Creating a protocol bridge

Creating a Protocol Bridge

ServiceMix can be used to create applications that serve as a bridge between two different protocols. Messages flow in via one protocol and out the other protocol. In this example, we will bridge the HTTP protocol with the JMS protocol using an InOnly message exchange pattern (MEP). We will also place an XSLT transformation service between the two protocols to further demonstrate that a service is not tied to a given protocol.

The following tutorial will explain how to create a JBI Service Assembly (aka a JBI application) to serve as a bridge between two different protocols. This tutorial will leverage some Maven archetypes for ServiceMix and also the Maven JBI plugin.

This tutorial has been written for ServiceMix 3.2 or greater. For ServiceMix 3.0, see this page.

Identifying the JBI Service Units

Our JBI Service Assembly (SA) will be composed of the following JBI Service Units (SUs)):

- An HTTP+SOAP consumer
- A EIP pipeline to call the XSLT transformer and the JMS provider
- An XSLT transformer
- A JMS provider

Directory layout

We aim to have the following project structure:

```
bridge/
  pom.xml
  bridge-http-su/
    ...
  bridge-eip-su/
    ...
  bridge-xslt-su/
    ...
  bridge-jms-su/
    ...
  bridge-sa/
    ...
```

Creating the Service Units

First, create a directory named bridge and move into that directory:

```
$ mkdir bridge
$ cd bridge
```

Create a file in the bridge directory named pom.xml and copy/paste the following content:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">

  <modelVersion>4.0.0</modelVersion>

  <groupId>org.apache.servicemix.samples.bridge</groupId>
  <artifactId>bridge</artifactId>
  <packaging>pom</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>ServiceMix Bridge Demo</name>

</project>
```

This is known as the root pom.xml because it is for the main bridge project. We'll come back to this in a bit.

In the bridge directory, run the following commands (remove the trailing backslashes and make each command fit on one line) to create the JBI service units (SUs):

```
$ mvn archetype:create \
        -DarchetypeGroupId=org.apache.servicemix.tooling \
        -DarchetypeArtifactId=servicemix-http-consumer-service-unit \
        -DarchetypeVersion=3.2.1 \
        -DgroupId=org.apache.servicemix.samples.bridge \
        -DartifactId=bridge-http-su

mvn archetype:create \
        -DarchetypeGroupId=org.apache.servicemix.tooling \
        -DarchetypeArtifactId=servicemix-jms-provider-service-unit \
        -DarchetypeVersion=3.2.1 \
        -DgroupId=org.apache.servicemix.samples.bridge \
        -DartifactId=bridge-jms-su

mvn archetype:create \
        -DarchetypeGroupId=org.apache.servicemix.tooling \
        -DarchetypeArtifactId=servicemix-eip-service-unit \
        -DarchetypeVersion=3.2.1 \
        -DgroupId=org.apache.servicemix.samples.bridge \
        -DartifactId=bridge-eip-su

mvn archetype:create \
        -DarchetypeGroupId=org.apache.servicemix.tooling \
        -DarchetypeArtifactId=servicemix-saxon-xslt-service-unit \
        -DarchetypeVersion=3.2.1 \
        -DgroupId=org.apache.servicemix.samples.bridge \
        -DartifactId=bridge-xslt-su

mvn archetype:create \
        -DarchetypeGroupId=org.apache.servicemix.tooling \
        -DarchetypeArtifactId=servicemix-service-assembly \
        -DarchetypeVersion=3.2.1 \
        -DgroupId=org.apache.servicemix.samples.bridge \
        -DartifactId=bridge-sa
```

Each of the commands above will create a directory containing a Maven project skeleton that is ready to be configured. The rest of the tutorial will focus on this.

Back to the Root pom.xml

Now that we have created the SUs and SA structure, let's take a look at the root pom.xml. Below is what you should see now:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">

  <modelVersion>4.0.0</modelVersion>

  <groupId>org.apache.servicemix.samples.bridge</groupId>
  <artifactId>bridge</artifactId>
  <packaging>pom</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>ServiceMix Bridge Demo</name>

  <modules>
    <module>bridge-http-su</module>
    <module>bridge-jms-su</module>
    <module>bridge-eip-su</module>
    <module>bridge-xslt-su</module>
    <module>bridge-sa</module>
  </modules>

</project>
```

This is what allows all the projects to be built using a single command from the bridge directory. Maven will descend into each subproject directory, figure out the order in which to build each subproject and build them all.

