Plan a PXE System

Every system consists of the same basic ingredients:

- a system deployment descriptor that defines the system in terms of PXE services and their interconnections.
- a resource repository containing the WSDL definitions for the port types to be used in the system along with other resources (e.g., additional WSDL definitions) that services may need to access by URI.
- a root WSDL document that the deployment descriptor will reference by its URI within the resource repository.
- any additional artifacts or resources that one of the services may require at runtime such as compiled BPEL processes.

This tutorial uses the Geolp Example to walk through the process of planning a system. The Geolp BPEL process and system are simplistic but provide a good tradeoff between illustrative features and unnecessary clutter. The system uses a BPEL process as a proxy between a client and a remote service. The WSDL for the remote service is available at online.

Omitting a number of details, the process is:

```xml
<bpws:process>
  <bpws:receive partnerLink="inboundPartnerLink" variable="inVar" createInstance="yes" />
  <bpws:invoke partnerLink="outboundPartnerLink" inputVariable="inVar"
              outputVariable="responseVar" />
  <bpws:reply partnerLink="inboundPartnerLink" variable="responseVar" />
</bpws:process>
```

In the Geolp example, both the client and remote service communicate synchronously over HTTP, so the BPEL process isn’t doing anything interesting in this case. The same process could be used, however, to perform protocol impedance matching between a client that communicates synchronously over HTTP and a remote service that communicates over JMS or over HTTP with a call-back.

Step 1: Lay Out the System

The first step in planning a PXE system is to enumerate the services, ports, and channels needed.

The Geolp example follows a common pattern of a single single BPEL process with its partner links bound to concrete protocols. For this pattern, the system will consist of:

- A service to execute the BPEL process. This means that the BPEL Service Provider will be required in the target deployment domain.
- One or more services to provide communications on the required protocols. The selection of individual service providers will depend both on the requirements of the system and on the target deployment domain. (See below.)
- One imported port on the service for each partner role on each partner link where the BPEL process plays the role of the client.
- One exported port on the service for each partner role on each partner link where the BPEL process plays the role of the server.
- One channel for each port on the BPEL service.

The Geolp process has two partner links with one role each:

```xml
<bpws:partnerLinks>
  <bpws:partnerLink name="inboundPartnerLink" partnerLinkType="gis:GeoIpServiceLinkType"
                    myRole="inbound" />
  <bpws:partnerLink name="outboundPartnerLink" partnerLinkType="gis:GeoIpServiceLinkType"
                    partnerRole="outbound" />
</bpws:partnerLinks>
```

(The port types are drawn from the WSDL for the external service.)

The system will have two channels, and the BPEL service will have one imported port and one exported port:

- One exported port with type `gis:GeoIpServiceLinkType` for the inbound role on the `inboundPartnerLink`.
- One imported port with type `gis:GeoIpServiceLinkType` for the outbound role on the `outboundPartnerLink`.

Step 2: Assess the Target Deployment Domain

PXE services are defined in terms of service providers that are configured at the domain level, so assembling the ingredients for a system involves some level of knowledge about the domain in which it will eventually be deployed. See Anatomy of a System Descriptor and Anatomy of a Domain Configuration for more detailed information.
For present purposes, we’ll assume that the target domain is the standalone server run by the `pxe` command with the configuration specified in `/etc/pxe-config.properties` in the PXE distribution. The domain configuration declares that there are (at least) three kinds of services available:

<table>
<thead>
<tr>
<th>Domain-Specific Provider URI</th>
<th>Service Provider Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>uri:protocoladapter.soap.inbound</code></td>
<td>HTTP-SOAP Protocol Adapter</td>
</tr>
<tr>
<td><code>uri:protocoladapter.soap.outbound</code></td>
<td>HTTP-SOAP Protocol Adapter -- Outbound</td>
</tr>
<tr>
<td><code>uri:bpelProvider</code></td>
<td>BPEL Service Provider</td>
</tr>
</tbody>
</table>

These three provide the functionality required to implement the system, with the inbound HTTP service provider to communicate with the client, the BPEL service provider to execute the process, and the outbound HTTP service provider to communicate with the remote service.

