Chapter II
Web Services Technology:
An Overview

Fani A. Tzima
Aristotle University of Thessaloniki, Greece

Pericles A. Mitkas
Aristotle University of Thessaloniki, Greece

ABSTRACT

This chapter examines the concept of service-oriented architecture (SOA) in conjunction with the Web services technology as an implementation of the former’s design principles. Following a brief introduction of SOA and its advantages, a high-level overview of the structure and composition of the Web services platform is provided. This overview covers the core Web services specifications as well as features of the extended architecture stack, which together form a powerful and robust foundation for building distributed systems. The chapter concludes with a discussion of the scope of applicability of SOA and Web services. The overall goal of this chapter is to portray the key assets of the presented technologies and evaluate them as tools for handling adaptability, portability and interoperability issues that arise in modern business environments.

INTRODUCTION

Nowadays, organizations are facing a highly dynamic and challenging environment, characterized by a rising demand for customized, high quality services and products in several segments of business and industry. This environment, combined with the pace of technological innovation and the globalization of economy, has triggered the development of new value-creating economic paradigms, where the concept of the virtual enterprise (VE) has a central position. Sets of economic actors are combining their resources, forming temporary enterprise alliances (VEs), to effectively respond to the shifts of market demand, identify new opportunities and minimize organizational costs through cooperative and dynamic solutions.
The need to realize new forms of collaboration has forced organizations to shift their focus from intra- to inter-enterprise system and process integration. However, most enterprises have made extensive investments in system resources over the course of years and own an enormous amount of data stored in legacy enterprise information systems (EIS). Since it is impractical to discard existing EIS, there is a constant effort to evolve and enhance them. Thus, IT professionals are currently faced with the challenge of capturing and controlling legacy technology in a way that transcends organizational boundaries and heterogeneities, but also promotes system evolution. In this direction, service oriented architecture (SOA) provides a cost-effective solution, with Web services being a promising implementation, intended to enable the construction of interoperable components that can be assembled and deployed in a distributed environment (Estrem, 2003).

**SERVICE-ORIENTED ARCHITECTURE (SOA)**

**Service-Oriented Architecture Overview**

A service oriented architecture (SOA) is a design principle intended for the construction of reliable distributed systems that deliver functionality as services, with an additional emphasis on loose coupling between interacting services (Srinivasan & Treadwell, 2005). In this context, services are typically characterized by the following properties (Orchard, Ferris, Newcomer, Haas, Champion, Booth & McCabe, 2004):

- **Logical view:** The service is an abstracted, logical view of an actual business-level operation, defined as an implementation-independent interface. Services may be completely self-contained, or they may depend on the availability of other services, or on the existence of specific resources such as a database.

- **Message orientation:** A service communicates with its clients by exchanging messages and is formally defined in terms of the message exchange patterns it supports. The internal structure of the provider and requester agents is deliberately abstracted away in the SOA, in order to maintain control of which aspects of an endpoint are revealed to external services.

- **Description orientation:** A service is described by machine-processable metadata. This description only exposes information important for the use of the service, such as its capabilities, interfaces, policies and supported protocols. Further, the description documents, directly or indirectly, the semantics that will govern the interaction between the requester and provider agents.

- **Granularity:** Services tend to use a small number of operations with relatively large and complex messages. However, various levels of granularity are possible, as services may be individually useful, or they can be integrated to provide higher-level services. Among other benefits, this promotes re-use of existing functionality.

- **Network orientation:** Services tend to be oriented toward use over a network. This property emphasizes the need for services to be automatically discoverable.

- **Platform neutrality:** Messages are delivered through the interfaces using a platform-neutral and standardized format, such as XML.

Additionally, services can participate in a workflow, where the order in which messages are exchanged affects the outcome of the operations performed by a service. This notion is defined as a “service choreography” and is actually a model of the sequence of operations, states and conditions that control the interactions involved in the
Web Services Technology

participating services. The interaction prescribed by a choreography results in the completion of some useful function.

Figure 1 illustrates the interaction cycle of a service. This interaction cycle follows the “find-bind-execute” paradigm and begins with service providers advertising their service (1) through a public registry, used by consumers to look-up available services. A potential client (consumer), that may or may not be another service, queries the registry (2) for a service that matches certain criteria. The registry returns a (possibly empty) list of suitable services along with their endpoint addresses, and the client selects one and passes a request message to it, using any mutually recognized protocol (3). The service may respond (4) either with the result of the requested operation or with a fault message.

