web services orchestration

a review of emerging technologies, tools, and standards

Abstract

Web services technologies are beginning to emerge as a defacto standard for integrating disparate applications and systems using open, XML-based standards. In addition to building web services interfaces to existing applications, there must also be a standard approach to connecting these web services together to form more meaningful business processes. In 2002, a number of new standards were introduced to address this problem, including BPEL4WS and WSCI. The purpose of this paper is to provide a review of these emerging standards, to help the reader better understand how web services orchestration can be accomplished today.

It is assumed that the reader has an understanding of the basic web services standards, including SOAP, WSDL, and UDDI.

Chris Peltz
Hewlett Packard, Co.
January 2003

table of contents

introduction

In a recent CSC survey [CSC], senior IT executives identified "connecting to customer, suppliers, and/or partners electronically" as the top ranked global management issue. META Group predicts that B2B web services projects will begin appearing around 2004 [META]. There is a clear need to extend current businesses to support the integration of web services, both internal and external. However, existing methods for creating business processes are not designed to interact with components that cross organizational boundaries. These methods are also not flexible enough to handle the technical interfaces (e.g., SOAP and WSDL) introduced by web services.

Processes being built today need the business agility to quickly adapt to customer needs and market conditions. This would include incorporating new customers, partners, or suppliers used in a process. A single standard is desired that can manage both EAI and B2B interactions involving web services. **Web services orchestration is about providing an open, standards-based approach for connecting web services together to create higher-level business processes**. Standards such as BPEL4W, WSCI, and BPML are designed to reduce the complexity required to orchestrate web services, thereby reducing time-to-market and costs, and increasing the overall efficiency and accuracy of business processes. Without a common set of standards, each organization is left to build their own set of proprietary business protocols, leaving little flexibility for true web services collaboration.

Today, developers are faced with understanding the capabilities of the various standards and tools available for orchestrating web services. The objective of this paper is to help the reader understand how the technologies available for web services orchestration can be used today. The concept of web services orchestration will be defined, along with key requirements for orchestrating web services. The major standards in this space will be discussed, followed by a review of products and tools currently available. Then, a specific case study will be explored in order to give the reader a more concrete example of how web services orchestration works. The paper will conclude with a look at future activities with web services orchestration.

terms and definitions

The industry has used a number of terms to describe how components can be connected together to build complex business processes. Workflow and document management systems have existed as a means to handle the routing of work between various resources in an IT organization. These resources might include people, systems, or applications, and typically involve some human intervention. Business process management systems (BPMS) have also been used to enable a business to build a tops-down process design model, consisting of various integration activities (e.g., integration to a legacy system). BPMS systems would typically cover the full lifecycle of a business process, including the modeling, executing, monitoring, management, and optimization tasks.

With the introduction of web services, terms such as "web services composition" and "web services flow" were used to describe the composition of web services in a process flow. More recently, the terms orchestration and choreography have been used to describe this. **Orchestration** describes how web services can interact with each other 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, multi-step process model. **Choreography** tracks the sequence of messages that may involve multiple parties and multiple sources, including customers, suppliers, and partners. Choreography is typically associated with the public message exchanges that occur between multiple web services, rather than a specific business process that is executed by a single party.

There is an important distinction between web services orchestration and choreography. Orchestration refers to an executable business process that may interact with both internal and external web services. For orchestration, the process is always controlled from the perspective of one of the business parties. Choreography is more collaborative in nature, in which each party involved in the process describes the part they play in the interaction. Many of the standards that will be discussed in this paper initially focused on either orchestration or choreography. However, recent enhancements and standards convergence has somewhat blurred this distinction. In this paper, the term **web services orchestration** will be used to describe the creation of business processes, either executable or collaborative, that utilize web services.

requirements

There are a number of important technical requirements that must be addressed when designing business processes involving multiple web services running over a long duration. Knowledge of these requirements will help in positioning the various standards that have been introduced for web services orchestration.

The ability to invoke services in an asynchronous manner is vital to achieving the reliability, scalability, and adaptability required by today's IT environments. With asynchronous support, a business process can invoke web services concurrently rather than sequentially in order to enhance performance. For example, a purchasing system may want to interact with multiple supplier web services at the same time, looking to find the supplier that can offer the lowest price of earliest shipment date. Asynchronous support can be achieved through web services through various correlation techniques.

Orchestrated web services that are long-running must also manage exceptions and transactional integrity. The architecture must take into account how the system will respond if there is an error or if the service invoked does not respond in a given time. Since nearly 80% of the time spent in building business processes is spent in exception management [HUR], one can see why this is a critical component. The architecture must also have a way to manage transactional integrity if something goes wrong. Traditional transactional systems that are ACID-based are typically not sufficient for long-running, distributed transactions. Resources cannot be locked in a transaction that runs over a long period of time. The notion of "compensating transactions" has been introduced to undo an action if a process or user cancels it.

Web services orchestration must be dynamic, flexible, and adaptable to meet the changing needs of a business. Flexibility can be achieved by providing a clear separation between the process logic and the web services used. This separation can usually be achieved through an orchestration engine. The engine handles the overall process flow, calling the appropriate web services and determining the next steps to complete. With this approach, an IT organization can swap out services in the overall process flow. Reusability is also a major benefit with web services, and there is often a need to compose higher-level services from existing orchestrated processes. As the following figure shows, this can be accomplished by exposing the process with its own web service interface so that other processes can then use it.

