By Francisco Curbera, Rania Khalaf, Nirmal Mukhi,
Stefan Tai, and Sanjiva Weerawarana

The Next Step in Web Services

How three specifications support creating robust service compositions.

The Web services framework intends to provide a standards-based realization of the service-oriented computing paradigm, which has emerged in response to a fundamental shift in the way enterprises conduct their business. Fully integrated enterprises are being replaced by business networks in which each participant provides the others with specialized services. Traditional IT infrastructures in which infrastructure and applications were managed and owned by one enterprise are giving way to networks of applications owned and managed by many business partners. Standards and the pervasiveness of network technologies provide the technology support for this trend.

This new computing environment defines a set of requirements that distinguish SOC from other computing paradigms. To operate in a SOC environment, applications ("services") must declaratively define their functional and nonfunctional requirements and capabilities in an agreed,

machine-readable format. Based on declarative service descriptions, automated service discovery, selection and binding become a native capability of SOC middleware and applications. A consequence of the dynamic binding capability is a looser coupling model between applications.

A componentized model emerges as the natural architecture for the SOC model. Services become the basic building blocks out of which new applications are created, and service composition becomes the main concern of the application development process. A service composition combines services following a certain composition pattern to achieve a business goal, solve a scientific problem, or provide new service functions in general. Service compositions may themselves become services, making composition a recursive operation. Service composition provides a mechanism for application integration that seamlessly supports cross-enterprise (business-to-business) and intra-enterprise application integration.

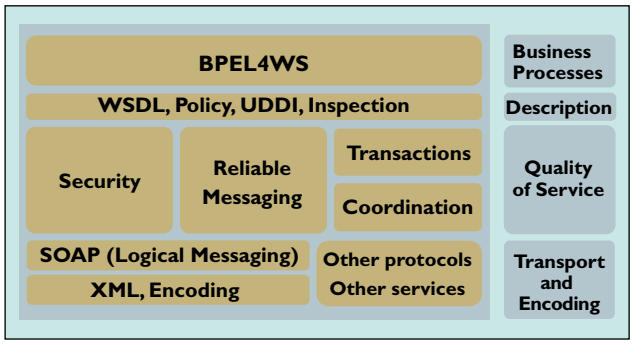

Figure 1. The Web services stack.

To support the SOC architecture, Web services must provide standards-based definitions of an interoperability communication protocol, mechanisms for service description, discovery, and composition as well as a basic set of quality of service (QoS) protocols. The initial trio of Web services specifications, SOAP, WSDL and UDDI, provided open XML-based mechanisms for application interoperability, service description, and service discovery. For a detailed look at these specifications and how they fit together, see [3]. SOAP is now a W3C standard, and WSDL and UDDI are being considered by standard bodies.

To move beyond this basic framework ("describe, publish, interact") mechanisms for service composition and quality of service protocols are required. Several specifications have been proposed in these areas, most notably the Business Process Execution Language for Web Services (BPEL4WS) [1] for service composition, Web services coordination (WS-Coordination) [10] and Web services transactions (WS-Transaction) [11] to support robust service interactions, Web services security (WS-Security), and Web services reliable messaging (WS-ReliableMessaging). All these aspects are critical elements of meaningful business interactions. The descriptive capabilities of WSDL are enhanced by the Web Services Policy Framework (WS-Policy), which extends WSDL to allow the encoding and attachment of QoS information to services in the form of reusable service "policies." It is important to note that the Web services stack (see Figure 1) is designed modularly, allowing one to use only the pieces of the stack required in a particular setting; for example, one may use a local proprietary registry to find Web services.

Here we focus on the problem of creating service compositions and review how three specifications, BPEL4WS, WS-Coordination, and WS-Transaction, support creating robust service compositions. BPEL4WS (or BPEL for short) provides a mechanism for defining service compositions in the form of choreographies of Web services; a choreography consists of the aggregation of services according to certain business rules. WS-Coordination and WS-Transaction complement BPEL to provide mechanisms for defining specific standard protocols for use by transaction processing systems, workflow systems, or other applications that wish to coordinate multiple Web services. We describe the key aspects of each specification and finally explain how the three fit together to provide a framework for composing and coordinating distributed Web services.

Service Composition

BPEL defines a language for creating service compositions in the form of business processes and is now being standardized by the Organization for the Advancement of Structured Information Standards (OASIS). Overviews and comparisons of several proposed Web services-based business process modeling standards have been presented at www.ebml.org and [12].

The Nature of BPEL Compositions. BPEL supports a process-oriented form of service composition: each BPEL composition is a business process or workflow that interacts with a set of Web services to achieve a certain goal. BPEL compositions are thus called processes; the services the process inter-

acts with are called partners. A process, like any Web service, supports a set of WSDL interfaces that enable it to exchange messages with its partners. The process interacts with them by invoking the operations they support and receiving messages through the process service interface. Figure 2 illustrates one such set of interactions. Observe that in this model, the constituent services (partners) are external to the composition itself.

