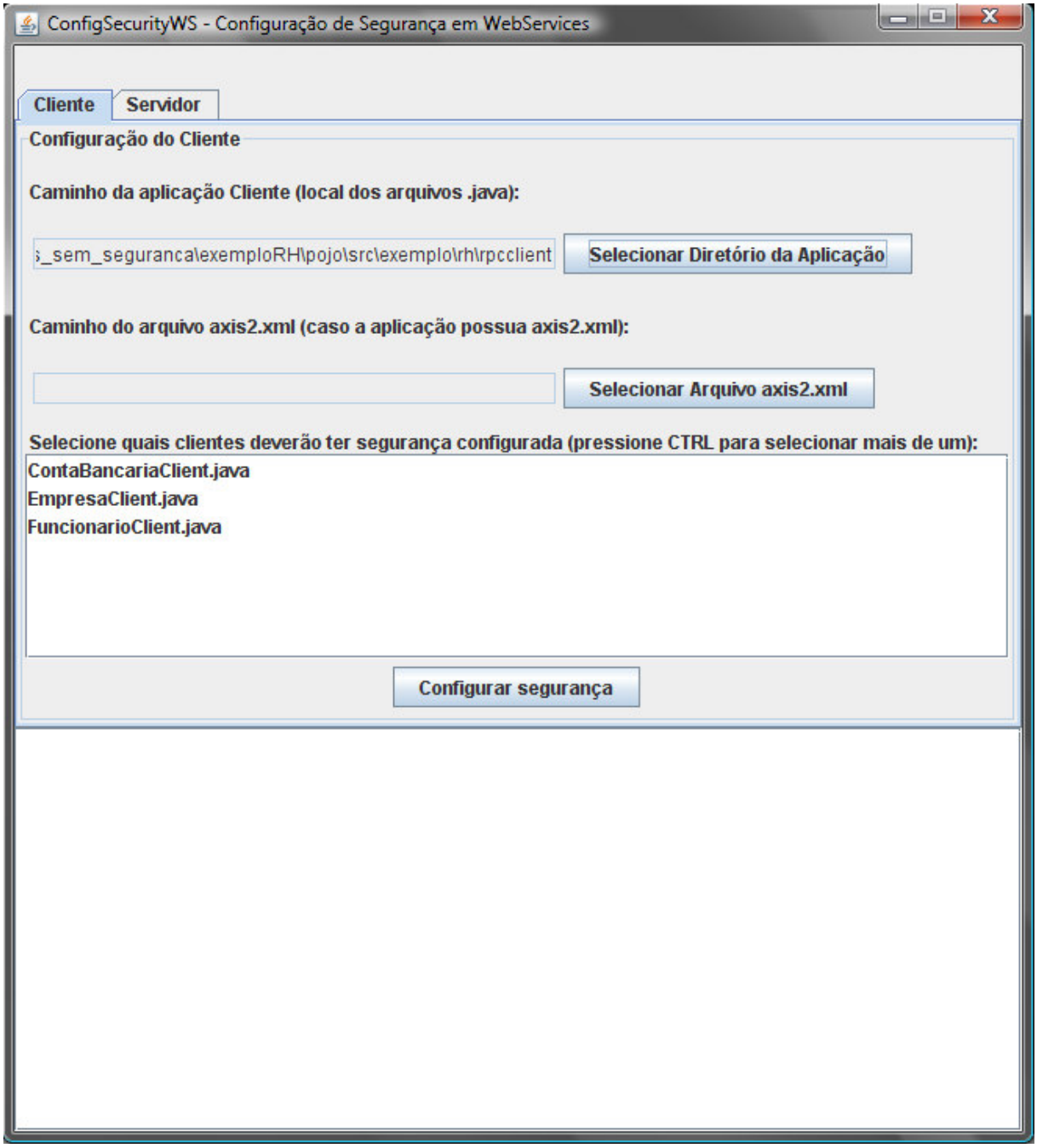


**Figura 4.3 – Janela principal exibindo classes encontradas no lado Cliente**

A partir desse momento o usuário poderá selecionar o arquivo axis2.xml, caso a aplicação possua este arquivo. Portanto a seleção do arquivo axis2.xml é opcional, e é realizada de forma semelhante ao apresentado anteriormente para o diretório da aplicação, onde o usuário poderá selecionar na lista de arquivos do diretório o arquivo axis2.xml. Ao clicar no botão “Selecionar Arquivo axis2.xml” a janela de seleção é apresentada, como será mostrado na figura 4.4.

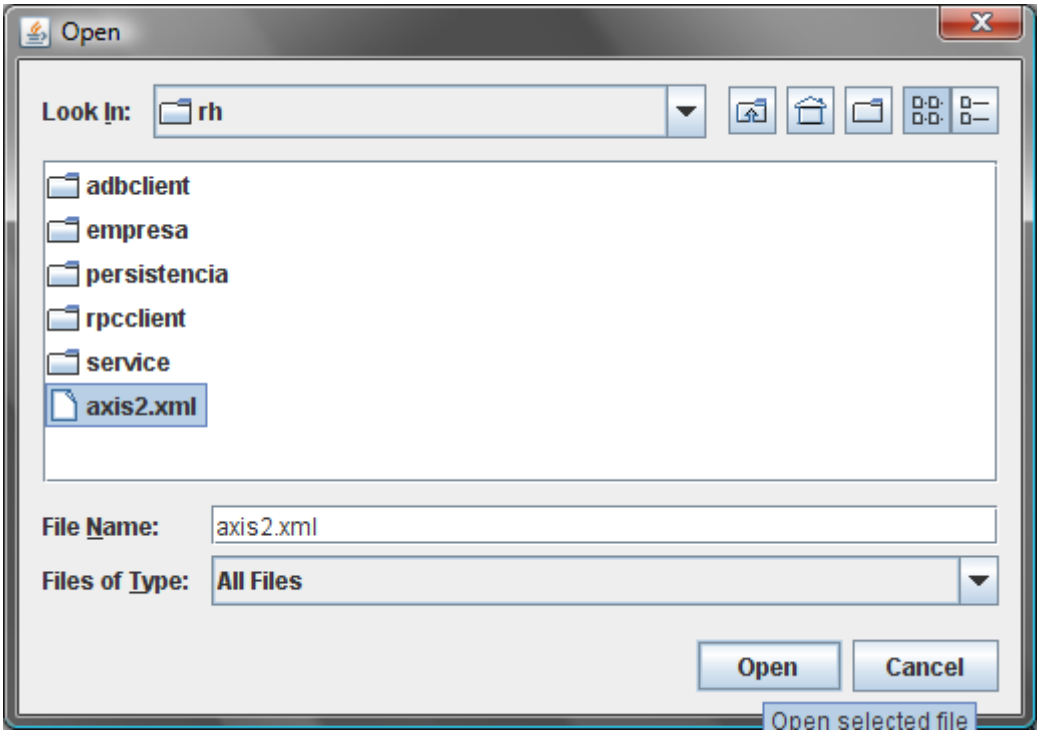


**Figura 4.4 – Janela de seleção do axis2.xml**

Após selecionar o arquivo axis2.xml, o caminho completo é exibido na interface de configuração do cliente. Neste momento o usuário poderá selecionar quais classes ele deseja incluir segurança, então ao clicar no nome da classe listada ele poderá selecioná-la. Utilizando a tecla CTRL ele poderá selecionar mais de uma classe. A figura a seguir demonstra a seleção de mais de uma classe.

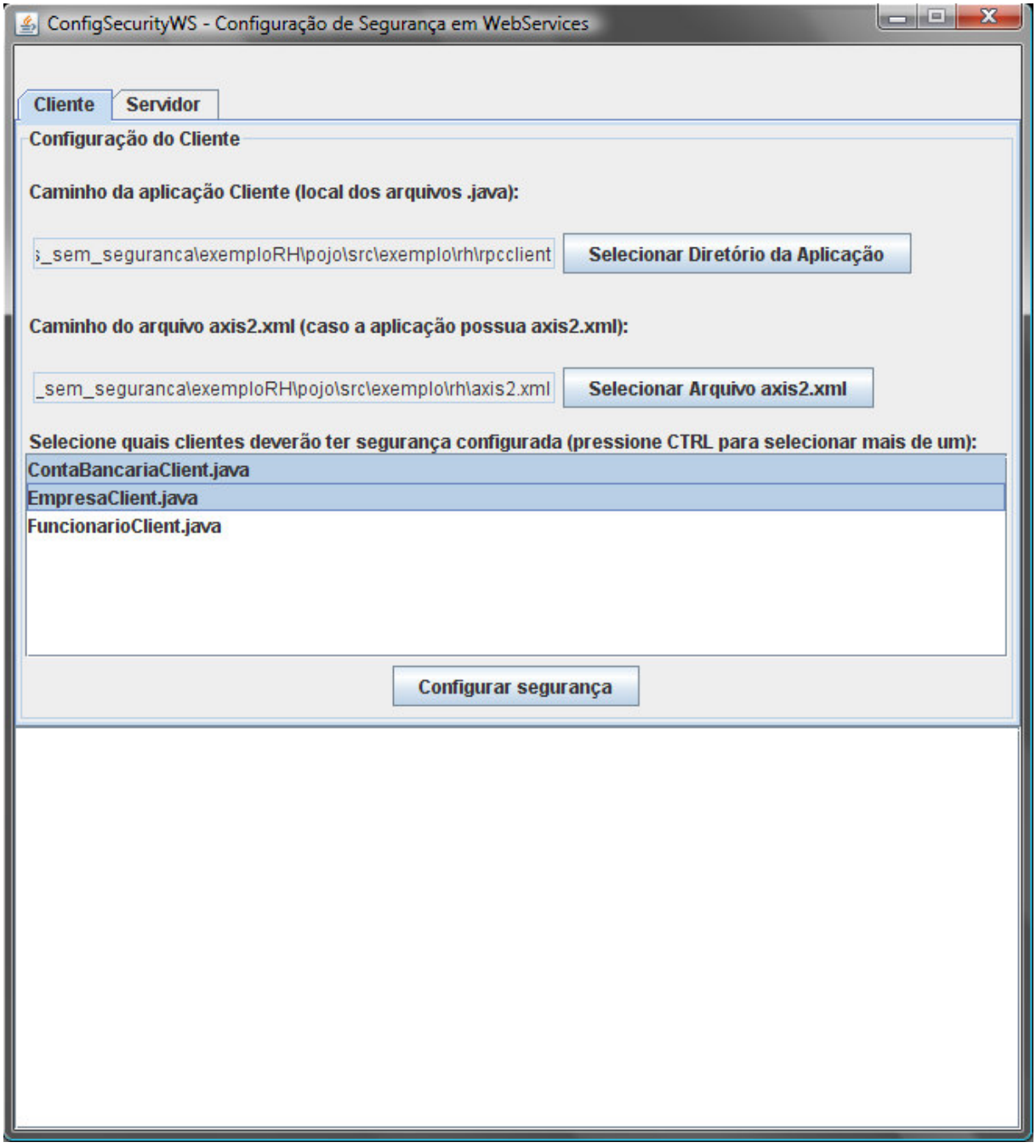


**Figura 4.5 – Seleção das classes a serem configuradas no lado Cliente**

Portanto, como demonstrado na figura 4.5, as classes que serão configuradas são ContaBancariaCliente.java e EmpresaClient.java. Após ter selecionado as classes o usuário poderá clicar em “Configurar segurança”. A seguir é demonstrado o resultado da configuração no lado cliente.