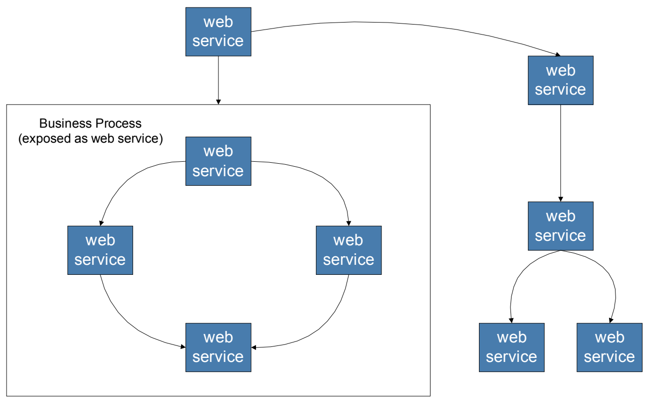

Figure 1. Recursive Composition of Web Services

web services orchestration standards

This section will take a look at the various standards that exist for web services orchestration. It will begin with a look at some early work in this technology space, and then focus on three of the more promising standards for orchestration -- BPEL4WS, WSCI, and BPML. Each standard will be presented, along with a discussion of how they meet the specific technical requirements outlined above. The section will conclude with a look at the current state of each standard and potential for convergence moving forward.

early work

Early work in web services orchestration included eCo, WSCL, XLANG, and WSFL. CommerceNet initially created the eCo framework to demonstrate the value of integrating e-commerce services, with a focus on the document exchanges required for B2B integration. The specification had a vague notion of orchestration, showing how a process can be composed of web services. The Web Services Conversation Language (WSCL) outlined a simple conversation language standard, focused on modeling the sequencing of interaction between web services. This was somewhat analogous to web services choreography.

Microsoft initially developed the XLANG specification for the Microsoft BizTalk Server. XLANG focused on the creation of business processes and the interactions between web service providers. The specification provided support for sequential, parallel, and conditional process control flow. It also included a robust exception handling facility, with support for long-running transactions through compensation. XLANG used WSDL as a means to describe the service interface of a process.

The Web Services Flow Language (WSFL) was an IBM proposal to describe both public and private process flows. WSFL defines a specific order of activities and data exchanges for a particular process. It defines both the execution sequence and the mapping of each step in the flow to specific operations, referred to as flow models and global models. The flow model represents the series of activities in the process, while the global model binds each activity to a specific web service instance. A WSFL definition can also be exposed with a WSDL interface, allowing for recursive decomposition. WSFL supports the handling of exceptions but has no direct support for transactions.

BPEL4WS

The web services workflow specifications outlined by XLANG and WSFL have recently been superseded by a new specification from IBM, Microsoft, and BEA called BPEL4WS (Business Process Execution Language for Web Services)[1]. BPEL4WS is a specification that models the behavior of web services in a business process interaction [WEE]. The specification provides an XML-based grammar for describing the control logic required to coordinate web services participating in a process flow. This grammar can then be interpreted and executed by an orchestration engine, which is controlled by one of the participating parties. The engine coordinates the various activities in the process, and compensates the system when errors occur.

BPEL4WS is essentially a layer on top of WSDL, with WSDL defining the specific operations allowed and BPEL4WS defining how the operations can be sequenced. A BPEL document leverages WSDL in three ways:

1. Every BPEL process is exposed as a web service using WSDL. The WSDL describes the public entry and exit points for the process.

[1] The terms "BPEL4WS" and "BPEL" will be used interchangeably throughout this paper.

2. WSDL data types are used within a BPEL process to describe the information that passes between requests.
3. WSDL might be used to reference external services required by the process.

BPEL4WS provides support for both executable and abstract business processes. An executable process models the behavior of participants in a specific business interaction, essentially modeling a private workflow. Abstract processes, modeled as business protocols in BPEL4WS, specify the public message exchanges between parties. Business protocols are not executable and do not convey the internal details of a process flow. Essentially, executable processes provide the orchestration support described earlier while the business protocols focus more on the choreography of the services.

The specification includes support for both basic and structured activities. One can think of a basic activity as a component that interacts with something external to the process itself. For example, basic activities would handle receiving or replying to message requests as well as invoking external services. The typical scenario is that there is a message received into the BPEL process. The process may then invoke a series of external services to gather additional data, and then respond to the requestor in some fashion. In the figure below, the <receive>, <reply>, and <invoke> messages all represent basic activities for connecting the services together.

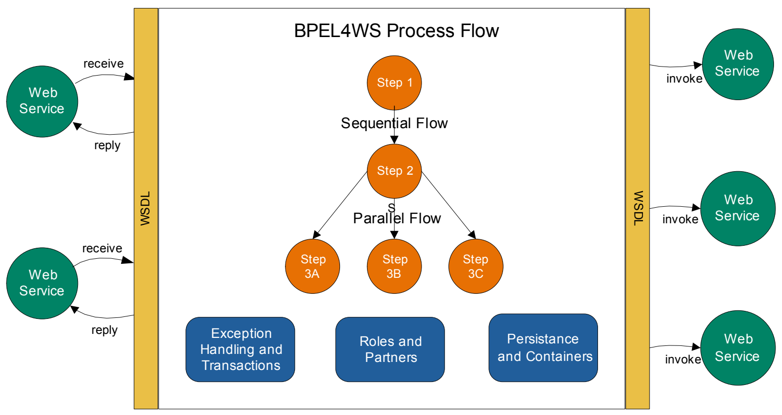

Figure 2. BPEL4WS Process Flow

In contrast, structured activities manage the overall process flow, specifying what activities should run and in what order. Structured activities might specify that certain activities should run sequentially or in parallel. These activities also provide support for conditional looping and dynamic branching. One can think of structured activities as the underlying programming logic for BPEL4WS. The following is a simple illustration of how a sequential activity would be described, containing basic activities to receive a message, invoke a partner, and reply back to the caller:

```
<sequence>
      <receive partner="buyer" … operation="sendOrder" container="request"/>
      <invoke partner="supplier" … operation="request" container="order"/>
      <reply partner="buyer" … operation="response" container="proposal"/>
</sequence>
```

Listing 1. Illustration of a sequence in BPEL4WS

Containers and partners are two other important elements within BPEL4WS. A **container** identifies the specific data exchanged in a message flow, which typically maps to a WSDL messageType. When a BPEL process receives a message, the appropriate container is populated so that subsequent requests can access the data. A **partner** could be any service that the process invokes or any service that invokes the process. Each partner is mapped to a specific role that it fills within the business process. A specific partner might play one role in one business process but a completely different role in another process. Containers are then used to manage the persistence of data across web services requests.

