PLATAFORMA DE DESENVOLVIMENTO PINHÃO PARANÁ

Bedel
Manual de Acoplagem

Janeiro – 2008

Sumário de Informações do Documento

Tipo do Documento: Manual

Título do Documento: Manual de Acoplagem do Bedel

Estado do Documento: Finalizado

Responsáveis: Cleverson Budel, Giuliano Diego Barbarra

Palavras-Chaves: bedel, web services, soap, acoplagem

Resumo: Esse documento contém o manual de acoplagem para sistemas hospedeiros prover e/ou consumir 'web services' utilizando o proto-agente Bedel

Número de páginas: 28

Software utilizados: Open Office 2.0.4

Versão	Data	Mudanças
1.0	04/08/2006	Elaboração do Manual
1.1	03/10/2006	Pequenos ajustes no texto sem mudar sua essência
1.2	18/05/2007	Atualização de informações de configuração do Bedel
1.3	05/07/2007	Inclusão das orientações de envio/recebimento de anexos
1.4	31/08/2007	Inclusão das orientações sobre questões de segurança
1.5	17/09/2007	Revisão e inclusão de informações sobre a arquitetura do Bedel (Capítulo 2)
1.6	09/01/2008	Retirada referência ao arquivo bedel.xml pois não é mais necessário.
1.7	22/08/2013	Alteração de responsáveis e complementação do item UserName Token

SUMÁRIO

1 INTRODUÇÃO

O sistema Bedel disponibiliza através de manuais, bibliotecas Java e do Sistema Administrativo, funcionalidades que facilitam e padronizam os procedimentos para consumir e prover *web services*. Este manual destina-se aos responsáveis pelo desenvolvimento de sistemas que necessitem consumir e/ou prover[1] estes serviços.

Nos tópicos subseqüentes será apresentada a ferramenta Bedel e os procedimentos necessários para a acoplagem ao sistema hospedeiro[2]. Dúvidas a respeito destes procedimentos devem ser verificadas junto à GTI.

1 Consumidor/Provedor: Estas expressões serão utilizadas no decorrer do documento para fazer referência à sistemas que utilizam e que disponibilizam os *web services*, respectivamente.

2 Sistema Hospedeiro, neste documento, refere-se a todo sistema que fará uso do Bedel. Os sistemas hospedeiros poderão ser consumidores ou provedores, dependendo da maneira que utilizam os *web services*.

2 INTRODUÇÃO AO BEDEL

O Bedel é uma ferramenta destinada à disponibilização, padronização e controle de sistemas que utilizem *web services*. Através dele é possível a criação provedores e consumidores de *web services* de forma rápida e padronizada; o Bedel também gera automaticamente bibliotecas para o consumo desses *web services* (caso sejam feitos utilizando o Framework Pinhão), facilitando a execução das chamadas aos serviços disponíveis. Além disso, o Bedel age como gerenciador dos *web services* através de sua interface administrativa, permitindo o cadastro e controle dos serviços disponibilizados por pessoas autorizadas no Sentinela.

Outra função do Bedel é o controle de segurança dos *web services* disponibilizados pela Celepar no padrão do Framework Pinhão, permitindo e negando acesso à funcionalidades dos sistemas provedores de acordo com configurações feitas no Bedel.

Basicamente, o Bedel pode ser dividido em duas partes: o sistema administrativo (Bedel Administrador) e as bibliotecas cliente (com funções para as aplicações provedoras e consumidoras), como veremos a seguir:

2.1 Bedel Administrador

A principal finalidade desse módulo é gerenciar os acessos às aplicações que disponibilizam web services (os provedores) utilizando o Framework Pinhão. O Bedel Administrador permite que sejam cadastrados todos os serviços a serem disponibilizados pelas aplicações provedoras, bem como todas as máquinas da rede e que aplicações consumidoras poderão utilizar esses *web services*.

A idéia é que uma instância do Bedel seja utilizada de forma semelhante à do Sentinela, para centralizar e gerenciar todos os servidores de web services que são mantidos pela Celepar, para que haja um controle das aplicações que estão utilizando *web services* e

quais *web services* são utilizados.

O Bedel Admin utiliza o Sentinela como sistema de segurança, através do qual o acesso às funções pode ser liberado ou bloqueado; por isso, o Bedel Admin já é cadastrado no Sentinela. Da mesma forma, os sistemas hospedeiros provedores também precisam estar cadastrados no Sentinela, pois os serviços só poderão ser cadastrados caso exista a permissão para eles no Sentinela. Já os sistemas consumidores precisam ser cadastrados quando utilizam as bibliotecas do Cliente Bedel para consumir os *web services*; porém é aconselhável que todos os sistemas, privados ou públicos, sejam cadastrados, para que se exista um controle dos consumidores de *web services* que são utilizados pela Celepar.

