Chapter VII
Introduction to Web Services

Cary Pennington
University of Georgia, USA

Jorge Cardoso
University of Madeira, Portugal

John A. Miller
University of Georgia, USA

Richard Scott Patterson
University of Georgia, USA

Ivan Vasquez
University of Georgia, USA

ABSTRACT

This chapter introduces the theory and design principles behind Web Service technology. It explains the models, specifications, and uses of this technology as a means to allow heterogeneous systems to work together to achieve a task. Furthermore, the authors hope that this chapter will provide sufficient background information along with information about current areas of research in the area of Web Services that readers will come away with an understanding of how this technology works and ways that it could be implemented and used.

INTRODUCTION

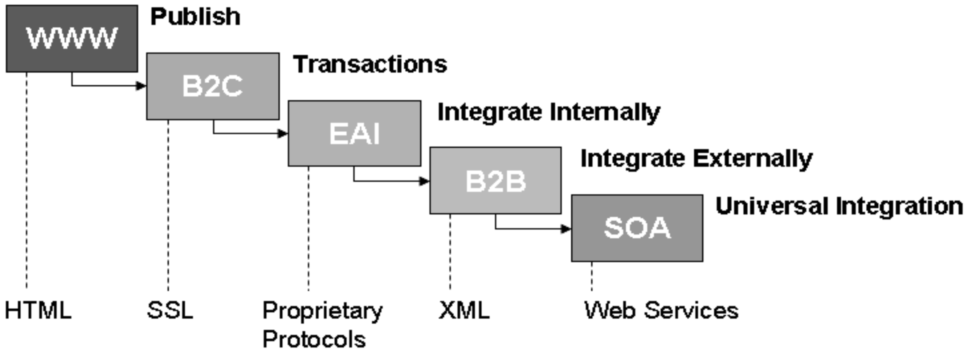
As the World-Wide Web (WWW) exploded into the lives of the public in the 1990s, people suddenly had vast amounts of information placed at their fingertips. The system was developed to allow information sharing within internationally dispersed working groups. The original WWW consisted of documents (i.e., Web pages) and links between documents. The initial idea of the WWW was to develop a universal information database to publish information that could be accessed in a reliable and simple way by consumers. The information would not only be accessible to users around the world, but information would be linked so that it could be easily browsed and quickly found by users. Organizations soon realized the importance of this technology to manage, organize, and distribute their internal data and information to customers and partners.

As organizations started to implement business-to-customer and e-commerce solutions, they realized that the initial technologies associated with the WWW were not sufficient to sell products over the Internet. Additional functionality was required to guarantee that transactions were conducted in a secure way. To this end, SSL (Secure Sockets Layer), a protocol defined by Netscape, was developed for transmitting private documents via the Internet. Using SSL, organizations were able to implement a solution to obtain confidential user information, such as credit card numbers.

With globalization, organizations were progressively undertaking mergers and acquisitions. This has created organizations with an IT environment composed of disparate legacy systems, applications, processes, and data sources. In order to meet increasing customer and business partner expectations for real-time information, organizations were required to link their heterogeneous, autonomous and distributed systems to improve productivity and efficiency. This important requirement led to the development and deployment of EAI (enterprise application integration) solutions. EAI platforms were used for integrating incompatible and distributed systems such as ERP (enterprise resource planning), CRM (customer relationship management), SCM (supply chain management), databases, data sources, data warehouses, and other important internal systems across the corporate enterprise. While useful, most EAI frameworks required costly and proprietary protocols and formats, which presented many technical difficulties when it was needed to integrate internal systems with external systems running on partners' computers.

The limitations of EAI solutions made most organizations realize that integrating internal systems with external systems to business supply chain members was a key to staying competitive, since the majority of business processes spanned across several organizations. Internal and external systems needed to communicate over networks to allow businesses to complete a transaction or part of a transaction. To achieve this level of integration, business-to-business (B2B) solutions were developed. B2B infrastructures were directed to help organizations to streamline their processes so they could carry out business transactions more efficiently with their business partners (such as resellers and suppliers). To reach a higher level of integration, most B2B solutions have relied on the use of XML as the language to represent data. XML allows one to model data at any level of complexity since it is extensible with the addition of new tags. Data can be published in multiple

Figure 1. The evolution of business usage on the WWW



formats. In contrast to the proprietary protocols used by EAI platforms, XML is vendor and platform independent allowing standard commercial software to process any conforming document.

Many organizations have already seen and experience the advantages in using XML to represent data for Web-based information exchanges (such as B2B communications). Nevertheless, organizations realized that their B2B strategies have lead the development of architectural solutions that often exhibited a tight-coupling among interacting software applications which limited the flexibility and dynamic adaptation of IT systems. As a result and to overcome these limitations, the concept of service-oriented architecture (SOA) was introduced and defined a method of designing, developing, deploying and managing discrete pieces of computer logic (i.e., services) within the WWW. The SOA goals are to achieve structuring, loose coupling, and standardization of business functionality among interacting software applications. Applications invoke a series of discrete services in order to perform a certain task. A service is carried out by a service provider in response to the request of a service consumer. The most prominent implementation of the SOA principle uses XML and Web services as its technological backbone.

Web services are based on distributed computing technology and provide a standard means of interoperating between different software applications across, and within, organizational boundaries, using XML protocols and formats. Web Services comply with several WWW standards, such as Web Services Definition Language (WSDL) and Simple Object Access Protocol (SOAP). These standards enable interoperability by using XML-based communications protocols and service definitions. The use of standard XML protocols makes Web services platform, language, and vendor independent, and an ideal candidate for use in SOA implementations.