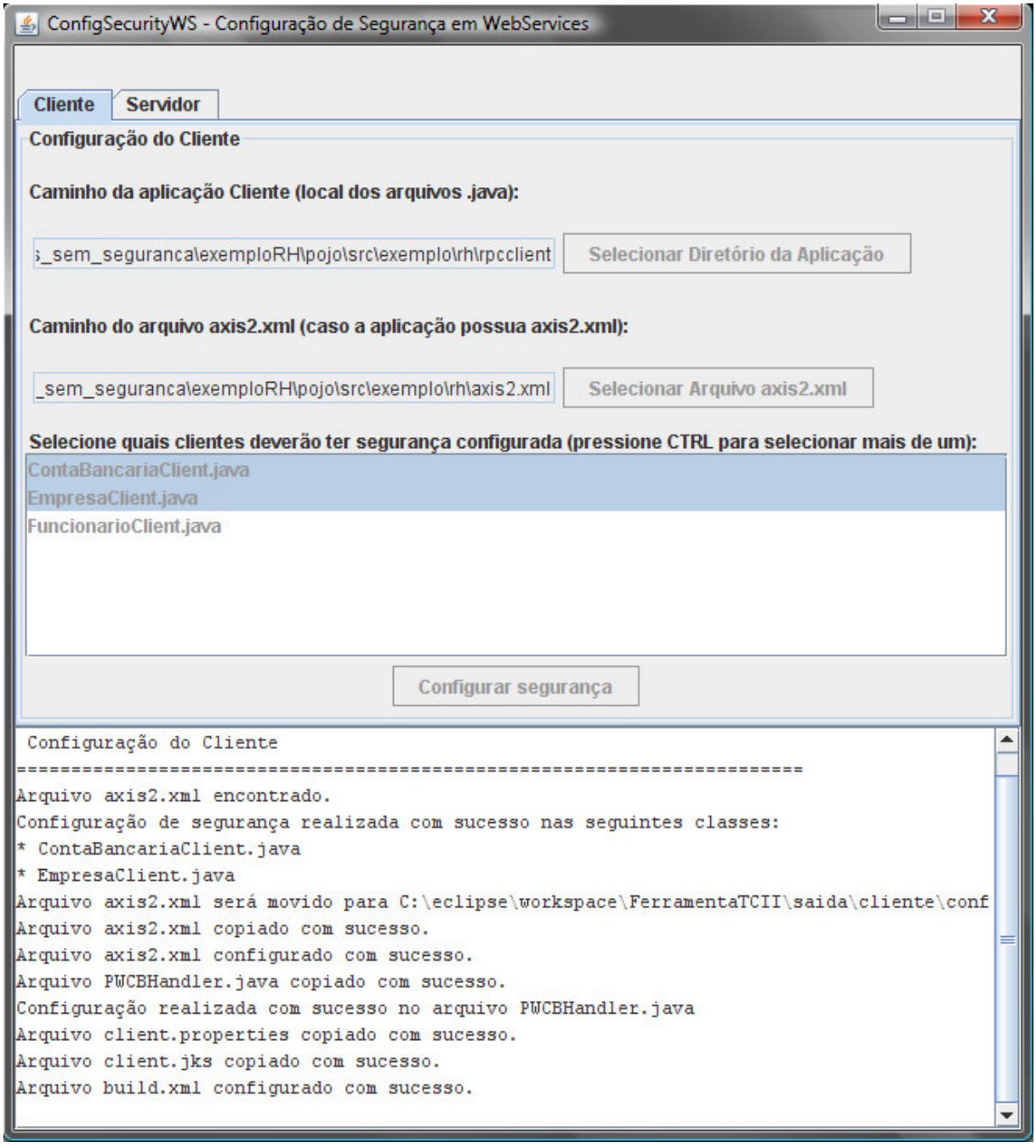


**Figura 4.6 – Resultado da configuração de segurança no lado Cliente**

## 4.4 INTERFACE DE CONFIGURAÇÃO DO LADO SERVIDOR

Nessa seção será demonstrada a continuação da configuração de segurança, com os passos a serem executados no lado servidor.

Para realizar a configuração o usuário deverá clicar na aba “Servidor”. Então será exibida a interface para configurar o lado servidor, como está representado na figura a seguir.

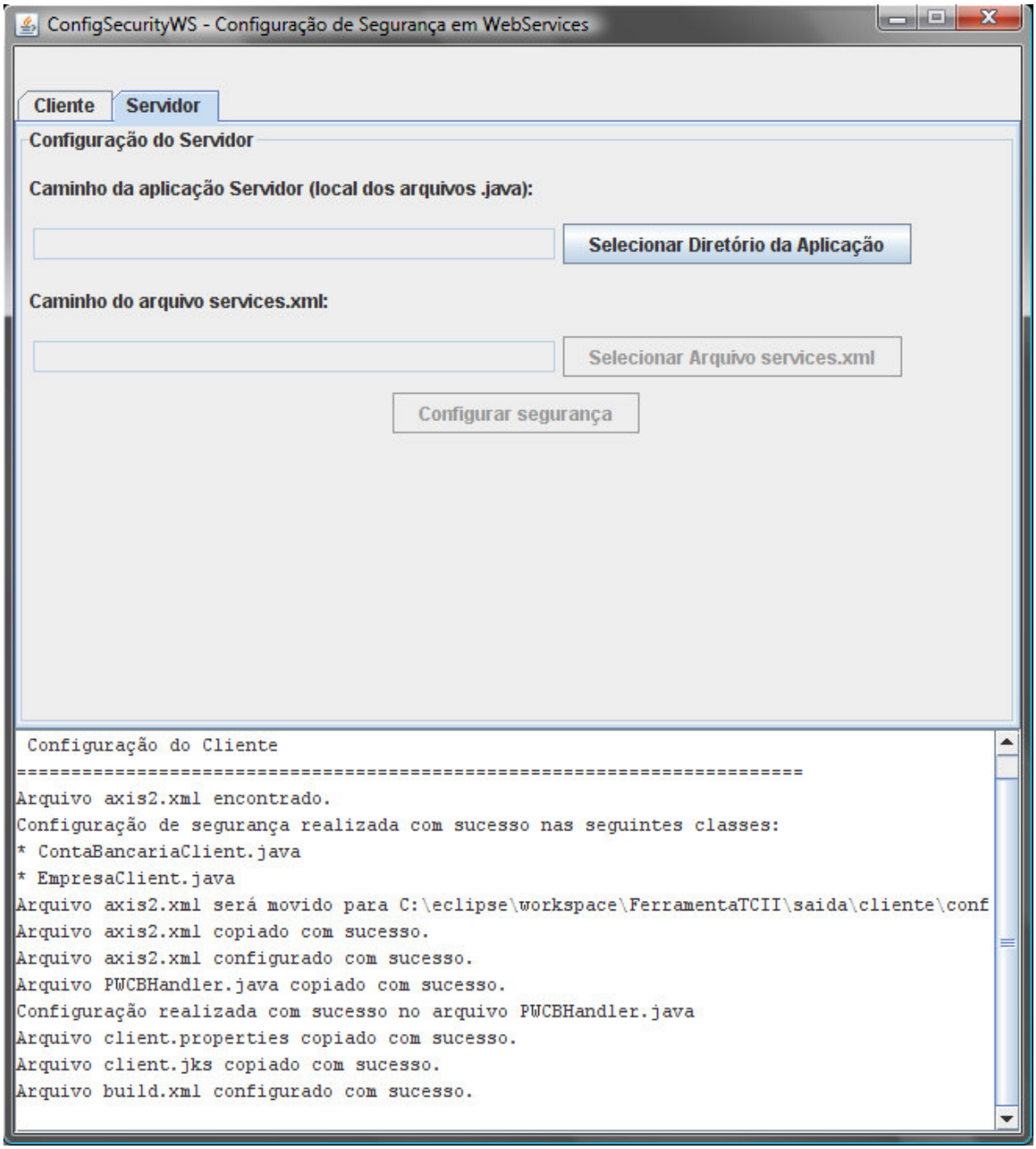


**Figura 4.7 – Janela exibindo interface de configuração do lado Servidor**

Neste momento o usuário pode selecionar o local do(s) Web Services, ou seja, onde se encontram as classes Java, clicando no botão “Selecionar Diretório da Aplicação”, então é apresentada a janela para a seleção do diretório como está representado na figura 4.8.

