10

Service-Based Development Using BPEL

Up until now, we have only seen services implemented in Java. In this chapter, we describe how components can be developed using BPEL. BPEL is very well-suited to SCA component development, because it was explicitly designed for handling long-running conversations and bidirectional interfaces.

BPEL is very well-suited to SCA component development, because it was explicitly designed for handling long-running conversations and bidirectional interfaces.

What Is BPEL?

The first release of BPEL was called Business Process Execution Language for Web Services, with an acronym of BPEL4WS. It has since been changed to WS-BPEL, although people usually just call it BPEL (pronounced beep'uhl).

History

In December of 2000, Microsoft published its proposal for a business process language called XLANG. Four months later, IBM published its proposal, which was called WSFL. The two companies then collaborated to merge the ideas of the two languages into BPEL4People 1.0, which was published in July 2002, as a proposal from IBM, Microsoft, and BEA.

In May 2003, some minor cleanups were done, and version 1.1 was published and submitted for standardization to OASIS. OASIS then worked for the next four years on WS-BPEL 2.0, which was published in April 2007.

The 2.0 version of the specification included improvements to the extensibility of the language, and added some important new features, but most important, it improved the description of the semantics of the language, which is critical to achieving the portability goal of the language. Any significant new projects built with BPEL should be based on version 2.0 of the specification.

A Language for Web Services

If you were going to design a language from scratch that was designed for use with asynchronous web services, there is a good chance you would design something very similar to BPEL.

Ignore, for the time being, the concept of business processes. If you were going to design a language from scratch that was designed for use with asynchronous web services, there is a good chance you would design something very similar to BPEL. The asynchronous qualifier in that statement is critical, but before dealing with that, the features that make it a language for web services include the following:

- Variables and parameters typed by XML Schema
- Operation signatures specified by WSDL
- Expressions and conditionals specified using XPath
- An XML syntax for the language itself

Nonetheless, if it weren't for asynchrony, the language would likely look very different. Two of the asynchronous patterns that BPEL adopts as central patterns are exactly the patterns that SCA has defined for asynchrony, as follows:

- Bidirectional interfaces, which BPEL calls partner link types
- Conversations that are embedded in BPEL's concept of a correlation set

Each of these topics, and how they are used within an SCA environment, will be dealt with in more detail later in the chapter.

Finally, there is the concept of the business process. Actually, the concept supported by BPEL is more precisely described by the term **orchestration**. It is a much more flexible form of control flow than exists in imperative languages, where the language primitives only

support a single thread of control, and calls to special-purpose libraries are needed in order to introduce concurrency (such as tasks in Java). In BPEL (as in any language designed to support workflow patterns), the language supports static and dynamic forking and various thread-joining patterns. **Static forking** is where the number of new threads is known at compilation time and can be represented by transitions in a graph. **Dynamic forking** is where the number of threads is determined only at runtime, usually based on the data being handled by a business process (such as a list of part vendors). By "thread" here, we do not necessarily mean operating system thread, but rather the more general concept of a thread of control. The control flow is best understood and is typically viewed as a graphical representation, such as that shown in Figure 10.1.

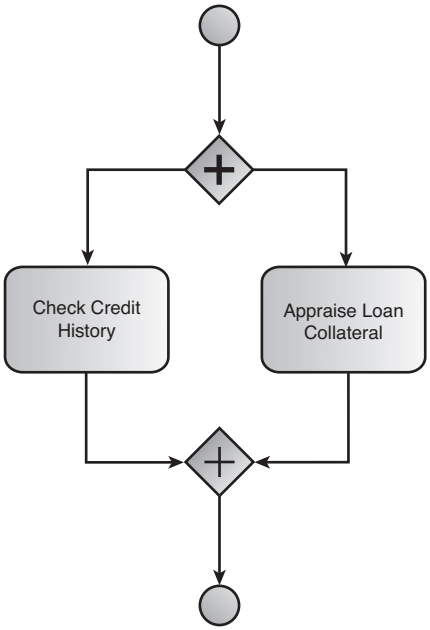

Figure 10.1 Parallel tasks

This representation of control flow is often used to represent business processes, because formal procedures that have been followed by businesses (or any organization) can be easily represented by such graphs. Think of the planning that goes on for building construction. The steps involved in construction are not typically tracked by computers, but if they were, you would need control flow constructs to support various complex forking and joining scenarios in order to accurately represent the work on the ground.

