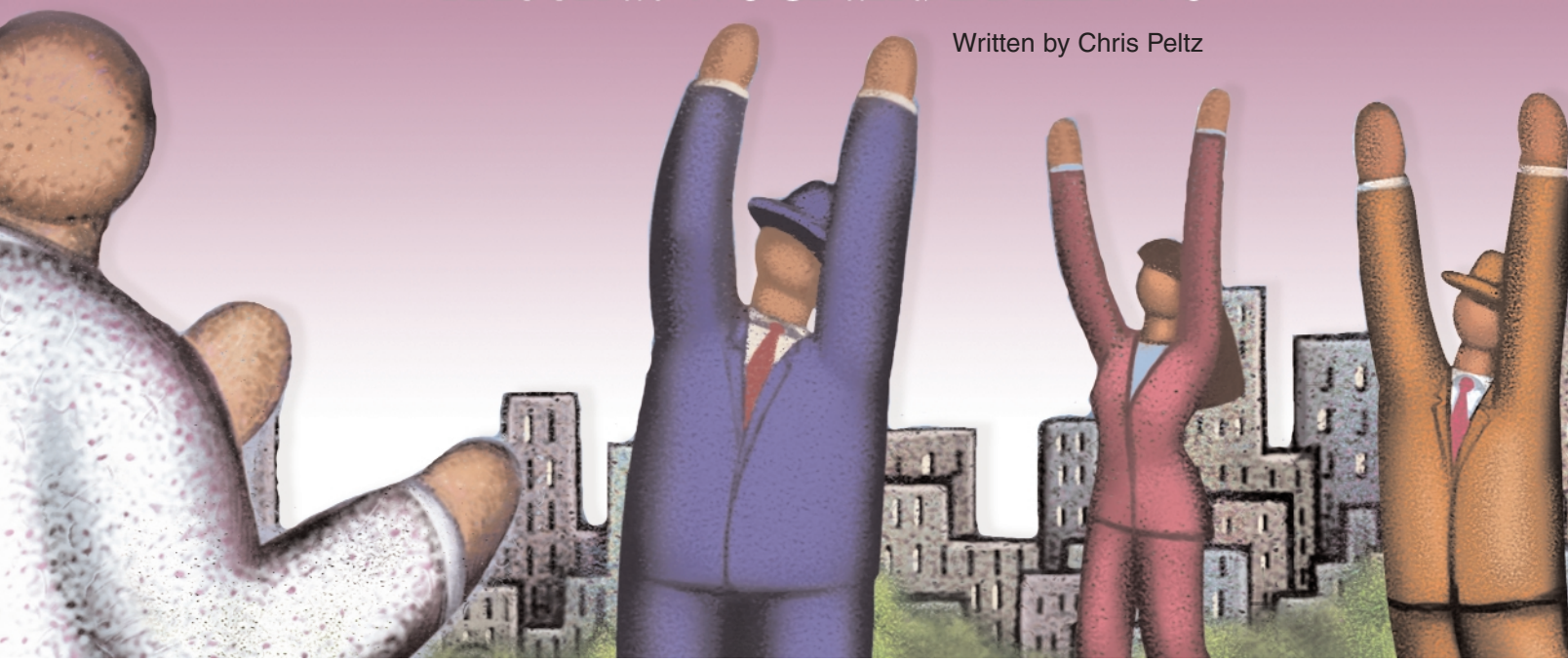
Web Service Orchestration and Choreography

A look at WSCI and BPEL4WS

Written by Chris Peltz



Web services are rapidly emerging as the most practical approach for integrating a wide array of customer, vendor, and business-partner applications. While many companies have begun to deploy individual Web services, the real value will come when enterprises can connect services together, providing higher value to an organization.

Early experience shows that to make the most of new Web services investments there must be a standard approach to Web services composition.

AUTHOR BIO:

Chris Peltz is a senior software consultant in HP's Developer Resources Organization (devresource.hp.com), providing technical consulting on J2EE and Web services architectures. He brings over 10 years of software development experience in helping customers select technologies, tools, and platforms for building enterprise applications.
CHRIS.PELTZ@HP.COM

IT organizations need the agility to adapt to customer requirements and changing market conditions. But existing business process languages do not directly support Web services standards and, as a result, IT organizations may be tempted to take a short-term approach and create their own proprietary protocols for composing services together. Web services orchestration and choreography standards are efforts that can be long-term solutions for business connectivity. By connecting services through open, standards-based methods, organizations spare themselves the burden of maintaining these proprietary interfaces.