2.2 Bibliotecas cliente do Bedel

As bibliotecas cliente do Bedel são utilizadas tanto nas aplicações provedoras quanto nas consumidoras. Nas aplicações provedoras, o Bedel é utilizado para a disponibilização dos *web services*; nas consumidoras, ele dispõe de uma série de bibliotecas Java para que o sistema hospedeiro consumidor possa interagir com o provedor de *web services*, caso ambos tenham sido feitos no padrão do Framework Pinhão.

3 ACOPLANDO AO SISTEMA HOSPEDEIRO

Um sistema hospedeiro pode ser do tipo consumidor de serviços, provedor de serviços ou ambos. Para facilitar o entendimento, os procedimentos de acoplagem serão listados conforme o tipo de hospedeiro (consumidor ou provedor). Caso o hospedeiro seja consumidor e provedor, o desenvolvedor deverá realizar ambos os procedimentos descritos a seguir.

3.1 Acoplagem no Provedor de Serviços

A acoplagem do sistema Bedel a um sistema hospedeiro (Provedor) deve passar por alguns procedimentos, são eles:

- Cópia dos arquivos do sistema Bedel para o diretório do projeto do sistema hospedeiro. Neste passo alguns arquivos deverão ser importados pela aplicação. Este passo será detalhado no tópico 3.1.1.
- Configuração do sistema hospedeiro para o correto funcionamento do Bedel. Este passo será detalhado no tópico 3.1.2.
- Cadastramento das informações necessárias para funcionamento do Bedel. Esta etapa deve ser feita pelo administrador do sistema hospedeiro. Mais detalhes no tópico 3.1.3.

3.1.1 Cópia de arquivos

Os seguintes arquivos devem ser colocados dentro do *classpath* (pasta context/WEB-INF/lib caso a aplicação seja no padrão Pinhão) do sistema hospedeiro. Estes arquivos podem ser obtidos na página do Framework Pinhão: http://www.frameworkpinhao.pr.gov.br

- axis.jar
- bedel_client_x_x_x.jar
- commons-discovery-x.x.jar

- commons-logging.jar
- commons-http-client
- commons-codec
- properties-plugin.jar
- wsdl4j-x.x.x.jar
- jaxrpc.jar
- saaj.jar

É necessário também inserir as bibliotecas do **hibernate3**, caso seu projeto ainda não as tenha.

Caso a aplicação utiliza Maven, basta adicionar no arquivo pom.xml do projeto a dependência do bedel.

```
<dependency>
  <groupId>bedel</groupId>
  <artifactId>bedel-client</artifactId>
  <version>1.3.3</version>
</dependency>
```

3.1.2 Configuração do sistema hospedeiro

A configuração do sistema hospedeiro (Provedor) se resume em editar dois arquivos de configuração: web.xml e server-config.wsdd. No arquivo web.xml acrescentaremos configurações de forma que o sistema hospedeiro passe a responder requisições de chamadas a *web services*. No arquivo server-config.wsdd relacionaremos os *web services* que estarão disponíveis para consumo. As configurações dos arquivos citados será descrita a seguir:

- Arquivo WEB-INF/web.xml: Inserir o bloco de configuração a seguir (em negrito)

antes das configurações do sentinela (em itálico):

```
<!-- Filtro Bedel -->
<filter>
        <filter-name>BedelFilter</filter-name>
        <filter-class>gov.pr.celepar.bedel.client.BedelFilter</filter-class>
</filter>
<filter-mapping>
        <filter-name>BedelFilter</filter-name>
        <url-pattern>/services/*</url-pattern>
</filter-mapping>

<!-- Config Sistema de Seguranca -->
<filter>
        <filter-name>SegurancaFilter</filter-name>
        <filter-class>gov.pr.celepar.sentinela.client.SegurancaFilter</filter-class>
</filter>
<filter-mapping>
        <filter-name>SegurancaFilter</filter-name>
        <url-pattern>/*</url-pattern>
</filter-mapping>
```

Ainda no web.xml, inserir o bloco de configuração a seguir:

```
<!-- INICIO CONFIGURACOES DO AXIS -->
<listener>
        <listener-class>org.apache.axis.transport.http.AxisHTTPSessionListener</listener-
class>
</listener>
<servlet>
        <servlet-name>AxisServlet</servlet-name>
        <servlet-class>org.apache.axis.transport.http.AxisServlet</servlet-class>
</servlet>
<servlet-mapping>
        <servlet-name>AxisServlet</servlet-name>
        <url-pattern>/services/*</url-pattern>
</servlet-mapping>
<session-config>
        <!-- Default to 5 minute session timeouts -->
        <session-timeout>5</session-timeout>
</session-config>
<mime-mapping>
        <extension>wsdl</extension>
        <mime-type>text/xml</mime-type>
</mime-mapping>
<mime-mapping>
        <extension>xsd</extension>
        <mime-type>text/xml</mime-type>
</mime-mapping>
<!-- FIM CONFIGURACOES DO AXIS -->
```