The interaction between a BPEL composition and its partners is assumed to be, in the general case, a peer-to-peer conversational one in which each party invokes operations on (or sends messages to) the public interfaces of the other. This general model covers the more traditional partner roles found in client/server environments: certain (client) applications may only use the process as a service without offering any function themselves, while others are simply used (invoked) by the process as utility services.

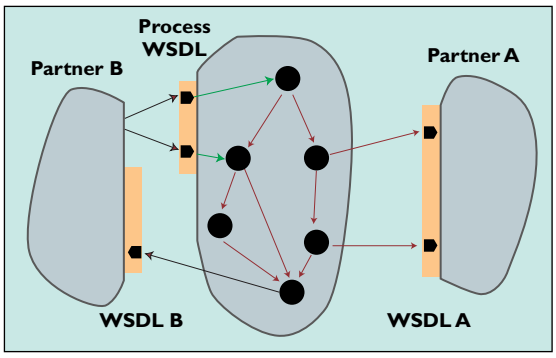

Figure 2. A BPEL process interacting with two partners: black circles are activities; black arrows are control links; other arrows illustrate message exchanges.

Defining Business Protocols. The core of a BPEL process composition is thus the definition of the message exchanges that take place between the process and each one of its partners. First, partners are defined in a BPEL process by declaring the WSDL interfaces over which the interaction with each partner will take place, including both the interfaces supported by the partner and by the process. To achieve the goal of providing multi-protocol access to a service, WSDL separates abstract service descriptions (interfaces and messages) from specific deployments of the service [3]. Only abstract interfaces are used in the partner definitions, which makes BPEL compositions platform-and transport-independent: the same BPEL process may be accessed over standard SOAP over HTTP, as well as, say, J2EE protocols such as IIOP and JMS [8].

Once the process partners are defined, a set of primitive activities are used to define how messages are exchanged with each partner. A message is sent to a partner using an invoke activity; the process can wait for a process operation to be invoked by some external client using the receive activity; the response of an input-output operation is sent back using the reply activity. In addition, BPEL provides other primitive activities to perform actions such as signaling faults, terminating the process execution, and manipulating data.

These primitive activities can then be combined into complex algorithms using the structured activities provided in the language. These are the ability to define an ordered sequence of steps (sequence), to have conditional branching (switch), to define a loop (while), to execute one of several alternative paths (pick) and to indicate that a collection of steps should be executed in parallel (flow). Within activities executing in parallel, execution order constraints can be specified by defining control links between the activities. All structured activities can be recursively combined.

To maintain the state of an interaction BPEL uses a mechanism called message correlation. Key fields within the data messages are identified that will be used to correlate messages received from a partner to a particular conversation. For example, in an order fulfillment system, the invoice number may be used to identify the conversation between the process and one of its partners. The stateful nature of business interactions is thus naturally captured by business data fields, as opposed to middleware and system-generated artifacts.

Correlation-based stateful interactions map quite naturally into an "instance-oriented" process lifecycle model. Unlike in traditional object systems, process instances are not created via a factory or referenced using an explicit instance identifier. Instead, some of the messages sent to a BPEL process implicitly lead to the creation of a new process instance; a set of correlation fields is then initialized that will allow the location of the process instance in subsequent interactions. Note that a process may have more than one set of correlation fields: different correlation sets may be used for each partner of the process; moreover, the interaction with each partner may rely on different correlation sets at different times.

Fault Handling and Compensation. BPEL provides extensive support for dealing with errors, through the use of fault and compensation handlers. Fault handlers provide a structured model to deal with errors occurring within the process; the model is similar to "try-catch" blocks in Java, but results in

significant simplification of the process modeling required to handle faults [2, 4]. Fault handling is closely tied in BPEL to the notion of compensation. Compensation [5, 6, 9] is a common technique used to "undo" the effects of actions that have already been completed (such as canceling a prior completed flight booking in a travel reservation process). A process designer defines the compensating actions to be performed should an error occur in the course of executing the primary process. Hence, compensation handlers are typically invoked by a fault handler. The units of fault handling and/or compensation in BPEL processes are called scopes. If a fault occurs in a scope, all activities are disabled and the fault is either handled or thrown to the enclosing scope. Completed scopes nested within a faulting one are compensated in reverse order of completion. The BPEL compensation model is closely related to the protocols defined by the WS-Transaction specification, presented later.