The two standards discussed here – the Web Service Choreography Interface (WSCI) and Business Process Execution Language for Web Services (BPEL4WS) – are designed to reduce the inherent complexity of connecting Web services together. Without them, an organization is left to build proprietary business protocols that shortchange true Web services collaboration. Recently, the terms orchestration and choreography have been employed to describe this collaboration:

- ***Orchestration:*** Refers to an executable business process that may interact with both internal and external Web services. Orchestration describes how Web services can interact at the message level, including the business logic and execution order of the interactions. These interactions may span applications and/or organizations, and result in a long-lived, transactional process. With orchestration, the process is always controlled from the perspective of one of the business parties.
- ***Choreography:*** More collaborative in nature, where each party involved in the process describes the part they play in the interaction. Choreography tracks the sequence of messages that may involve multiple parties and multiple sources. It is associated with the public message exchanges that occur between multiple Web services.

Orchestration differs from choreography in that it describes a process flow between services, controlled by a single party. More collaborative in nature (see Figure 1), choreography tracks the sequence of messages involving multiple parties, where no one party truly "owns" the conversation.

In this article, I'll highlight key technical

requirements for Web services orchestration and choreography, and point out key standards used to meet these needs.

Technical Requirements for Orchestration and Choreography

Before introducing the standards, it's important to define the technical requirements for orchestrating Web services. The following requirements are important for both the language and the underlying infrastructure that supports it:

- ***Flexibility:*** One of the most important considerations is the flexibility offered by the language. Flexibility can be achieved by providing a clear separation between the process logic and the Web services invoked. This separation can usually be achieved through an orchestration engine that handles the overall process flow. With this flexibility, an organization can easily swap out services as business needs change.
- ***Basic and structured activities:*** An orchestration language must support activities for both communicating with other Web services and handling workflow semantics. One can think of a basic activity as a component that interacts with something external to the process itself. In contrast, structured activities manage the overall process flow, specifying what activities should run and in what order.
- ***Recursive composition:*** A single business process can interact with multiple Web services. However, a business process can itself be exposed as a Web service, enabling business processes to be aggregated to form higher-level processes.

In addition, both Web services orchestration and choreography must support some basic requirements for managing the overall integrity and consistency of the interactions. These requirements include:

- ***Persistence and correlation:*** The ability to maintain state across Web services requests is an important requirement, especially when dealing with asynchronous Web services. The language and infrastructure should provide a mechanism to manage data persistence and correlate requests in order to build higher-level conversations.
- ***Exception handling and transactions:*** Orchestrated Web services that are long-running must also manage exceptions and transactional integrity. For example, resources cannot be locked in a transaction that runs over a long period of time.

WSCI

WSCI defines an extension to WSDL for Web services collaboration. Initially authored by Sun, SAP, BEA, and Intalio, it was recently published as a W3C note. WSCI is a choreography language that describes the messages exchanged between Web services that participate in a collaborative exchange. A key aspect of WSCI is that it describes only the observable behavior between Web services. It does not address the definition of an executable business process.

A single WSCI interface describes only one partner's participation in a message exchange. As Figure 2 illustrates, a WSCI choreography would include a set of WSCI interfaces, one for each partner in the interaction. In WSCI, there is no single controlling process managing the interaction.

WSCI can be viewed as a layer on top of the existing Web services stack. Each action in WSCI represents a unit of work, which typically would map to a specific WSDL operation. WSCI can be thought of as an extension to WSDL, describing how the operations can be choreographed. In other words, WSDL describes the entry points for each service, while WSCI would describe the interactions among these WSDL operations.

WSCI defines an <action> tag for specifying a basic request or response message. Each activity specifies the WSDL operation involved and the role being played by the participant. External services can then be invoked through the <call> tag. A wide variety of structured activities are supported, including sequential and parallel processing and condition looping. WSCI also introduces an <all> activity, used to indicate that the specific actions have to be performed, but not in any particular order.

Listing 1 is a simple example of WSCI. In this example, a purchasing interface is created containing two activities, "Receive Order" and "Confirm". Note that this is the WSCI document from the perspective of the agent. There would also be a WSCI interface for the buyer and the supplier in the interaction.

BPEL4WS

The BPEL4WS standard represents a convergence of ideas originally proposed by two early workflow languages, XLANG and WSFL. Microsoft, IBM, Siebel Systems, BEA, and SAP authored the 1.1 release of the specification in May 2003. It provides an XML-based grammar for describing the control logic required to coordinate Web services participating in a process flow and is layered on top of WSDL, with BPEL4WS defining how the WSDL operations should be sequenced.