The following illustrates a simple example of how partners and containers can be defined in BPEL4WS:

```
<partners>
    <partner name="buyer" … myRole="agent"/>
    <partner name="supplier" … myRole="requestor" partnerRole="supplier"/>
</partners>
<containers>
    <container name="request" messageType="tns:orderRequest"/>
    <container name="response" messageType="tns:orderResponse"/>
</containers>
```

Listing 2. BPEL4WS example illustrating partners and containers

Finally, BPEL4WS provides a robust mechanism for handling transactions and exceptions, building on top of the WS-Coordination and WS-Transaction specifications. These corollary specifications include the support necessary to manage and coordinate the operations of a business activity.

In BPEL4WS, a set of activities can be grouped in a single transaction through the <scope> tag. This tag signifies that the steps enclosed in the scope should either all complete or all fail. Within this scope, the developer can then specify compensation handlers that should be invoked if there is an error. For example, if part of a travel reservation process fails, the handler would identify how to rollback the other parts of the transaction. The transactional mechanisms within BPEL work hand-in-hand with exception handling. BPEL provides a robust exception handling mechanism through the use of throw and catch clauses, similar to the Java programming language.

WSCI

The Web Services Choreography Interface (WSCI, pronounced "Whiskey") is a specification from Sun, SAP, BEA, and Intalio that defines an XML-based language for web services collaboration [ARK]. It defines the overall choreography describing the messages between web services that participate in a collaborative exchange. The specification supports message correlation, sequencing rules, exception handling, transactions, and dynamic collaboration.

A key aspect of WSCI is that it only describes the observable or visible behavior between web services. WSCI does not address the definition of executable business processes as defined by BPEL4WS. Furthermore, a single WSCI document only describes one partner's participation in a message exchange. As the following illustrates, a WSCI choreography would include a set of WSCI documents, one for each partner in the interaction. In WSCI, there is no single controlling process managing the interaction.

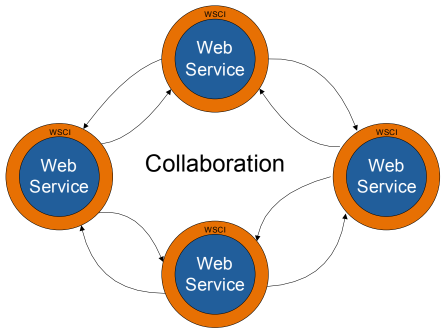

Figure 3. Web Services Choreography Interface (WSCI)

WSCI can also 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 the glue around WSDL, describing how the operations can be choreographed. In other words, WSDL would be used to describe the entry points for each service available and WSCI would describe the interactions among these WSDL operations, very similar to how BPEL4WS leverages WSDL.

WSCI supports both basic and structured activities:

- The <action> tag is used to define a basic request or response message. Each activity specifies the WSDL operation involved and the role being played by this 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.

