Spago JBI Adapter

Authors Gianfranco Boccalon

Document Goal

The goal of this document is to explain the implementation of the JBI adapter of Spago.

Versions history

Version/Release n°:	1.0	**Date Version/Release:**	17/07/2006
Description:	First release		

References

You can find more information about Spago framework in the following documents available on ObjectWeb forge (http://forge.objectweb.org/projects/spago):

[1] Spago Overview

[2] Spago User Guide.

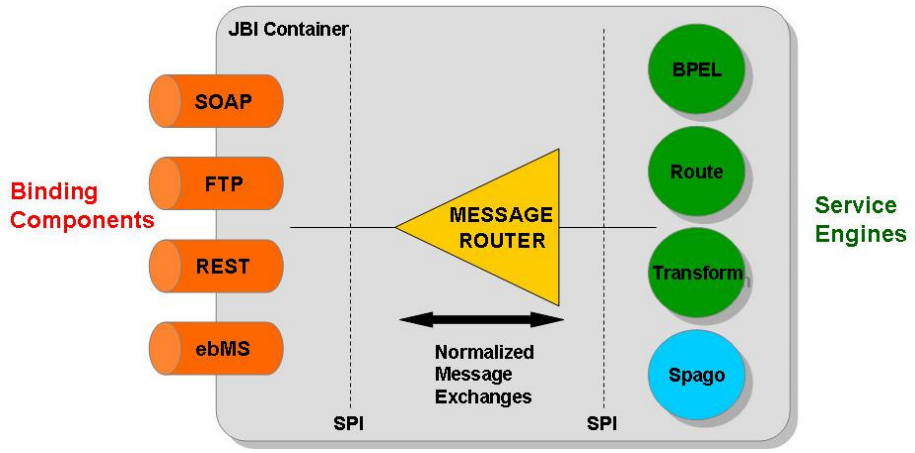
1 Introduction

An *adapter* is a component that accepts requests on a particular channel and translates them in a channel independent format (Spago's format is XML).
The JBI adapter allows exposing the services written with Spago as JBI components. All the services will be available through the binding components supplied by the container.

2 JBI Adapter

The JBI standard defines two different types of components:

- Binding components: provides communication services.
- Service Engines: provides all other kind of services.



Spago JBI Adapter is a Service Engine that allows the developer to write the services as standard Spago Services. These services can then be deployed in a JBI container, packaging them in a Service Engine archive using the predefined descriptors supplied by Spago.
You can use the action or modules dispatching mode, using the usual Spago infrastructure services.

2.1.1 How to develop a service

The services are built according the usual Spago rules: please refer to the User Guide for the details.
In the released sample we developed a test action that was invoked by a Java client through a HTTP call.
Here you find the code of the sample action:

```
public class TestAction extends AbstractAction {

    public void service(SourceBean serviceRequest, SourceBean serviceResponse)
            throws Exception {
        serviceResponse.setAttribute("TEST_ATTRIBUTE",
                                "Servizio di test invocato");
    }

}
```

Following there is the content of the configuration file actions.xml:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ACTIONS>
      <ACTION name="TEST_ACTION"
            class="it.eng.spago.jbi.test.TestAction"
            scope="REQUEST">
            <CONFIG></CONFIG>
      </ACTION>
</ACTIONS>
```

There is no publisher associated to this action.

2.1.2 Create the Service Engine

To deploy a component in a JBI container, you have to package the component according to the JBI specifications.
For Spago services, the package (a .zip file) has the following structure:

<spago libraries>.jar	Spago libraries
<dependencies>.jar	Dependencies libraries
<application classes>.jar	Application libraries
META-INF/	
jbi.xml	JBI descriptor for the component
WEB-INF/	
conf/	
spago/	Spago configuration files
<spago configuration files>	

The configuration files are inside a folder called *WEB-INF*, only because they are the same files used in the Web sample called *SpagoStartup* (the *master.xml* file refer to all the configuration files as *WEB-INF\conf\spago\<filename>.xml*).
In the *jbi.xml* you have to define all the libraries used by the component, including the application libraries where are stored your services.

This is exactly the structure of the file *spago-jbi-component.zip*, released in the Spago JBI sample. We show here the descriptor of the sample:

```
<jbi version="1.0" xmlns='http://java.sun.com/xml/ns/jbi' xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'>
  <component type="service-engine">
    <identification>
      <name>SpagoServiceEngine</name>
      <description>Description of your component.</description>
    </identification>

    <component-class-name
          description="JBI Adapter for Spago">it.eng.spago.jbi.SpagoComponent</component-class-name>
    <component-class-path>
                <path-element>spago-jbi-2.1.0.jar</path-element>
                <path-element>……</path-element>
    </component-class-path>