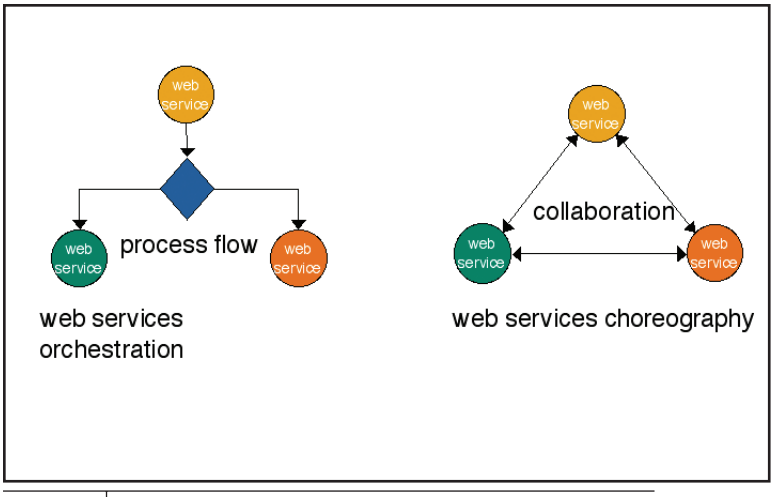


FIGURE 1 Orchestration and choreography

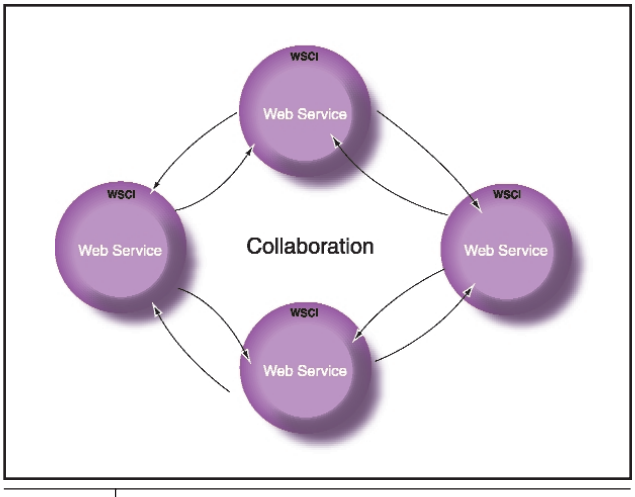


FIGURE 2 Web Services Choreography Interface (WSCI)

BPEL4WS provides support for both abstract business protocols and executable business processes. A BPEL4WS business protocol specifies the public message exchanges between parties. Business protocols are not executable and do not convey the internal details of a process flow, similar to WSCI. An executable process models the behavior of participants in a specific business interaction, essentially modeling a private workflow. Executable processes provide the orchestration support described earlier, while the business protocols focus more on Web services choreography.

The BPEL4WS specification supports basic activities for communicating with Web services. The typical scenario is that there is a message received into a BPEL4WS executable process. The process may then invoke a series of external services to gather additional data, and then respond back to the requestor. In Figure 3, the <receive>, <reply>, and <invoke> messages all represent basic activities for connecting the services together.

BPEL4WS also supports structured activities for constructing the business logic for a process. These activities include sequential and parallel activities, as well support for conditional looping and dynamic branching. Listing 2 is a simple illustration of how a sequential activity would be described.

Variables and partners are two other important elements within BPEL4WS that satisfy the requirements for persistence and correlation.

- ***Variable:*** Identifies the specific data exchanged in a message flow, which typically maps to a WSDL message type or XML schema type. When a BPEL4WS process receives a message, the appropriate variable is populated so that subsequent requests can access the data.
- ***Partner:*** Defines the various parties that interact with the process.