As businesses do more and more of their work as tasks on a computer, keeping track and controlling the flow of work in the business *can* be handled by a computer. This also has the advantage that tasks can change over time from being manual tasks to automated tasks (or vice versa) without having to change the fundamental business process that is controlling everything. Ultimately, this kind of workflow representation remains the best representation, even when *all* the tasks have been automated—especially when the work lasts long enough to require asynchrony (conversations and callbacks).

Using BPEL with SCA

A BPEL process definition can be used as the implementation of an SCA component.

BPEL fits very well into the world of SCA. A BPEL process definition can be used as the implementation of an SCA component. The BPEL partner links become services and references (more on this later), and the interfaces of those services and references are specified using the WSDL port types that make up the BPEL partner link types. SCA's conversational interfaces provide what BPEL refers to as **engine-managed correlation**, which removes the need for developers to specify correlating information explicitly.

What the BPEL specification lacks is exactly where the SCA assembly specification steps in. BPEL provides no mechanism for specifying *what* will provide the services at the other end of the partner links. SCA's wiring fills this need. BPEL also has no way of specifying bindings or policies for the partner links. The SCA binding specifications and policy specification provide for this. Together, the SCA specifications and the BPEL specification provide a complete answer. And because not all services make sense to be developed in BPEL, the SCA Java specifications round out a complete programming model for SOA development.

BPEL Versus Java for Conversational Services

When a Java class is used to implement a conversational service, every operation in the interface is always active. In the conversational loan service introduced in Chapter 4, "Conversational Interactions Using Java," the interface had operations for `apply()`, `getStatus()`, and `cancel()`. There is nothing in that interface or in the code that allows the system to automatically handle situations where messages arrive in an order that makes no sense, such as a `getStatus()` or `cancel()` request that arrives before `apply()`. The developer has to have code that explicitly checks that the operation has been invoked at an appropriate time for the conversation. For more complex conversations, the check is hard to do, and the result is hard to understand, because there is no single place you can go to see a representation of the acceptable sequence. By contrast, take the case in BPEL, where you have the following activities connected in a sequence:

1. Receive X from client.
2. Reply to X.
3. Receive Y from client.
4. Reply to Y.
5. Receive Z from client.
6. Reply to Z.

The acceptable order for requests from the client is clear, and an attempt to send requests in any unexpected order will generate a fault without any code on the part of the service.

Using BPEL for the Loan Service

To get a sense for using BPEL to implement a component, we will replace the Java implementation of the loan service from the application introduced in Chapter 4 with a BPEL implementation of that service.

BPEL does not define a graphical representation for processes, although the expectation has always been that a graphical representation would be the most common way for developers to work with these processes. However, the OASIS technical committee that standardized BPEL was not chartered to standardize such a representation. One common notation for business processes is the

Business Process Modeling Notation (BPMN). Its concepts don't align perfectly with BPEL, because it has a number of constructs with no equivalent in BPEL, and BPEL has constructs without a representation in BPMN. Nonetheless, it is close enough to represent the basic control flow of the process.

For the sake of this example, imagine that the process for handling new loan applications is a little bit more complicated than it was in Chapter 4; let's create a loan-approval process based on the loan-application process used as an example at the end of the BPEL specification. In this process, if the loan amount is less than some designated amount (say $10,000), a call is made to a risk assessment service, which tries to determine whether the risk is low enough to immediately approve it, or high enough to immediately deny it. If, however, the amount is larger than $10,000 or if the automatic risk assessment service can't make a clean determination, the full loan review is initiated. The process might look like that shown in Figure 10.2.

Unfortunately, BPMN has no representation of the partner links behind the communication activities. Listing 10.1 is some of the partner link declaration section of that process.