This chapter will introduce SOA, Web service technology and its standards. It begins in the second section, with a brief history of distributed computing, which serves as the backdrop for the development of today's Web service technology. The guiding principle behind the development of Web service technology is SOA which is described in the third section. The fourth section gives an overview of the role of Web services in the context of SOA. This section gives a description of today's standards and technologies for Web services. The fifth section introduces the second-generation of Web Services Protocols. It looks in detail at the threats and standards relevant to the Web Services Security landscape and examines problems and solutions in reliability and transactions of Web Services. Clearly, these areas must be addressed before Web service technology will be widely adopted. The sixth section explains how to develop Web services starting from the initial design and continuing until deployment and publication. A summary and conclusions can be found in the last section of this chapter.

A BRIEF HISTORY OF DISTRIBUTED COMPUTING

Once networking became widespread across academia and industry, it became necessary to share data and resources. In the early years of distributed computing, message passing (e.g., using for example sockets developed in the early 1980s) was the prevailing method for communication. This involved encoding the data into a message format (i.e., how a structured piece of information is encoded prior to transmission) and sending the encoded data over the wire. The socket interface allowed message passing using send and receive primitives on transmission control protocol (TCP) or user datagram protocol (UDP) transport protocols for low-level messaging over Internet protocol (IP) networks. Applications communicated by sending and receiving text messages. In most cases, the messages exchanged conformed to an application-level protocol defined by programmers. This worked well but was cumbersome in the fact that the data had to be coded and then decoded. Using this approach, two programmers developing a distributed application must have knowledge of what the other is doing to the data. Programmers had to spend a significant amount of time specifying a messaging protocol and mapping the various data structures to and from the common transmission format.

As the development of distributed computing applications increased, new mechanisms and approaches became necessary to facilitate the construction of distributed applications. The first distributed computing technology to gain widespread use was remote procedure call (RPC). RPC technology was made popular in the 1980s by Sun Microsystems. RPC uses the client/server model and extends the capabilities of traditional procedure calls across a network. Remote procedure calls are designed to be similar to making local procedure calls. While in a traditional local procedure call paradigm the code segments of an application and the procedure it calls are in the same address space, in a remote procedure call the called procedure runs in another process and address space across the network on another processor.

RPC (Birrell, 1995) proved to be an adequate solution for the development of two-tier client/server architectures. As distributed computing became more widespread, the need to develop, for example, N-tier applications emerged and RPC could not provide the flexibility and functionality required.

With such applications, multiple machines may need to operate simultaneously on the same set of data. Therefore, the state of that data became of great concern. Research in the area of distributed objects allowed overcoming this problem with the specification of two competing technologies: common object request broker architecture (CORBA) and distributed common object model (DCOM). Later, Java remote method invocation (RMI) was developed and also became a competitor.

The CORBA [4, 5] standard was developed by the Object Management Group (OMG) starting in the 1990's and defines an architecture that specifies interoperability between distributed objects on a network. With CORBA, distributed objects can communicate regardless of the operating system they are running on (for example, Linux, Solaris, Microsoft Windows, or MacOS). Another primary feature of CORBA is its interoperability between various programming languages. Distributed objects can be written in various languages (such as Java, C++, C, Ada, etc.). The main component of CORBA is the ORB (object request broker). Objects residing in a client make remote requests using an interface to the ORB running on the local machine. The local ORB sends the request to the remote ORB, which locates the appropriate object residing in a server and passes back an object reference to the requester. An object residing in a client can then make the remote method invocation of a remote object. When this happens the ORB marshals the arguments and sends the invocation over the network to the remote object's ORB which invokes the method locally and sends the results back to the client.

DCOM (Brown & Kindel, 1996) is a protocol, developed by Microsoft, which enables communication between two applications running on distributed computers in a reliable, secure, and efficient manner. DCOM is an extension of the Component Object Model (COM). COM is an object-based programming model and defines how components and their clients interact. COM allows the development of software components using a variety of languages and platforms to be easily deployed and integrated. The distributed COM protocol extends the programming model introduced by COM to work across the network by using proxies and stubs. Proxies and stubs allow remote objects to appear to be in the same address space as the requesting object. When a client instantiates a component that resides outside its address space, DCOM creates a proxy to marshal methods calls and route them across the network. On the server-side, DCOM creates a stub, which unmarshals method calls and routes them to an instance of the component.

Java RMI (Dwoning, 1998) is a package for writing and executing distributed Java programs by facilitating object method calls between different Java Virtual Machines (JVM) across a network. Java RMI hides most of the aspects of the distribution and provides a conceptually uniform way by which local and distributed objects can be accessed. An RMI application consists of a server interface, a server implementation, a server skeleton, a client stub, and a client implementation. The server implementation creates remote objects that conform to the server interface. These objects are available for method invocation to clients. When a client wishes to make a remote method invocation it invokes a method on the local stub, which is responsible for carrying out the method call on the remote object. The stub acts as a local proxy. A server skeleton exists for each remote object and is responsible to handle incoming invocations from clients.