The above described interaction cycle, of course, covers only the simplest case. In a real-world setting, such as a commercial application, the process may be significantly more complex—protocol configuration, user authorization, interaction patterns and transaction control are only a few of the issues that may arise and need to be resolved.

Benefits of a SOA

A key feature of SOA is that it promotes loose coupling between software components. The interacting entities (services and/or other software components) have no built-in knowledge of each other and discover the information necessary for their interaction on-the-fly, when needed. This notion of loose coupling, imposed by the architecture, is exactly what turns services into valuable reusable “building blocks” and, more importantly, enables the creation of new services from existing assets and IT infrastructure. In other words, SOA promises to deliver interoperability between heterogeneous applications and technologies.

Clearly, the above benefits are of great value in a dynamic distributed environment. However, SOA also provides an unprecedented level of flexibility, as the benefits of service-oriented development also include the following (Srinivasan & Treadwell, 2005):

- **Flexibility:** Prospective clients can always locate (or relocate) a service, as long as the service registry entry is maintained.
Scalability: Services can be added and removed according to the variations of demand.

Replaceability: Service implementations may be updated or completely altered—provided that the original interfaces are preserved—without any disruption to clients.

Fault tolerance: Clients can always query the registry for alternative services providing the required functionality. In this way, uninterrupted operation is ensured, irrespective of the availability of independent services.

The real value of SOA, though, comes in later stages of development when new applications can be developed entirely, or almost entirely by composing existing services (Newcomer & Lomow, 2005). This may require significant investment in initial service development, but once this point has been reached, the best value for effort can be realized: new applications can be assembled out of a collection of existing, reusable services, with low cost and in fast times, thus resulting in substantial return of investment.

WEB SERVICES

Web services are a promising implementation of the service-oriented architecture, intended to provide a standard means of interoperating between different software applications, running on a variety of platforms and/or frameworks. This interoperability is gained through the use of a set of XML-based open standards for defining, publishing and using Web services.

Web Services: Definitions

A Web service is defined by W3C (Orchard et al., 2004, p. 7) as “a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.”

It should be pointed out that, according to W3C (Orchard et al., 2004), a Web service should be perceived as an abstract notion, whose implementation is realized by use of a concrete agent. In this context, the agent is a basic entity, responsible for the exchange of messages and encapsulating the service, while the service itself is the resource abstractly defined by the provided functionality.

The purpose of a Web service is to provide some functionality on behalf of its owner—a person or organization, such as a business or an individual. The provider entity is the person or organization that provides an appropriate agent to implement a particular service. On the other hand, a requester entity is a person or organization that uses a requester agent to exchange messages with the provider entity’s agent, in order to make use of the latter’s Web service.

The message exchange between the requester and provider agents conforms to certain rules documented in a Web service description (WSD). The WSD is written in Web service description language (WSDL) and provides a machine-processable specification of the Web service’s interface. Starting with the invocation address (network location) of the provider agent, the WSD also defines supported message formats, message exchange patterns, data types, transport protocols and transport serialization formats. In essence, the service description represents the contract governing the mechanics of interaction with a particular service.

Besides the WSD though, there is also a need for the service requester and provider entities to share a common understanding of the purpose and consequences of their interaction. This shared expectation about the behavior of the service constitutes the semantics of a Web service, on which
the two interacting parties should agree upon. This agreement is not necessarily the product of a negotiation process or a legally expressed consensus and may take the form of an informal or implicit contract.

**Web Services Infrastructure**

As an implementation of SOA, Web services employ an infrastructure that provides the following: a discovery mechanism to locate Web services, a service description mechanism defining how to use those services and standard wire formats with which to communicate. Using this infrastructure, a requester entity might engage and use a Web service in many ways, but still the interaction cycle of a Web service is compatible with that defined for a SOA. In general, the following broad steps are required (Orchard et al., 2004), as illustrated in Figure 2:

1. *The requester and provider entities become known to each other*, in the sense that whichever party initiates the interaction must become aware of the other party. Typically, the requester agent will be the initiator and will obtain the invocation address of the provider agent, through the latter’s service description. This description may be available for retrieval from a registry service.