Listing 10.1 Partner Links for the Loan Application Process

```
<partnerLinks>
   <partnerLink name="customer"
      partnerLinkType="lns:loanPartnerLT"
      myRole="loanService" />
   <partnerLink name="approver"
      partnerLinkType="lns:loanApprovalLT"
      partnerRole="approver" />
   <partnerLink name="assessor"
      partnerLinkType="lns:riskAssessmentLT"
      partnerRole="assessor" />
</partnerLinks>
```

All the partner links in this process have only a single role. The partner link with `myRole="loanService"` is the only service offered by the process. The other two (named `"approver"` and `"assessor"`) are references. The algorithm for the mapping of partner links to services and references is described in the next section.

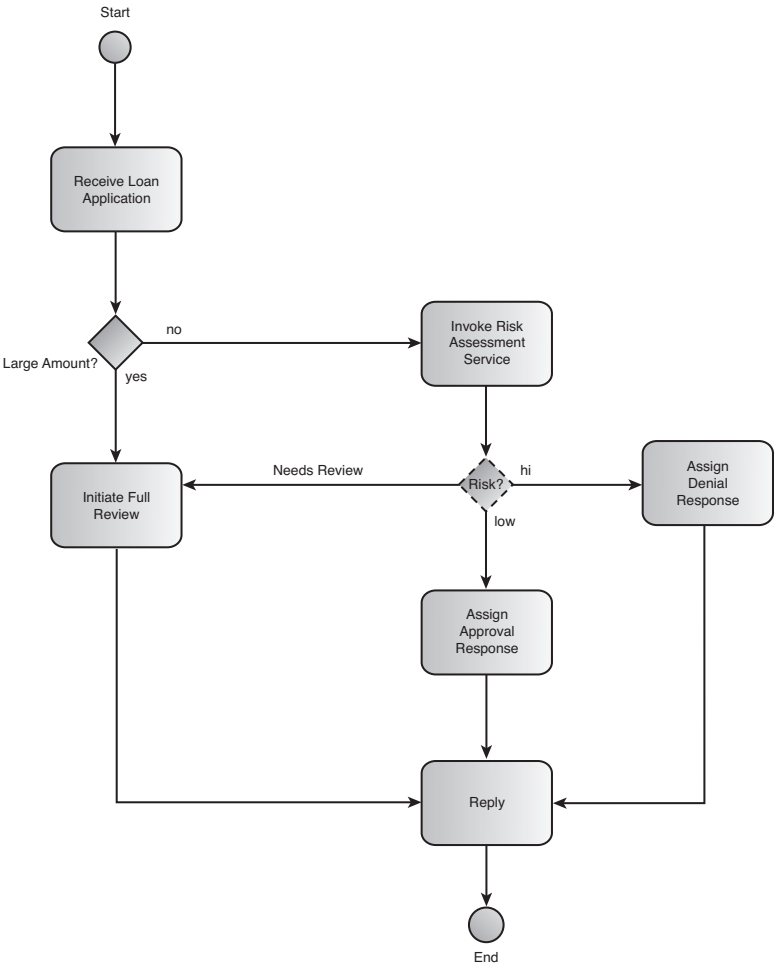

Figure 10.2 The loan application process

Because this implementation makes use of an approver and an assessor, our assembly needs a couple of additional components that existed in the composite described in Chapter 4. The new composite would look like what is shown in Figure 10.3.