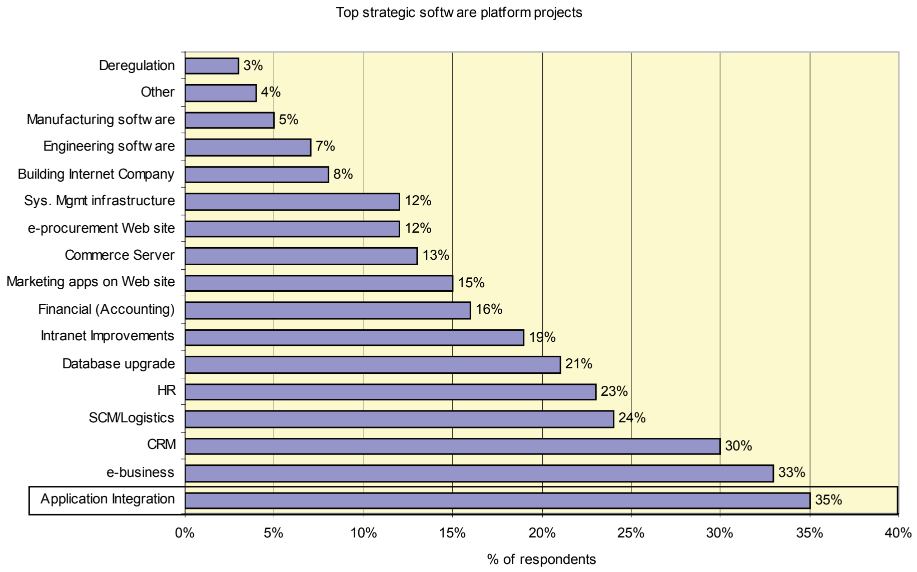
CORBA, DCOM, and Java RMI enjoyed considerable success, but they present a set of shortcoming and limitations when used in Web environments. For example, they tend to create tightly-coupled distributed systems, some are vendor and platform specific (e.g., COM/DCOM only runs on Windows), the distributed systems developed run on closely administered environment, some use complex and proprietary protocols, and specific message formats and data representation. With the growth of the Web, the search soon started for a Web compliant replacement for this technology. In the next sections, we will see that Web services are currently the most natural solution to develop distributed systems in the Web.

SERVICE-ORIENTED ARCHITECTURE

As we have seen, in the 1980s distributed computing was introduced. This research led to the development of distributed objects architectures through the 1990's. The distributed platforms developed, such as Java RMI and DCOM, had several restrictions. For example, RMI was limited to Java, while DCOM was limited to Microsoft platforms. Moreover, distributed applications developed using different platforms were difficult to integrate. Integration was and is still one of the major concerns for Chief Information Officers. Figure 2 gives us a very good indication that application integration tops the priority list of high ranking business people.

To cope with the restrictions of more traditional distributed objects architectures, in the early 2000's, the concept of service-oriented architecture (SOA) was introduced (or reintroduced, since

Figure 2. Priority list of CIOs (Channabasavaiah & Tuggle, 2003)



in reality, the concept SOA was defined by Sun in the late 1990's to describe Jini (Waldo, 1999)). SOA describes an approach which facilitates the development and composition of modular services that can be easily integrated and reused to create distributed applications. It promises the development of truly flexible and adaptable IT infrastructures. According to the W3C, a Service-Oriented Architecture is a set of components which can be invoked, and whose interface descriptions can be published and discovered. Components are made available as independent services that are accessed in a standardized way.

In order for SOA to enjoy greater success than it predecessors, it should consider the following attributes:

- **Scalable:** The past solutions were not designed with the scale of the Web in mind. SOA should work in a variety of settings, such as within an organization, between business partners and across the world.
- **Loosely-coupled:** SOA is an evolution from tightly coupled systems to loosely coupled ones. Senders and receivers of a SOA should be independent of each other; the source can send the message independently of the target. Tight coupling is not suitable for SOA since it leads to monolithic and brittle distributed applications. Even trivial changes in one component lead to catastrophic breaks in function. Small changes in one application require matching changes in partner applications (Channabasavaiah & Tuggle, 2003).
- **Interoperability:** One party should be able to communicate with another party regardless of the machine they are running on.
- **Discovery:** One party should be able to communicate with a second party selected from a set of competent candidates. Services need to be dynamically discoverable. This is accomplished through services such as a directory of service descriptions.
- **Abstraction:** A SOA abstracts the underlying technology. Developers can concentrate on building services for business users rather than connecting systems and applications.
- **Standards:** Interaction protocols must be standardized to ensure the widest interoperability among unrelated institutions. Contracts should also be standardized. Explicit contracts define what may be changed in an application without breaking the interaction. Furthermore, standards are the basis of interoperable contract selection and execution.

When comparing SOA with previous approaches we can find the following major differences. Traditional Middleware, such as distributed object systems, are based on the client-server paradigm, have heavily asymmetric interaction model, are biased towards synchronous protocols, assign public interfaces to network accessible objects, and support "name-oriented" object discovery. On the other hand, service-oriented Middleware are based on a peer-to-peer paradigm, have symmetric interaction models, mixes synchronous and asynchronous protocols, assigns public contracts to network accessible objects, and supports capability based service discovery (Cardoso, Curbera, Sheth, 2004).

Service Oriented Architecture and Web Services

Most distributed computing technologies have the concept of services and are defined by interfaces. While there are many different possibilities for developing an SOA (e.g., Web services, Java RMI, DCOM, and CORBA), Web services is currently the most desirable solution since it eliminates many of the interoperability problems between applications and services. Web services provide many of the necessary standards that are crucial for making a distributed system work. It should be noticed that using Web services does not necessarily mean that there is an SOA. Also, it is possible to have a service-oriented architecture without Web services.

There are three common actions associated with a service in SOA—discovery, request, and response. Discovery is the process of finding the service that provides the functionality that is required. A request provides the input to the service. The response yields the output from the service. It follows easily that this architecture must have three primary actors: requestor, provider, and registry.