2. *The requester and provider entities agree on the service description (a WSDL document) and semantics that will govern the interaction between the requester and provider agents*. This does not necessarily presuppose communication or negotiation between the requester and provider. A shared understanding of the service description and semantics and a commitment to uphold them is sufficient and can be achieved in various ways, such as:
   
a. An explicit agreement through a communication or negotiation procedure.

   b. The service description and semantics being published by the provider entity in the form of a “contract” that the requester entity must accept unmodified as conditions of use.

   c. The service description and semantics (excepting the network address of the particular service) being published by the requester entity in the form of a specification that the provider entity must conform to.

   d. The service description and semantics (excepting the network address of the particular service) being a defined industry standard that both parties

---

*Figure 2. The general process of engaging a Web service*
3. The service description and semantics are realized by the requester and provider agents, in the sense that both the semantics and the service description must somehow be input to, or implemented in, both sides before the two agents can interact.

4. The requester agent and provider agent exchange SOAP messages on behalf of their own entities, thus performing some desirable task.

Some of these steps may be automated, while others may be performed manually.

Core Web Services Specifications

The core of the Web services architecture consists of specifications, such as XML, SOAP and WSDL that support the interaction of a Web service requester with a web service provider. The potential discovery of the web service description may be realized as a universal description, discovery and integration (UDDI) or other type of registry. Though it is clear that a service registry is a required part of the Web services platform, UDDI has not yet been widely accepted by corporations, as most of them are reluctant against the notion of a public registry, due to security and intellectual property concerns.

The technologies considered in this section, in relation to the Web services Architecture, are XML, SOAP, WSDL and UDDI.

XML (eXtensible Markup Language)

XML solves the key technology requirement for the success of Web services: the requirement for a standardized, platform-neutral, flexible and inherently extensible data format. In this sense, the XML initiative within the W3C provides the core standards that Web services are based on. The important aspects of XML, for the purposes of this architecture, are the core syntax itself, the concepts of the XML Infoset, XML Schema and XML Namespaces.

A Web service message is an XML document information item as defined by the XML Information Set, or Infoset (XML Information Set, 2004). The Infoset is a formal set of information items that comprise an abstract description of an XML and map to the various features in an XML document, such as elements, attributes, namespaces and comments. Each information item has an associated set of properties that provide a more complete description of the item. Infoset’s information items along with their associated properties provide for a consistent and rigorous set of definitions that can be used by other specifications when referring to the information in a well-formed XML document.

The abstract data model defined by the Infoset is compatible with the text-based XML 1.0 (extensible markup language (XML) 1.0 Fourth Edition, 2006) and the foundation of all XML specifications: XML Schema (XML Schema, 2004), XML Query (XML Query Working Group Public Page), and XSLT 2.0 (XSL Transformation Version 2.0, 2006). By basing the Web services architecture on the XML Infoset rather than on a specific representation format, both the architecture and its core protocols are compatible with alternative encodings and serialization formats, allowing for broader interoperability between agents in the system.

SOAP (Originally “Simple Object Access Protocol”)

SOAP (SOAP 1.1, 2000) provides a simple and lightweight mechanism for exchanging structured and typed messages in a distributed environment. It is an XML-based standard and defines a message as an “envelope” containing an optional header element and a mandatory body element (Figure 3).

The envelope is the root element of a SOAP
message and specifies its XML namespace and the encoding. The body element is always the last child element of the envelope and acts as a container for the content of the message, intended for its ultimate recipient. Finally, the header element provides a generic mechanism for adding referencing capabilities to SOAP messages. If a header is provided, it extends the SOAP message in a modular way, indicating additional processing that is to be done at an intermediate node, independent of the message’s final destination. Typically, the SOAP header is used to convey security-related information to be processed by runtime components.

In addition to the message format described above, SOAP also defines a transport binding framework for exchanging envelopes using a variety of underlying network protocols, including HTTP, SMTP, FTP, RMI/IIOP, or even proprietary messaging protocols. Moreover, there are three optional components to the SOAP specification: (1) a serialization framework defining encoding rules for expressing instances of application-defined data types such as numbers and text, (2) a convention for representing remote procedure calls (RPC) and responses, and (3) a set of rules for using SOAP with HTTP/1.1.

SOAP was originally an acronym for Simple Object Access Protocol, but since SOAP Version 1.2 (SOAP 1.2 Part 0, 2003; SOAP 1.2 Part 1, 2003) it is technically no longer an acronym.

**WSDL (Web Service Description Language)**

WSDL (WSDL Version 2.0, 2006) is a widely adopted mechanism for describing the basic characteristics of a Web service. A WSDL description is a first step in automatically identifying all characteristics of the target service starting with the messages that are exchanged between the requester and provider agents. The messages are described abstractly and then bound to concrete physical deployment information to define an endpoint. Typically, messages are bound to the SOAP protocol and the HTTP transport, but these are not the only alternatives. Since the notation that a WSDL file uses to describe message formats is based on XML Schema, the described service interface can be mapped to any implementation language, platform, object model or messaging system.