- **Arquivo WEB-INF/server-config.wsdd:** Se este arquivo não existir, criar e inserir o bloco de configuração a seguir:

```
<?xml version="1.0" encoding="UTF-8"?>
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
	<globalConfiguration>
		<requestFlow>
			<handler type="java:gov.pr.celepar.bedel.client.Handler" />
		</requestFlow>
		<parameter name="attachments.implementation"
value="org.apache.axis.attachments.AttachmentsImpl" />
	</globalConfiguration>
	<transport name="http">
		<requestFlow>
			<handler name="URLMapper"
type="java:org.apache.axis.handlers.http.URLMapper" />
			<handler type="java:org.apache.axis.handlers.http.HTTPAuthHandler" />
		</requestFlow>
	</transport>

	<!--  INICIO DECLARACAO DE SERVICOS -->
	<service name="NomeServico" provider="java:RPC">
		<parameter name="allowedMethods" value="*" />
		<parameter name="className" value="gov.pr.celepar.NomeClasse" />
	</service>
	<!--  FIM DECLARACAO DE SERVICOS -->

</deployment>
```

Detalhamento da tag service:

```
1.	<service name="NomeServico" provider="java:RPC">
2.		<parameter name="allowedMethods" value="*" />
3.		<parameter name="className" value="gov.pr.celepar.NomeServico" />
4.	</service>
```

Na linha 1 temos a declaração no nome do serviço (parâmetro *name*), neste exemplo é NomeServico. Na linha 2 e 3 declaramos os métodos disponíveis (allowedMethods) e a classe (className) relacionada ao *web service*. Ou seja, estamos disponibilizando um *web service* denominado NomeServico que na prática é a classe gov.pr.celepar.NomeServico (como mostrado na linha 3). Todos os métodos desta classe estão acessíveis (*value=”*”*). Caso seja necessário especificar quais os métodos que estarão disponíveis, declarar o(s) nome(s) do(s) método(s) no campo *value* (linha 2), separados por vírgula.

Observação importante: Para o correto funcionamento das questões de segurança e de geração automática do arquivo .jar é **imperativo** que o nome do *web service*, configurado na linha 1, seja **idêntico** ao nome da classe a ser disponibilizada como *ws* (linha 3).

3.1.3 Cadastro dos serviços no Bedel Admin

Neste momento, os *web services* estão configurados no sistema hospedeiro, porém, ainda não podem ser consumidos. Para disponibilizá-los é preciso cadastrá-los corretamente no sistema Bedel Admin. O procedimento para o cadastro dos serviços é detalhado no Manual do Usuário disponível em http://www.frameworkpinhao.pr.gov.br.

3.2 Acoplagem no Consumidor de Serviços

ATENÇÃO: ATUALMENTE O BEDEL APENAS DISPONIBILIZA MÉTODOS PARA O CONSUMO DE SERVIÇOS PROVIDOS POR APLICAÇÕES GERENCIADAS PELA CELEPAR E FEITAS NA ARQUITETURA DO FRAMEWORK PINHÃO. AS BIBIOTECAS PARA CONSUMO DE OUTRAS APLICAÇÕES PODEM SER GERADAS UTILIZANDO-SE AS CLASSES WSDL4JAVA CONTIDAS NO PACOTE AXIS.

A acoplagem do sistema Bedel a um sistema hospedeiro (Consumidor) deve passar por alguns procedimentos, são eles:

- Cópia dos arquivos do sistema Bedel para o diretório do projeto do sistema hospedeiro. Neste passo alguns arquivos deverão ser importados pela aplicação. Ver tópico anterior 3.1.1;
- Importação do arquivo .jar gerado pelo Bedel Admin contendo uma estrutura para facilitar as chamadas ao *web service*. Neste passo o desenvolvedor deverá realizar o download do pacote e importá-lo no sistema hospedeiro. Este passo será detalhado no tópico 3.2.1;
- Configuração do sistema hospedeiro para o correto funcionamento do Bedel. Este passo será detalhado no tópico 3.2.2;
- Implementação das chamadas a *web services* com o uso das classes *proxy* geradas pelo Bedel Admin. Este passo será detalhado no tópico 3.3.

3.2.1 Importação do pacote gerado pelo Bedel Admin

Com o objetivo de facilitar as chamadas aos *web services* nos sistemas baseados na plataforma Pinhão, o sistema Bedel Admin, disponibiliza para cada *web service* um conjunto de classes proxy. Essas classes proxy constituem um artefato que simplifica a implementação das chamadas aos *web services*. Maiores detalhes sobre como realizar o download do arquivo JAR são encontrados no Manual do Usuário disponível em http://www.frameworkpinhao.pr.gov.br.