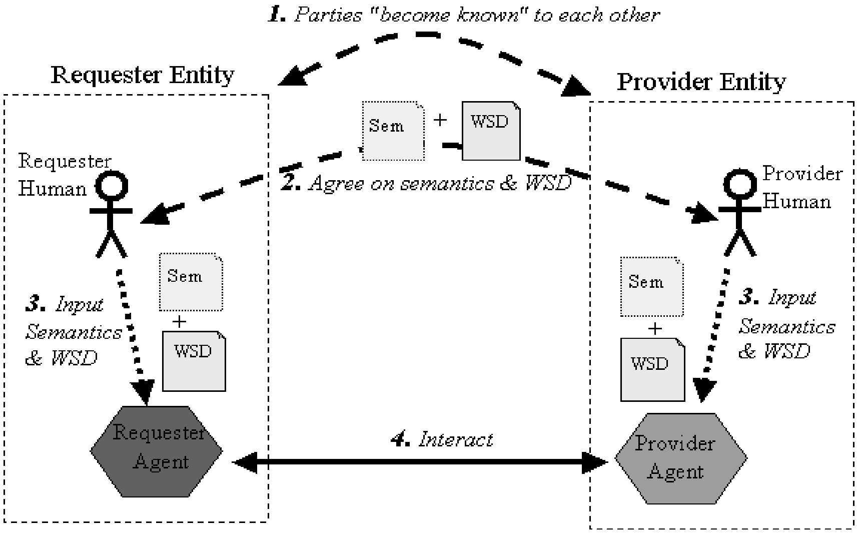
The beginning of this figure (step 1) shows the process that two participants would become aware of one another. This is accomplished as the service provider publishes the Web Service Description (WSD) and Semantics (Sem.) to a registry after which the service requestor would discover that service. In step 2, the semantics and description are agreed upon so that there will be no misunderstanding about the data that is being exchanged during this communication. Once the WSD and semantics are accepted by and loaded into both the participants (step 3) then they can interact to carry out the operation that was needed.

A service provider may develop and deploy one or more Web services. Each service must contain at least one operation. Operations are also referred to as endpoints because they are the part of the service that actually does the processing.

What are Web Services?

Web services are modular, self-describing, self-contained applications that are accessible over the Internet (Curbera & Nagy, 2001). They are the most popular realization of the service-oriented architecture. A Web service is a software component invokable over the Web via an XML (XML, 2005) message that follows the SOAP (SOAP, 2003) standard. The component provides one or more operations for performing useful actions on behalf of the invoking client. These operations and the formats of the input and output messages are described using WSDL (Christensen & Curbera, 2001). Being based on these Web standards makes Web services both implementation language and platform independent. Description of services in a language neutral manner is vital for the widespread use of Web services. For general usability, a service must be described and advertised. WSDL takes

Figure 3. Process of discovery (Booth, 2004)



care of the description by providing a language to describe a service in enough detail to invoke any of its operations. Service providers describe their Web services and advertise them in a universal registry called UDDI (UDDI, 2002). This enables service requestors to search the registry and find services, which match their requirements. UDDI allows for the creation of registries that are accessible over the Web. A registry contains content from the WSDL descriptions as well as additional information such as data about the provider. Clients may use one or more registries to discover relevant services.

To describe Web services further, let us look at an example scenario. A company called Moon Company is a product distributor. They keep track of their clients, goods, and orders through a system that they have in-house. They do not want to provide unlimited access to this system to their customers, but they would like their customers to be able to place orders easier. Using Web services, the Moon Company can create an interface to their interior system so that a customer can be looked up, and once authenticated, order products. With these services in place, Moon needs only provide the WSDL definitions of the services to their clients and the clients will be able to compose any system on their side to handle ordering in any way they see fit. Since Moon does not know what type of system their customers are using, other remote technologies would be more difficult to implement.

SOA and Web Service Standards

The use of standard protocols is one of the aspects that allow SOA to deploy technically compatible services. Currently, Web service standards are the preferred solution to develop SOA-based products. Web services technology has gained a suitable degree of maturity and is being used to easily publish business functions to an intranet or the Internet for remote execution. Business functions can reside in popular applications such as ERP (enterprise resource planning), CRM (customer relationship management), and SCM (supply chain management) systems.

Some of the standards associated with Web services are indispensable to developing SOA-based solutions as illustrated in Figure 4.

The most well-known protocols will be presented and discussed in this section, while the second-generation Web services standards, such as WS-Security, WS-Coordination, WS-Transaction, and WS-Policy will be discussed in the next section.

Figure 4. Web Services and list standards (Cardoso, Curbera, & Sheth, 2004)

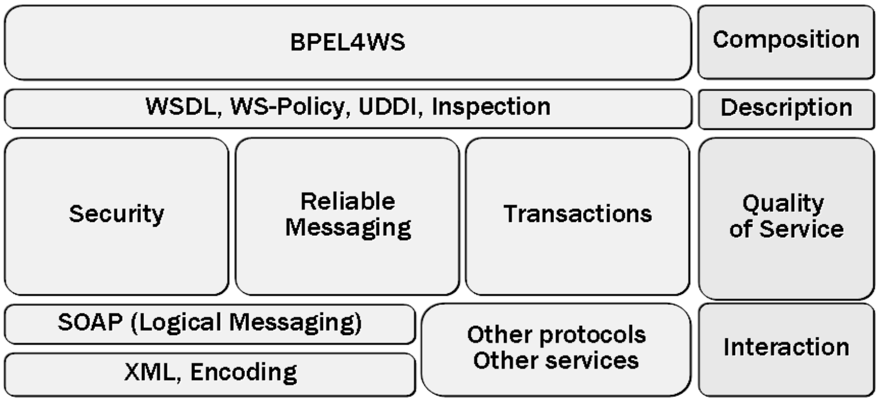
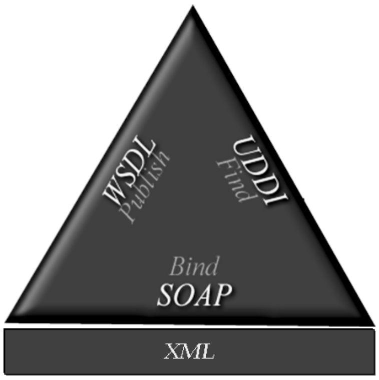


Figure 5. The relationship between XML/SOAP/ WSDL/UDDI



Basic Web Service Standards

XML, SOAP, WSDL and UDDI (Graham & Simenov, 2002) are the fundamental elements to deploy SOA infrastructures based on Web services (see Figure 5). XML is the standard for data representation; SOAP specifies the transport layer to send messages between consumers and providers; WSDL describes Web services; and UDDI is used to register and lookup for Web services.