In addition to describing message contents, a WSDL document uses the following elements in the definition of a web service (Figure 4):

- **Types:** A container for data type definitions using some type system (such as XML Schema Definition, XSD)
• **Message**: An abstract, typed definition of the data being communicated
• **Operation**: An abstract description of an action supported by the service
• **Port type**: An abstract set of operations supported by one or more endpoints
• **Binding**: A concrete protocol and data format specification for a particular port type

**Port**: A single endpoint defined as a combination of a binding and a network address

**Service**: A collection of related endpoints

### UDDI (Universal Description, Discovery and Integration)

The universal description, discovery, and integration protocol (UDDI Executive Overview, 2004)

---

**Figure 4. Sample WSDL document**

```xml
<definitions
  targetNamespace="http://OnLineMath.com"
  xmlns="http://schemas.xmlsoap.org/soap/">
  <message name="SubSoapIn">
    <part name="parameters" element="s:Sub"/>
  </message>
  <message name="SubSoapOut">
    <part name="parameters" element="s:SubResponse"/>
  </message>
  <message name="SCalc/Service1.asm">
    <operation name="Sub">
      <input message="s:0/SubSoapIn"/>
      <output message="s:0/SubSoapOut"/>
    </operation>
  </message>
  <binding name="Service1Soap" type="s:0:Service1Soap">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
  </binding>
  <service name="Service1">
    <port name="Service1Soap" binding="s:0:Service1Soap">
      <soap:address location="http://OnLineMath.com/WSCalc/Service1.asmx"/>
    </port>
  </service>
</definitions>
```
defines a standard method for publishing and discovering services through a common directory of Web service information. It is an open industry initiative (sponsored by OASIS) designed to be interrogated by SOAP messages and to provide access to WSDL documents. Additionally, the specification defines a UDDI schema identifying the types of XML data structures that comprise an entry in the registry for a service.

UDDI registries can be deployed in one of three ways: public, extra-enterprise or intra-enterprise UDDI registries. Intra- and extra-enterprise approaches employ private registries deployed by an organization, or a group of business partners respectively. On the other hand, the UDDI business registry (hosted by a group of vendors led by Microsoft, IBM and SAP) is a public UDDI registry, serving as both a resource for Internet-based Web services and a test bed for Web services developers.

The fundamental goal of UDDI is to enable businesses to register information about the services they provide, in order for prospective clients to locate them easily by consulting a repository. A UDDI business registration consists of three components, commonly referred to as “white pages”, “yellow pages” and “green pages” information:

- White pages provide basic information about a company, such as the business name, address and contact information. White pages also allow the discovery of services according to unique business identifiers.
- Yellow pages describe business services using different categorizations—“taxonomies” in UDDI terminology. This information allows for the discovery of business services based on their categorization.
- Green pages provide technical information on the behaviors and supported functions of services hosted by a business. This includes references to specifications for Web services, as well as support for pointers to various file and URL based discovery mechanisms if required. Moreover, green pages in UDDI are not limited to describing XML-based Web services: the technical description is applicable to any service type exposed by a business entity.

UDDI version 2.0 was approved in 2003 and was integrated into the Web Services Interoperability (WS-I) standard as a central pillar of Web services infrastructure. The UDDI Version 3.0 specification (UDDI Version 3.0.2, 2004), approved by the OASIS International Standards Consortium in early 2005, represents another significant milestone in UDDI’s evolution, as it provides key capabilities for enterprise-level deployment and is a mature, well-supported standard.

Extended Web Services Specifications

Although the core Web services specifications (XML, SOAP, WSDL and UDDI) comprise an accepted industry-wide basis for interoperability, significant effort has been put in enhancing the scope of the Web services platform and addressing higher-level issues that arise in real-world application domains. In this direction, a wide array of Web services specifications for security, reliability, transactions, metadata management and orchestration have emerged and are currently in their way toward standardization. Figure 5 provides an illustration of some of these technology families.

The extended “architecture stack”, depicted above, will hopefully provide WS-based solutions with the necessary qualities of service to support enterprise-level projects.

Metadata Management

Metadata management includes the description information about Web services necessary for a
service requester to invoke them. In order for this invocation to be successful, the requester needs to effectively construct a message—a message body, including its data types and structures, and message headers—processable by the service provider.

Effective construction of messages depends on the metadata published by the provider and discovered by the requester. Such metadata include not only the data types and structures to be sent, but also the additional qualities of the service provided (if any), such as security, reliability or transactions. The absence of one or more of these features may hinder successful message processing.