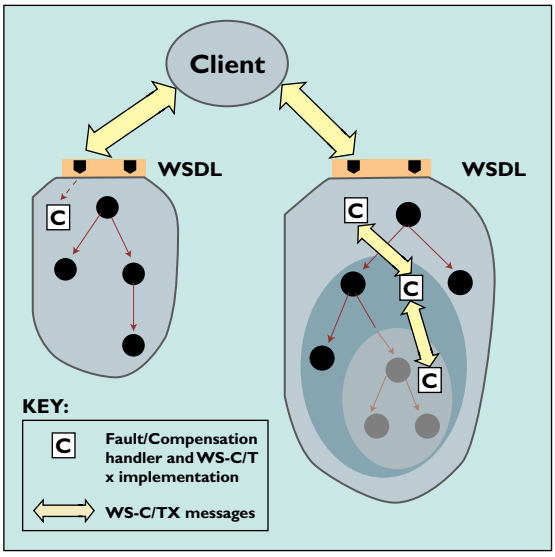

Figure 3. Using WS-Coordination/ WS-Transaction to coordinate BPEL Web services and nested BPEL scopes.

Service Composition Using BPEL4WS, WS-Coordination, and WS-Transaction

WS-Coordination and WS-Transaction are two specifications that address the reliable, transactional coordination of Web services. They can be used individually, or in combination with BPEL. Each of these specifications has a well-defined purpose in the context of robust service compositions:

- BPEL allows a set of existing Web services to be composed into a new Web service using well-defined process modeling constructs;
- WS-Coordination is a general framework for implementing specific coordination types, where the coordination of Web services requires a shared context; and
- WS-Transaction defines two particular coordination types for (short-running) atomic transactions and (long-running) business transactions.

The combined use of these three specifications allows the BPEL composition model to be extended with distributed coordination capabilities. The fault and compensation handling relationship between BPEL scopes can be expressed as a WS-Coordination coordination type, and distributed BPEL implementations can register for fault handling and compensation notifications using the coordination framework. WS-Coordination defines the coordination context for use in environments where BPEL scopes are distributed or span different vendor implementations; a context that is understood across the participants (BPEL implementations) is required in such environments. The use of the Business Activity coordination type defined in the WS-Transaction specification for coordinating nested BPEL scopes is also described in [1]. Figure 3 illustrates these two uses of WS-Coordination/WS-Transaction to implement distributed BPEL process coordination and the coordination of nested BPEL scopes.

WS-Coordination and WS-Transaction

WS-Coordination and WS-Transaction address the problem of coordinating multiparty service interactions. The rationale behind WS-Coordination is to provide generic coordination mechanisms that can be extended for specific coordination protocols. Such coordination includes the execution of short-running transactions within an organization (similar to traditional distributed transactions) and long-running transactions across organizations. WS-Transaction defines two such specific coordination protocols for atomic (short-running) and business (long-running) transactions. Another notable specification in this area is the Business Transaction Protocol (BTP), described in [7].

WS-Coordination. WS-Coordination defines a framework that supports the notion of pluggable coordination models, similar to frameworks for extended distributed object transactions like the OMG/J2EE Activity Service for Extended Transac-

tions (JSR 95). The proposed approach to implementing a specific coordination model is to extend the mechanisms provided by WS-Coordination.

Specific coordination and transaction models are each represented as a coordination type supporting a set of coordination protocols; a coordination protocol is the set of well-defined messages that are exchanged between Web services participants. Coordination protocols such as completion protocols, synchronization protocols, or outcome notification protocols address the problem of correct execution of a set of distributed activities to reach a consistent, defined outcome.

The WS-Coordination framework defines three main elements commonly required by different coordination models:

- A CoordinationContext, the shared, extensible context representing the coordination that is propagated to the distributed participants;
- An Activation service, the service used by clients to create a coordination context; and
- A Registration service, the service used by participants to register resources for inclusion in specific coordination protocols.

The Activation service and the Registration service are generic. Together with the set of services that represent the specific coordination protocols for a given coordination type, they make up a Coordination service (or coordinator for short).

In order to coordinate a set of Web services, the coordination client starts the coordination by sending a request message to the Activation service of a chosen coordinator. A CoordinationContext is then created by the Activation service. The CoordinationContext contains a global identifier, expiration data, the port reference for the Registration service, and can also be extended to include other information relevant to specific coordination protocols (such as an isolation level element for an atomic transaction). The port reference is a WSDL definition type that is used to identify an individual port; it consists of the URI of the target port as well as contextual information that may include service-specific instance data.