The following listing illustrates a simple example of WSCI. A purchasing process is created containing two sequential activities, "Receive Order" and "Confirm". Each activity maps to a WSDL portType, and a correlation is established between the two steps. Note that this is the WSCI document from the perspective of the Agent. There would also be WSCI files for the buyer and the supplier in the process.

```
<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 3. WSCI Example

Both business transactions and exception handling are supported by WSCI. Specific transactional contexts can be set up within WSCI, similar to the scope activity in BPEL4WS. When a set of activities is defined within a context, any failure will result in the entire group being rolled back.

BPML

The Business Process Management Language (BPML) is a meta-language for describing business processes. The specification was developed by Business Process Management Initiative (BPMI.org), an independent organization chartered by Intalio, Sterling Commerce, Sun, CSC, and others. BPML was initially designed to support business processes that could be executed by a BPMS system. However, the first draft of BPML also incorporated the WSCI protocol. WSCI could be used to describe the public interactions and choreographies and the private implementations could be developed with BPML. Both BPML and WSCI share the same underlying process execution model and similar syntaxes.

The specification can also be loosely compared to BPEL4WS, providing similar process flow constructs and activities [SHA]. Basic activities for sending, receiving, and invoking services are available, along with structured activities that handle conditional choices, sequential and parallel activities, joins, and looping. BPML also supports the scheduling of tasks at specific times.

Other features supported in BPML include persistence, roles, instance correlation, and recursive decomposition. The language was designed to manage long-lived processes, with persistence supported in a transparent manner. XML exchanges occur between the various participants, with roles and partner components similar to the BPEL constructs. BPML also supports recursive composition, the ability to compose sub-processes into a larger business process.

BPML includes both transactional support and exception handling mechanisms. Both short and long-running transactions are supported, with compensation techniques used for more complex transactions. BPML uses a scoping technique similar to BPEL4WS to manage the compensation rules. It also provides the ability to nest processes and transactions, a feature that BPEL currently does not provide. Finally, a robust exception handling mechanism is available within BPML, following many of the constructs in XLANG. Timeout constraints can also be specified for specific activities defined within the process.

summary

This section has presented an overview of the emerging standards in the web services orchestration arena. Each standard has taken somewhat of a different approach to orchestration. BPEL4WS primarily focuses on the creation of executable business processes, while WSCI is concerned with the public message exchanges between web services. WSCI takes more of a collaborative and choreographed approach, requiring each participant in the message exchange to define a WSCI interface. BPEL takes more of an "inside-out" perspective, describing an executable process from the perspective of one of the partners. BPML has some complimentary components to BPEL4WS, both providing capabilities to define a business process. WSCI is now considered a part of BPML, with WSCI defining the interactions between the services and BPML defining the business processes behind each service.

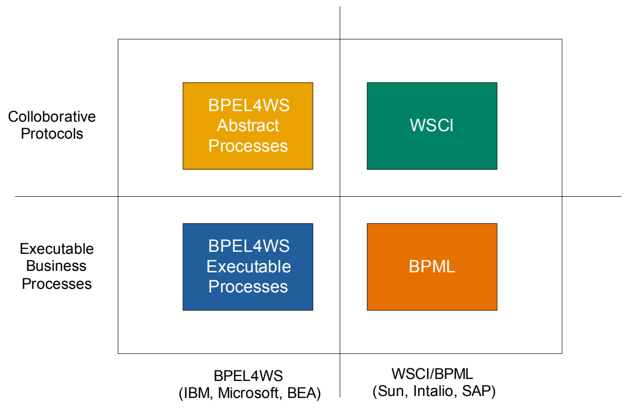
Colloborative Protocols
Executable Business Processes
BPEL4WS Abstract Processes
WSCI
BPEL4WS Executable Processes
BPML
BPEL4WS (IBM, Microsoft, BEA)
WSCI/BPML (Sun, Intalio, SAP)

Figure 4. Relationship between BPEL4WS, WSCI, and BPML

As these standards are being defined, it is still very unclear which ones will emerge as industry standards for web services orchestration and choreography. There are key advantages to each standard:

- BPEL4WS includes a number of major supporters behind it, including IBM, Microsoft, and BEA. This could make it very difficult for the BPML/WSCI initiative to gain broad acceptance. There is also more technical documentation and developer tools currently available for BPEL4WS.
- WSCI appears to be more robust in its support for collaboration/choreography as compared to BPEL4WS. The W3C recently announced on Jan 14, 2003 a web services choreography work group that will consider both HP's WSCL and WSCI, but not BPEL4WS. BPEL4WS wasn't considered because it was never submitted to W3C and does not offer a royalty-free condition of its use.

A potential scenario could also include merging the BPEL4WS executable process language with WSCI for abstract, collaborative processes.

The following is a quick overview of a few of the products available for web services orchestration and choreography. This list is not meant to be exhaustive, and represents a sampling of vendors that support that XLANG, BPEL4WS, WSCI, and BPML standards. Of the tools listed, BEA WebLogic Workshop, Collaxa Orchestration Server, IBM's BPWS4J, and SunONE WSCI Generator were directly evaluated. Information presented on the other tools was gathered from the available technical papers and product brochures.

BEA WebLogic Workshop

While BEA contributed to both the BPEL4WS and WSCI initiatives, the current release of the BEA WebLogic Workshop product does not provide native support for either standard[2]. However, Workshop does provide a very elegant user interface for connecting web services together, with support for asynchronous messages and web services conversations. The tool is probably one of the best choices for developers wanting to get some experience with composing web services together in a conversational context.