Metadata extended specifications include among others:

- **WS-Addressing**: Enables messaging systems to support message transmission in a transport-neutral manner through networks that include processing nodes such as endpoint managers, firewalls and gateways. It provides support for endpoint addressing and reference properties associated with endpoints for many of the other extended specifications.

- **WS-Policy**: Provides a general-purpose framework and corresponding syntax to describe and communicate quality of service requirements of a Web service; policy declarations cover various aspects of security, transactions and reliability.

- **WS-MetadataExchange**: Defines request-response message pairs to incrementally retrieve three types of metadata: WSDL files, policy definitions and XML schemata associated with a Web service.

### Addressing

SOAP messages must include the endpoint address information within the message, as no directory of Web services endpoint addresses exists on the Web. Proper message routing relies on a common mechanism, which will allow for critical messaging properties to be carried across multiple transports in cases of communication failure or complicated message exchange patterns. WS-Addressing replaces earlier proposals called WS-Routing, WS-Referral and SOAP routing protocol (SOAP-RP) and promises to provide an effective solution for message transmission problems.
The WS-Addressing (WS-Addressing, 2004) specification specifies how addressing information (typically used for routing) should be conveyed in a SOAP message, by defining common headers such as message identifiers and to and from fields (Figure 6). These headers rely on a WS-Addressing-defined structure called an endpoint reference that is an access point of a Web service and bundles together the information needed to properly address a SOAP message.

Policy

An emerging problem in Web services is describing the capabilities and requirements of endpoints, in addition to the data requirements for the messages expressed in the WSDL file. As WSDL does not cover many aspects of a Web service’s operation, such as security, transaction or reliability, these aspects have to be specified or negotiated on a point-to-point basis. WS-Policy aims to fill this gap by providing an open framework for the definition of extended requirements, capabilities and configuration issues—together referred to as the policy of a Web service.

WS-Policy provides a machine-readable expression of assertions necessary for achieving interoperability for the extended features. It specifies a base set of constructs that can be used and extended by other Web service specifications to describe a broad range of service requirements and capabilities. For example, WS-PolicyFramework (WS-Policy 1.2 Framework, 2006) describes the overall framework of expressing policy assertions and combining them. WS-PolicyAssertions (WS-Policy Assertions Version 1.0, 2002) defines a set of common message policy assertions, such as document encoding or specification support, that can be specified within a policy. Finally, WS-PolicyAttachment (WS-Policy 1.2 Attachment, 2006) defines general-purpose mechanisms for associating policies with XML elements, WSDL-type definitions, and UDDI entries.

Acquiring Metadata

WSDL and WS-Policy both define metadata formats, but do not specify mechanisms for acquiring or accessing metadata for a given service. In general, service metadata can be discovered using a variety of techniques—a requester might obtain it using WS-MetadataExchange or another similar mechanism that queries the WSDL and associated policy files directly.

WS-MetadataExchange (WS-MetadataExchange Version 1.1, 2006) is a SOAP-based access protocol for metadata designed to provide information about a Web service description, essentially replacing UDDI that, despite its existence, does not provide the metadata management facilities required to support interoperability requirements at the extended specification level. The WS-MetadataExchange specification, though, ensures that service requesters have all the necessary information to achieve interoperability with providers using extended features.

Security

Security concerns apply at every level of the Web services architecture, involving threats to the host system, the application and the entire network infrastructure. To secure Web services, a range of XML-based security mechanisms are built around encryption, authentication and authorization mechanisms and typically include comprehensive logging for problem tracking. Currently, the industry has achieved consensus around a single specification framework, WS-Security (WS-Security Policy Version 1.1, 2004), although ongoing work is necessary to complete the profiles and additional related specifications.

WS-Security describes enhancements to SOAP messaging to provide quality of protection through message integrity, message confidentiality, and single message authentication. The WS-Security specification supports tokens for common security mechanisms in use today, such
as Kerberos tickets and X.509 certificates, and can use XML Encryption and XML Signature technologies for further protecting the message contents. XML-based security tokens also include the security assertion markup language (SAML) (OASIS Security Services TC Webpage).