XML, the emerging standard for data representation, has been chosen as the language for describing Web services. XML is accepted as a standard for data interchange on the Web allowing the structuring of data on the Web. It is a language for semi-structured data and has been proposed as a solution for data integration problems, because it allows a flexible coding and display of data, by using metadata to describe the structure of data (using DTD or XSD). A well-formed XML document creates a balanced tree of nested sets of open and closed tags, each of which can include several attribute-value pairs.

Simple object access protocol **(**SOAP**)**. This standard defines the types and formats of XML messages that may be exchanged between peers in a decentralized, distributed environment. One of the main objectives of SOAP is to be a communication protocol that can be used by distinct applications developed using different programming languages, operating systems, and platforms. Many software vendors are producing an implementation of SOAP into their systems. Examples of major vendors include Sun, Microsoft, and IBM. The latest version of the standard is SOAP 1.2 (http://www.w3.org/TR/soap). SOAP specification is not completed yet and as it goes through the W3C standardization process some minor changes will certainly occur.

The current specification defines a skeleton that looks like the listing below. The envelope defines the namespace of the SOAP specification and the encoding style that was used to create this message. The Header section is optional and contains additional information about the mes-

Figure 6. SOAP skeleton listing (SOAP, 2002)

```
<?xml version=”1.0”?>
<soap:Envelope
xmlns:soap=”http://www.w3.org/2001/12/soap-envelope”
soap:encodingStyle=”http://www.w3.org/2001/12/soap-encoding”>
<soap:Header>
 ...
</soap:Header>
<soap:Body>
 ...
 <soap:Fault>
 ...
 </soap:Fault>
</soap:Body>
</soap:Envelope>
```

sage. The Body section contains the data that is being transferred.

Web Service Description Language (WSDL). WSDL is the major language that provides a model and an XML format to describe the syntactical information about Web services. It is a W3C standard XML language for specifying the interface of a Web service. This standard enables the separation of the description of the abstract functionality offered by a service from concrete details of a service implementation by defining the interface that Web services provide to requesters. The definition of the interface (called a port type in version 1.x and called interface in version 2.0) gives the signatures for all the operations provided including operation name, inputs, outputs and faults. Beyond the interface, information about the service itself and allowed bindings is included in WSDL documents. The latest version of the standard is WSDL 1.1 (http://www.w3.org/TR/wsdl), although WSDL 2.0 has become a candidate recommendation (http://www.w3.org/TR/wsdl20). WSDL 1.1 uses XML Schema Definition (XSD) which provides constructs for creating complex types (http://www.w3.org/XML/Schema).

The following is brief and incomplete copy of a WSDL file. Notice how it defines the type of data to be used, the operations that exist in the service and the type of inputs and outputs that those operations require. With this information, a call to invoke any operation in this service can be made and carried out successfully.

UDDI (universal description, discovery, and integration). Currently, the industry standards available to register and discover Web services are based on the UDDI specification (UDDI, 2002). Once a Web service is developed, it has to be advertised to enable discovery. The UDDI registry is

Figure 7. Partial WSDL listing (Semantic Web Services Challenge, 2006)

```
<wsdl:definitions
targetNamespace=”mooncompany”
xmlns:wsdl=”http://schemas.xmlsoap.org/wsdl/” xmlns:wsdlsoap=”http://schemas.xmlsoap.org/
wsdl/soap/” xmlns:xsd=”http://www.w3.org/2001/XMLSchema”>
<wsdl:message name=”SearchCustomerResponseMessage”>
 <wsdl:part element=”impl:SearchCustomerResponse”
   name=”SearchCustomerResponse”/>
  </wsdl:message>
 <wsdl:portType name=”SearchCustomer”>
                    <wsdl:operation name=”search”>
   <wsdl:input message=”impl:SearchCustomerRequestMessage”/>
                    <wsdl:output message=”impl:SearchCustomerResponseMessage”/>
                    </wsdl:operation>
                    </wsdl:portType>
                    <wsdl:binding name=”CRMServiceSoapBinding”
    type=”impl: SearchCustomer “>
                    <wsdlsoap:binding style=”document”
    transport=”http://schemas.xmlsoap.org/soap/http”/>
                    <wsdl:operation name=”search”>
                                <wsdlsoap:operation soapAction=”search”/>
                                </wsdl:operation>
                    </wsdl:binding>
 <wsdl:service name=”CRMService”>
                    <wsdl:port binding=”impl:CRMServiceSoapBinding” name=”CRMService”>
                    <wsdlsoap:address
     location=”http://138.232.65.158/moon/services/CRMService”/>
                   </wsdl:port>
 </wsdl:service>
</wsdl:definitions>
```

supposed to open doors for the success of service oriented computing, leveraging the power of the Internet. Hence the discovery mechanism supported should be scaled to the magnitude of the Web by efficiently discovering relevant services among tens and thousands (or millions according to industry expectations) of Web services. UDDI standard defines a SOAP-based Web service for locating WSDL descriptions of Web services. This standard defines the information content and the type of access provided by service registries. These registries provide the advertisement of the services that can be invoked by a client. UDDI can store descriptions about internal Web services across an organization and public Web services located in the Internet.

OTHER WEB SERVICES STANDARDS AND PROTOCOLS: WS-*

Besides the core standards discussed in section 4, there are several other standards needed for Web services to be used in practice. This section gives a quick tour of some of these standards.