Após recuperar o arquivo JAR o desenvolvedor deverá copiar o arquivo para a pasta WEB-INF/lib[1] da aplicação hospedeira. Após a cópia do arquivo JAR o próximo passo será editar alguns arquivos de configuração do sistema hospedeiro (Consumidor).

3.2.2 Configuração do sistema hospedeiro

Para invocar um *web service* via classes *proxy* duas informações são necessárias: a **localização** do *web service* (endereço http que o serviço está respondendo - URL) e o **nome do consumidor**. O nome do consumidor é opcional e desnecessário na invocação de um serviço do tipo **público**. A seguir, serão explicados os mecanismos e suas respectivas configurações.

Mecanismo para obtenção do endereço (URL) de um *web service:*

As bibliotecas para acesso aos *web services* (proxy), descritas anteriormente, não armazenam internamente suas respectivas URL's que respondem às requisições SOAP. A URL é obtida, dinamicamente via *web service* do Bedel Admin, no momento da primeira chamada de cada serviço. Esta URL é armazenada em um *buffer* ficando disponível para as

1 No caso de aplicações Web

chamadas posteriores. Desta forma, o mesmo arquivo .jar importado para a aplicação hospedeira poderá acessar um serviço do ambiente de desenvolvimento ou de produção dependendo da resposta do *Bedel Service*. O algoritmo tentará recuperar a URL do *Bedel Service* primeiramente do banco de dados, se não encontrar buscará do arquivo de properties do servidor Jboss (properties-service.xml) e se não encontrar retornará um erro.

Mecanismo para obtenção do nome do consumidor

O nome do consumidor é opcional, sendo desnecessário caso o *web service* a ser invocado seja do tipo público (veja no manual do Usuário como cadastrar um *web service* público). Se não for do tipo público o nome do consumidor poderá ser configurado no arquivo WEB-INF/web.xml como será demonstrado a seguir:

No arquivo **web.xml** inserir o bloco de configuração:

```
<!-- Identificação do consumidor de web services -->
<context-param>
    <param-name>nomeAplicacao</param-name>
    <param-value>aquiONomeDoConsumidor</param-value>
</context-param>
```

O campo em negrito (**`aquiONomeDoConsumidor`**) deverá ser substituído pelo nome do consumidor correspondente ao sistema hospedeiro.

Opcionalmente ou para aplicações não Web, o desenvolvedor poderá configurar a localização do *web service* e o nome do consumidor de forma manual no momento da invocação, através da classe de configuração gov.pr.celepar.client.BedelConfig.

3.3 Invocando web services

ATENÇÃO: ATUALMENTE O BEDEL APENAS DISPONIBILIZA MÉTODOS PARA O CONSUMO DE SERVIÇOS PROVIDOS POR APLICAÇÕES GERENCIADAS PELA CELEPAR E FEITAS NA ARQUITETURA DO FRAMEWORK PINHÃO. AS

BIBIOTECAS PARA CONSUMO DE OUTRAS APLICAÇÕES PODEM SER GERADAS UTILIZANDO-SE AS CLASSES WSDL4JAVA CONTIDAS NO PACOTE AXIS.

O arquivo JAR gerado pelo Bedel Admin encapsula uma estrutura de classes e interfaces Java que realizam o trabalho de invocação dos *web services.* Esta característica permite ocultar a complexidade, simplificando a implementação no sistema hospedeiro. O arquivo JAR é composto por, pelo menos, 2 interfaces e 2 implementações. A saber:

Interfaces	Implementações
xxxxService.class	xxxxServiceLocator.class
xxxx_PortType.class	xxxxSoapBindingStub.class

Onde xxxx é o nome do *web service* e as classes ServiceLocator e SoapBindingStub implementam as interfaces Service e _PortType respectivamente.

O proxy gerado pelo Bedel Admin poderá ser utilizado em aplicações Java Web ou por aplicações Java **não** feitas para Web. A diferença é que nas aplicações Java Web temos a possibilidade de configurar o endereço do localizador de serviços e o nome do consumidor (web.xml) – *ver tópico 3.2.2*. Nas demais aplicações Java temos que setar manualmente essas informações. A seguir, serão listados dois exemplos: Invocação com configuração manual e Invocação com carga automática dos parâmetros contidos nos arquivos de configuração:

Exemplo de invocação com configuração manual dos parâmetros:

```
BedelConfig config = BedelConfig.getInstance();
config.setEnderecoProvedorDeNomes("http://swebdesenv01.celepar.parana:8080/bedel/services/Lis
taServicosFacade");
config.setNomeAplicacao("NomeDoConsumidor");
Calculadora service = new CalculadoraServiceLocator(config);
Calculadora_PortType calculadora = service.getCalculadora();
System.out.println(calculadora.soma(1,2));
```

Exemplo de invocação com carga automática dos parâmetros:

```
BedelConfig config = BedelConfig.getInstance(context);
Calculadora service = new CalculadoraServiceLocator(config);
Calculadora_PortType calculadora = service.getCalculadora();
System.out.println(calculadora.soma(1,2));
```

Neste caso, será necessário passar como parâmetro o objeto context da requisição.