WS-Security is further extended and complemented by additional specifications, including:

- **WS-SecurityPolicy:** Indicates the policy assertions for WS-Policy, which apply to WS-Security (WS-SecurityPolicy Version 1.1, 2005)
- **WS-Trust:** Defines extensions building on WS-Security to request and issue security tokens (such as Kerberos tickets) and to manage trust relationships (WS-Trust, 2005)
- **WS-SecureConversation:** Defines extensions building on WS-Security to provide secure communication; specifically, it defines mechanisms for establishing and maintaining persistent security contexts, and deriving session keys from security contexts (WS-SecureConversation, 2005).
- **WS-Federation:** Defines mechanisms to bridge multiple security realms into a federated session by allowing and brokering trust of identities, attributes and authentication between participating Web services; the defined mechanisms allow for a Web service to be authenticated only once in order to access all federated Web services (WS-Federation Version 1.0, 2003).

XML-based security technologies are also important for protecting the XML data (metadata or content) in a SOAP message. Such technologies include:

- **XML encryption:** Specifies a process for encrypting data, using a variety of supported encryption algorithms; the encryption result is an XML Encryption element that contains or references the cipher data, ensuring that the contents of the document cannot be intercepted and read by unauthorized persons (XML Encryption Syntax and Processing, 2002).
- **XML signature:** Specifies XML digital signature processing rules and syntax. The specification provides integrity, message authentication and signer authentication services for data of any type, ensuring that documents cannot be altered in transit and are received exactly once (XML Signature Syntax and Processing, 2002).

### Reliable Messaging

In any distributed system there are fundamental limits to the reliability of messages sent over a public network. However, in practice there are techniques that can be used to increase the reliability of messages and provide valuable feedback in cases where communication fails. The Web services architecture does not by itself provide specific support for reliable messaging, or for reporting in the event of failure. However, since all messages are structured according to SOAP, overall message reliability can be incorporated within the SOAP message structure.

Though the industry has not yet achieved consensus on a single, unified set of specifications, there are some examples of specifications for an acknowledgement infrastructure that leverage the SOAP extensibility model:

- **WS-Reliability:** An OASIS standard (WS-Reliability 1.1, 2004)
- **WS-ReliableMessaging:** From BEA, IBM, Microsoft and TIBCO (WS-ReliableMessaging, 2005)

Reliable messaging specifications define mechanisms for ensuring that the sending and receiving parties know whether or not a message was delivered. These mechanisms guarantee reliable delivery of SOAP messages, over poten-
Web Services Technology

tially unreliable networks, with no duplicates, and guaranteed message ordering. Additionally, reliable messaging automates recovery from certain transport-level error conditions, such as in the presence of software component, system or network failures.

In the general messaging area, there are also specifications for message exchange patterns such as event notification and publish/subscribe, which basically extends the asynchronous messaging capability of Web services. Specifications in this area include:

- **WS-Eventing**: Describes how to construct an event-oriented message exchange pattern, allowing Web services to act as event sources for subscribers; it defines the operations required to manage subscriptions to event sources, as well as how the actual event messages are constructed (WS-Eventing, 2006).

- **WS-Notification**: Implements the notification pattern, where a service provider, or other entity, initiates messages based on a subscription or registration of interest from a service requestor; it defines how the “publish/subscribe” pattern can be realized using Web services. This includes brokered as well as direct “publish/subscribe” patterns, allowing the publisher/subscribers to be decoupled and provides greater scalability (OASIS WSN TC Webpage).

**Transactions**

Transactions are a fundamental concept in building reliable distributed applications, as they allow multiple operations, usually on persistent data, to succeed or fail as a unit. Traditionally, transactions have held the following properties collectively referred to as *ACID*:

- **Atomicity**: A transaction allows for the grouping of multiple operations, so that either all of them are performed or none of them is.

- **Consistency**: The application returns to a valid state when the transaction ends, hence the transaction should leave the processing resource in a logical state instead of ambiguity.

- **Isolation**: Operations in a transaction appear isolated from all other operations, until the transaction completes successfully.

- **Durability**: Once a transaction successfully completes, the changes are guaranteed to persist, even in the case of system failure.

Moreover, one of the most important aspects of transaction processing technologies is their ability to recover an application to a known state following an operating system or hardware failure.

A Web service environment may require only the coordination behavior provided by a traditional transaction mechanism to control the operations and outcome of an application. However, it may also require the capability to handle the coordination of processing outcomes or results from multiple services, in a more flexible manner. This requires more relaxed forms of transactions—not strictly abiding to the ACID properties—such as collaborations, workflow, real-time processing and so forth. In this direction, Web services transaction specifications extend the concept of the transaction coordinator, adapt the familiar two-phase commit protocol for Web services, and define new extended transaction protocols for more loosely coupled Web services and orchestration flows.