Whenever the client initiates an invocation on a Web service, the CoordinationContext must be propagated along with the application message (WS-Coordination implementations can be used to append the context to the application message). The service being invoked can then find out about the Registration service's port reference (contained in the context) to register for the coordination pro-

WEB SERVICES ARE MOVING FROM THEIR INITIAL "DESCRIBE, PUBLISH, INTERACT" CAPABILITY TO A NEW PHASE IN WHICH ROBUST BUSINESS INTERACTIONS ARE SUPPORTED.

tocol that it wishes to participate in; it can either directly register with the client's coordinator, or use another (local) coordinator, and have the coordinator register with the client's coordinator (see [11] for an example in the context of atomic transactions).

WS-Transaction: Atomic Transactions and Business Activities. WS-Transaction leverages WS-Coordination by defining two particular coordination types: "Atomic Transaction (AT)" and "Business Activity (BA)." ATs model short-running atomic transactions; BAs model business transactions that are potentially long-lived.

ATs compare to traditional distributed transactions. The AT coordination type supports the property of atomicity ("all-or-nothing" with respect to the execution of distributed Web services operations) based on the premise that the data resources manipulated by service operations can be held. The coordination type correspondingly comprises protocols common to atomic transactions, including the two-phase commit protocol.

The BA coordination type supports transactional coordination of potentially long-lived activities. BAs do not require resources to be held, but business logic to be applied to handle exceptions. Participants are viewed as business tasks that are children to the BA for which they register. The participant list is dynamic (a participant may choose to leave a transaction) and participants are loosely-coupled. Unlike in the more tightly coupled two-phase commit protocol, a BA participant may also declare its outcome before being solicited to do so.

Conclusion

The Web services framework has emerged to address the movement toward service-oriented computing, where applications are offered as services both within and across enterprises. Aiming to leverage the heterogeneity of the IT landscape, its key enabler is in the definition of a modular technology stack based on open, XML-based standards. As the technology continues to evolve, a number of specifications are being proposed to address the areas necessary to support SOC, such as security, reliability, and service composition. The specifications presented here demonstrated how Web services are moving from their initial "describe, publish, interact" capability to a new phase in which robust business interactions are supported.

SOC is still in the early stages of development; fully dynamic business interactions following the SOC model are not foreseen in the immediate future. The specifications presented here, however, are important milestones on the way toward a complete standards-based framework to support service orientation. Other specifications are already filling in the remaining gaps and industry support is slowly consolidating behind a set of basic standards. Over the next few years, we will likely see the deployment and adoption of the full SOC model by business and scientific communities. C

References

1. *Business Process Execution Language for Web Services, version 1.1.*; www.ibm.com/developerworks/library/ws-bpel/.
2. Curbera, F., Khalaf, R., Leymann, F., and Weerawarana, S. Exception handling in the BPEL4WS language. In *Proceedings of the International Conference on Business Process Management, BPM 2003* (Eindhoven, The Netherlands, June 2003).
3. Curbera, F. et al. Unraveling the Web services Web: An introduction to SOAP, WSDL, and UDDI. *IEEE Internet Computing 6*, 2 (Mar./Apr. 2002).
4. Hagen, C. and Alonso, G. Flexible exception handling in the OPERA process support system. In *Proceedings of the International Conference on Distributed Computing Systems (ICDS 98)*, 526–533.
5. Leymann, F. Supporting business transactions via partial backward recovery in workflow management systems. In *Proceedings of BTW '95*, Springer-Verlag, Berlin, 1995.
6. Leymann, F. and Roller, D. *Production Workflow*. Prentice Hall, 2000.
7. Little, M. Transactions and Web services. *Commun. ACM 46*, 10 (Oct. 2003).
8. Mukhi, N., Khalaf, R., and Fremantle, P. Multi-protocol Web services for enterprises and the grid. In *Proceedings of EuroWeb '02* (Oxford, UK, December 2002).
9. van der Aalst, W. and van Hee, K. *Workflow Managment: Methods, Models, and Systems*. MIT Press, 2002.
10. *Web Services Coordination (WS-Coordination) 1.0*; www-106.ibm.com/developerworks/library/ws-coor.
11. *Web Services Transaction (WS-Transaction) 1.0;* www-106.ibm.com/developerworks/library/ws-transpec.
12. *Workflow Patterns, Standard Evaluation*. Technische Universiteit Eindhoven; tmitwww.tm.tue.nl/research/patterns/standards.htm.

Francisco Curbera (curbera@us.ibm.com) is a research staff member at IBM Research in Hawthorne, NY.
Rania Khalaf (rkhalaf@us.ibm.com) is a software engineer at IBM Research in Hawthorne, NY.
Nirmal Mukhi (nmukhi@us.ibm.com) is a software engineer at IBM Research in Hawthorne, NY.
Stefan Tai (stai@us.ibm.com) is a research staff member at IBM Research in Hawthorne, NY.
Sanjiva Weerawarana (sanjiva@us.ibm.com) is a research staff member at IBM Research in Hawthorne, NY.