

BPEL Tutorial

This document is a brief tutorial on how to get started using NetBeans 5.5 and Sun Java Application Server for the creation of BPEL-based Web services. It shows how to deploy a simple BPEL process as part of a “composite application” (a service-based component interface and implementation) to the application server and how to test the composite service’s interface with JUnit.

The tutorial is based on Sun’s synchronous hello-world BPEL composite application example available at <http://www.netbeans.org/kb/55/helloworldca.html>. I haven’t changed any of the steps, but only added my comments, software installation instructions, and a quick overview of how to use NetBeans.

Introduction

As commercial technology for service-oriented architecture (SOA) matures, open source alternatives are also starting to appear. As we discussed in class, the Web Services Business Process Execution Language (WS-BPEL) provides a standard platform-independent method for specifying Web service orchestrations. There are at least three major open source efforts to enable BPEL-based development:

- Sun’s NetBeans Enterprise Pack (<http://www.netbeans.org/products/enterprise/>) offers a graphical environment for BPEL design and XML editing as well as Java code generation based on the Service Component Architecture (SCA) specification for service components.
- The Eclipse SOA Tools Platform Project (<http://www.eclipse.org/stp>) offers a graphical business process modeling tool that generates BPEL, a BPEL to Java translation engine, and tools and frameworks for deploying services to runtime containers.
- JBoss jBPM (<http://www.jboss.com/products/jbpm>) provides a business process modeling language, an execution engine for the language, visual modeling tools, and support for BPEL.

At the time of this writing (May 2007), the Sun tools are by far the most mature and easy to use. Expect a new release from Eclipse in mid-2007, and jBPM is also under active development. Here we’ll focus on the BPEL and composite application support in NetBeans 5.5. NetBeans is a bit resource hungry but runs quite well on a 2 GHz Pentium 4 with 1 GB of RAM running Ubuntu Edgy Eft (6.10). Its main benefits are close integration with Java EE 5 (JPA, EJB 3.0, etc.) and strong support for Web services development, including asynchronous Web Services via JAX-WS 2.0. With this API, we don’t have to create threads on our own — the library handles it for us.

JDK installation

The Java EE tools require at least JDK 1.5.0 version 9. On Ubuntu, the JDK version Synaptic knows about is version 8, so you have to do a manual install, downloading the self-extracting file `jdk-1_5_0_09-linux-i586.bin` from http://java.sun.com/javase/downloads/index_jdk5.jsp (the Linux self-extracting file format).

To install the JDK on Ubuntu, as root, in `/usr/lib/jvm` directory, unpack the software and make the new version your default:

```
% cd /usr/lib/jvm
% sh jdk-1_5_0_09-linux-i586.bin
% rm java-1.5.0-sun
% ln -s java-1.5.0-sun-1.5.0.09 java-1.5.0-sun
```

You may or may not want to change your `JAVA_HOME` environment variable, depending on what Java applications you use. I changed mine in `~/.bashrc`:

```
export JAVA_HOME=/usr/lib/jvm/java-1.5.0-sun-1.5.0.09
```

NetBeans and NetBeans Enterprise Pack installation

The easiest way to get NetBeans with the Enterprise Pack add on is to install the Sun Java EE 5 Tools Bundle (<http://java.sun.com/javaee/downloads/index.jsp>, select “Download with Tools”). This set includes:

- NetBeans IDE 5.5, Multi-language
- NetBeans Enterprise Pack 5.5, Multi-language
- Sun Java System Application Server Platform Edition 9 Update 1, Multi-language
- Project Open ESB Starter Kit, Multi-language
- Examples and documentation

The download is `java-tools-bundle-linux-ml.sh`. The most convenient setup of NetBeans is in your home directory:

```
% cd ~
% chmod 755 java-tools-bundle-linux-ml.sh
% ./java-tools-bundle-linux-ml.sh
```

I found the defaults suitable except the JDK location had to be explicitly specified:

- JDK home directory: `/usr/lib/jvm/java-1.5.0-sun-1.5.0.09`
- NetBeans install directory: `${HOME}/netbeans5-5`
- Install all of the tools
- Install bundled Java EE SDK update (this includes the application server) and put it in `${HOME}/SUNWappserver`.
- Use default username and password (admin/adminadmin)
- Admin Port 4848, HTTP Port 8080, HTTPS Port 8081.