The product supports asynchronous messaging through callbacks and timer controls. The client can specify a callback location (e.g., a URL) that the service can invoke when the request is completed. If a callback cannot be specified, the developer can set up a timer control. Timer controls essentially implement a polling approach, periodically checking the service to determine whether the data is ready. This is typically used when interacting with legacy systems that do not have a built-in callback mechanism.

Conversations within Workshop can be built with JavaDoc annotation tags. For example, to start a conversation, a developer would include the tag, "@jws:conversation: phase=start". Conversations can be started, continued and completed in Workshop. With this simple Java tagging mechanism, the process will automatically manage the conversation through a unique SOAP conversation ID. The following figure illustrates how conversations and asynchronous support can be done with WebLogic Workshop today.

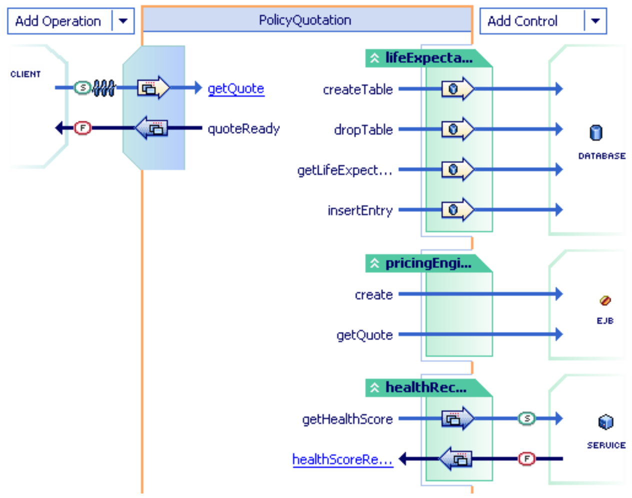

Figure 5. BEA WebLogic Workshop Example

[2] BEA announced on Dec 13th, 2002 that they would be releasing a major upgrade to the WebLogic platform in 1H'03. The effort, code-named "Gibraltar", will provide additional support for working with business processes, including BPEL4WS support.

The product can connect a number of external components together through the use of controls. Within Workshop, a control can be set up to connect the process to a web service through a WSDL interface. Controls can also be defined to connect to Enterprise JavaBeans, JDBC, and JMS. Once the connection is established, a developer can add additional programming logic for controlling the process. Today, this process logic would be written using the Java programming language. BEA WebLogic Workshop also provides an automated mechanism to generate a web-based user interface, enabling the developer to test functionality of the web service very quickly.

Microsoft BizTalk Server

Microsoft BizTalk Server can be used to design, build, and execute dynamic business interactions that span applications, platforms, and organizations. The current version of BizTalk supports the XLANG specification for modeling business processes. The product includes an orchestration engine for executing and monitoring processes. It also includes a very robust visual development environment, BizTalk Orchestration Designer, for defining and connecting processes. With BizTalk, a distinct separation is made between the process definition and implementation. This provides additional flexibility for dynamically changing the process flow or the services used within the interaction.

Bindings within BizTalk are done using ports, similar to WSDL portTypes for sending and receiving messages. Process bindings can be built for COM components, MSMQ queues, script components, and other BizTalk server processes. BizTalk 2002 does not provide native support for web services integration. Today, a port has to be mapped to a COM object, and the web service has to be then wrapped with a COM-based proxy. Future versions of BizTalk will include more integrated support for .NET where XLANG schedules can be compiled as .NET assemblies within the framework.

BizTalk provides the basic primitives for conditional branching, sequential, and parallel execution. It also includes support for long-running transactions. A transactional context can be defined with appropriate compensation logic if parts of the transaction fail. The product includes robust management and monitoring support, with the ability to query a process state, manage processes, and debug them.

In October 2002, Microsoft detailed a vision for their next-generation business collaboration environment, code-named "Jupiter". The goal of the project is to unify and extend current e-business server technologies, including business process management and monitoring capabilities. It will provide tighter integration with the .NET framework and provide additional support for the BPEL4WS specification. The project will be implemented in a two phases, with BPEL support coming in 2H'03 and enhanced e-commerce capabilities being delivered in 1H'04.

Collaxa Orchestration Server

Collaxa provides a simple, standards-based software infrastructure for integrating collaborative business processes together. The Collaxa product supports the BPEL4WS, WS-Coordination, and WS-Transaction specifications and includes an orchestration server and a management console. The orchestration server provides the underlying infrastructure; handling asynchronous processing, flow and transaction coordination, and monitoring of business processes. Collaxa also plans to provide a graphical designer tool for building processes, but this tool is not yet available for download.

Every process in Collaxa is modeled as a BPEL Scenario, which allows Java developers to easily compose asynchronous, long-running, multi-step workflows and can automatically be published as its own web service, enabling higher-level composition. BPEL Scenarios can be implemented in standard BPEL4WS or in a JSP-like scripted programming abstraction called JBPEL. With JBPEL, Java developers indicate how a process should flow using a JSP-like syntax and Java annotation tags, somewhat similar to the BEA WebLogic Workshop approach. Within a JBPEL Scenario, each web service interface uses a JavaBean proxy interface that Collaxa calls a "two-way proxy". These interfaces are generated through command-line tools provided by Collaxa, and the orchestration server automatically handles the marshalling and unmarshalling of the web services. The following is an example of a JBPEL script:

```
public ITripReceipt process (ItripInfo info)
{
    /** @bpel: flow */
    {
        /** @bpel:sequence AirlineBranch */
        {
            airline = Airline.initiate(info);
            ticket = Airline.receiveResult(airline);
        }
        /** @bpel:sequence CarBranch */
        {
            car = CarRental.initiate(info);
            reservation = carRental.receiveResult(car);
        }
    }
}
```

Listing 4. Collaxa BPEL Scenario Example

Collaxa has support for all of the basic control flow constructs, including support for parallel processing and sequential activities. Dynamic branching and looping is supported, along with complex join patterns for parallel flow. Collaxa provides a unique implementation for asynchronous messaging. Given a web services interface (WSDL), the WSDLC tool can be used to automatically generate the Java interface or proxy. This generation process automatically creates a patented 2-way proxy class for handling asynchronous interactions. Essentially, there is an interface created for invoking the class and another one for receiving a callback. When the orchestration server detects an asynchronous call, it automatically pauses execution and passivates the state of the current activity in a database. The activity is then re-activated when the result comes back from the invoked service.

The product supports both atomic and business transactions, based on the WS-Coordination and WS-Transaction standards. With Collaxa, a developer can assign a transactional context to a section of a BPEL Scenario. Exception handling is fully supported in the product. If an exception is thrown, the orchestration server will either cancel or compensate the appropriate activities in the scenario.

Collaxa includes a robust user interface for managing and interacting with deployed BPEL Scenarios. The management console can be used for testing, monitoring, and viewing transaction logs. The visual audit trails within Collaxa display process state graphically for BPEL4WS flows along with an XML debugger. The user interface also includes a simple portal for interacting with a business process. The portal enables a developer to build a quick client for the process, but there are also APIs available to customize the user experience. For example, customized JSPs can be written to interact with a BPEL Scenario

IBM BPWS4J

IBM provides an alpha release of their Business Process Execution Language for Web Services Java Run Time (BPWS4J) from their alphaWorks website. The BPWS4J platform contains two basic components: a runtime platform for executing BPEL documents and an editor for creating them. The download includes the engine, a web-based interface to deploy a BPEL process, an Eclipse plug-in to edit the BPEL documents, and documentation for using it. While the documentation was somewhat incomplete, the samples provides were very helpful in gaining a better understanding of BPEL4WS.

The BPWS4J editor provides a simple user interface for creating BPEL4WS documents. The editor is delivered as a plug-in to Eclipse. Eclipse provides an environment for quickly constructing BPEL documents, with support for task lists to help identify problems in the BPEL definition. Through the editor, a developer can view both a visual and XML representation of the process, with bi-direction synchronization included. The view is basically a hierarchical representation of the process and the activities within it. Through the editor, you can add sequences, flows, and receive/reply requests. Containers and partners can also be added, describing the information flow and partner roles used within the process. The following illustrates how a BPEL document looks within the BWS4J editor:

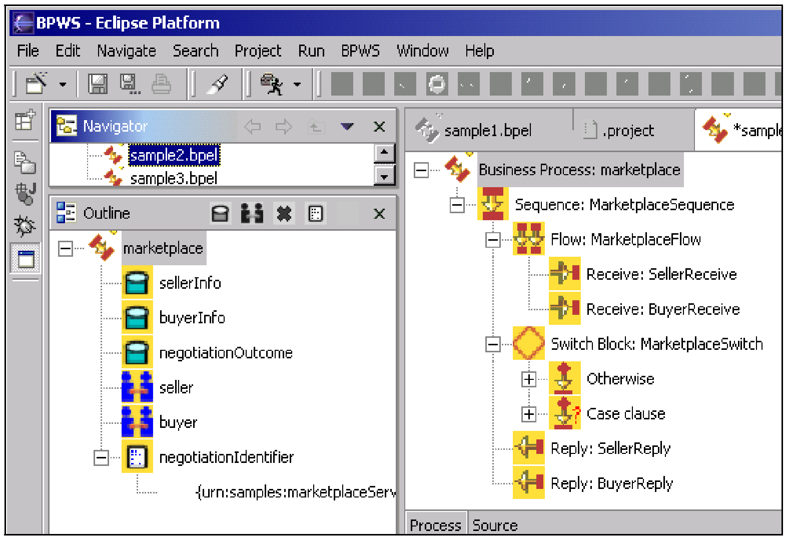

Figure 6. BPWS4J Example within Eclipse

The runtime engine for BPWS4J can be deployed on either Apache Tomcat or IBM WebSphere. Installation on Apache Tomcat was very easy and all but one of the samples could be executed. BPWS4J includes a very simple web-based interface for viewing deployed BPEL processes, deploying new processes, and undeploying existing ones. The deployment process requires three basic pieces of information:

- The BPEL4WS document that corresponds to the process being defined.
- The corresponding WSDL file for the BPEL process. This WSDL file specifies the public interface being exposed by the process and also includes all data types and messages used.
- Any dependent web services used by the process, indicated as a WSDL reference

Once the process has been deployed, it is assigned a SOAP endpoint. A developer could then automatically build a SOAP client from the WSDL, providing a quick way to interact with the business process.

Sun WSCI Editor

Sun provides a free, alpha release of their SunONE WSCI Editor. This release is a fairly lightweight editor, but does provide basic functionality to build a WSCI interface file. The tool requires the developer to input a WSDL file for the port types that will be referenced in the choreography. It supports some of the basic WSCI constructs, including actions, sequences, and switch statements. The download provided minimal documentation and sample code for the tool. Additionally, it was not possibly to test the WSCI interface created because a WSCI interface by definition is not executable.

Intalio|n^3

Intalio provides a complete Business Process Management System (BPMS) for managing discrete and transactional business processes. The Intalio|n^3 product supports the full process lifecycle, including design, deploy, execute, analyze, and optimize. It is marketed as a complete, end-to-end solution, with a single development environment for both creating and managing business processes. Key components of the product include a process design tool, a console, and a portal. The process design tool claims to generate over 80% of the code required to build an executable business process, allowing for shortened development time. The console provides a management environment to administer, configure, and monitor deployed processes. Users can interact with deployed processes through a web-based portal interface. The product brochure indicates support for the BPML specification, and while Intalio was the main contributor to the WSCI specification, it was unclear whether the product provided this support.

summary

This section provided an overview of some of the available products for orchestrating and choreographing web services. It is not meant to be an exhaustive list as new products are already emerging on the market today. A few recommendations can be made from the evaluation of the tools listed here.

- For developers who want to gain a quick understanding of how conversations and asynchronous messaging can be done with web services, BEA WebLogic Workshop is one of the best tools available today.
- Microsoft BizTalk Orchestration Server also provides an easy-to-use interface for creating business processes, but does not have direct support to invoke web services from within a process definition.
- The Collaxa Orchestration Server would make a very good learning tool to understanding the basic BPEL constructs, such as process flow, joins, and conditional processing. The product supports BPEL4WS as well as a JSP-like programming language, rich management console, and samples available are very helpful.
- IBM's BPWS4J product is the only product that provides a graphic tool for creating BPEL4WS documents and would be recommended for those developers wanting experience with writing BPEL4WS documents.
- The SunONE WSCI Generator is the only product that supports WSCI, but there was no mechanism to test out the WSCI generation.
- Finally, while the Intalio product was not directly evaluated here, it would be a worthwhile product to check out if BPML was of interest.

A case study will now be introduced to illustrate the various concepts around web services orchestration. The scenario revolves around a purchasing system where a PC manufacturer wishes to build a set of PC configurations using a list of available suppliers. A PC configuration consists of a set of component parts, such as a motherboard, processor, and a power supply. 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 requested by the buyer. 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 order for the parts or cancel the request. The following illustrates a simplified view of the process. It shows the initial request from the buyer to the agent, with subsequent requests to each supplier:

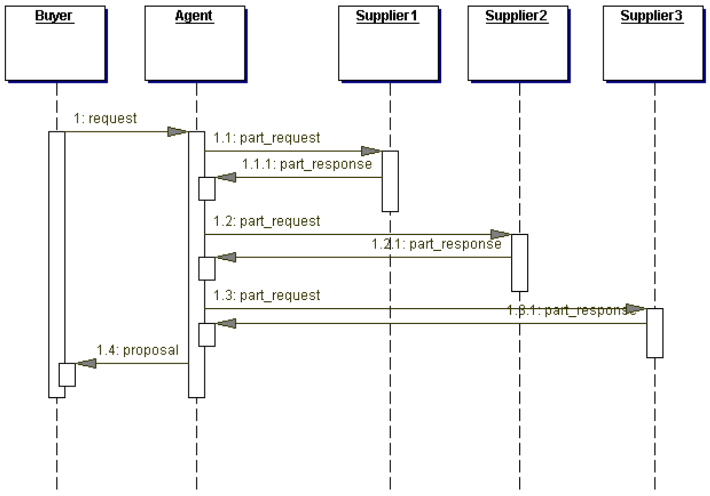

Figure 7. Case Study Sequence Diagram

In addition to accepting requests from the buyer and sending requests to the various suppliers, the purchasing agent must also handle the transactional aspects of the process. A supplier may or many not fulfill a specific request from the agent. A failure could result because the supplier does not have the part specified or the agent cannot contact the supplier because of networking issues. In these cases, the agent must manage the process appropriately, either rolling back the request or providing some mechanism for the client to place an incomplete order.

There are three distinct partners in this orchestrated process: the buyer, the purchasing agent, and a set of suppliers. Each partner 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 buyer and suppliers may have their own internal workflows, but these will not be discussed.

a BPEL4WS example

This case study used the BPEL4WS because of the technical documentation and developer tools available. The tool used, BPWS4J, provided support for both modeling and executing business processes[3]. The

[3] At the time of this writing, there were no tools available for modeling and executing a WSCI collaboration.

discussion below will focus on the definition of the BPEL4WS document itself. References are made to specific WSDL portTypes and messageTypes, but these WSDL definitions will not be provided here.

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 BPEL document.

The next step is to define the partners in the process and the roles they play. In this case study, 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 implemented in BPEL4WS with the <partners> and <partner> tags. In the example below, the serviceLinkType is essentially a reference to a WSDL that defines the port types available and the roles allowed for that port type. Here, we will assume there is a WSDL port type defining a request_quote operation that is initiated from the buyer. The purchasing agent also has a request_quote link to the supplier.