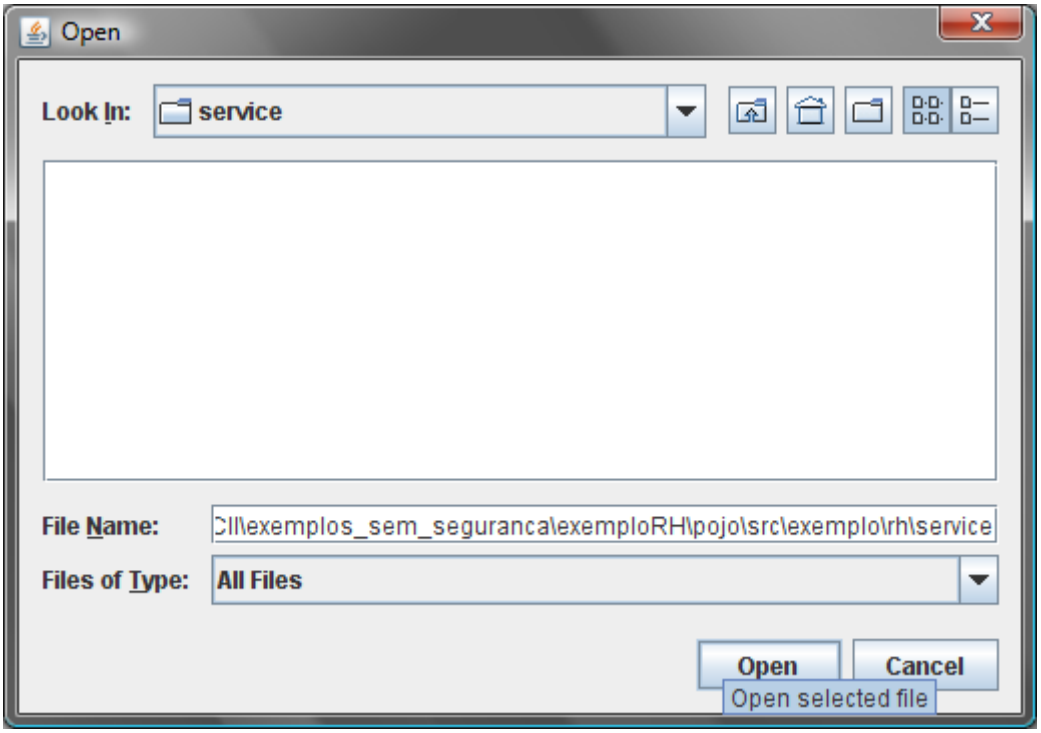


**Figura 4.8 – Janela de seleção do diretório das classes no lado Servidor**

Então é exibido o caminho completo da localização das classes do servidor na interface, como será demonstrado na figura 4.9.

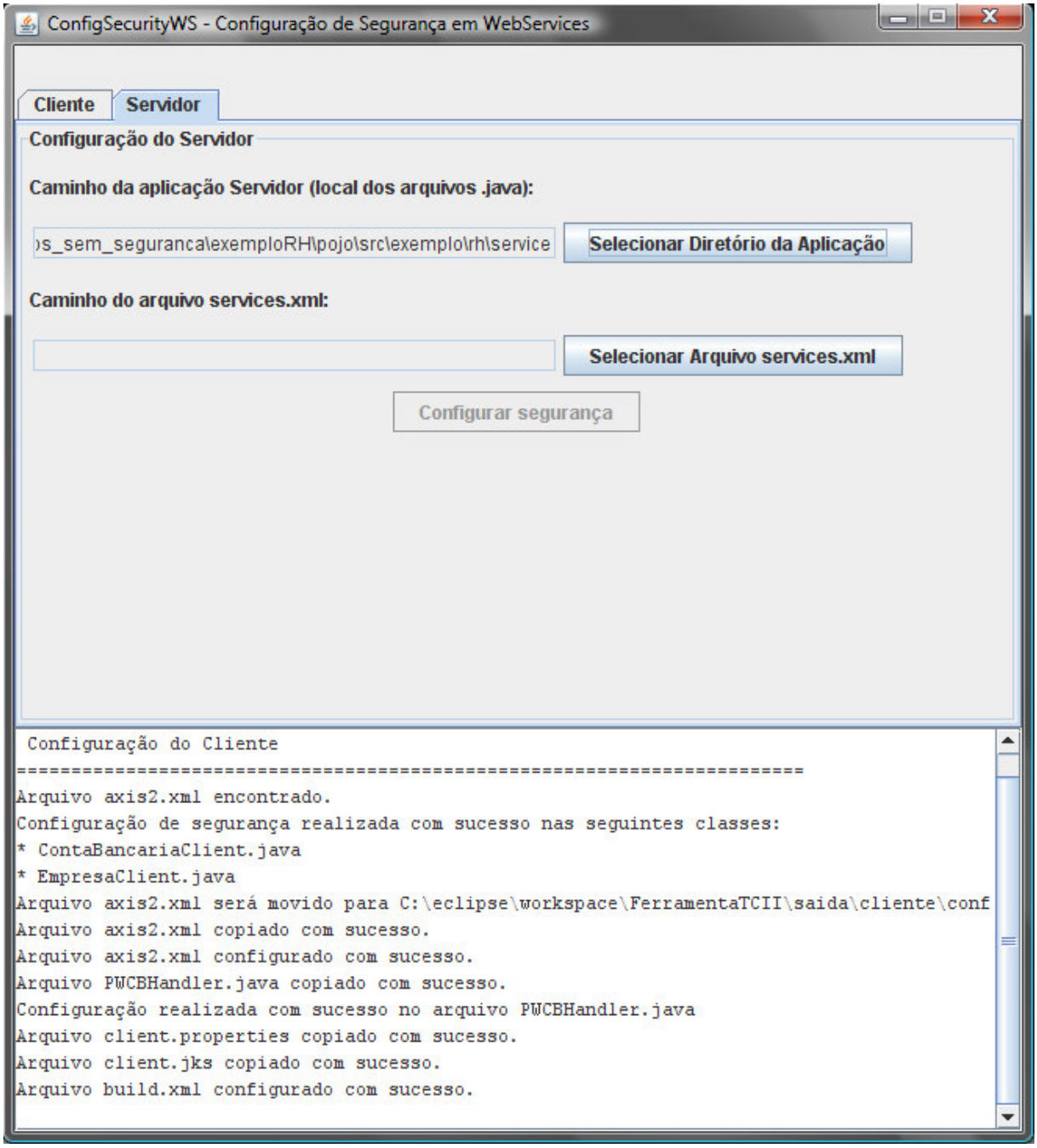


**Figura 4.9 – Janela após seleção do diretório das classes no lado Servidor**

Após ter selecionado o diretório da aplicação no servidor, o usuário deverá selecionar o arquivo services.xml. Ao clicar no botão “Selecionar Arquivo services.xml” é apresentada uma janela para a seleção desse arquivo, como é demonstrado na figura a seguir.

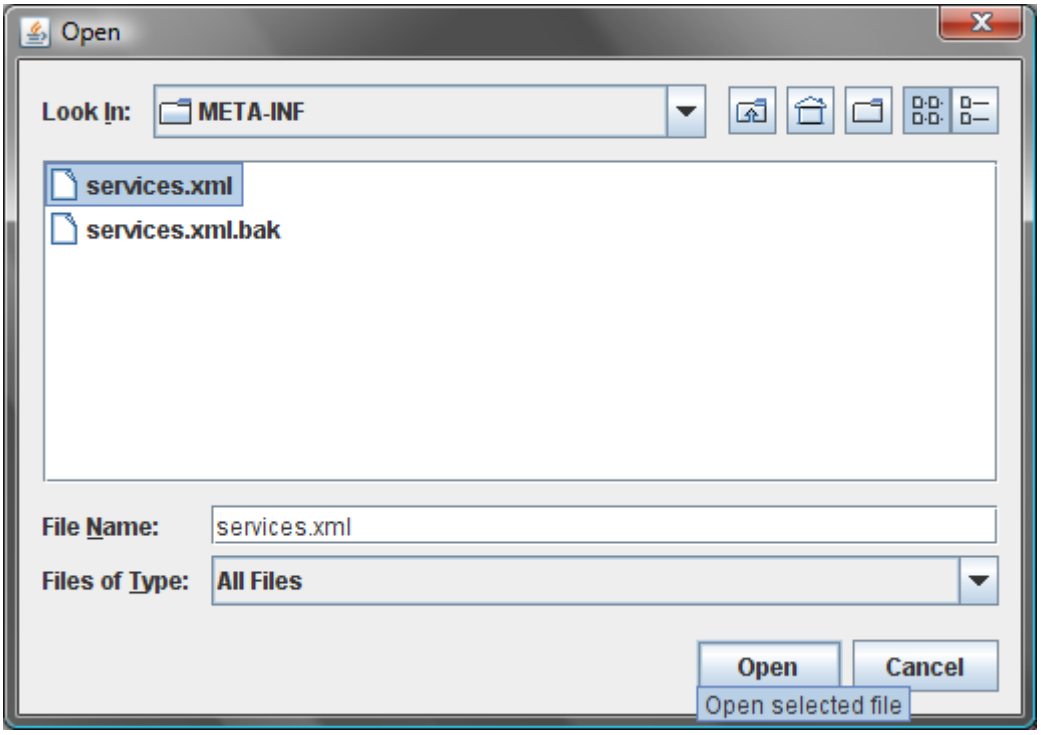


**Figura 4.10 – Janela de seleção do arquivo services.xml**

Neste ponto o usuário poderá realizar a configuração de segurança no lado servidor clicando no botão “Configurar segurança”. Na figura 4.11 é apresentado o resultado após o usuário ter clicado neste botão. A configuração de segurança está completa.