In the Gnome environment on Ubuntu, NetBeans is automatically added to the Applications → Programming menu and a shortcut is created on the desktop.

Hello world

To get the feel of NetBeans, let’s just do a standard Java hello world program before building our first BPEL module. Fire up the IDE and go to File → New Project → General/Java Application. Give the project a name such as “Hello World,” select “Set as Main Project” and “Create Main Class.” The `.java` file will be created and opened in the source view. Just replace the comment in `main()` with your hello world message:

```
System.out.println("Hello World!");
```

To compile, use Build → Build Main Project, or you can press F11 or click the build icon in the toolbar. You should see a successful compile, then you can run with Run → Run Main Project, or you can press F6 or the run icon in the toolbar.

Synchronous BPEL application

This is a quick version of the full tutorial, “Developing a Hello World Composite Application” at <http://www.netbeans.org/kb/55/helloworldca.html>. If you get stuck or want to see more details, go there. They even have videos you can watch!

Preliminaries

Server runtime configuration: First we need to set up the application server. Click the “Runtime” tab in the left panel and expand “Servers.” If you installed correctly you should see “Sun Java System Application Server 9.” If not, go to the tutorial site just mentioned and follow instructions to install an application server after the fact. Otherwise, right click on the server and select “Start.” It will take a minute or two. You should see the messages “Application server startup complete” and “BPEL service engine started” in the console, and you should see a little green arrow icon next to the server in the “Runtime” tab.

Create your BPEL Module project: Now create your first BPEL project. Go to File → New Project → Service Oriented Architecture → BPEL module. Name the project “SynchronousSample.”

WSDL and XSD

Our BPEL module is going to be invoked through a Web service endpoint that we provide to remote clients. To allow clients to access our service, we have to publish a specification of the service’s interface so that developers of client service consumers can properly format their XML messages and interpret responses.

WSDL (Web Services Description Language) is how we specify the characteristics of a Web service endpoint. A WSDL specification contains

- An abstract description of the Web service interface, containing a list of operations (actions the client can invoke) and corresponding input/output messages;
- A concrete definition, binding operations to specific transport technology such as SOAP.

The input and output messages for our service have to be defined using XSD (XML Schema Definition). We can either *embed* the XSD in our WSDL document or keep it in a separate `.xsd` file and *import* the definition in the WSDL document. NetBeans likes us to keep the XSD separate, so we’ll take the approach of creating a XSD file for our message formats.

See Erl, T. (2005), *Service Oriented Architecture (SOA): Concepts, Technology, and Design*, Prentice-Hall, for more detail on how WSDL and XSD are used to specify Web service endpoints.

XML schema definition

Create the XML schema: To create your XSD document, right click on “Process Files” under the SynchronousSample project, select New → File/Folder → XML → XML Schema. Use the filename “SynchronousSample.”

You will see SynchronousSample.xsd displayed in the Source view, using the Schema viewer tab. You can switch to the Source tab to see the raw XML.

Create a complex type: We will declare a new type called `simpleType` and use it for both our input and output message formats (our service will only have one operation). It will be an XML “complex type” to demonstrate the approach but for simplicity our complex message type will contain just a single string.

Click on the “Design” tab in the editor view and turn on the Palette view (Window → Palette) if it’s not already there. Expand “XML schema components” in the palette and drag a “Complex Type” onto the design view, under the “Complex Types” box. Change the name of the type to “simpleProcess.”