Web Service Policy

In the process of discovering a service, there is an inherent problem. We might write a query that yields ten services that match our keyword, or meet our input and output specifications. Yet, at this point, we do not know what these services require of the messages that will be exchanged. Policy in Web services adds this information to the description. It allows the provider of the service to give all the information they see fit about the service; requirements, capabilities, and quality. With this information, the best service can be chosen from the discovered services based on much more complete information than just functional requirements and keywords. (Verma, Akkiraju, & Goodwin, 2005).

WS-Policy

WS-Policy is a specification of a framework for defining the requirements and capabilities of a service. In this since, a policy is nothing more that a set of assertions that express the capabilities and requirements of a service. The specification WS-Policy (http://www-128.ibm.com/developerworks/library/specification/ws-polfram/) defines terms that can be used to organize a policy. Once a provider has a policy defined in XML, then he must publish that information by referencing it in the description of the service.

WS-PolicyAttachment

This defines the method for attaching a policy to a WSDL file so that it can be published to the UDDI and thus used in deciding on services. There are several mechanisms defined for accomplishing this task. The simplest method is to write the policy directly into the WSDL file. A more complex, and more powerful method is to construct the policy as a stand alone file that is referenced in the WSDL file as a URI. These references can exist at any element of the WSDL. WS-Policy and WS-PolicyAttachment together give us hierarchy based on to which element the policy is attached and direction for merging policies together to create an effective policy for an element (WS-PolicyAttachment, 2005).

Both WS-Policy and WS-PolicyAttachment have recently been submitted to W3C for standardization.

Web Service Security

In this section, we examine some of the concepts, theories, and practices in securing Web services at an introductory level. Our aim is for you to become familiar with these as well as the terms used. Security is a constantly changing arena driven by the changes in associated technologies.

The World Wide Web, or Web, has in some way touched the lives of most people living in an economically developed country. Unfortunately, it has not always been in a positive way. This is because once a computer is connected to the Web; it becomes part of a system that was not designed with security and privacy in mind. Computers hold information, sometimes sensitive information, for much longer than most users realize. Even during the simple event of entering information into a Web browser, information is stored onto disk. This may take place in a temporary file. Although once the information is sent to a Web server and the file is deleted, the information is still present on the disk; even though the file reference is gone. Many unsavory characters have learned how to glean this information off of remote systems through vulnerabilities of the operating system.

A basic definition of security can be thought of as "keeping unauthorized access minimal." This is true not only on the Web but also in our daily lives. We lock our doors when we leave our houses in an effort to keep unauthorized visitors out. This definition is simple, but it is clear. A more complete definition may become too convoluted. Let us consider a definition for privacy, "not making public what may be considered personal." Not a fancy definition, rather straight to the point. We all have different ideas of what is personal to us, and what being made public means. However, I think we can all agree that having our Social Security Number and checking account information sold to the highest bidder is a violation of our privacy.

Now that security and privacy are defined, let us consider how this fits into the Web. Suppose you would like to purchase a book online. Once you have found the book and placed it in your "Cart" it is time to checkout. In order to checkout you must pass through some security. Typically, you will be asked for your credit card information and billing address. This is the first security checkpoint and this information is verified with your bank; as well as making sure the card has not been reported stolen. The next checkpoint is physical possession of the card, which is verified by a security code on the back of your card. So, you the consumer trust this Web site to send the book, or you would not have placed the order, and the Web site trusts you for payment since it has verified all your information. Trust is a key component of security and privacy as we shall see. As a consumer using sensitive personal information to make a purchase, have you considered privacy of your information? Somewhere in the information exchange between you and the Web site an agreement has been made; whereas, the Web site has promised not to sell your personal information. However, how well is it protected? Your credit card information, billing address, and security code are now stored in two places, the Web sites server and on your PC. More than likely one of those unsavory characters will not spend the time and effort to get one credit card number off a PC when with a little more work they could have access to thousands of entries. So this brings us back to security. This time that of the Web site server. As you can see, security and privacy go hand and hand, with mutual trust holding them together.

The above scenario is a simple client-server process, much like those that currently encompasses the Web. However, Web services extend the client-server model and are distributed as discussed in earlier sections. Although this combination is what gives Web services such promises in the SOA, it is also an area of concern for security and privacy. The more doors and windows a home has, the more opportunities a thief has, the more vigilant the home owner must be. This can be applied to Web services as well. Web services increases the number of access points to data and ultimately machines. Furthermore, because the access to data is increased, the sharing of information is increased. This in itself is opens the possibility of privacy invasion.

Now that the stage has been set, let us look at the specific security and privacy considerations. Web services are a distributed cross-domain environment. Therefore, it is difficult to determine the identity of the actors; in this case who is the service *requester* and who is the service *provider*. Message level security and privacy is important since these invocations may cross un-trusted intermediaries. It is necessary for the requester and provider to have a protocol for discovering each others policies and negotiating constraints at run-time, prior to interaction. Privacy rights and agreements should be explicitly described and agreed upon. We will look more closely at these considerations in the following paragraphs.

Message level security involves securing all aspects of the SOAP message. Encryption plays a large role in providing integrity of messages between the requester and the provider while traversing intermediaries. In addition, the requester and the provider can not always be trusted.

Man-In-The-Middle attack is when an attacker is able to compromise a SOAP message in transit. An attacker may gain access to confidential information contained in the message or may alter the message.

Unauthorized Access attack takes place when an attacker is able to gain access to a Web service which they do not have permissions to use. This can happen through brute-force or by compromising a SOAP message thereby gaining a username and token. An attacker may also pose as a legitimate Web service in order to gain an authentication mechanism, this is known as Spoofing.

The above threats can be alleviated using proper authentication and encryption techniques. However, there are other attacks that can only be alleviated through good programming habits and proper verification of parameters.

SQL injection attack is the insertion of malicious SQL statements. This requires preprocessing of any parameters passed to an operation which queries a SQL database to alleviate this threat. Command injection attacks are similar to SQL injection attacks in that malicious system commands are injected into the SOAP in an effort to exploit the systems vulnerabilities. This threat can be alleviated by proper configuration permissions and preprocessing.