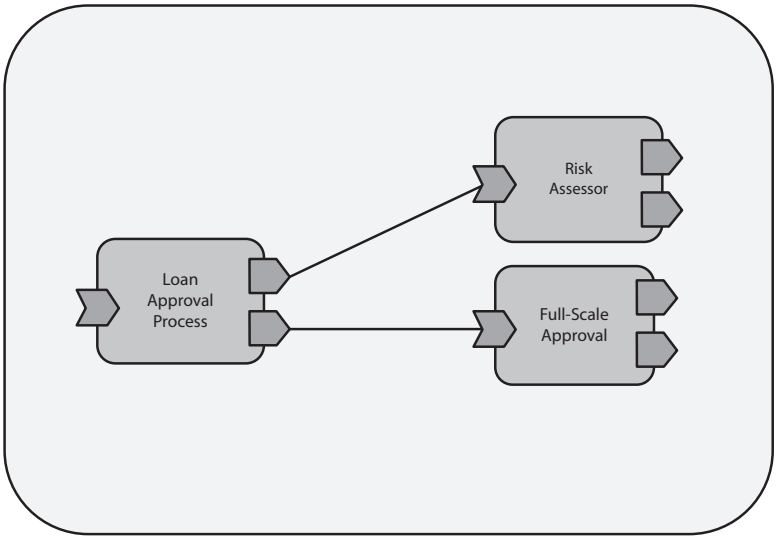

Figure 10.3 Loan application components

Partner Links Are Services and References

BPEL identifies the external services that it communicates with through ***partner links****, and like SCA services and references, they can be bidirectional.*

BPEL identifies the external services that it communicates with through **partner links**, and like SCA services and references, they can be bidirectional. The client may need to provide services that can be used by the process for callbacks. A partner link's type is specified by a **partner link type**, which consists of either one or two port types (two if it is bidirectional).

Symmetry of Partner Link Types

There is one difference between SCA's concept of bidirectional interfaces and BPEL's partner link types, which is that partner link types are symmetric. An SCA bidirectional interface is made of an **interface** and a **callback interface**. There is an implication that the service provider is providing the interface and the client is providing the callback interface. However, BPEL makes no such distinction. In fact, BPEL does not refer to the two roles as "service provider" and "client." The roles can have any name, and there is nothing else to distinguish one role as more important than the other.

So, what *is* the difference between a "client" and a "service provider?" With bidirectional interfaces, both roles provide services that can be used by the other role. The only difference is that the "client" sends the first message of the conversation. The direction of that first message provides the asymmetry, and then after that, there are no constraints on the messages that might be sent between the two roles.

In BPEL, even that distinction is not made. It is theoretically possible that a partner link type could be created where either role could initiate the conversation. In that case, there would be no asymmetry at all. However, although that may make the model more elegant and theoretically more powerful, in fact, almost every conversational service is designed where it is known in advance which role will send the first message. The distinction between a client and a service provider exists in the mind of the developer, so it makes sense to recognize it in the programming model.

Static Control Flow Analysis with SCA BPEL

When you use a BPEL process definition as the implementation of a component, SCA needs to be able to tell which of the partner links should be represented as services and which should be represented as references. If the partner link type has only one role, the choice is easy: The partner links with the one role as "myRole" are services and the partner links with the one role as "partnerRole" are references.

When the partner link type has two roles (that is, it is bidirectional), SCA determines which are services and which are references by depending on the fundamental asymmetry between clients and service providers described previously—figure out which role will send the first message of the conversation. The SCA BPEL specification says that this is accomplished by static control flow analysis of the BPEL process.

SCA determines which are services and which are references by depending on the fundamental asymmetry between clients and service providers.

Partner links are used for either inbound or outbound communication. Inbound communication occurs as either a receive, onMessage, or pick activity. Outbound communication is done with either an invoke or a reply activity. The order in which the activities can occur is constrained by the sequence of activities they

are in and by any links present in the process. The SCA processor can analyze each use of a partner link and determine whether the first activity for the partner link is inbound or outbound communication. If it is inbound (for example, receive), the partner link is turned into a service. If it is outbound (for example, invoke), it is turned into a reference.

According to SCA, the static analysis is not allowed to try to guess the path taken through any condition. (Although in most cases, it would be impossible anyway, but the specification disallows it so that different processors don't come up with different answers.)

In the rare case that it can't be determined which will occur first, the partner link is turned into a reference. This can happen when a receive and an invoke activity for the same partner link are in the same flow, and there is no link that causes one of the two to occur before the other.

Usually, the programmer does not need to think about any of this. The partner links that the developer thinks of as services become services, and the partner links that the developer thinks of as references become references.