Add a local element to the simpleProcess type: Drag an “Element” from the “XML Components” pane of the palette onto the `simpleProcess` type. This lets us add a new element to the type. Expose the Properties window if not already exposed (Windows → Properties). Select the new element in the design view, go to the properties view, change the element name to `paramA` and the element definition to “Built-in type” `string`.

Add a global element to the schema: Now we create a global element using the `simpleProcess` type. Drag another “Element” from “XML Components” to just under “Elements” (this makes it a global element). Set the element’s name to `typeA` and its definition to the “Complex Type” `simpleProcess`.

Save your work with Ctrl-s or File → Save All.

WSDL definition

Create the WSDL file: Right click on “Process Files” under the SynchronousSample project, select “New WSDL File.” In the WSDL wizard, set the filename to SynchronousSample. To have the XSD previously created imported into your WSDL specification, check the “Import XML Schema” box, click “Browse,” then find SynchronousSample.xsd.

As previously described, a WSDL specification contains an abstract definition and a concrete definition. The WSDL wizard has one screen for each part.

On the “Abstract Configuration,” screen, we want to create an input and output message containing the `typeA` element we created in the XSD. For the input message, set the “Message Part Name” to `inputType` and set the “Element Or Type” to `typeA`. For the output message, set the “Message Part Name” to `resultType` with “Element or Type” `typeA`.

Click “Next” to go to the “Concrete Configuration” screen. You’ll see an error message at first but it will go away. We want the SOAP binding (this is actually the only choice for NetBeans). For the “Binding Subtype,” select “Document Literal” then click Finish. You can view your raw WSDL in the Source view.

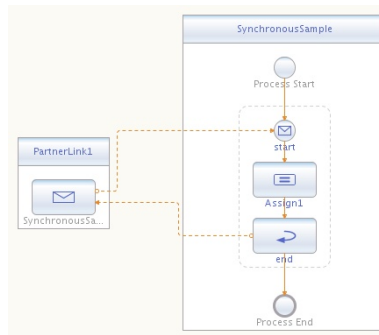
Aren’t you glad there’s a wizard for WSDL file creation?

In any case, the parts of the WSDL should be straightforward to understand now. In the `<types>` element we see that our XSD is being imported. We see two `<message>`s defined, the `SynchronousSampleOperationRequest` message and the `SynchronousSampleOperationReply`. They both have a single part of type `typeA`. We have just the one `<operation>` called `SynchronousSampleOperation` defined — it uses these two messages.

The concrete part of the WSDL specification contains the `<binding>` of our operation to the SOAP transport method and says the messages will be encoded in “`literal`” form. Then it specifies the URI of the provided service.

The BPEL process

Here we’ll create the BPEL process using NetBeans’ nifty graphical editor. The process will look like this once we’re done:



In BPEL, any party the process interacts with is called a *partner*, and a link to any of those parties is called a *partner link*. Partners could be clients that invoke the process or external Web services that the process invokes. There is always at least one partner link, for the client invoking the service. In the diagram above, we see that there is in fact only one partner, the invoking client (called `PartnerLink1` in the diagram). This BPEL process does not invoke any external Web services (in fact its meat is just a trivial assignment of the input message to the output message).

OK, let's define the process.

Create the BPEL process: Right click on "Process Files" and select New → BPEL Process. Set the name to `SynchronousSample` and click "Finish." An initial process flowchart should appear in the Design view.

Create and attach a partner link: Drag from the `SynchronousSample` WSDL file in the Projects window to the BPEL designer window. This is one way to quickly create a partner link (you can also use the "Partner Link" element in the Palette). You'll get a property editor with defaults already set up for a partner link to the invoking client, called "PartnerLink1." This is exactly what we want. Just accept the defaults and click OK.

Now to connect the invoking client partner link to the input of our process, first drag a Receive node from the "Web Services" area of the Palette to the process diagram, just above the Empty node. Double-click the Receive1 node to get its Property Editor, change the name to "start," click the Partner Link pulldown, select "PartnerLink1," click to "Create" an Input Variable, change the name of the input variable to "inputVar," then click OK and close the property editor.