Proper authentication and encryption schemes can alleviate threats which compromise message integrity. Point-to-Point schemes which are implemented at the transport layer, such as VPN, SSL, or IPSec, provide a "secure tunnel" for data to flow, however, they can not guarantee the integrity of the message. End-to-End schemes, which are implemented at the application layer, can guarantee the confidential integrity of the message and that the message has not been altered. This is because the message is encrypted and digitally signed with a key. End-to-End schemes also offer the granularity necessary for Web services such that sections of the SOAP message may be encrypted while other sections are not.

WS-Security Framework

The WS-Security specification provides a framework and vocabulary for requesters and providers to secure messaging as well as communicate information regarding security and privacy. There are other security related specifications worth mentioning. XML-Encryption specifies the process of encrypting data and messages. XML-Signature provides a mechanism for messages integrity and authentication, and signer authentication. XACML is an XML representation of the Role-Based Access Control standard (RBAC). XACML will likely play an important function in Web services authorization. Security Assertion Markup Language, or SAML, is an OASIS framework for conveying user authentication and attribute information through XML assertions. There are many specifications and standards for Web services security. We would like to encourage you to investigate these on your own as an exercise.

WS-SecurityPolicy

Policies for Web services that describe the access permissions as well as actions which a requester or provider are required to perform. For example, a policy may indicate that requesters must have an active account with the service and that messages be encrypted using a PKI scheme from a trusted certificate authority. A requester may also have a policy indicating which encryption schemes it accepts.

WS-Trust

Before two parties are going to exchange sensitive information, they must establish a secure communication. This can be done by the exchange of security credentials. However, one problem remains, how one party can trust the credentials of the other. The Web Service Trust Language (WS-Trust) was developed to deal with this problem. It offers extensions to the WS-Security elements to exchange security tokens and establishing trust relationships (WS-Trust, 2005).

WS-SecureConversation

The Web services protocol stack is designed to be a series of building blocks. WS-Secure Conversation is one of those blocks. WS-Security provides message level authentication, but is vulnerable to some types of attacks. WS-SecureConversation uses SOAP extensions to define key exchange and key derivation from security context so that a secure communication can be ensured (WS-SecureConversation, 2005).

WS-Authorization

Authorization for Web services still remains an area of research at the time of this publication. The difficulty of authorization is the inability to dynamically determine authorization for a requester whom a Web service has just been introduced. Some authorization frameworks being suggested include assertion based, role based, context based and a hybrid approach.

Assertion based authorization uses assertions about the requester to decided on the level of authorization. In a role based approach, requesters are given "user" labels and these labels are associated with roles, which in turn have permissions assigned to them. Context based authorization examines the context in which a requester is acting. For instance: proximity to the resource, on behalf of a partnership, or even the time of day. Obviously a hybrid approach is some combination of two or more approaches.

WS-Privacy

Privacy is in the context of data and can be associated with the requester or the provider. The requester may be concerned that the information given to a provider will be propagated to other entities. Such information could be a credit card number, address, or phone number. A provider may be concerned with the proliferation of information which they have sold to a requester. In this case the provider does not want the requester to resell this information without proper compensation.

Transaction Processing

The perceived success of composite applications in a service-oriented architecture depends on the reliability of participants that are often beyond corporate boundaries. In addition to already frequent errors and glitches in application code, distributed applications must cope with external factors such as network connectivity, unavailability of participants and even mistakes in service configuration. Web services transaction management enables participating services to have a greater degree of confidence in that the actions among them will progress successfully, and that in the worst case, such transactions can be cancelled or compensated as necessary.

WS-Transaction

To date, probably the most comprehensive effort to define transaction context management resides in the WS-Coordination (WS-C) (Microsoft, BEA, IBM,`Web Service Coodination', 2005), WS-AtomicTransaction (WS-AT) (Microsoft, BEA, IBM, `Web Service Atomic Transaction', 2005) and WS-BusinessActivity (WS-BA) (Microsoft, BEA, IBM,`Web Service Business Activity', 2005) specifications. WS-C defines a coordination context, which represents an instance of coordinated effort, allowing participant services to share a common view. WS-AT targets existing transactional systems with short interactions and full ACID properties. WS-BA, on the other hand, is intended for applications involved in business processes of long duration, whose relaxed properties increase concurrency and suit a wider range of applications.

Neither the Web services architecture nor any specifications prescribe explicit ways to implement transactional capabilities, although it is clear that delivering such features should minimally impact existing applications. Some propose approaching the problem of transaction monitoring and support by means of intermediary (proxy) services (Mikalsen, 2002), while others by providing a lightweight programming interface requiring minimal application code changes (Vasquez, Miller, Verma, & Sheth, 2005). Whichever the case, protocol-specific messages should also be embedded in exchanged messages and propagated though all participants.

WS-Composite Application Framework

Reliability and management are aspects highly dependent on particular Web service implementations and therefore no specification mandates or comments on them. However, just like the J2EE Enterprise JavaBeans (EBJ) technology has made available container-managed transactions (CMT) for some time, a way to procure increased Web service reliability could be through their deployment in *managed* environments, in which the hosting application server becomes responsible for support activities such as event logging and system recovery. These additional guarantees could potentially improve many aspects of Web services reliability, taking part of the burden away from their creators with regards to security, auditing, reliable messaging, transactional logging and fault-tolerance, to cite just a few. Some implementations leading this direction are already available from enterprise software companies such as Arjuna Transaction Service (Arjuna Transaction Service, 2005), IBM Transactional Attitudes (IBM Transactional Attitudes, 2005), and from open source projects like Apache Kandula (Apache Kandula Project, 2005) and the academic community (Trainotti, Pistore, Pistore, et al., 2005; Vasquez et al., 2005).