```
<partners>
    <partner name="Buyer" serviceLinkType="tns:requestQuoteLinkType"
        myRole="Purchaser" partnerRole="Requestor"/>
    <partner name="Supplier1" serviceLinkType="tns:requestQuoteLinkType"
        myRole="Requestor" partnerRole="Purchaser"/>
    <!--Set up other suppliers used in this process -->
</partners>
```

Listing 5. Define partner roles in BPEL4WS

Since the process is defined from the perspective of the purchasing agent, only the buyer and supplier partner relationships have to be defined. When the buyer interacts with the agent, the buyer is the requestor and the purchasing agent acts as the receiver or purchaser. The roles are reversed when the agent interacts with one of the suppliers.

The process must also manage the flow of information between the partners. BPEL4WS uses containers to store the information that has to be persisted in the business process. In this scenario, a buyer makes their 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 the buyer. Here, there are potentially four containers to model this interaction, two for each request/response interaction. Each container shown in the listing below references a specific WSDL messageType.

```
<containers>
    <container name="request" messageType="tns:request"/>
    <container name="part_request messageType="tns:part_request"/>
    <container name="part_quote" messageType="tns:part_quote"/>
    <container name="proposal" messageType="tns:proposal"/>
</containers
```

Listing 6. BPEL containers required for the process

There must also be a way to correlate the message requests to each other. In this scenario, there might be unique identifiers for each quote received from the suppliers and the final proposal constructed by purchasing agent. These identifiers must be documented in the BPEL process using the <correlationSet> tag, which would reference specific message types within the WSDL document, e.g.:

```
<correlatonSets>
      <correlationSet name=”Quote” properties=”cor:quoteID”/>
      <correlationSet name=”Proposal” properties=”cor:proposalID”/>
</correlationSets>
```

Listing 7. Defining correlation sets in BPEL.

A key part of the BPEL4WS document is the definition of the basic sequence of steps required to handle the request. This is where basic and structured activities come into play. The process flow in this scenario consists of an initial request from the buyer, followed by invocations to multiple suppliers in parallel, followed by reply back to the buyer of the completed proposal. To model this, the SEQUENCE tag is used for running 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.