Nos exemplos acima são realizadas chamadas ao serviço Calculadora executando-se o operador soma().

3.4 Web services com anexos

O envio e recebimento de anexos com *web services* é similar ao envio e recebimento de anexos em um e-mail. O provedor possui a mensagem, no caso o método a ser invocado e dentro dele o anexo. No consumidor, é recebida a mensagem e depois é extraído o anexo dela. Para o tratamento de anexos o proto-agente Bedel disponibiliza duas classes, Provedor e Consumidor. Por meio delas que os arquivos são manipulados nas mensagens de *web services*. A seguir será apresentado um exemplo de um método provedor de web-services com anexo, um exemplo de como obter o anexo no consumidor e um exemplo do arquivo client_config.wsdd que deve ser utilizado no consumidor.

Exemplo de método no provedor anexando arquivo:

```
public static void buscarArquivo(){
               FileInputStream anexo = new FileInputStream("/tmp/anexo.zip");
               Provedor.adicionaAnexo(anexo);
}
```

Neste caso, será necessário importar a classe “gov.pr.celepar.bedel.client.anexo.Provedor”.

Exemplo de invocação no consumidor recuperando arquivo anexado:

```
        //recuperando client_config que trabalha com anexos
        EngineConfiguration configuracao = new FileProvider("WEB-INF/client_config.wsdd");
        ExemploService service = new  ExemploServiceLocator(bedel,configuracao);
        Exemplo_PortType exemploInterface= service.getExemplo();

        //invocação do método que possui anexo
        exemploInterface.buscaAnexo();

        //recuperação do anexo dentro da mensagem recebida
        Call c = ((Stub)exemploInterface)._getCall();
        MessageContext msgContext = c.getMessageContext();
        Message msg= msgContext.getResponseMessage();

        InputStream[] arquivo = Consumidor.recuperaAnexos(msg);
```

Neste caso, será necessário importar as seguintes classes:

- "gov.pr.celepar.bedel.client.BedelConfig"
- “gov.pr.celepar.bedel.client.anexo.Consumidor”
- "gov.pr.celepar.bedel.ex.services.ExemploServiceLocator"
- "gov.pr.celepar.bedel.ex.services.Exemplo_PortType"
- "gov.pr.celepar.bedel.ex.services.ExemploLocalService"
- "org.apache.axis.EngineConfiguration"
- “org.apache.axis.client.Call”
- “org.apache.axis.Message”
- “org.apache.axis.MessageContext”
- “org.apache.axis.client.Stub”
- "org.apache.axis.configuration.FileProvider"

Exemplo de arquivo client_config.wsdd:

```
<?xml version="1.0" encoding="UTF-8"?>
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
   xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
     <!-- o CommonsHTTPSender que é o responsável pelo envio de anexos -->
     <transport name="http" pivot="java:org.apache.axis.transport.http.CommonsHTTPSender"/>
</deployment>
```

Esse arquivo deverá ficar dentro do diretório WEB-INF do projeto do sistema consumidor.

4 SEGURANÇA

Este capítulo irá explicar o funcionamendo da segurança no Bedel com restrições por consumidor, por IP e com o uso do framework WSS4J (Web Services Security for Java) da Axis. As restrições por consumidor e por IP, que são feitas internamente no Bedel por meio de um filtro configurado na aplicação provedora, ocorrem quando o serviço é configurado como privado. Já o framework WSS4J trabalha por meio de handlers que processam as requisições de pacotes SOAP e faz a segurança de acordo com as especificações do OASIS Web Service Security (WSS) . O WSS4J possui os handlers **WSDoAllSender** e **WSDoAllReceiver** que controlam a criação e o consumo de segurança em requisições SOAP. Os handlers trabalham por trás das aplicações e normalmente são transparentes para sistemas consumidores e provedores de *web services*. O arquivo descritor de deployment do *web services* (*.wsdd) pode conter toda a informação necessária para controlar o processo de segurança.