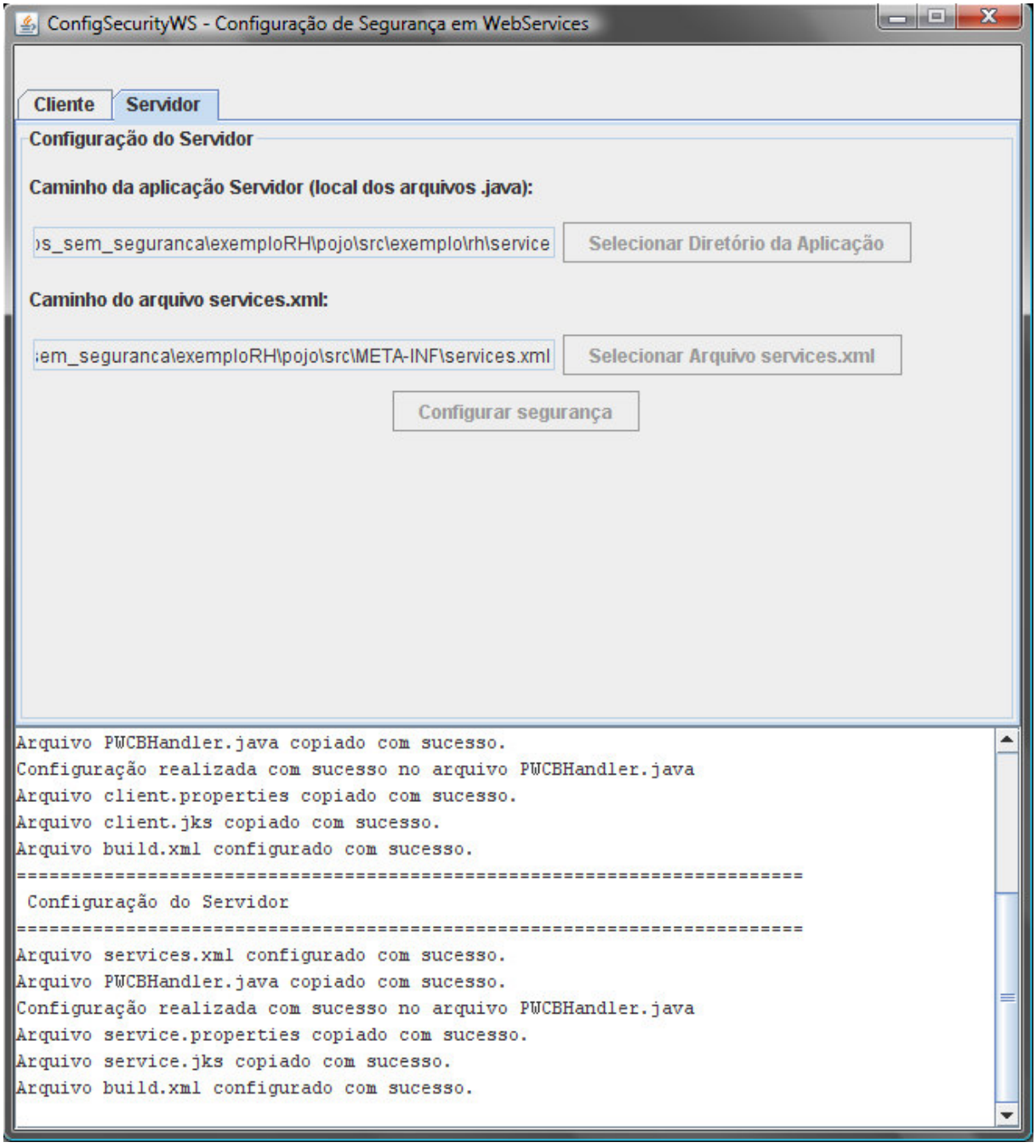


**Figura 4.11 – Resultado da configuração de segurança no lado Cliente**

Após ter configurado a segurança no cliente e no servidor o usuário poderá visualizar no subdiretório “saida”, localizado sob o diretório ConfigSecurityWS, as aplicações com a segurança configurada, como é demonstrado na figura 4.12.

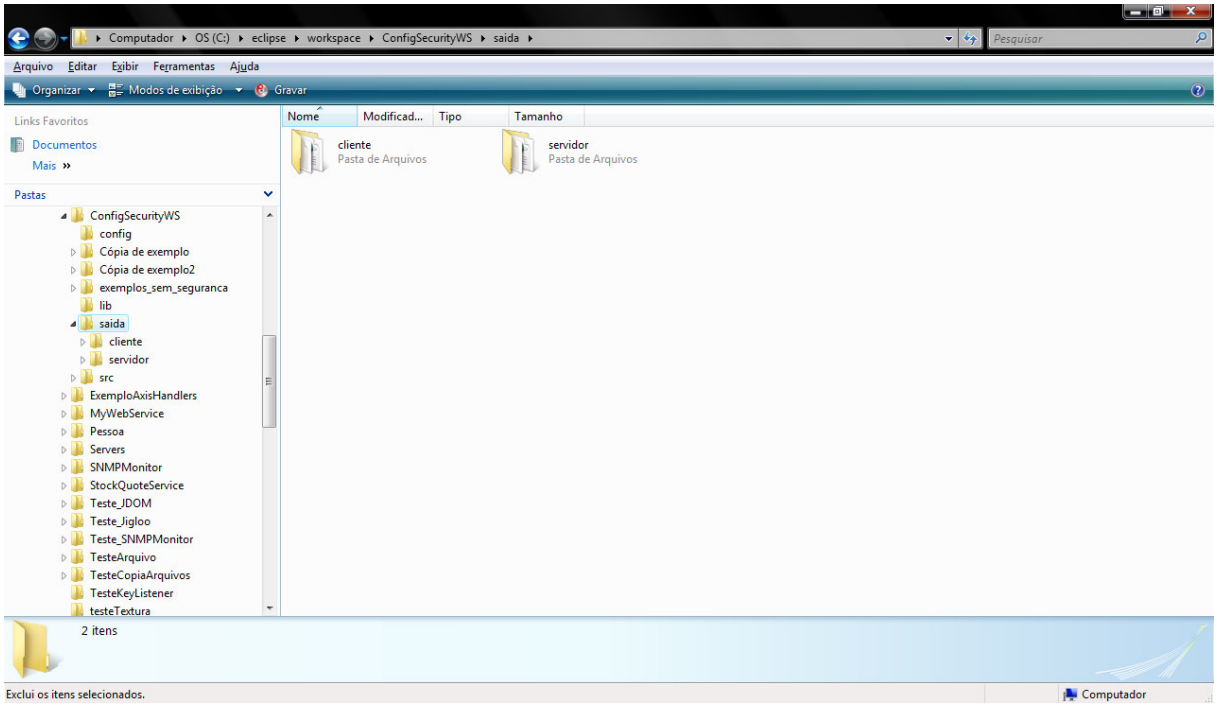


**Figura 4.12 – Diretório saída após configuração do Cliente e Servidor**

# 5 APLICAÇÃO EXEMPLO

Para que fossem possível os testes e validação do ConfigSecurity-WS, implementou-se uma aplicação seguindo o padrão Web Services. Essa aplicação consiste em um sistema de gerência de Recursos Humanos, onde o uso de segurança é essencial para manter em privacidade os dados de funcionários, empresas e os dados das contas bancárias desses funcionários. Esse exemplo é distribuído juntamente com a ferramenta, e pode ser visualizado no subdiretório “exemplos_sem_seguranca”.

A classe RHService possui as operações incluirFuncionario, pesquisarFuncionario, incluirEmpresa, pesquisarEmpresa, incluirContaBancaria e pesquisarContaBancaria. Estas são operações executadas pelo Web Service.

As classes ContaBancaria, Funcionário e Empresa são as entidades a serem gerenciadas pelo sistema. No lado cliente foram implementadas as seguintes classes:

- ContaBancariaClient: cria uma instância de ContaBancaria invocando o método incluirContaBancaria.
- EmpresaClient: cria uma instância de Empresa invocando o método incluirEmpresa.
- FuncionarioClient cria uma instância de Funcionário chamando incluirFuncionario; após criar esse funcionário faz uma pesquisa e imprime os dados deste invocando o método pesquisarFuncionario.

Os códigos dessas classes estão anexados ao final do trabalho.

# 6 AVALIAÇÃO DA FERRAMENTA

Neste capítulo serão apresentados os métodos de testes utilizados para a avaliação da ferramenta, com o objetivo de verificar se as mensagens trocadas possuem os itens de segurança incluídos pela ferramenta. Após verificar se as mensagens possuem os itens de segurança, serão realizados ataques contra as aplicações de Web Services configuradas pela ferramenta, visando certificar-se do bloqueio desses ataques.

## 6.1 ANÁLISE DAS MENSAGENS SOAP REQUEST E SOAP RESPONSE

Inicialmente realizou-se um teste antes das aplicações ter os itens de segurança configurado. Para analisar as mensagens de request e response, utilizou-se o software Wireshark (WIRESHARK, 2008), que permite capturar os pacotes enviados e recebidos pela interface de rede. No teste foram utilizadas duas máquinas, uma atuando como cliente e outra como servidor. A máquina com IP 192.168.1.100 atuará como cliente, e a máquina com IP 192.168.1.101 será o servidor.

### 6.1.1 Comunicação sem segurança

Nessa seção serão apresentados os resultados obtidos na execução do exemplo proposto anteriormente no capítulo 5, onde não há uso de segurança na comunicação com os Web Services. As figuras 6.1 e 6.2 apresentadas a seguir são capturas de tela do Wireshark mostrando respectivamente a mensagem SOAP Request enviada e a mensagem SOAP Response recebida após a execução da classe FuncionarioClient.

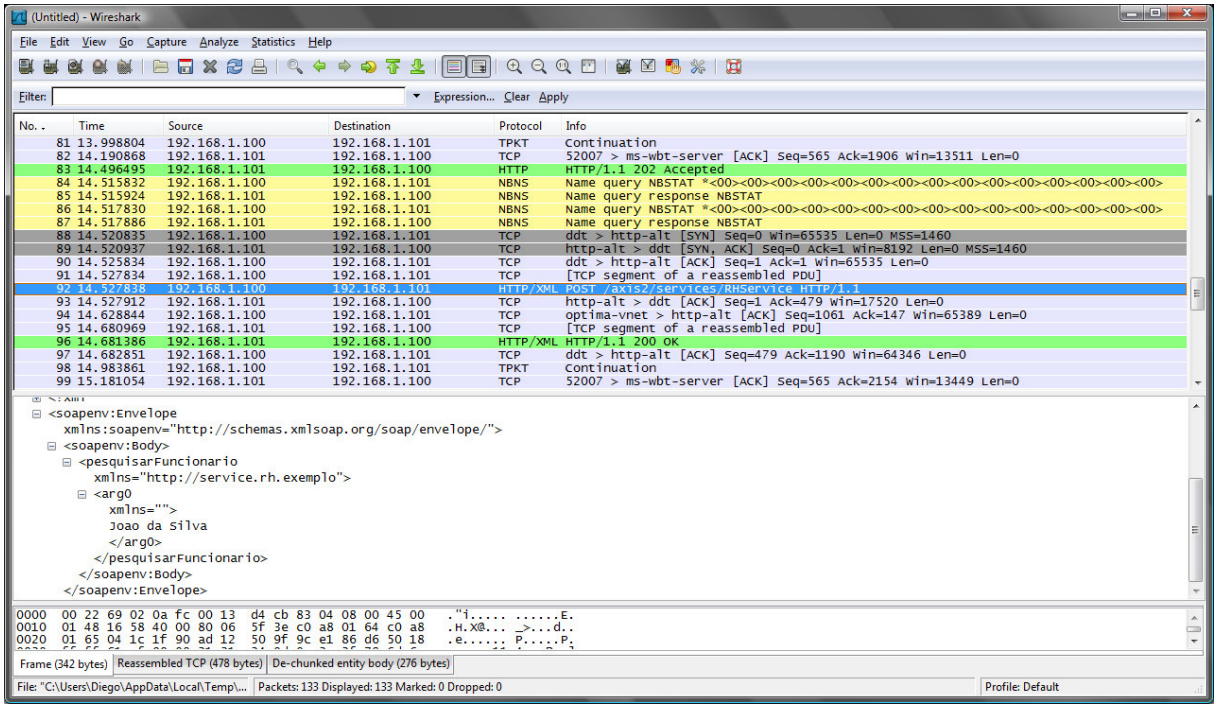


**Figura 6.1 – Mensagem SOAP Request sem segurança**

Na figura 6.1 é mostrado o resultado da execução do método pesquisarFuncionario, onde é informado como parâmetro o nome “João da Silva”. A seguir é demonstrado na figura 6.2 a mensagem SOAP response para esse método, que retorna os dados do funcionário pesquisado.