Next, to connect the output of our process to the response expected by the invoking client, drag a Reply node from the Palette to the process diagram, just below the Empty node. Double-click the Reply1 node to get its Property Editor, change the name to "end," set the Partner Link to `PartnerLink1`, select the Normal Response radio button, click to "Create" an Output Variable, change the name to "outputVar," and click OK to accept then close the property editor.

Add an Assign activity: This trivial BPEL process actually will just copy its input message to the output. This is easy to set up in the NetBeans editor. Right click and delete the empty activity in the middle of the diagram. Drag an "Assign Node" from the "Basic Activities" pane in the Palette and place it between the "start" and "end" nodes. If the "BPEL Logical View" window is not showing expose it (Window → BPEL Mapper). In the BPEL mapper pane on the right, expose Variables → `outputVar` → `resultType` → `paramA`, and in the left pane, expose Variables → `inputVar` → `inputType` → `paramA`. Drag from `paramA` on the left to `paramA` on the right. This creates the assignment of input variable to output variable.

Create and Deploy the Composite Application

The Java code for our sample business process needs to be packaged for deployment on the application server.

In Service Component Architecture (SCA), an emerging industry-led open specification for service component creation, deployment, and composition, BPEL modules are deployed inside of so-called "Composite Applications." A composite application is similar in some ways to an EJB.

Here we'll create the composite application that will wrap our BPEL module.

Select File → New Project → Service-Oriented Architecture → Composite Application. Name the application `SynchronousSampleApplication` and click "Finish."

Now we have our composite application. We want to plug in the BPEL module we created into the composite application as a Java Business Integration (JBI) component (JBI includes other types of integration components such as XSLT transformations). This is just a matter of grabbing the JAR file encapsulating our BPEL module.

Right click on the project node and select “Add JBI module.” Find the SynchronousSample project in the list then click “Add Project Jar Files.” Now `SynchronousSample.jar` should appear under the JBI Modules node.

Lastly, right click on SynchronousSampleApplication and select Deploy. If you did everything right, you should see the BPEL module get built, then it will be combined with the deployment information in SynchronousSampleApplication. At the end of the build you should see a success message something like

```
BUILD SUCCESSFUL (total time: 18 seconds)
```

Test the composite application

Yet another nice feature of NetBeans for SOA development is its automated construction of JUnit tests for composite applications. Here we’ll use this functionality to test the BPEL process we wrapped in the SynchronousSampleApplication composite application in the previous step.

First expand the SynchronousSampleApplication node, select “Test,” then right click and select “New Test Case.” Accept the default test case name “TestCase1” and click Next. Now find your WSDL file under SynchronousSample, click Next, select the SynchronousSampleOperation for testing, and click Finish.

You need to specify the input for the test case. NetBeans has already created the shell of a SOAP request message embedding the `<typeA>` element required by the WSDL specification. All you have to do is find the tag `<syn:paramA>?string?<syn:paramA>` and replace the `?string?` placeholder with your input text, e.g., `Hello World`.

If all is well, you’re ready to test. Select File → Save All.

Before you run the test, double click on the Output message for the test case in the Project window and verify that it’s empty. It will get filled in when we run the test case.

Run the test case by right clicking on it and selecting “Run.” The first time you run, you’ll get a message about whether to overwrite the empty `Output.xml` file this time and compare to the original. Select “Yes.”

Run the test case a second time; now the test case should pass since JUnit is comparing the new result to the result we just saved.

Conclusion

NetBeans is the most mature open source tool environment for creating, deploying, and testing BPEL processes with Java EE.

Now you’ve seen how to install NetBeans, use the tool a bit, create a BPEL process as part of a composite application, and test that BPEL process using JUnit.

There are many more examples and tutorials on BPEL and SOA development with NetBeans available at <http://www.netbeans.org/kb/trails/soa.html>.