4.1 Níveis de Segurança

O Bedel oferece diversos níveis de segurança para as aplicações provedoras e consumidoras de *web services.* A restrição por Consumidor e por IP são configuráveis no Administrativo do Bedel. Já o UsernameToken, o XML Signature e o XML Encryption são configurados no arquivo descritor de deployment de *web services* que serão detalhados nos próximos tópicos. A complexidade de utilização dessas opções de segurança aumenta de acordo com o nível de segurança aplicado. A seguir uma figura que ilustra o grau de segurança/complexidade de cada opção de segurança:

4.2 Segurança por Consumidor

A comunicação entre provedor e consumidor é feita por meio de pacotes SOAP. Cada um desses pacotes possui um campo que identifica qual é o consumidor do pacote. Quando uma requisição é feita para um provedor de *web services,* o filtro do Bedel analisa essa requisição antes dela chegar no provedor. No caso da segurança por consumidor, o Bedel verifica se o consumidor indicado no pacote SOAP é um consumidor autorizado. Sendo um consumidor com acesso permitido, o Bedel encaminha o pacote SOAP para a aplicação provedora. Caso contrário, retorna para o consumidor um erro de acesso e registra um log no Banco de Dados do Bedel como tentativa indevida de acesso. Para utilizar esta validação é preciso que o serviço seja cadastrado no Bedel como privado e que todos os consumidores autorizados sejam cadastrados na interface administrativa. Quando o serviço disponibilizado possui anexos, esse tipo de segurança não pode ser utilizado por causa de uma característica específica do controlador de anexos.

4.3 Segurança por IP

O filtro do Bedel analisa cada pacote SOAP antes dele chegar na aplicação provedora. Essa análise, permite saber se é uma aplicação cadastrada no Administrativo do Bedel e se ela possui restrição por IP. Então o Bedel analisa o IP do pacote recebido verificando se ele possui acesso. Sendo um consumidor com IP permitido, o Bedel encaminha o pacote para a aplicação provedora. Caso contrário, retorna para o consumidor um erro de acesso e registra um log no Banco de Dados do Bedel como tentativa indevida de acesso. Para utilizar esta validação é preciso que o serviço seja cadastrado no Bedel como privado e que todos os IP autorizados sejam cadastrados na interface administrativa. Esta opção de segurança não possui nenhuma restrição para quando o serviço disponibilizado possui anexos.

4.4 Acoplagem do WSS4J

Para utilizar dos recursos de segurança disponibilizados pelo WSS4J, além das bibliotecas citadas no capítulo 3.1.1, é preciso adicionar também:

- wss4j-1.5.5.jar
- xmlsec-1.4.1.jar

Como a configuração do controle de segurança é feito no arquivo descritor de deployment do *web services*, esse deverá ser carregado manualmente nas aplicações consumidoras, pois o AXIS fornece um descritor de deployment padrão e utiliza ele quando o serviço é carregado sem a especificação desse. A seguir temos um exemplo de carga manual do arquivo de deployment na parte consumidora.

```
BedelConfig bedel = BedelConfig.getInstance();
EngineConfiguration configuracao = new FileProvider("WEB-INF/client_config.wsdd");
CalculadoraService service = new CalculadoraServiceLocator(bedel,configuracao);
```

O WSS4J consegue gerar e processar as seguintes opções de segurança do protocolo SOAP:

- XML Security
  - XML Signature
  - XML Encryption
- Tokens
  - Username Tokens
  - Timestamps
  - SAML Tokens

Essas opções de segurança serão descritas detalhadamente nos próximos tópicos. Esse framework de segurança também suporta os certificados X.509.

4.5 Classe de resposta de senhas

Uma das classes que o WSS4J utiliza para prover a senha necessária para criar um UsernameToken ou para assinar as mensagens é a classe PWCallBack que vem junto com o pacote do Bedel Client. Essa classe valida os usuários e atribui senhas a eles. Como esses

usuários são os consumidores cadastrados no Bedel Admin, é necessário que no handler o parâmetro “user” seja um desses consumidores cadastrados para que a validação possa ser feita.

4.6 Certificados

O padrão de certificado utilizado pelo WSS4J é o X.509. Ele precisa estar instalado no java (JSDK) utilizado pelas aplicações provedoras e consumidoras. O protoagente Tabelião oferece o seguinte manual para fazer essa instalação:

```
GIC_CertificadosDigitaisAplicacoesWEB
```

Além disso, o WSS4J possui um arquivo de propriedades (bedel-sign.properties e bedel-crypto.properties) para cada tipo de certificado. Esse arquivo é indicado no handler de XML Signature e no de XML Encryption. A seguir exemplo de conteúdo de arquivo de propriedades:

```
org.apache.ws.security.crypto.merlin.file=/tmp/sigilo/localhost.jks
org.apache.ws.security.crypto.merlin.keystore.alias=localhost
org.apache.ws.security.crypto.merlin.keystore.password=celepar
org.apache.ws.security.crypto.provider=org.apache.ws.security.components.crypto.Merlin
org.apache.ws.security.crypto.merlin.keystore.type=jks
```

onde:

org.apache.ws.security.crypto.merlin.file – caminho onde o certificado se encontra;
org.apache.ws.security.crypto.merlin.keystore.alias – alias do certificado;
org.apache.ws.security.crypto.merlin.keystore.password – senha do certificado;
org.apache.ws.security.crypto.provider – classe responsável pela criptografia;
org.apache.ws.security.crypto.merlin.keystore.type – tipo da chave

4.7 UsernameToken

UsernameToken é uma das formas de identificar o requisitor (consumidor) pelo seu username e por meio de uma senha para autenticar aquela identidade com o sistema provedor.