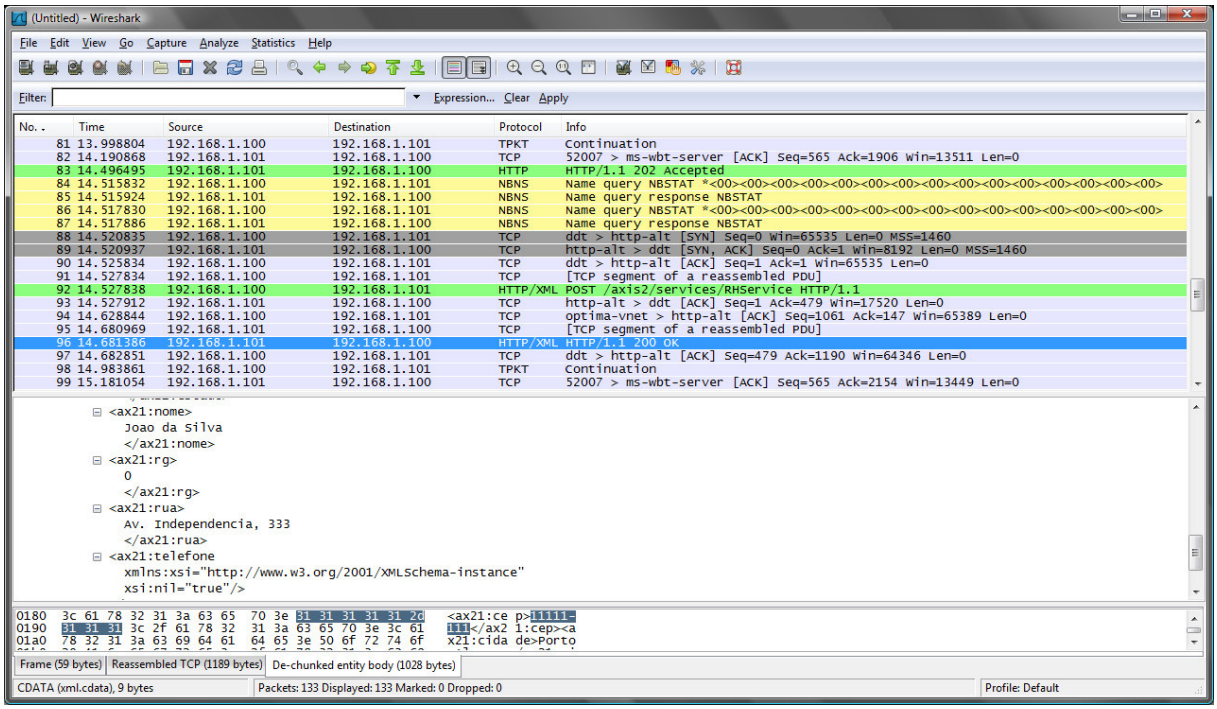


**Figura 6.2 – Mensagem SOAP Response sem segurança**

As figuras 6.1 e 6.2 demonstram que os dados estão explícitos na mensagem, sem uso de assinaturas ou criptografia dos dados. Portanto a mensagem poderá ser visualizada e modificada por qualquer pessoa não autorizada.

### 6.1.2 Comunicação com segurança

Nessa seção serão demonstrados os testes com o uso da segurança configurada pelo ConfigSecurity-WS. Na figura 6.3 apresentada a seguir, a mensagem de request possui os dados criptografados, como é possível observar abaixo do elemento <xenc:CipherValue>, na linha em destaque.

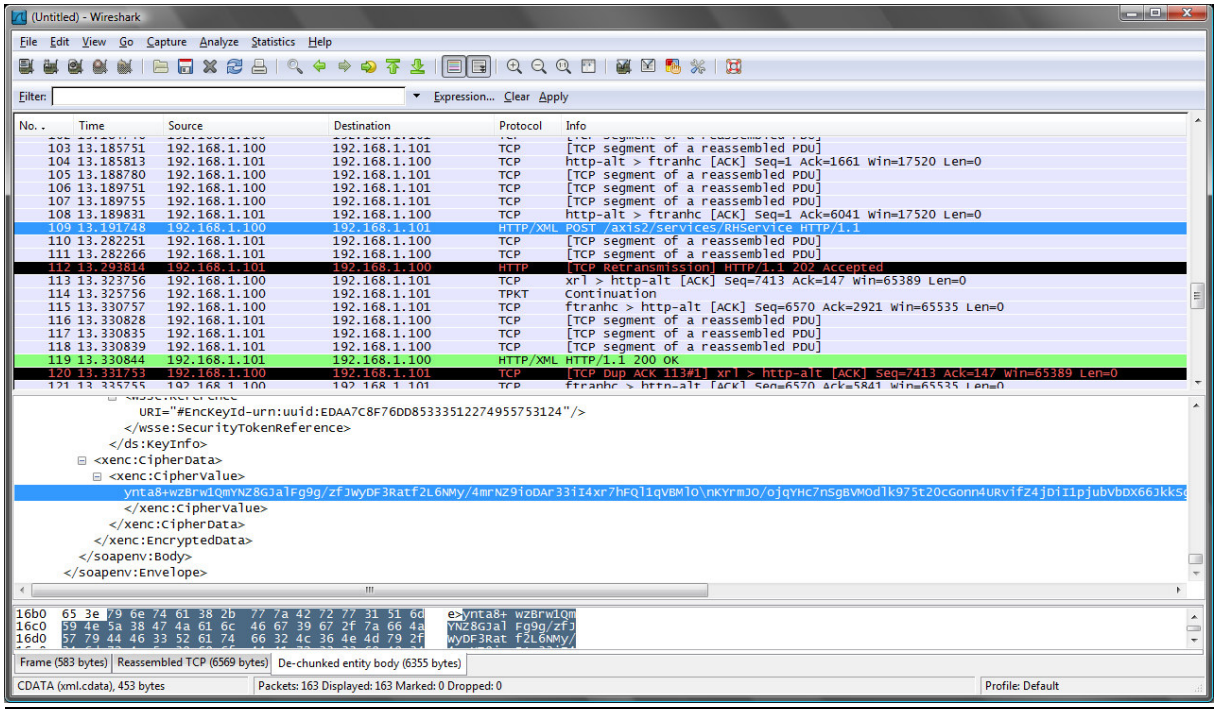


**Figura 6.3 – Mensagem SOAP Request com segurança (criptografia)**

Além da criptografia, existe também uma assinatura para a mensagem, representada pelo elemento <ds:signature> e um *timestamp*, o elemento <wsu:Timestamp> que possui data de criação e expiração, permitindo que essa mensagem seja aceita uma única vez. A assinatura e o *timestamp* estão representados na figura 6.4 a seguir.

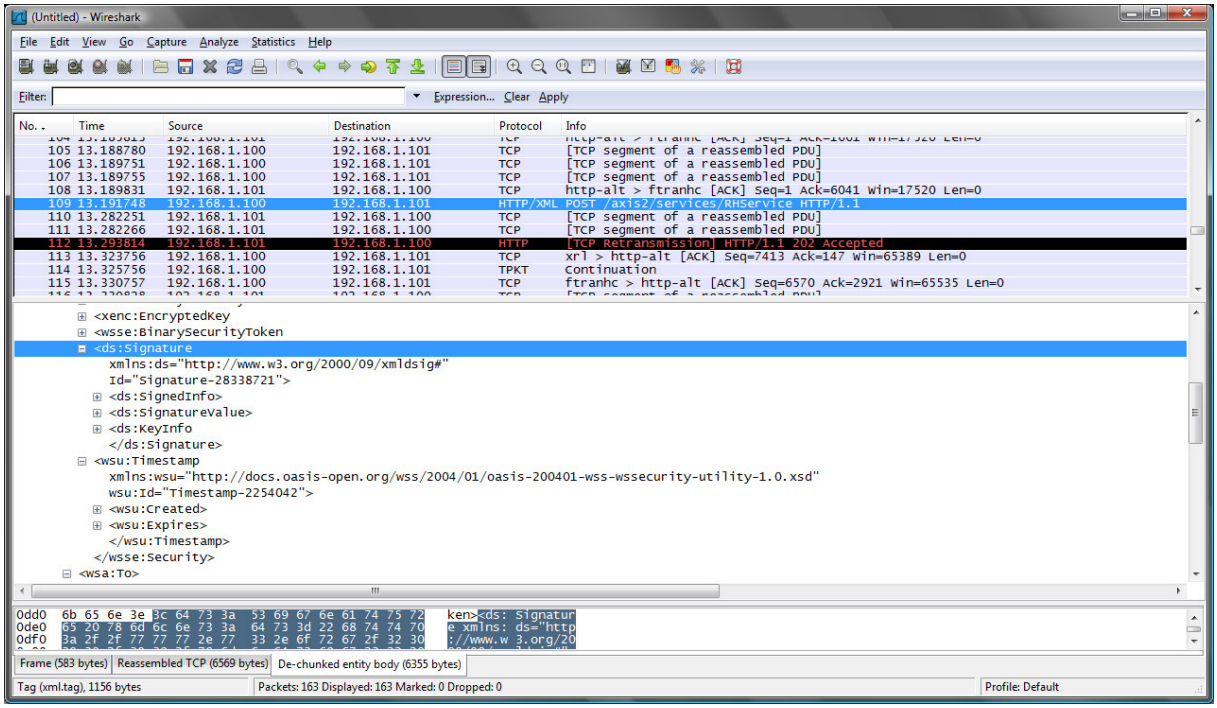


**Figura 6.4 – Mensagem SOAP Request com segurança (assinatura)**

A figura 6.5, a seguir, apresenta a mensagem response enviada pelo servidor, que também possui assinatura, *timestamp* e criptografia.