```
<sequence>
    <receive name="receive" partner="Buyer" operation="request"
       container="request" createInstance="yes">
    </receive>
    <flow name="supplier_flow">
       <invoke name="quote_supplier1" partner="Supplier1" operation="request_quote"
          inputContainer = "part_request" outputContainer="part_quote">
       </invoke>
       <!-- invoke other suppliers as part of the process, done in parallel -->
    </flow>
    <reply name="reply" partner="Buyer" operation="send_proposal container="proposal">
    </reply>
</sequence>
```

Listing 8. BPEL4WS process flow for the scenario

The first step in the process flow is the initial buyer request. The "createInstance" flag is used to identify the start of a new process instance. Once this request is received, there is a parallel set of activities that are executed using the FLOW tag. Here, each supplier will be contacted in order to receive quotes for specific PC components. Each references a specific WSDL operation (e.g., request_quote), and uses the available containers for input and output. Upon receiving the responses back from the suppliers, the purchasing agent would construct a message back to the buyer. This could involve use of the XPath language to take the various containers received from the suppliers and building a final proposal to the buyer.

The final step in this scenario is the management of exceptions. While the sequence diagram above doesn't specifically show the interactions when requests are cancelled or in error, the process must still handle these 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, e.g.:

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

Listing 9. Setting up fault handlers in BPEL4WS.

There may be a 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 rollback the order. To set up a transactional context in BPEL, the <scope> tag would be 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.

This completes the basic steps required to set up the business process using BPEL4WS. The completed process could then be deployed into an orchestration engine, such as the one provided by BPWS4J. During this deployment, the developer would specify this BPEL4WS document, the WSDL for the process, and any referenced WSDL used. Hopefully, this scenario demonstrated some of the key characteristics of web services orchestration, and how this could be implemented today using BPEL4WS.

conclusion

Orchestration, choreography, business process management, and workflow -- these are all terms related to connecting web services together in a collaborative fashion. The capabilities offered by web services orchestration 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.

To accomplish this, there are some basic requirements that have to be met. There is a need for asynchronous support in order to build reliability into the process. Strong transactional semantics and exception handling are required to manage both internal and external errors. There is also a need for a set of programming constructs to describe workflow. Finally, there has to be a way to link or correlate requests together to build higher-level conversations.

Some of the leading standards for web services orchestration include BPEL4WS, WSCI, and BPML. It is unclear where the industry is going with the various standards. There is a fair amount of traction behind BPEL4WS from major players in the industry, and WSCI and BPML have already converged with each other. All of the standards discussed here support the basic requirements for orchestrating web services, either as an executable process or as an abstract messaging protocol. There is no clear winner in this space yet, but the availability of developer tools will be one of the drivers to standards adoption. A number of tools have already been released that either support the basic concepts behind web services orchestration or support the specific standards mentioned here.

The other key driver to adoption of these standards will be the availability of use cases and scenarios. One specific case study was presented here, but the industry will need to develop additional use cases to demonstrate the business value of web services orchestration.

future initiatives

This paper concludes with a look at a few emerging areas for web services orchestration, including peer-to-peer conversations, security, and manageability.

The ability to support a conversational model between web services is an important area that must be addressed by the web services standards. A conversational model for web services provides a more loosely coupled, peer-to-peer interaction model. These conversations involve multiple steps between parties, often involving negotiation between the parties. A peer-to-peer conversational model takes more of a third-person perspective, quite different from the standards presented in this paper. Even WSCI, which offers a somewhat collaborative model between web services, still takes a first-person perspective for any given WSCI document. [HAN] offers a good analogy to illustrate the difference between the business process standards and the conversational model for web services. The current web services model is analogous to a vending machine. There are a set number of buttons that can be pressed in a pre-defined order. A conversational model is more analogous to a telephone call, involving a series of exchanges between the parties at each end in a more flexible, dynamic fashion. At this time, IBM's Conversation Support for Web Services (CS-WS) is the only standard that claims to support this capability [KUM].

Security is also a concern for web services orchestration and web services in general. Security is vital because we are now dealing with exposing interfaces on a somewhat less-secure protocol. There are a number of standards being discussed for web services security including XML Digital Signatures and Encryption, SAML, and WS-Security. These standards hope to fill specific web services security requirements for the authentication and authorization of users, and for securing the XML message itself. However, the orchestration standards presented in this paper do not offer direct support for security. For example, how do the roles defined for each partner relate or leverage the existing authentication and authorization standards? At this point, it is unclear how these standard relate, but there is an opportunity for the industry to work on the intersection of these two efforts.

The other area receiving a great deal of attention is web services management. Organizations have a need for an end-to-end management infrastructure that manages across applications, systems, and networks. Included in this is the ability to manage and monitor business processes involving web services. Businesses need to have more flexible control over their businesses process, with end-to-end visibility and the capability to control specific steps in the process. A robust management infrastructure must both monitor the health of the environment, as well as provide the capability to optimize and adapt itself in real-time. This is the main thrust behind Business Activity Monitoring (BAM), a term coined by Gartner to describe the process of providing real-time access to critical business information, potentially drawn from multiple application systems and sources.

For the management of business processes based on webMethods or Siebel's Universal Application Network (UAN), OMI is the most appropriate standard available. OMI provides a standards-based (XML/HTTP) interface allowing applications to expose management information to a management platform. With OMI, customers can include business processes into service level agreements with customers, suppliers, and partners. For example, it can be used to show the start of a process, the order of steps in a business process, and the relationship between these steps.

In addition to the OMI effort, HP has been leading many of the web services management initiatives. HP has extended its leadership in systems and network management to the management of web services. Today, web services can be managed with OpenView plug-ins that support SOAP and UDDI servers. HP has also announced Web Services Management Engine (WSME), a new technology preview that supports the active management of services. WSME is a collection of software components that enable an enterprise or service provider to provision and enforce service level agreements for web services. The need for this type of management will become increasingly important as web services are combined, aggregated, and orchestrated to form more meaningful business processes.

references

[ADA] Adams, Holt. *Asynchronous operations and Web services, Part 3*. IBM developerWorks, Oct 2002.

[ARK] Arkin, Assaf, et. al. Web Service Choreography Interface 1.0. www.sun.com/software/xml/developers/wsci/wsci-spec-10.pdf, 2002.

[BAK] Baker, Jeanne, P. Fingar, and H. Smith. *Value Chain Integration: The Next Frontier*. Internet World, July 2002.

[BAU] Bau, David. *SOAP Conversation Protocol (SOAP Conversation) 1.0*. BEA dev2dev (www.dev2dev.bea.com), June 2002.

[CSC] CSC. *13th Annual Critical Issues of IS Management Survey*. CSC (www.csc.com/survey), 2000.

[DUF] Duftler, Matthew and R. Khalaf. *Business Process with BPEL4WS: Learning BPEL4WS, Part 3*. IBM developerWorks, Oct 2002.

[DUI] Duivestein, Sander. *Web Services and Workflow*. Web Services Architect (www.webservicesarchitect.com), Sep 2001.

[HAN] Hanson, James E., P. Nandi, and D. Levine. *Conversation-enabled Web Services for Agents and e-Business*. IBM Research Paper.

[HUR] Hurwitz Group. *Ten Pillars for World Class Business Process Management*. The Hurwitz Group (www.hurwitz.com), July 2001.

[KUM] Kumaran, Santhosh and P. Nandi, *Conversational Support for Web Services: The next stage of Web services abstraction*. IBM developerWorks, Sep 2002.

[LEE] Lee, Juhnyoung, J. Yang, and J. Chung. *Winslow: A Business Process Management System with Web Services*. IBM Research Report, Nov 2002.

[LEY] Leymann, Frank and D. Roller. *Business processes in a Web services world*. IBM developerWorks, Aug 2002.

[LEY2] Leymann, Frank, D. Roller, and M. Schmidt. *Web services and business process management*. IBM Systems Journal, Volume 41-2, 2002.

[META] Meta Group, *Web Services: Stages of Adoption*. The Meta Group, 2002.

[MIC] Microsoft. *BizTalk Orchestration: A Technology for Orchestrating Business Interactions*. Microsoft Corporation, June 2000.

[ORI] O'Riordan, David. *Business Process Standards for Web Services*. Tect.

[SHA] Shapiro, Robert. *A Comparison of XPDL, BPML, and BPEL4WS*. Cape Visions, May 2001.

[SHE] Sherman, Doron. *Orchestrating Asynchronous Web Services*. Collaxa (www.collaxa.com), Feb 2002.

[SNE] Snell, James. *Automating business processes and transactions in Web services*. IBM developerWorks, Aug 2002.

[WEE| Weerawarana, Sanjiva and C. Francisco. *Business processes: Understanding BPEL4WS, Part 1*. IBM developerWorks, Aug 2002.

[WES] Wesley, Ajamu. *WSFL in action, Part 1*. IBM developerWorks (www.ibm.com/developerworks), Jan 2002.