A aplicação que irá consumir serviços utilizando UsernameToken deve ter um arquivo descritor de deployment, client_config.wsdd, no diretório context/WEB-INF. A seguir temos um exemplo de conteúdo desse arquivo para configuração do uso de UsernameToken:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
 <transport name="http" pivot="java:org.apache.axis.transport.http.HTTPSender"/>
  <globalConfiguration >
   <requestFlow >
    <handler type="java:org.apache.ws.axis.security.WSDoAllSender" >
        <parameter name="action" value="UsernameToken"/>
        <parameter name="passwordType" value="PasswordDigest"/>
        <parameter name="user" value="GTF-Exemplo"/>
        <parameter name="passwordCallbackClass" value="gov.pr.celepar.bedel.client.PWCallback"/>
    </handler>
   </requestFlow >
  </globalConfiguration >
</deployment>
```

No atributo *action* definimos o uso do *UsernameToken* como forma de segurança. Em seguida realizamos a configuração do *passwordType* que define como a senha deve ser trafegada, o bedel exige que seja utilizado como tipo o *PasswordDigest* pois ele realiza uma encriptação da senha para não ser passada em texto plano. O atributo *user* representa o usuário que está realizando a chamada, esse atributo deve ter como valor um consumidor vinculado a esse serviço via bedel administrativo (verificar item 4.2 desse manual e os itens 2.3 e 2.6 do manual do Usuário encontrado no link: https://www.documentador.pr.gov.br/documentador/pub.do?action=d&uuid=f1824be6-9539-42eb-b553-ec23e1b4d67a). E por fim, o atributo *passwordCallbackClass* define qual classe irá definir a senha a ser trafegada para o serviço de destino.

Para o cliente realizar a chamada para um serviço que utiliza o UsernameToken, deve alterar a chamada da seguinte forma:

```
//Estancia normal do bedelConfig
BedelConfig bedelConfig = BedelConfig.getInstance(context);

//obtendo o arquivo wsdd para obter as configurações de serviços
EngineConfiguration configuracao = new FileProvider("WEB-INF/client_config.wsdd");

    try {
         //Configurando o serviço passando o arquivo de configuração wsdd
         ExemploService service = new ExemploServiceLocator(bedelConfig,configuracao);
        ExemploAction_PortType exemploPortType = service.getExemploAction();
        //Chmando um método do serviço
       exemploPortType.metodoServico();


    } catch (Exception e) {
         e.printStackTrace();
    }
```

O sistema provedor de *web services* também deve configurar o seu arquivo de deployment, server_config.wsdd, para funcionar com UsernameToken. A seguir um exemplo de configuração desse arquivo, para cada serviço que deseja usar deve configurar o handler como descrito abaixo:

```
<service name="ServicoExemplo" provider="java:RPC">
        <parameter name="allowedMethods" value="*" />
        <parameter name="className" value="gov.pr.celepar.exemplo.action.ServicoExemplo" />
        <requestFlow>
                <handler  type="java:org.apache.ws.axis.security.WSDoAllReceiver">
                        <parameter name="action" value="UsernameToken" />
                        <parameter name="passwordCallbackClass"
                                   value="gov.pr.celepar.bedel.client.PWCallback" />
                </handler>
        </requestFlow>
</service>
```

4.8 XML Signature

O XML Signature garante que os dados presentes no pacote não foram alterados antes de chegar ao seu destino. Ele utiliza certificados X.509 para assinar os pacotes SOAP. A aplicação que irá consumir serviços utilizando XML Signature deve ter o certificado configurado conforme o tópico 4.3 e um arquivo descritor de deployment, client_config.wsdd, no diretório context/WEB-INF. A seguir temos um exemplo de conteúdo desse arquivo para configuração do uso de XML Signature:

```
<requestFlow>
    <handler type="java:org.apache.ws.axis.security.WSDoAllSender" >
        <parameter name="action" value="Signature"/>
        <parameter name="signaturePropFile" value="bedel-sign.properties" />
        <parameter name="user" value="localhost"/>
        <parameter name="passwordCallbackClass"
value="gov.pr.celepar.bedel.client.PWCallback"/>
    </handler>
</requestFlow>
```

O sistema provedor de *web services* também deve configurar o seu arquivo de deployment, server_config.wsdd, para funcionar com XML Signature. A seguir um exemplo de configuração desse arquivo:

```
<requestFlow>
    <handler type="java:org.apache.ws.axis.security.WSDoAllReceiver">
        <parameter name="action" value="Signature"/>
        <parameter name="signaturePropFile" value="bedel-sign.properties" />
        <parameter name="user" value="localhost"/>
        <parameter name="passwordCallbackClass"
value="gov.pr.celepar.bedel.client.PWCallback"/>
    </handler>
</requestFlow>
```

4.9 XML Encryption

O XML Encryption criptograva o conteúdo dos pacotes SOAP garantindo o sigilo das informações presentes nos pacotes. Também utiliza certificados x.509 para fazer a criptografia dos pacotes SOAP. A aplicação que irá consumir serviços utilizando XML Encryption deve ter o certificado configurado conforme o tópico 4.3 e um arquivo descritor de deployment, client_config.wsdd, no diretório context/WEB-INF. A seguir temos um exemplo de conteúdo desse arquivo para configuração do uso de XML Encryption:

```
<requestFlow>
    <handler type="java:org.apache.ws.axis.security.WSDoAllSender" >
        <parameter name="action" value="Encrypt"/>
        <parameter name="encryptionPropFile" value="bedel-crypto.properties" />
        <parameter name="user" value="localhost"/>
        <parameter name="passwordCallbackClass"
value="gov.pr.celepar.bedel.client.PWCallback"/>
    </handler>
</requestFlow>
```

O sistema provedor de *web services* também deve configurar o seu arquivo de deployment, server_config.wsdd, para funcionar com XML Encryption. A seguir um exemplo de configuração desse arquivo:

```
<requestFlow>
    <handler type="java:org.apache.ws.axis.security.WSDoAllReceiver">
        <parameter name="action" value="Encrypt"/>
        <parameter name="encryptionPropFile" value="bedel-crypto.properties" />
        <parameter name="user" value="localhost"/>
        <parameter name="passwordCallbackClass"
value="gov.pr.celepar.bedel.client.PWCallback"/>
    </handler>
</requestFlow>
```

5 ANEXOS

5.1 Web services e interoperabilidade

ATENÇÃO: ATUALMENTE O BEDEL APENAS DISPONIBILIZA MÉTODOS PARA O CONSUMO DE SERVIÇOS PROVIDOS POR APLICAÇÕES GERENCIADAS PELA CELEPAR E FEITAS NA ARQUITETURA DO FRAMEWORK PINHÃO. AS BIBIOTECAS PARA CONSUMO DE OUTRAS APLICAÇÕES PODEM SER GERADAS UTILIZANDO-SE AS CLASSES WSDL4JAVA CONTIDAS NO PACOTE AXIS.

Neste tópico serão descritos algumas práticas recomendadas na implementação de *web services*. Serão levantadas, principalmente, questões relativas aos tipos de dados retornados pelos métodos (operadores) dos *web services*. Veremos a seguir que dependendo da estrutura utilizada aumentaremos ou reduziremos o nível de interoperabilidade dos serviços.

Se o *web service* em questão será invocado por aplicações desenvolvidas em linguagens e/ou plataformas diferentes de Java, então é recomendado o uso dos tipos primitivos determinados pela especificação JAX-RPC[1]:

```
base64Binary        byte[]
xsd:boolean         boolean
xsd:byte            byte
xsd:dateTime        java.util.Calendar
xsd:decimal         java.math.BigDecimal
xsd:double          double
xsd:float           float
xsd:hexBinary       byte[]
xsd:int             int
xsd:integer         java.math.BigInteger
xsd:long            long
xsd:short           short
xsd:string          java.lang.String
```

O uso de Collection's, Map's ou outra estrutura que trabalhe com objetos agregados é permitido, porém, a especificação SOAP[2] não padroniza estes tipos de dados complexos.

1 Java API for XML-based RPC, JSR 101, version 1.1
2 Simple Object Access Protocol

Desta forma, é possível implementar essas estruturas entre aplicações que utilizam o sistema Bedel, mas o funcionamento em outras implementações do SOAP não é garantido. Em particular, a plataforma .NET não suporta esses tipos. Para contornar esta limitação o desenvolvedor poderá utilizar vetores dos tipos primitivos relacionados acima.

Observação: Em testes realizados com utilização de estruturas mais complexas (Java Map's por exemplo) foram detectados elevados tempos de resposta devido a lentidão na geração do XML(SOAP). Portanto, aconselha-se a utilização dos tipos primitivos ou vetores de tipos primitivos quando for possível.