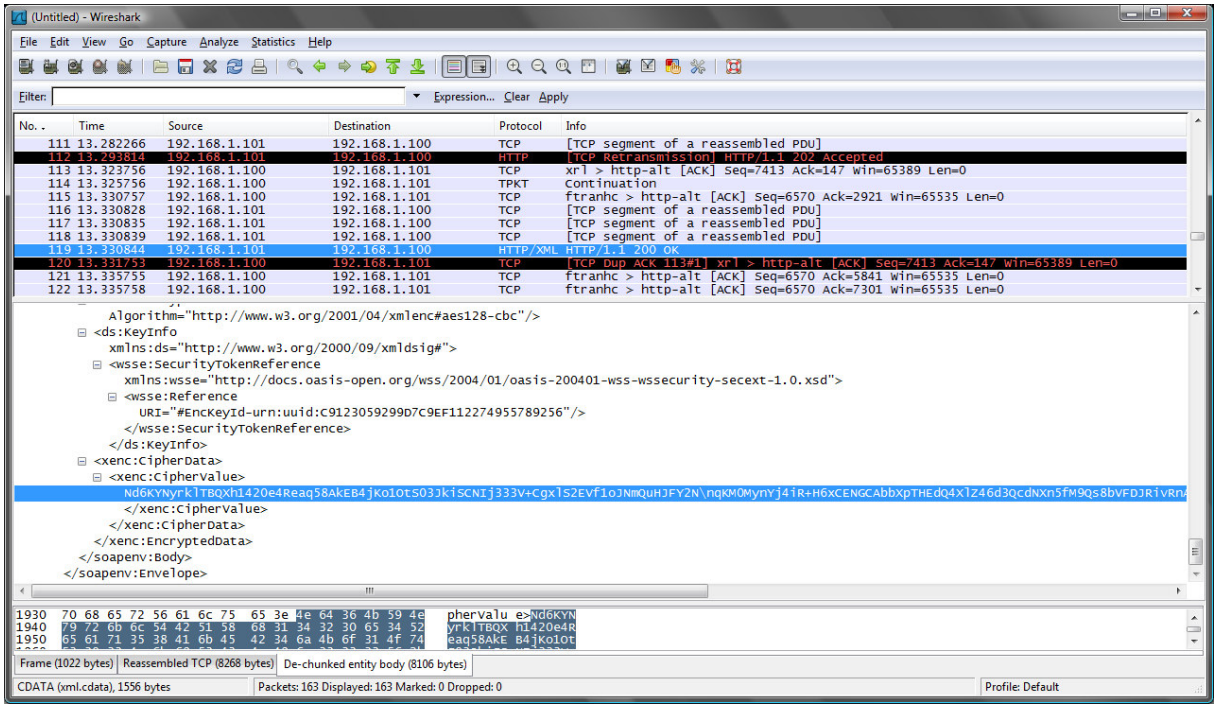


**Figura 6.5 – Mensagem SOAP Response com segurança**

## 6.2 ATAQUES CONTRA O WEB SERVICE

Com o objetivo de verificar a capacidade de bloquear ataques do Web Service que possui segurança configurada, alguns ataques foram testados. Os ataques testados são:

- Replay Attack (Ataque de Repetição): mensagens SOAP são capturadas e reenviadas exatamente como estão. O ataque executado consistiu nos seguintes passos:
    1. Captura da mensagem através do Wireshark. Nesse caso foi capturada uma mensagem de request do método pesquisarFuncionario.
    2. Verificou-se qual era o WSDL no endereço do servidor.
    3. Envio da mesma mensagem capturada utilizando o software soapUI (SOAPUI, 2006), que permite enviar mensagens diretamente através do formato XML, informando o WSDL do Web Service.

    Resultado: a primeira mensagem enviada foi aceita pelo Web Service, não sendo bloqueada, retornando uma mensagem de response como é demonstrado na figura 6.6. A mensagem que aparece selecionada é o request e a direita está representada a mensagem response.

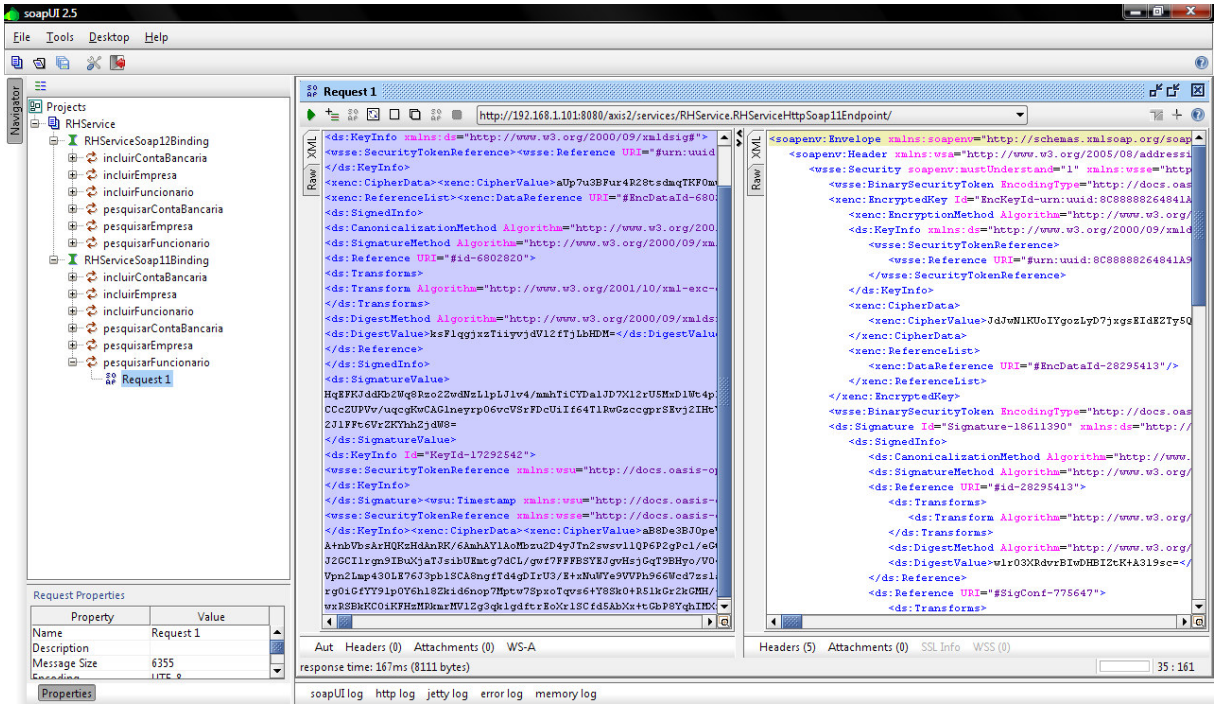


**Figura 6.6 – Replay Attack realizado com sucesso**

Após ter realizado esse ataque, foi realizado um novo ataque após um determinado tempo. A figura 6.7 mostra que a nova requisição não foi aceita, retornando uma mensagem de fault. Esse fato se deve ao tempo de expiração da mensagem, que por padrão é 5 minutos. Como o tempo esperado ultrapassou 5 minutos, a mensagem de request não pôde ser aceita. Portanto o tempo de expiração da mensagem deve ser reduzido de maneira a evitar esse tipo de ataque.

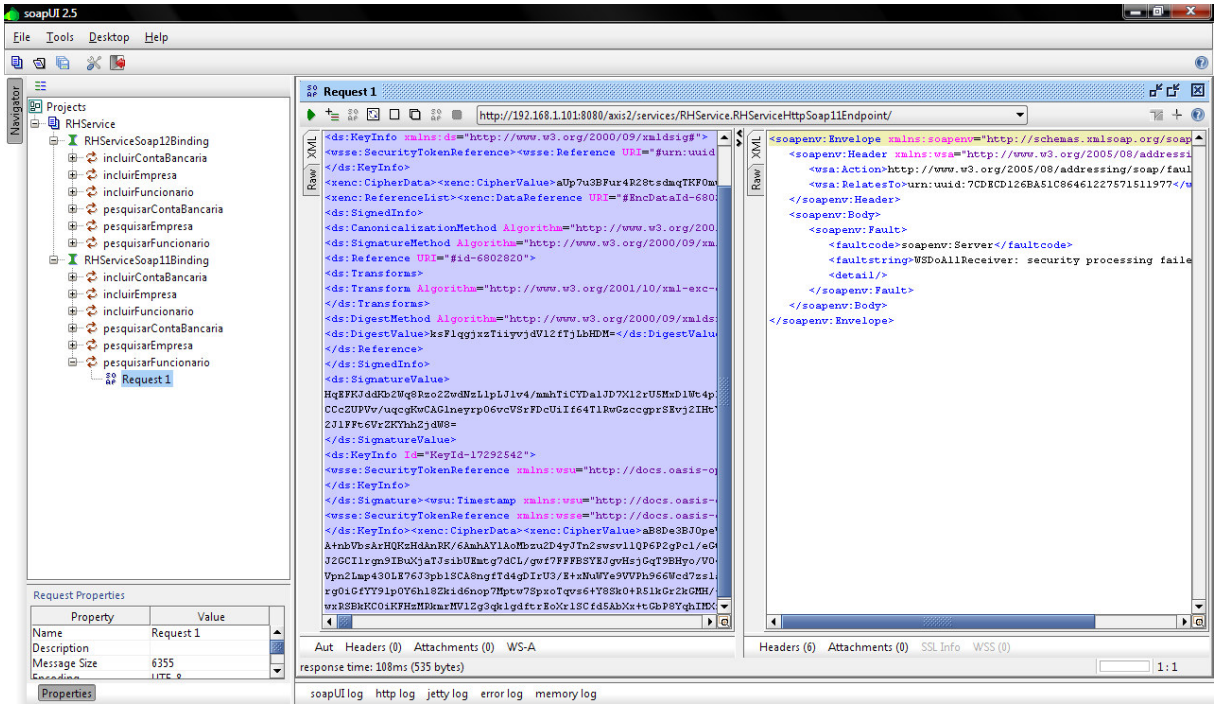


**Figura 6.7 – Replay Attack bloqueado**

A mensagem impressa no console do Tomcat indica a expiração do *timestamp* da mensagem, como é demonstrado na figura 6.8.