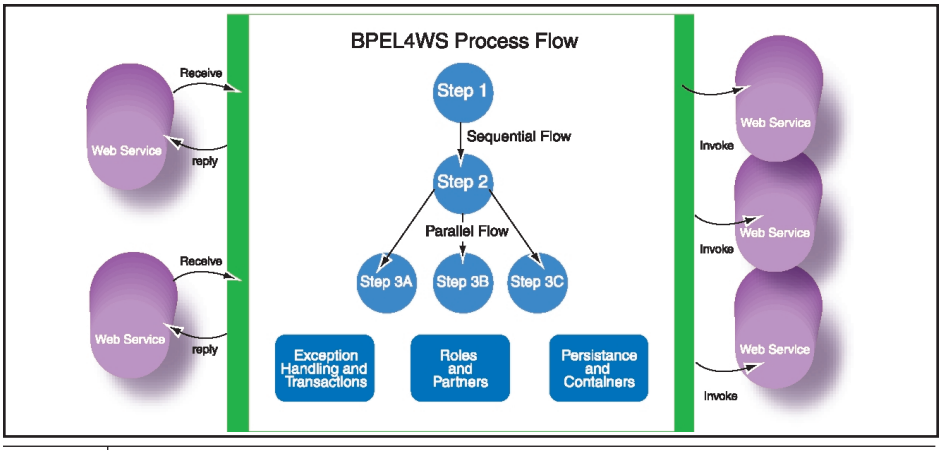


FIGURE 3 BPEL4WS Process Flow

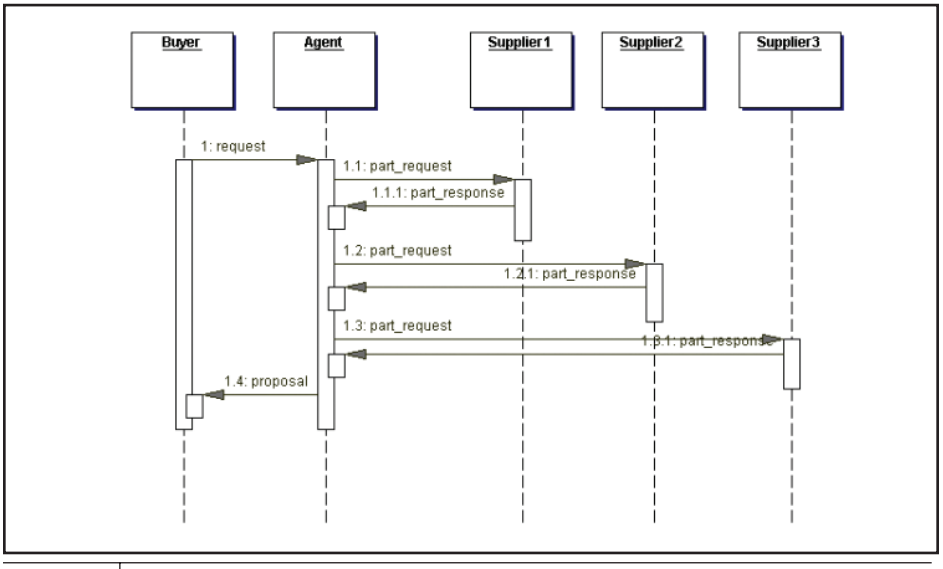


FIGURE 4 Case study sequence diagram

Comparing WSCI and BPEL4WS

Each standard takes a somewhat different approach to orchestration and choreography. While BPEL4WS supports the notion of "abstract processes," most of its focus is aimed at BPEL4WS executable processes. BPEL4WS takes more of an "inside-out" perspective, describing an executable process from the perspective of one of the partners. WSCI takes more of a collaborative and choreographed approach, requiring each participant in the message exchange to define a WSCI interface.

At the same time, WSCI and BPEL4WS both meet many of the technical requirements outlined earlier. They both provide strong support for persistence and correlation to manage conversations. WSCI and BPEL4WS also describe how exceptions and transactions should be managed. From a usability standpoint, WSCI does have a somewhat "cleaner" interface than BPEL4WS. Some of the difficulties in using BPEL4WS are attributed to the fact that the language includes artifacts from both XLANG and WSFL, each of which took a different approach to workflow.

It's also important to look at overall industry acceptance for each standard. BPEL4WS has a number of major supporters behind it, including IBM, Microsoft, and BEA. Moreover, the companies submitted BPEL4WS to OASIS in April 2003, further broadening its support. Sun, Intalio, and SAP initially submitted the WSCI specification to the W3C, which recently created a WS-Choreography working group to standardize on Web services choreography. While the OASIS BPEL technical committee focuses on standardizing the BPEL4WS specification, WS-Choreography will be defining a choreography language.