Partner Link Types as Interfaces

In SCA, a bidirectional interface is defined in Java with an `@Callback` annotation, as we saw back in Chapter 3, "Service-Based Development Using Java" (see Listing 10.2).

Listing 10.2 Bidirectional Interface in Java

```
@Callback(CreditCallback.class)
public interface CreditService { ...  }
```

To expose the service using WSDL, port types that correspond to the Java interfaces would be created (by hand or using JAX-WS to generate them), and the component type would identify the bidirectional interface of the service by specifying both the service provider interface and the callback interface. The component type would look like Listing 10.3.

Listing 10.3 Bidirectional Interface Using WSDL

```
<componentType xmlns="http://www.osoa.org/xmlns/sca/1.0"
  <service name="CreditService">
    <interface.wsdl
interface="http://www.bigbank.com/loanapplication#wsdl.interface(Credit
➥Service)"
callbackInterface="http://www.bigbank.com/loanapplication#wsdl.
➥interface(CreditCallback)"/>
```

BPEL requires that the pairing of an interface and its corresponding callback interface be specified in a partner link type. The partner link type name is then used as the interface name. This is appropriate because the two interfaces are not really independent. They were designed to be used together, so it is appropriate that there be a name for the combination.

In BPEL, the partner link type for this bidirectional interface would be in the WSDL file (usually with the corresponding port types) and would look like Listing 10.4.

Listing 10.4 Bidirectional Partner Link Type Definition

```
<partnerLinkType name="CreditServicePLT">
  <role name="creditBureau">
    <portType="bb:CreditService">
  </role>
  <role name="creditRequestor">
    <portType="bb:CreditCallback">
  </role>
</partnerLinkType>
```

The SCA BPEL specification then states that the partner link type can be used in the component type as an alternative to the typical way of specifying bidirectional interfaces. This is done by using `<interface.partnerLinkType>` instead of `<interface.wsdl>`. Because the partner link type is symmetric, you must also specify the name of the role that is provided by the service, as shown in Listing 10.5.

The partner link type can be used in the component type as an alternative to the typical way of specifying bidirectional interfaces.

Listing 10.5 Bidirectional Interface Using a Partner Link Type

```
<componentType xmlns="http://www.osoa.org/xmlns/sca/1.0"
  <service name="CreditService">
    <interface.partnerLinkType type="bb:CreditServicePLT"
      serviceRole="creditBureau"/>
  </service>
  ...
```

SCA Extensions to BPEL

Up until now, we have seen that standard BPEL processes may be used without using any extensions, APIs, or standardized services from SCA. The business process does not need to have any reference to SCA in it at all.

There are a few capabilities that SCA provides for BPEL processes that are only available by using the SCA extension to BPEL.

However, there are a few capabilities that SCA provides for BPEL processes that are available only by using the SCA extension to BPEL. Any BPEL engine that is working within an SCA domain should understand these extensions, although careful thought should be given before they are used, because they will cause the process to be unable to run in a BPEL engine that is not running in an SCA domain.

Nonetheless, BPEL does provide a mechanism for adding extensions. SCA's extension is declared at the beginning of any process that uses it (see Listing 10.6).

Listing 10.6 The SCA Extension Declaration for BPEL

```
<extensions>
  <extension
   namespace="http://docs.oasis-open.org/ns/opencsa/sca-bpel/200801"
   mustUnderstand="yes" />
</extensions>
```

The extension is marked with `mustUnderstand="yes"` because most of the extensions affect the semantics of the process, and an engine that did not understand the extensions would generate results different from what would have been desired by the developer.

SCA Properties

Properties are at least as valuable for BPEL processes as they are for any other implementation type.

BPEL has no capability to provide data that can be set by a deployer and used by the process. In other words, there is no equivalent to SCA's concept of properties. Nonetheless, properties are at least as valuable for BPEL processes as they are for any other implementation type.

Consider the loan application process again. In that process, $10,000 was hard coded into the process in the condition for the branch that determined whether to attempt automatic loan approval based on risk assessment rules. Rather than have that number hard coded into the process, the value should be represented as a property that can be set at the place where the process is used.