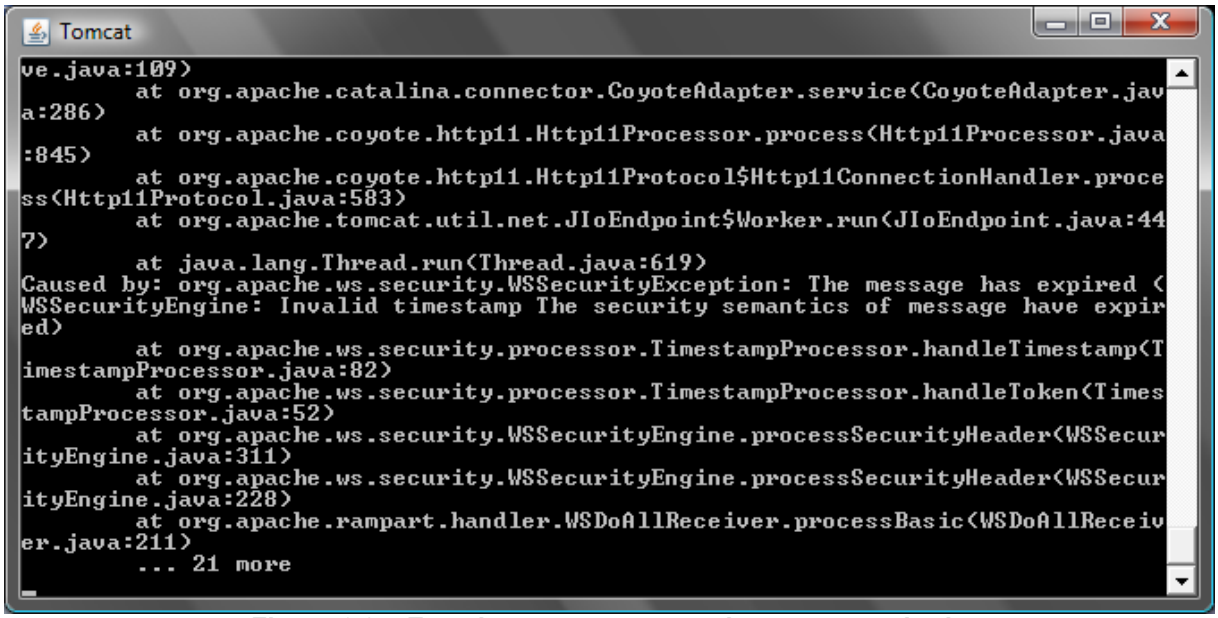


**Figura 6.8 – Erro de mensagem com timestamp expirado**

- Buffer Overflow (Estouro de Buffer): envio de uma mensagem SOAP com um parâmetro maior que o Web Service pode suportar. Dessa forma ele também é considerado um ataque do tipo Negação de Serviço. Por

exemplo, o invasor envia uma mensagem onde o parâmetro nome contém um número de caracteres maior do que o serviço permite. A execução desse ataque foi realizada através dos seguintes passos.

1. Captura da mensagem através do Wireshark. Nesse caso foi capturada uma mensagem de request do método pesquisarFuncionario.
2. Verificou-se qual era o WSDL no endereço do servidor.
3. Envio da mesma mensagem capturada utilizando o software soapUI. Nesse caso alterou-se a mensagem de request incluindo uma *string* que possui 1,5 MB de dados sob elemento <xenc:CipherValue>, a tag que contém os dados criptografados.

Resultado: o Web Service retornou a mensagem fault indicando "WSDoAllReceiver: security processing failed", como é demonstrado na figura 6.9, sem ocorrer interrupção na sua execução. Portanto a mensagem não foi aceita quando o Web Service processou a mensagem, no momento em que esta foi descriptografada.

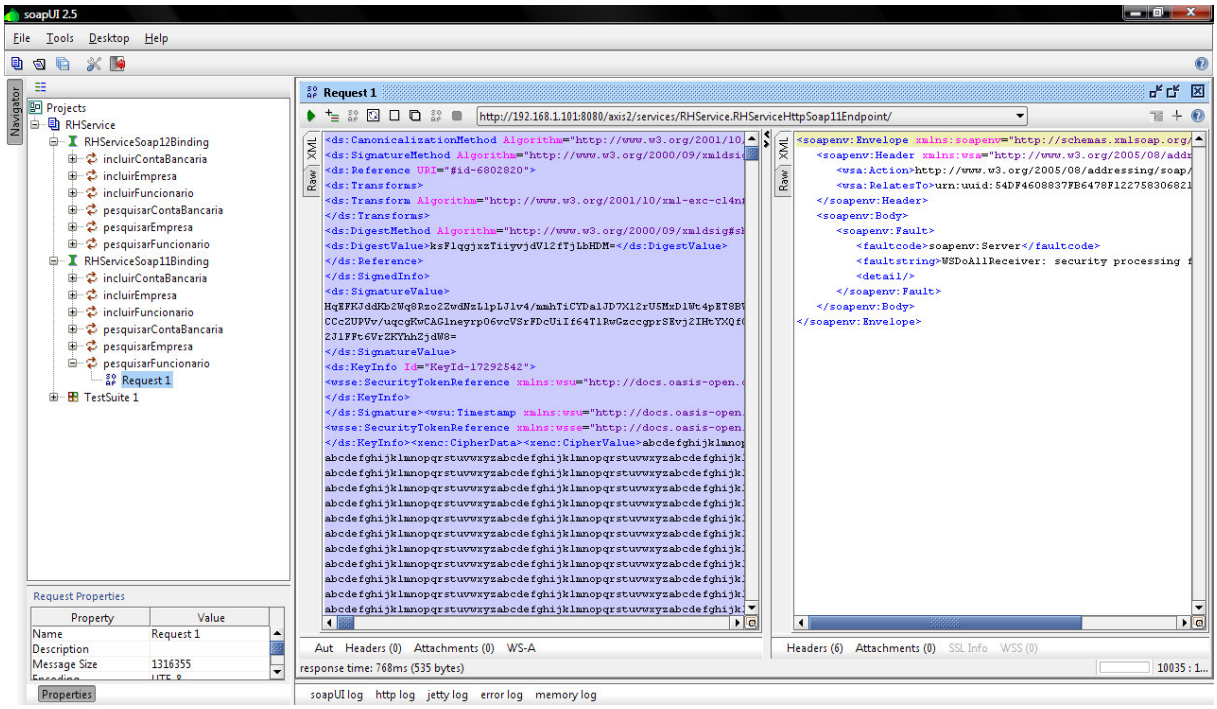


**Figura 6.9 – Ataque Buffer Overflow bloqueado**

Através dos resultados obtidos a eficácia da ferramenta foi comprovada. No caso do ataque de repetição da mensagem seria necessário apenas configurar o tempo de expiração de uma mensagem.

Para validar a ferramenta não foram testados todos os tipos de ataques, pois ataques também são usados de acordo com a arquitetura do sistema a ser atacado. Por exemplo, em um sistema que mantém um banco de dados, ataques do tipo SQL Injection poderiam ser usados com o objetivo de roubar dados restritos.

# 7 CONSIDERAÇÕES FINAIS

Inicialmente, através do estudo realizado, observou-se a necessidade do uso de segurança nos sistemas presentes na Web devido ao grande número de ataques possíveis de serem realizados. No caso dos Web Services existem os padrões de segurança a serem anexados nas mensagens para fornecer controle de segurança, mas existe também a necessidade de que este controle seja adicionado durante a fase de implementação nesse tipo de aplicação. A ferramenta ConfigSecurity-WS consegue fornecer segurança às aplicações após a fase de implementação, evitando que os desenvolvedores tenham de fazer um grande número de modificações nos códigos fontes para prover segurança.

Atualmente a ferramenta está operacional, e através desta foi possível realizar a configuração de segurança em mais de uma classe cliente Java. Os resultados puderam ser confirmados através dos testes apresentados no capítulo “Avaliação da Ferramenta”.

Uma das dificuldades encontradas durante a realização desse trabalho foi o fato de os padrões para Web Services serem resultados de artigos recentes. Durante a execução do trabalho ocorreu a necessidade de mudança da versão do Axis2 de 1.3 para a 1.4, pelo fato de a versão anterior não possuir uma exibição adequada para as mensagens de erro. Além disso, foi possível verificar que o uso de *timestamp* necessita de sincronização dos relógios da rede, que nos testes puderam ser solucionados sincronizando os relógios das máquinas através de um servidor NTP. Mas por outro lado, houve facilidade de acesso aos artigos, por estes estarem abertos ao público na Web, como foi o caso dos padrões WS definidos pela W3C e OASIS.

A ferramenta implementada neste trabalho fornece uma alternativa rápida e simples de configurar segurança em uma arquitetura de Web Services. Seu uso poderá ser de grande valia para empresas que adotam Web Services, sabendo que atualmente há um crescimento no número de empresas que adotam arquiteturas orientadas a serviços (SOA).

## 7.1 LIMITAÇÕES DO TRABALHO

O ConfigSecurity-WS possui algumas limitações: seu uso é possível somente para aplicações Java e que implementam a API Axis2. Além disso, a ferramenta gera a configuração no diretorio “saida” apenas para um projeto informado pelo usuário, que é sobrescrito caso o usuário realize uma nova configuração de segurança.

## 7.2 PERSPECTIVAS FUTURAS

O ConfigSecurity-WS foi desenvolvido de maneira que possam ser acrescentadas novas funcionalidades, pois se utilizou baixo acoplamento entre as classes implementadas. Algumas das sugestões de melhorias seriam permitir salvar as configurações de segurança para diferentes projetos e a opção de oferecer autenticação para um número n de usuários, que seriam cadastrados na ferramenta indicando o nome de usuário e senha.

# 8 REFERÊNCIAS BIBLIOGRÁFICAS

(APACHE, 2005) **Apache Web Services Wiki**
Disponível em: http://wiki.apache.org/ws/FrontPage/Axis/AxisGeneral
Acesso em setembro de 2007.

(APACHE-AXIS2, 2008) **Apache Axis2/Java Version 1.4 Documentation**
Disponível em: http://ws.apache.org/axis2/
Acesso em novembro de 2008.

(APACHE-AXIS2-CONFIG, 2007) **Axis2 Configuration Guide**
Disponível em: http://ws.apache.org/axis2/1_3/axis2config.html
Acesso em setembro de 2008.

(APACHE-RAMPART, 2008) **Securing SOAP Messages with Rampart**
Disponível em: http://ws.apache.org/axis2/modules/rampart/1_3/security-module.html
Acesso em março de 2008.