The vendor specifications have quickly moved into a number of product implementations. Vendors such as Intalio and Vergil Technologies have products that implement BPML (Business Process Markup Language), which incorporates WSCI. Sun also provides the Sun ONE WSCI editor, which supports the WSCI extensions to WSDL. Vendors supporting or planning to support the BPEL4WS specification include:

- ***Collaxa:*** Offers a complete orchestration platform for BPEL4WS

- ***IBM:*** Provides a BPWS4J runtime/editor for BPEL4WS from their alphaWorks Web site
- ***BindSystems:*** Provides a BPEL4WS modeling/editing tool
- ***Microsoft, BEA, and other vendors:*** Announced they will support BPEL4WS in their products

While the industry appears to be embracing the BPEL4WS initiative, it is still unclear what part WSCI and the W3C Web services choreography working group will play. Clearly, vendor backing and tools support will influence the adoption taken by the software industry.

Case Study

To illustrate some of the capabilities outlined here, let's look at how Web services orchestration and BPEL4WS can solve a typical use-case scenario. It revolves around a purchasing system where a PC manufacturer wishes to build a set of PC machine configurations using a list of available suppliers. In the process, a buyer works through a purchasing agent to fulfill these inventory requests. The purchasing agent then communicates with a number of suppliers, each offering specific components required to build the PC configuration. Once a complete configuration can be built across one or multiple suppliers, a proposal is constructed and sent back to the buyer. The buyer then has the opportunity to place the parts order or cancel the request. Figure 4 shows a simplified view of the process. It shows the initial request from the buyer to the agent, with subsequent requests to each supplier.

Each partner in the process has a WSDL describing the specific input and output interfaces that are being exposed. This example will demonstrate the workflow that is built from the perspective of the purchasing agent, as well as the public interfaces exposed by this workflow.

The first step in creating the BPEL4WS document is to define the process itself. This starts with a <process> tag at the root level. This tag provides a name for the process and lists specific references to XML namespaces used. This is where any WSDL references are placed in the BPEL4WS document. In this example, the xmlns:po (http://acme-manufacturing.com/purchaseorder) will be used to refer to the WSDL definitions.

The next step is to define the specific parties involved in the process. In this example, there are three basic roles: (1) the buyer making the purchase; (2) the purchasing agent working on behalf of the buyer; and (3) a set of suppliers offering computer parts. This is supported in BPEL4WS through the <partnerLinks> and <partner Link> tags.

In Listing 3, a "Buyer" partner link is defined between the buyer (requestor) and the purchasing agent (purchaser). The partnerLinkType (po:requestQuote LinkType) is a reference to a <partner LinkType> tag defined within the WSDL document (see Listing 4).

The partnerLinkType defines the dependencies between the services and the WSDL port types that are used. We will assume there is a WSDL port type defining a request_quote operation that is initiated from the buyer. The purchasing agent will also have a link to the supplier for requesting a quote for a single part.

The process must also manage the flow of information between the partners, modeled as variables within BPEL4WS. In this scenario, a buyer makes the initial request with a configuration number and quantity to purchase, and the agent then constructs individual quote requests to each supplier with a part number and quantity. The requests come back from the supplier with the pricing information. The purchasing agent then constructs a proposal back to the buyer. Here, there are potentially four variables required to model this interaction, two for each request/response interaction. Each variable is declared with a name, followed by a reference to a WSDL message type (see Listing 5).

In this process, there must be a way to correlate the message requests to each other. For example, there might be a unique identifier for the quote that is received back from a supplier. The WSDL document would first define a correlation property for this quoteID, which would then be referenced within the BPEL4WS process. Listing 6 highlights how the correlation property is described in the WSDL document.

A key part of the BPEL4WS document is the definition of the steps required to handle the request. This is where basic and structured activities are used. The process flow consists of an initial request from the buyer, followed by invocations to multiple suppliers in parallel, followed by a reply back to the buyer of the completed proposal. The <sequence> tag is used for executing components sequentially; the <flow> tag is used for parallel execution; and the <receive>, <reply>, and <invoke> tags handle the basic activities required to interact with the services (see Listing 7).

The first step in the process flow is the initial buyer request. Once this request is received, a parallel set of activities is executed using the <flow> tag. Each supplier will be contacted in order to receive quotes for specific PC components. Each references a specific WSDL operation (e.g., request_quote), using variables for input and output. Upon receiving the responses back from the suppliers, the purchasing agent would construct a message back to the buyer. This would involve use of the <assign> tag in BPEL4WS and the XPath language to take the data received from the suppliers and build a final proposal to the buyer.

The final step in this scenario is the management of exceptions. For example, if there is an error in contacting a supplier, the agent may want to send a message back to the buyer. Within BPEL4WS, this would be done with fault handlers (see Listing 8).