SCA's extension for declaring properties provides an attribute that can be added to a variable declaration to also designate the variable as a property. If the cutoff amount for automatic processing were a property, it would be defined as shown in Listing 10.7.

Listing 10.7 An SCA Property as a BPEL Variable

```
<variable name="cutoffAmount" type="xsd:integer"
        sca-bpel:property="yes">
  <from>
    <literal>10000</literal>
  </from>
</variable>
```

The variable's initialization value will be used as the value of the variable if the property is not set in the composite file that instantiates this process. If the property is set, the process uses that property value rather than the value in the initialization expression. Just in case the initialization expression has some side effects (which would be bad practice), the SCA BPEL specification requires that the expression be evaluated before its result is ignored and the SCA property value is used instead. Any subsequent property initialization expressions that access the variable should see the SCA property value, so the replacement can't wait until after all the initialization expressions have been evaluated.

Customizing the Generated Services and References

The developer may not like the way that SCA generates service and reference definitions for a BPEL process. The automatically generated component type uses the partner link names as the names of the services and references. Because these names are not guaranteed to be unique for the process (they are only unique within a single scope), the automatically generated name might need to include a disambiguation digit at the end of the name. For example, it might have to generate "myService1" and "myService2" as service names for two partner links named "myService." If this happens, the developer may want to choose better names. The developer may also want different names if there is a convention, such as ending partner links with "PL," which he does not want to have exposed in the service or reference names.

To customize the generated name, the developer can include an attribute from the SCA BPEL extension that explicitly specifies the name. This looks like Listing 10.8.

Listing 10.8 Partner Link with Customized SCA Service Name

```
<partnerLink name="CreditServicePL"
partnerLinkType="bb:CreditServicePLT" myRole="creditBureau"
sca:service="CreditService"/>
```

Because it is theoretically possible for the static analysis to come up with the wrong choice, when determining whether a partner link should be a service or a reference, this mechanism can also be used to force a partner link that would otherwise have been turned into a service into a reference, and vice versa.

References with Multiplicity

A partner link is used to communicate with a single service. There is nothing in BPEL that corresponds to SCA's concept of a reference that has a multiplicity of "0..n" (which we will refer to as a **multi-reference**). Nonetheless, as with properties, this is a useful concept that should be made available to developers. Also, SCA should be able to be used with a top-down development style, where an architect designs the components and the wires without knowing what implementation language will be used for each of the components. If such an architect included any multivalued references, it should still be possible to implement that component using BPEL.

Recall that in SCA, a multireference does *not* mean that outbound messages on the reference are automatically broadcast to all the targets. Instead, it just means that all the targets are somehow presented to the component developer to do with as she wants. She may send requests to every target in parallel, to every target sequentially, to a subset of the targets, or to just one of the targets. How the list of targets is presented to the component developer is up to the programming language.

In BPEL, most references are represented as partner links. However, there is no obvious way for SCA to extend BPEL links so that they instead represent a list of targets, instead of only one. Instead, SCA has to represent the list of targets as the contents of a variable and depend on the developer using BPEL's capability to set partner link targets at runtime.

Multiplicity Example with Credit Bureaus

To see the value of multiplicity in the context of our example, imagine that the process is modified so that a credit check is made with a list of credit bureaus before the rest of the loan approval process is done by the bank. We would want the list of credit bureaus to be expandable, rather than hardcoded into the process.

We will modify our process by adding a new activity immediately after the initial receive activity for getting credit scores from credit bureaus. The fact that it is done for multiple credit bureaus is represented in BPMN by the parallel bars at the bottom of the new activity (see Figure 10.4).

The BPEL representation of this is significantly more complicated. First, we need the variable that will hold the list of targets for the multi-reference that will represent the credit bureaus. At assembly time, the reference will be wired to services for each of the credit bureaus to be used. The variable that holds these targets looks like Listing 10.9.