Why have indirection in bindings between services and providers?

The indirection provided by a URI to identify the provider for a given service allows the use of different service providers in different deployment domains. For example, a synthetic endpoint could be used in a lightweight domain for automated unit testing, while the same system deployed in a different domain could use an enhanced endpoint in production deployment to integrate with security and management providers.

**Step 3: Write the Root WSDL Definitions**

The root WSDL definitions document for the system is expected to define the port types for the channels in the system, and it may include additional WSDL or BPEL definitions, depending on the needs of the service providers in the system. In fact, that is the usual recipe for assembling the root WSDL for a system:

1. Create a blank WSDL definitions document.
2. Import the definitions for the external web services that will interact with the system from their respective WSDL definitions.
3. Augment the WSDL with any additional definitions (port types, BPEL partner links, concrete services to be exposed) required for the system.

The root WSDL for the GeoIp system follows this recipe. It imports the WSDL used to define the remote web service, and the port type definitions for the system actually come from the imported WSDL. The root WSDL also includes one additional WSDL service definition, specifically for use by the inbound HTTP service provider.

Use the root WSDL to house both compile-time and runtime definitions.

The BPEL partnerLink definitions in the root WSDL for the GeoIp system are accessed by the BPEL compiler at compile time but then ignored at runtime. Nonetheless, it is convenient (and not detrimental) to work with a single artifact.

**Step 4: Write the System Deployment Descriptor**

The next step is to construct the system deployment descriptor, and this is essentially the task of filling-in the structure required by the XML schema.

For the GeoIp system, the assumption is that the root WSDL will be stored at the URI `file:GeoIp.wsdl`, so the root element of the descriptor is:

```xml

Partner link port type definitions are referenced from the root WSDL for BPEL4WS 1.1 processes.

For BPEL4WS 1.1 processes, the port types referenced by BPEL partner links should be defined either in the root WSDL for the system or in one of its imports. For WS-BPEL 2.0 processes, the `<import>` element draws referenced resources from the root resource repository directly by URI.

The next part of the descriptor declares the two channels with descriptive names:

```xml
<channels>
  <channel name="inboundChannel" />
  <channel name="outboundChannel" />
</channels>
```

**GeoIp System – Inbound HTTP**

The services come next, starting with the inbound HTTP service:
All of the basic service providers use port naming conventions as part of their configuration, and in the case of the **HTTP-SOAP Protocol Adapter**, the name of the PXE service and port should match the name of the WSDL service and port declared in the system WSDL:

```xml
<services>
  <service name="HttpInbound" provider="uri:protocoladapter.soap.inbound">
    <imports>
      <port name="GeoIpSoap" type="wsn:GeoIPServiceSoap" channel-ref="inboundChannel" />
    </imports>
  </service>
</services>
```

The **location** attribute on the `<soap:address>` element is used by the servlet to configure the mapping between the URL of a SOAP request and the target PXE service.

### GeoIp System – Outbound HTTP

The outbound HTTP service needs to be configured with the qualified name of the target WSDL service; this is configured via additional properties passed to the PXE service:

```xml
<services>
  <service name="HttpOutbound" provider="uri:protocoladapter.soap.outbound">
    <properties>
      <property name="serviceWsdlNamespace" value="http://www.webservicex.net" />
      <property name="serviceWsdlName" value="GeoIPService" />
    </properties>
    <exports>
      <port name="GeoIPServiceSoap" type="wsn:GeoIPServiceSoap" channel-ref="outboundChannel" />
    </exports>
  </service>
</services>
```

### GeoIp System – BPEL Execution

The service configuration for the BPEL process is declared next. Of note:

- The naming convention for ports is of the form **partnerLink role**.
- The name of the **compiledProcess property** is the name of the BPEL process (from the `<bpws:process>` element) with **.cbp** appended.

These conventions are explained in the documentation for the BPEL Service Provider.
And that completes the descriptor.

**Building a Deployment Bundle**

The next step is to pull the pieces together into a system deployment archive, and the Build a SAR from the Commandline and Build a SAR with Ant tutorials give two approaches that both pick up where this tutorial leaves off.