You may need to set up compensation handlers for the process. For example, if one of the suppliers can't be contacted while placing the order, there should be a way to roll back the order. To set up a transactional context in BPEL, the <scope> tag is used to group related activities together. In this scenario, the three parallel invocations to the suppliers might be a good candidate for a scope declaration.

In a Nutshell

Orchestration and choreography are terms related to connecting Web services in a collaborative fashion. The capabilities offered by the available standards will be vital for building dynamic, flexible processes. The goal is to provide a set of open, standards-based protocols for designing and executing these interactions involving multiple Web services.

Many vendors have announced support for BPEL4WS in their products, and the OASIS technical committee is looking to move this specification going forward. WSCI is being considered by the W3C for

Web services choreography. While BPEL4WS has defined a notion of choreography through abstract processes, it is still unclear whether this will be accepted over the W3C work. Clearly, market adoption will be driven by the direction taken by vendors and their support of the standards in their product implementations.

As these standards take shape, it will be important to pay close attention to the direction taken by standards bodies such as the W3C and OASIS. There is still some confusion on how these efforts will come together, if at all. And many organizations are concerned over how reliability and security will be addressed. The good news is that much progress has been made by the major vendors embracing these standards, bringing great promise for Web services orchestration and choreography going forward.

Listing 1: WSCI Example

```
<process name="Purchase" instantiation="message">
 <sequence>
  <action name="ReceiveOrder" role="Agent" operation=
    "tns:Order">
  </action>
  <action name="Confirm" role="Agent" operation="tns:Confirm">
   <correlate correlation="tns:ordered"/>
   <call process="tns:Purchase"/>
  </action>
 </sequence>
</process>
```

Listing 2: Illustration of a sequence in BPEL4WS

```
<sequence>
  <receive partner="buyer" … operation="sendOrder"
     cariable="request"/>
  <invoke partner="supplier" … operation="request"
     inputVariable="itemreq" outputVariable="itemqt"/>
  <reply partner="buyer" … operation="response" variable="
     proposal"/>
</sequence>
```

Listing 3: Define partner roles in BPEL4WS

```
<partnerLinks>
 <partnerLink name="Buyer"
partnerLinkType="po:requestQuoteLinkType"
  myRole="Purchaser"/>
 <partnerLink name="Supplier1"
partnerLinkType="po:requestPartQuoteLinkType"
  myRole="Requestor" partnerRole="Purchaser"/>
 <!--Set up other suppliers used in this process -->
</partners>
```

Listing 4: Partner link types defined in WSDL

```
 <!-- partnerLinkType defined in the WSDL document -->
 <plnk:partnerLinkType name="requestQuoteLinkType">
 .<plnk:role name="Purchaser">
   <plnk:portType name="po:request_quote"/>
  </plnk:role>
 </plnk:partnerLinkType>
```

Listing 5: BPEL4WS variables required for the process

```
<variables>
 <variable name="request" messageType="po:request"/>
     <variable name="part_request
messageType="po:part_request"/>
     <variable name="part_quote" messageType="po:part_quote"/>
     <variable name="proposal" messageType="po:proposal"/>
</variables>
```

Listing 6: Correlation properties defined in WSDL

```
<definitions name="properties" …>
<bpws:property name="quoteID" type="xsd:string"/>
</definitions>
 <definitions name="correlatedMessages ..>
  <bpws:propertyAlias propertyName="cor:quoteID"
messageType="po:part_quote" …>
 </definitions>
```

Listing 7: BPEL4WS process flow for the scenario

```
<sequence>
    <receive name="receive" partnerLink="Buyer"
operation="request"
   variable="request" initiate="yes">
    </receive>
    <flow name="supplier_flow">
      <invoke name="quote_supplier1" partnerLink="Supplier1"
operation="request_quote"
       inputVariable = "part_request"
outputVariable="part_quote">
      </invoke>
   <!-- invoke other suppliers as part of the process, done
       in parallel -->
    </flow>
    <!-- construct a proposal from the part quotes received
      -->
    <reply name="reply" partnerLink="Buyer"
operation="send_proposal" variable="proposal">
    </reply>
</sequence>
```

Listing 8: Setting up fault handlers

```
<faultHandlers>
 <catch faultName="cantFulfillRequest">
  <invoke partner="buyer" operation="sendError"
inputVariable="fault"/>
 </catch>
</faultHandlers>
```

Download the code at

sys-con.com/webservices