    <bootstrap-class-name>it.eng.spago.jbi.init.Bootstrap</bootstrap-class-name>
    <bootstrap-class-path>
                <path-element>spago-jbi-2.1.0.jar</path-element>
                <path-element>……</path-element>
    </bootstrap-class-path>
  </component>
</jbi>
```

2.1.3 Create the Service Assembly

To use the Service Engine you have to create a Service Assembly where the Service Engine is used together (probably) other JBI components.

The Service Assembly package (a .zip file) has the following structure:

META-INF/	
jbi.xml	JBI descriptor for the Service Assembly
<List of Service Unit packages>	Packages (.zip) of the Service Units used in the Service Assembly

The *jbi.xml* descriptor contains the list of all Service Units included in the Service Assembly.

Following there is the *jbi.xml* descriptor used in the Spago JBI sample:

```
<jbi version="1" xsi:schemaLocation="http://java.sun.com/xml/ns/jbi  ./jbi.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/jbi">
   <service-assembly>
      <identification>
         <name>SpagoServiceAssembly</name>
         <description>Spago sample Service Assembly</description>
      </identification>
      <service-unit>
        <identification>
          <name>SpagoSU</name>
          <description>Service Engine Sub-Assembly for JBI Demo</description>
        </identification>
        <target>
          <artifacts-zip>SpagoSU.zip</artifacts-zip>
          <component-name>SpagoServiceEngine</component-name>
        </target>
      </service-unit>
</service-assembly>
</jbi>
```

This sample uses only a Service unit in the Service Assembly.

2.1.3.1 Create the Service Unit

To create a Service Unit you have to create a package (a .zip file) with the following structure:

META-INF/	
jbi.xml	JBI descriptor for the Service Unit
servicelist.xml	Packages (.zip) of the Service Units used in the Service Assembly

The *jbi.xml* descriptor should define the Service Engine that compose the Service Unit (the SEs are usually unusable if they are not referenced by a Service Unit in a Service Assembly).
Following you find the descriptor of the Service Unit used in the Spago JBI sample:

```
<jbi version="1.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns="http://java.sun.com/xml/ns/jbi"
        xmlns:spago="http://spago.eng.it/spago.wsdl"
        xsi:schemaLocation="http://java.sun.com/xml/ns/jbi jbi.xsd">
  <services binding-component="false">
    <provides service-name="spago:SpagoServiceEngine" endpoint-name="SpagoEndpoint" />
```

```
    </services>
</jbi>
```

2.1.4 Deploy and test the service

Once created the Service Engine package and the Service Assembly package, you can deploy them to your JBI container. The deploy process is dependent on the JBI container: we tried our sample on ServiceMix, where you first put the Service Engine package in the /install folder, and then you put the Service Assembly package in the /deploy folder.

Configuring the ServiceMix HTTP connector (for example) to call our Service Engine (please refer to the README.txt of the sample for details) you can use the test class *it.eng.spago.jbi.test.client.**HTTPClient*** to activate the Service Engine and obtain a response like:

```
<?xml version="1.0" encoding="UTF-8"?>
<XML encodings of some characters.....>
<RESPONSE>
 <SERVICE_RESPONSE TEST_ATTRIBUTE="Servizio di test invocato"/>
 <ERRORS/>
</RESPONSE>
```

3 Dependency from JBI Container

Sometimes it's useful to use specific classes of the JBI container where the component will be deployed.
For example in the class *it.eng.spago.jbi.**ServiceUnitManager*** that implements the JBI interface *javax.jbi.component.**ServiceUnitManager*** , if the deploy is successful we have to produce a XML fragment containing some informations about the deploy. To do this we rely on a utility class of ServiceMix called *org.apache.servicemix.jbi.framework.**ManagementSupport***.
To separate this kind of code, from the core code of JBI adapter, we created a separate package called ***spago-jbi-providers*** where we put all the container dependent code.
In the configuration file *conf/spago/jbi.xml* we choose which provider to use by the attribute *DEFAULT_CONTAINER*, as you can see from this configuration sample:

```
<JBI>
      <CONTAINER
        class="it.eng.spago.jbi.containerproviders.ServiceMixContainerProvider"
        name="SERVICEMIX" />
      <CONTAINER
        class="it.eng.spago.jbi.containerproviders.PetalsContainerProvider"
        name="PETALS" />
      <CONTAINER
        class="it.eng.spago.jbi.containerproviders.SunRIContainerProvider"
        name="SUN_RI" />

      <DEFAULT_CONTAINER name="SERVICEMIX" />
</JBI>
```

Actually we provide the implementation only for ServiceMix.