Configure the Service Assembly

We want the JBI Service Assembly to include the four SUs that we just created. So let's edit the *bridge-sa* pom.xml and add the <dependencies> which is a child of <project>:

```
<dependencies>
  <dependency>
    <groupId>org.apache.servicemix.samples.bridge</groupId>
    <artifactId>bridge-http-su</artifactId>
    <version>1.0-SNAPSHOT</version>
  </dependency>
  <dependency>
    <groupId>org.apache.servicemix.samples.bridge</groupId>
    <artifactId>bridge-eip-su</artifactId>
    <version>1.0-SNAPSHOT</version>
  </dependency>
  <dependency>
    <groupId>org.apache.servicemix.samples.bridge</groupId>
    <artifactId>bridge-xslt-su</artifactId>
    <version>1.0-SNAPSHOT</version>
  </dependency>
  <dependency>
    <groupId>org.apache.servicemix.samples.bridge</groupId>
    <artifactId>bridge-jms-su</artifactId>
    <version>1.0-SNAPSHOT</version>
  </dependency>
</dependencies>
```

Now, we should be able to build from the bridge directory and Maven will generate the SA in the bridge-sa/target/ directory. So let's run the following command from the bridge directory:

```
mvn install
```

But this is just to make sure that everything builds correctly. Now we need to go through each subproject and configure each SU.

HTTP SU

We need to define a simple HTTP consumer endpoint in the HTTP SU. As we do not use the request/response pattern, we will use an InOnly MEP. Now you need to edit the src/main/resources/xbean.xml for the HTTP SU to match the following:

```
<?xml version="1.0"?>
<beans xmlns:http="http://servicemix.apache.org/http/1.0"
       xmlns:b="http://servicemix.apache.org/samples/bridge">

  <http:endpoint service="b:http"
                 endpoint="endpoint"
                 targetService="b:pipeline"
                 role="consumer"
                 locationURI="http://localhost:8192/bridge/"
                 defaultMep="http://www.w3.org/2004/08/wsdl/in-only" />

</beans>
```

EIP SU

The EIP SU contains the definition of the pipeline pattern, which can be used as a bridge between two InOnly MEPs and an InOut MEP. The XSLT transformer primary MEP is an InOut, as it will receive an incoming message, transform it, and send the answer back to the consumer. But in our case, the HTTP component and JMS component should send and receive an InOnly MEP, hence the need for the pipeline. Edit the src/main/resources/xbean.xml file for the EIP SU to match the following:

```
<?xml version="1.0"?>
<beans xmlns:eip="http://servicemix.apache.org/eip/1.0"
       xmlns:b="http://servicemix.apache.org/samples/bridge">

  <eip:pipeline service="b:pipeline" endpoint="endpoint">
    <eip:transformer>
      <eip:exchange-target service="b:xslt" />
    </eip:transformer>
    <eip:target>
      <eip:exchange-target service="b:jms" />
    </eip:target>
  </eip:pipeline>

</beans>
```

XSLT SU

The XSLT transformation will be deployed to the servicemix-saxon JBI service engine using the following xbean.xml configuration file:

```
<?xml version="1.0"?>
<beans xmlns:saxon="http://servicemix.apache.org/saxon/1.0"
       xmlns:b="http://servicemix.apache.org/samples/bridge">

  <saxon:xslt service="b:xslt" endpoint="endpoint"
              result="string"
              resource="classpath:bridge.xslt" />

</beans>
```

Note the `classpath:bridge.xsdl` URI used to point to the XSL stylesheet. The classpath is automatically defined by the component and includes the root directory of the SU (see Classloaders).

We need to create the bridge.xslt stylesheet and put it in the src/main/resources directory of this SU, along the xbean.xml configuration file. Let's use the following identity stylesheet:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
  <xsl:template match="node() | @*">
    <xsl:copy>
      <xsl:apply-templates select="node() | @*"/>
    </xsl:copy>
  </xsl:template>
</xsl:stylesheet>
```

Nothing special in the stylesheet above. It just copies all nodes and attributes.

JMS SU

For the JMS provider, we will reuse the embedded ActiveMQ broker which is started with ServiceMix, and send the request in the bridge.output JMS queue:

```
<?xml version="1.0"?>
<beans xmlns:jms="http://servicemix.apache.org/jms/1.0"
       xmlns:b="http://servicemix.apache.org/samples/bridge">

  <jms:endpoint service="b:jms"
                endpoint="endpoint"
                role="provider"
                destinationStyle="queue"
                jmsProviderDestinationName="bridge.output"
                connectionFactory="#connectionFactory" />

  <bean id="connectionFactory" class="org.apache.activemq.ActiveMQConnectionFactory">
    <property name="brokerURL" value="tcp://localhost:61616" />
  </bean>

</beans>
```

Building, Deploying and Testing

To build the SUs and SA, run the following command in the bridge root directory:

```
mvn install
```

Then, deploy the SA and needed components to a started ServiceMix container:

```
cd bridge-sa
mvn jbi:projectDeploy
```

Then you can send an HTML POST request to http://localhost:8192/bridge/ and use a JMX console to check that the **bridge.output** queue contains a JMS message.

Summary

This tutorial has demonstrated how to send a message to a URL via the HTTP protocol and route that message through a XSLT transformation service and on to a JMS queue via the JMS protocol. Although this is a simple demonstration of what ServiceMix can do, it is a good start to illustrate just one use case.

What's Next?

Of course, this example is not very useful as is, especially with an identity XSLT sheet. However, adding an XPath router as the target of the pipeline, could be useful to normalize different requests to the same format, because the XSD changed between versions of the service, or because different clients use different XSDs which can be services by the same service.