The specifications in this area include (Newcomer & Lomow, 2005):

- **WS-Transactions family** from BEA, IBM, and Microsoft:

  - **WS-AtomicTransaction**: Provides the definition of the atomic transaction coordination type that is to be used with
the extensible coordination framework described in the WS-Coordination specification; the specification defines three specific agreement coordination protocols (completion, volatile two-phase commit, and durable two-phase commit) for short-lived distributed activities that have all-or-nothing semantics (WS-AtomicTransaction 1.1, 2006).

- **WS-BusinessActivity:** Provides the definition of the business activity coordination type that is to be used with the extensible coordination framework described in the WS-Coordination specification; the specification defines protocols for long-running distributed activities (WS-BusinessActivity 1.1, 2006).

- **WS-Coordination:** Describes an extensible framework for providing protocols that coordinate the actions of distributed applications (WS-Coordination 1.1, 2006).

- **WS-Composite Application Framework (WS-CAF)** from OASIS that proposes standard, interoperable mechanisms for managing shared context and ensuring business processes achieve predictable results and recovery from failure. WS-CAF has three sub-parts (OASIS WS-CAF TC Webpage):
  - **WS-Context (WS-Context):** Defines a standalone and lightweight context management system (WS-Context, 2006)
  - **WS-CoordinationFramework (WS-CF):** A sharable mechanism to manage context augmentation and lifecycle, and guarantee message delivery; a software agent is defined as the coordinator for the basic context specification and the pluggable transaction protocols in the WS-TransactionManagement specification (WS-CF, 2005).
  - **WS-TransactionManagement (WS-TXM):** Comprises three distinct protocols for interoperability across multiple transaction managers and supports multiple transaction models (two phase commit, long running actions and business process flows) (WS-TXM Version 1.0, 2003)

### Orchestration and Choreography

Terms such as orchestration and choreography are used to describe the composition of Web services in a process flow. More formally, the term **orchestration** (Peltz, 2003, p. 3) “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.” On the other hand, the term **choreography** (Orchard et al., 2004, p. 32) “defines the sequence and conditions under which multiple cooperating independent agents exchange messages in order to perform a task to achieve a goal state.” In other words, a choreography tracks the sequence of operations, states and messages that may involve multiple parties and multiple sources, interacting for the completion of some useful function.

There is an important distinction between Web services orchestration and choreography. An orchestration refers to a business process that is executed by a single Web service interacting with other parties—both internal and external Web services—but controlling this interaction from its personal perspective. Choreography, on the other hand, is more collaborative in nature, and is typically associated with the public message exchanges that occur between multiple Web services, rather than with a single party’s executable
business process. In this context, a choreography permits the definition of how several Web services can be composed, how service roles and associations can be established, and how the state, if any, of composed services is to be managed.

Orchestration

The industry has reached a consensus around a single orchestration specification: the business process execution language for Web services (BPEL4WS). BPEL4WS (Business Process Execution Language for Web Services Version 1.1, 2003) is essentially a layer on top of WSDL, with WSDL defining the specific operations allowed and BPEL4WS defining how the operations can be sequenced. More specifically, BPEL4WS includes support for both basic and structured activities, with structured activities managing the overall process flow and specifying which basic activities should be executed and in what order. Moreover, BPEL4WS provides a robust mechanism for handling transactions and exceptions, building on top of the WS-Transactions family of specifications.

It should be pointed out, that BPEL4WS provides support for both executable and abstract business processes. An executable process essentially models a private workflow, while abstract processes—modelled as business protocols—focus more on the choreography of services.

Choreography

W3C’s Web services choreography description language (WS-CDL) (Web Services Choreography Description Language Version 1.0, 2005) is the dominant specification in the area of choreography. It is geared towards composing interoperable, peer-to-peer collaborations between any variety of participants, where ordered message exchanges result in accomplishing a common business goal. In this context, WS-CDL provides a standard method for defining, from a global viewpoint, the collaborating parties’ common and complementary observable behavior.

Figure 6. Sample SOAP message using extended specifications
Web Services Specification

Composability

Composability is a key feature of the Web services architecture stack and has been a guiding principle in the development of all specifications, core or extended. As already mentioned, SOAP and WSDL are designed to support composition inherently. On the other hand, all extended specifications although independently addressing specific concerns, are also designed to work seamlessly with each other in order to provide increasingly powerful functionality.

The composition of Web services specifications is illustrated in Figure 6, by means of a simple SOAP message that contains elements associated with three different specifications: WS-Addressing, WS-Security and WS-ReliableMessaging.