(BAJAJ, SIDDHARTH et al, 2006) **Web Services Policy Framework (WSPolicy)**
Disponível em: http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-polfram/ws-policy-2006-03-01.pdf
Acesso em novembro de 2007.

(BOHREN, JEFF et al, 2005) **Web Services Secure Conversation Language (WS-SecureConversation)**
Disponível em: http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-secon/ws-secureconversation.pdf
Acesso em novembro de 2007.

(DA CRUZ, 2005) **Serviços Web – Uma breve introdução**
Disponível em: http://www.nce.ufrj.br/conceito/artigos/2005/01p2-1.htm
Acesso em novembro de 2007.

(DELLA-LIBERA, GIOVANNI et al, 2005) **Web Services Security Policy Language (WS-SecurityPolicy)**
Disponível em: http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-secpol/ws-secpol.pdf
Acesso em novembro de 2007.

(IATAC, 2007) **State-of-The-Art Report - Software Security Assurance**
Disponível em: http://iac.dtic.mil/iatac/download/security.pdf
Acesso em setembro de 2007.

(IBM, 2002) **Use SOAP-based intermediaries to build chains of Web service**
Disponível em: http://www.ibm.com/developerworks/webservices/library/ws-soapbase/
Acesso em outubro de 2007.

(JAYASINGHE, 2006) **Axis2 Execution Framework**
Disponível em: http://www.developer.com/java/web/article.php/3529321
Acesso em junho de 2007.

(JDOM, 2007) **JDOM – Java Document Object Model**
Disponível em: http://jdom.org/
Acesso em setembro de 2008.

(OASIS, 2006) **Web Services Security: SOAP Message Security 1.1 (WS-Security 2004)**
Disponível em: http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf
Acesso em setembro de 2007.

(SOAPUI, 2006) **SOAP Testing Tool**
Disponível em: http://www.soapui.org
Acesso em novembro de 2008.

(RAHAMAN, MOHAMMAD, 2006) **Towards secure SOAP message exchange in a SOA**
Disponível em: http://delivery.acm.org/10.1145/1190000/1180382/p77-rahaman.pdf?key1=1180382&key2=4422806911&coll=portal&dl=ACM&CFID=15151515&CFTOKEN=6184618
Acesso em novembro de 2007.

(RAVUTHULA, 2002) Sridhar Ravuthula. **Web Services Applications and Security**
Disponível em: http://www.developer.com/services/article.php/1550461
Acesso em setembro de 2007.

(SINGHAL, 2006) Anoop Singhal. **A Guide to Secure Web Services**
Disponível em: http://homes.cerias.purdue.edu/~bhargav/WS-Workshop/august9-wsss-online-proceedings.pdf
Acesso em setembro de 2007.

(STOUPA, 2005) Konstantina Stoupa. **Policies for Web Security Services**
Disponível em: http://oswinds.csd.auth.gr/papers/idea2005.pdf
Acesso em setembro de 2007.

(WASC, 2004) **Web Aplication Security Consortium: Threat Classification**
Disponível em: http://www.webappsec.org/projects/threat/v1/WASC-TC-v1_0.pdf
Acesso em setembro de 2007.

(WIRESHARK, 2008) **Wireshark: Network Protocol Analyzer**
Disponível em: http://www.wireshark.org
Acesso em setembro de 2008.

(WS-I, 2005) **Security Challenges, Threats and Countermeasures**
Disponível em: http://www.ws-i.org/Profiles/BasicSecurity/SecurityChallenges-1.0.pdf
Acesso em setembro de 2007.

# ANEXO

```
 1  package exemplo.rh.rpcclient;
 2
 3  import javax.xml.namespace.QName;
 4
 5  import org.apache.axis2.AxisFault;
 6  import org.apache.axis2.addressing.EndpointReference;
 7  import org.apache.axis2.client.Options;
 8  import org.apache.axis2.rpc.client.RPCServiceClient;
 9
10  import exemplo.rh.persistencia.ContaBancaria;
11
12  public class ContaBancariaClient {
13
14      public static void main(String[] args1) throws AxisFault {
15
16          RPCServiceClient serviceClient = new RPCServiceClient();
17
18          Options options = serviceClient.getOptions();
19
20          EndpointReference targetEPR = new EndpointReference(args1[0]);
21          options.setTo(targetEPR);
22
23          // ////////////////////////////////////////////////////////////////////
24
25          /*
26           * Cria uma ContaBancaria e armazena no cadastro de RH.
27           */
28
29          // QName do metodo destino
            QName opIncluirContaBancaria = new QName("http://service.rh.exemplo",
30  "incluirContaBancaria");
31
32          /*
33           * Construindo nova conta
34           */
35          ContaBancaria conta1 = new ContaBancaria();
36
37          conta1.setContaCorrente(123456789);
38          conta1.setAgencia(123);
39          conta1.setNomeBanco("Banco do Brasil");
40
41
42          // Construção do array de argumentos para a chamada do metodo
43          Object[] opIncluirContaBancariaArgs = new Object[] { conta1 };
44
45          // Chamando o metodo
46          options.setAction("urn:incluirContaBancaria");
47          serviceClient.invokeRobust(opIncluirContaBancaria, opIncluirContaBancariaArgs);
```

```
48
49          ////////////////////////////////////////////////////////////////////
50
51      }
52  }
```

Código 11: ContaBancariaClient.java

```
 1  package exemplo.rh.rpcclient;
 2
 3  import javax.xml.namespace.QName;
 4
 5  import org.apache.axis2.AxisFault;
 6  import org.apache.axis2.addressing.EndpointReference;
 7  import org.apache.axis2.client.Options;
 8  import org.apache.axis2.rpc.client.RPCServiceClient;
 9
10  import exemplo.rh.persistencia.Empresa;
11
12
13
14  public class EmpresaClient {
15
16      public static void main(String[] args1) throws AxisFault {
17
18          RPCServiceClient serviceClient = new RPCServiceClient();
19
20          Options options = serviceClient.getOptions();
21
22          EndpointReference targetEPR = new EndpointReference(args1[0]);
23          options.setTo(targetEPR);
24
25          // ////////////////////////////////////////////////////////////////////
26
27          /*
28           * Cria uma Empresa e armazena no cadastro de RH.
29           */
30
31          // QName do metodo destino
32          QName opIncluirEmpresa = new QName("http://service.rh.exemplo", "incluirEmpresa");
33
34          /*
35           * Construindo nova empresa
36           */
37          Empresa empresa1 = new Empresa();
38
39          empresa1.setNomeComercial("Dell Computadores do Brasil");
40          empresa1.setRua("Av. Ipiranga, 6681");
41          empresa1.setBairro("Partenon");
42          empresa1.setCidade("Porto Alegre");
43          empresa1.setEstado("RS");
44          empresa1.setEstado("Tecnologia da Informacao");
45
46          // Construção do array de argumentos para a chamada do metodo
```

```
47          Object[] opIncluirEmpresaArgs = new Object[] { empresa1 };
48
49          // Chamando o metodo
50          options.setAction("urn:incluirEmpresa");
51          serviceClient.invokeRobust(opIncluirEmpresa, opIncluirEmpresaArgs);
52
53          ////////////////////////////////////////////////////////////////////
54
55      }
56  }
```

Código 12: FuncionarioClient.java

```
 1  package exemplo.rh.rpcclient;
 2
 3  import javax.xml.namespace.QName;
 4
 5  import org.apache.axis2.AxisFault;
 6  import org.apache.axis2.addressing.EndpointReference;
 7  import org.apache.axis2.client.Options;
 8  import org.apache.axis2.rpc.client.RPCServiceClient;
 9
10  import exemplo.rh.persistencia.Funcionario;
11
12
13
14  public class FuncionarioClient {
15
16      public static void main(String[] args1) throws AxisFault {
17
18          RPCServiceClient serviceClient = new RPCServiceClient();
19
20          Options options = serviceClient.getOptions();
21
22          EndpointReference targetEPR = new EndpointReference(args1[0]);
23          options.setTo(targetEPR);
24
25          // ////////////////////////////////////////////////////////////////
26
27          /*
28           * Cria um Funcionario e armazena ele no cadastro de RH.
29           */
30
31          // QName do metodo destino
            QName opIncluirFuncionario = new QName("http://service.rh.exemplo",
32  "incluirFuncionario");
33
34          /*
35           * Construindo novo Funcionario
36           */
37          Funcionario funcionario1 = new Funcionario();
38
39          funcionario1.setNome("Joao da Silva");
40          funcionario1.setRua("Av. Independencia, 333");
```

```
41          funcionario1.setCidade("Porto Alegre");
42          funcionario1.setEstado("RS");
43          funcionario1.setCep("11111-111");
44
45          // Construção do array de argumentos para a chamada do metodo
46          Object[] opIncluirFuncionarioArgs = new Object[] { funcionario1 };
47
48          // Chamando o metodo
49          options.setAction("urn:incluirFuncionario");
50          serviceClient.invokeRobust(opIncluirFuncionario, opIncluirFuncionarioArgs);
51
52          //////////////////////////////////////////////////////////////////////
53
54
55          //////////////////////////////////////////////////////////////////////
56
57          /*
58           * Pesquisando um Funcionario do RH
59           */
60
61          // QName do metodo a ser invocado
            QName opPesquisarFuncionario = new QName("http://service.rh.exemplo",
62   "pesquisarFuncionario");
63
64          //
65          String name = "Joao da Silva";
66
67          Object[] opPesquisarFuncionarioArgs = new Object[] { name };
68          Class[] returnTypes = new Class[] { Funcionario.class };
69
70          options.setAction("urn:pesquisarFuncionario");
71          Object[] response = serviceClient.invokeBlocking(opPesquisarFuncionario,
72                  opPesquisarFuncionarioArgs, returnTypes);
73
74          Funcionario result = (Funcionario) response[0];
75
76          if (result == null) {
77              System.out.println("Nenhum funcionario encontrado para " + name);
78              return;
79          }
80
81          System.out.println("Nome   :" + result.getNome());
82          System.out.println("Rua    :" + result.getRua());
83          System.out.println("Cidade :" + result.getCidade());
84          System.out.println("Estado :" + result.getEstado());
85          System.out.println("Cep    :" + result.getCep());
86
87
88          //////////////////////////////////////////////////////////////////////
89      }
90  }
```

Código 13: FuncionarioClient.java