Listing 10.9 Partner Link with Customized SCA Service Name

```
<variable name="bureaus" element="sca-bpel:serviceReferenceList">
  <sca-bpel:multiReference partnerLinkType="pos:CreditBureauPT"
    partnerRole="bureau" />
</variable>
```

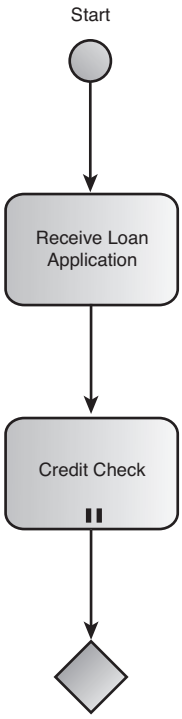

Figure 10.4 The new credit check activity

This causes the `"bureaus"` variable to hold endpoint references for each service that the multireference is wired to within the composite file. The child element of the variable declaration is used to specify the type of the reference, which may be either a single-direction interface or a bidirectional interface, specified using a `partnerLinkType` and a `partnerRole`; this is analogous to the way single-valued references are typed.

At runtime, the contents of the `"bureaus"` variable will be a document that looks like Listing 10.10.

Listing 10.10 A ServiceReferenceList with Two Endpoints

```
<<sr:serviceReferenceList
  xmlns:sr="http://docs.oasis-open.org/wsbpel/2.0/serviceref"
  xmlns:wsa="http://schemas.xmlsoap.org/ws/2003/03/addressing">
  <sr:serviceRef>
    <wsa:EndpointReference>
      <wsa:Address>http://creditBureau1.com/svc</wsa:Address>
```

```
    </wsa:EndpointReference>
  </sr:serviceRef>
  <sr:serviceRef>
    <wsa:EndpointReference>
      <wsa:Address>http://creditBureau2.com/svc</wsa:Address>
    </wsa:EndpointReference>
  </sr:serviceRef>
</sr:serviceReferenceList>
```

Next is the BPEL representation of a parallel call to each credit bureau. We take advantage of BPEL's parallel "for-each" activity to accomplish this. It looks like Listing 10.11.

Listing 10.11 Accessing All (0..n) Reference Targets in Parallel

```
<forEach counterName="idx">
  <startCounterValue>1</startCounterValue>
  <finalCounterValue>
     count($bureaus/sref:service-ref)
  </finalCounterValue>
  <scope>
     <partnerLinks>
       <partnerLink name="bureauLink"
          partnerLinkType="pos:CreditBureauPT"
          partnerRole="bureau"
          sca-bpel:multiRefFrom="bureaus" />
     </partnerLinks>
     <sequence>
       <assign>
          <copy>
             <from>$bureaus/sref:service-ref[$idx]</from>
             <to partnerLink="bureauLink"/>
          </copy>
       </assign>
       <invoke partnerLink="bureauLink"
           operation="getCreditReport"
           inputVariable="applicantSSN"
           outputVariable="creditReport">
       </invoke>
       ... do something with the credit report
     </sequence>
   </scope>
 </forEach>
```

In this code sample, you can see that the `forEach` loop variable only maintains the variable for the index. If there are three credit bureaus in our list (that is, there were three wires), the `forEach` will initiate three parallel branches of control flow. Within each scope, a local partner link is defined, and that partner link is assigned one

of the targets (the bold part of the preceding code). This means that in this example, three scopes will be simultaneously started, each scope will have its own partner link, and each partner link will be assigned to one of the three reference targets.

When each `invoke` completes, it now has the credit report for the applicant from one of the three credit bureaus. The next thing to do would be to add that credit report to a list of credit reports that will be given to the reviewers that come later in the process. (This part of the code was not shown in the preceding sample.)

The entire `forEach` will complete when the scope of all three of the initiated scopes have been completed. The rest of the process can then run.

Summary

BPEL is a language that was designed for orchestrating web services. It is easy to use standard BPEL process definitions as the implementation of components within SCA. It is also possible to use SCA extensions to standard BPEL to take advantage of SCA-specific features, such as properties and multireferences, or to customize things like the names of the service and references that are generated for a process definition.