Each of these elements can be used independently of other elements present and without affecting their processing functions. In this way, transactions, security, reliability or any other extended capability can be incorporated into a SOAP message in terms of composable message elements. Moreover, this extension can be implemented incrementally, with necessary assets being added in later stages of development, when the need for new functionality arises.

The practical value of composable Web services is also evident in the case of service consumers seeking to determine to what extent a particular service provides the desired functionality and assurances (Weerawarana et al., 2005). In order for this to be possible, the service must explicitly document its requirements, in terms of its specific support for transactions, security etc. In this direction, WS-Policy provides a flexible mechanism for Web services to incrementally augment their WSDL and specify the supplementary SOAP elements that are necessary in order to successfully interact with the service.

REALIZING SOA WITH WEB SERVICES

Since 1996 when Service Oriented Architecture was first introduced by Gartner (1996) a lot of effort has been put worldwide in this area. SOA has captured the interest of many software architects and developers but only recently with the advent of Web services, SOA has found its way to real applications. Other technologies have been tried in the meantime, but undoubtedly Web services is the most prominent technology that forms a solid base to develop robust SOA applications.

An important advantage of using Web services as the technology platform for an SOA, is that Web services address a fundamental challenge of distributed computing: to provide a uniform way of describing components or services within a network, locating them and accessing them. The difference between the Web services approach and traditional approaches (for example, distributed object technologies such as the Object Management Group – Common Object Request Broker Architecture (OMG CORBA) or Microsoft Distributed Component Object Model (DCOM)) lies in the loose coupling aspects of the architecture. Instead of building applications that result in tightly integrated collections of objects or components, the Web services approach is much more dynamic and adaptable to change (Weerawarana, Curbera, Leymann, Storey & Ferguson, 2005).

Another key feature (Weerawarana et al., 2005) is that Web services-based software is designed using technology and specifications developed in an open way, utilizing industry partnerships and broad consortia such as W3C and the Organization for the Advancement of Structured Information Standards (OASIS), and based on standards and technology that are the foundation of the Internet. In fact, Web services “build” on the way in which the World Wide Web
achieved its tremendous success (Newcomer & Lomow, 2005), namely with HTML providing a powerful interoperability solution and HTTP an effective and lightweight universal data transfer mechanism. Web services provide the same level of abstraction for IT systems: Web services-based solutions are only required to understand, process and reply to an XML-formatted message received using a supported communications transport.

Despite the apparent benefits of Web services-based solutions, distributed object systems have a number of architectural challenges that apply irrespective of their implementation technology (Orchard et al., 2004). Web services are no less appropriate than the alternatives for building distributed systems, but still they may not add enough benefits to justify their costs in performance. For example, they are not a cost-effective solution for organizations that have relatively small and static application portfolios and do not need to interoperate with other external environments (Weerawarana et al., 2005).

On the other hand, though, SOA and Web services are most appropriate for applications:

- That must operate over the Internet where reliability and speed cannot be guaranteed and high performance is not a primary criterion
- Where there is no ability to manage deployment so that all requesters and providers are upgraded at once
- Where integration of components within heterogeneous environments (e.g., different platforms and vendor products) is at the core of the problem being addressed
- Where an existing application needs to be exposed for use over a network and be made available to various kinds of requesters—this presupposes, however, that the application can indeed be wrapped as a Web service

CONCLUSION

Collaborating companies need their applications and services to interoperate effectively. This is exactly the force driving the industry towards SOA and Web services technologies, which promise significant benefits in terms of adaptability, ease-of-integration, portability and interoperability (Protogeros, 2006).

Projects deployed with Web services and SOA can achieve an important level of business process abstraction. The interoperability and integration issues can successfully be addressed through SOA in a two-step process involving publishing services and orchestrating them. Publishing means making the Web services available through a supported interface/protocol but does not require that all existing systems be “wrapped” with a new XML/SOAP Web service layer. Orchestration means assembling and coordinating these services into a manageable business application.

However, there is work to do, specifically to the wider standards adoptions between medium and small enterprises. The lack of custom and user friendly tools drives developers to manually recode services or provide “glue code” so that they can interconnect with one another. Such painstaking labor deprives SOAs of much of their virtue—namely, rapid integration and composite application.

Wider virtual enterprise models acceptance relies heavily on the ease of integration at the business process level, and this in turn relates closely with SOA acceptance and adoption. BPEL4WS promise for universal remote integration makes us more optimistic about the future of virtual enterprises.

REFERENCES


Web Services Technology