Messaging

WS-ReliableMessaging

Communication over a public network such as the Internet imposes physical limitations to the reliability of exchanged messages. Even though failures are inevitable and unpredictable, certain techniques increase message reliability and traceability even in the worst cases.

At a minimum, senders are interested in determining whether the message has been received by the partner, that it was received exactly once and in the correct order. Additionally, it may be necessary to determine the validity of the received message: Has the message been altered on its way to the receiver? Does it conform to standard formats? Does it agree with the business rules expected by the receiver?

WS-Reliability and WS-ReliableMessaging have rules that dictate how and when services must respond to other services concerning the receipt of a message and its validity.

WS-Eventing

Web services eventing (WS-Eventing) is a specification that defines a list of operations that should be in a Web service interface to allow for asynchronous messaging. WS-Eventing is based on WS-Notification that was submitted to OASIS for standardization.

WS-Notification

Web service notification (WS-Notification) is a family of specifications that provide several capabilities.

- Standard message exchanges for clients
- Standard message exchanges for a notification broker service provider

- Required operations for services that wish to participate in notifications
- An XML model that describes topics.

WS-Notification is a top layer for the following specifications: WS-BaseNotification, WS-BrokeredNotification, and WS-Topics.

WS-BaseNotification defines the operations and message exchanges that must take place between the two parties. WS-BrokeredNotification defines messages and operations required of a Notification Broker Service and those that wish to use it. WS-Topics define the "topics" that are used to organize the elements in a notification message. It also defines XML to describe the metadata associated with different topics.

DEVELOPING WEB SERVICES

The starting point of using Web service technology is to create Web services. Although it is similar to developing other software, there are some differences in that early focus on interfaces and tool support is of even greater importance. One can start by creating a WSDL specification, or alternatively, by creating, for example, a Java interface or abstract class. Since tools such as Axis (Apache Axis Documentation, 2006) or Radiant (2005) can convert one form to the other, it is a matter of preference where to start. In this chapter we will give a guide to developing Web services starting by designing the Java classes.

We will do this by following fundamental software engineering techniques to create the Web services. Start by creating a UML Class Diagram to define the requirements of the system. To illustrate the ideas in this section, we will use an example from the Semantic Web Services Challenge 2006 (Semantic Web Services Challenge, 2006). The Challenge scenario is to create a process to create a purchase order. The first step in this process is to confirm that a given business is a customer of the fictitious "Moon Company." Our example implements this service. Below are the eight steps to create this service:

1. **Create a UML Class Diagram:** Following software engineering practices, the initial step is to create a UML Class Diagram to define the classes that will be needed for the service. UML provides a succinct representation of modeling classes. The following is an example of a UML class diagram for a service that will take as input the name of a business and search a database to return the profile for this business if they are a partner of the Moon Company.
2. **Generate Java Code:** Using a UML tool such as Poseidon, the UML Class Diagram can easily converted into a Java class skeleton. It is important to note that while you are developing objects to be used for Web services that you must follow the Java bean programming conventions, for example, implementing "getters" and "setters" for every member variable. Fortunately, this is exactly the code that will be generated thanks to the UML tool based on the diagram that we have created in step one. For simplicity, we have generated our Web service as an abstract class.
3. **Adding in Web Services Annotations:** Java 6 includes annotations so that the compiler will know that the program code is a Web service. A partial list of available annotations is as follows:

- javax.jws.WebService
- javax.jws.WebMethod
- javax.jws.WebParam
- javax.jws.WebResult
- javax.jws.HandlerChain
- javax.jws.soap.SOAPBinding

Figure 9 illustrates an example of a Java service which has been annotated. Note that in the example the @WebService and @WebMethod are the annotations. The complier will recognize these tags and create the WSDL document.

Figure 8. UML class diagram

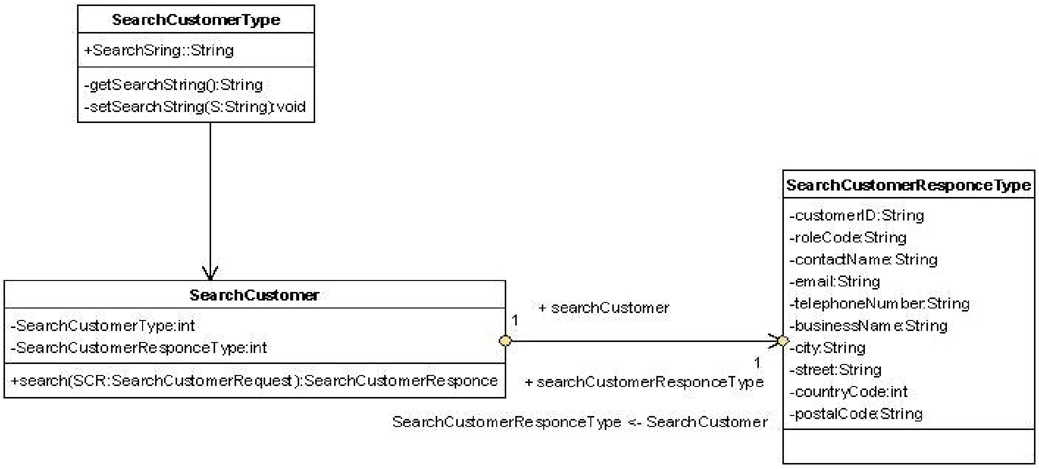


Figure 9. Annotated Java example

```
import javax.jws.WebService;
import javax.soap.SOAPinding;

@WebService
public class SearchCustomer
{

   @WebMethod
   public SearchCustomerResponce search (SearchCustomerRequest)
          //call to backend to verify Customer
          if(! verifyCustomer(SearchCustomerRequest))
          {
                        return err;
                        }

EarchCustomerResponce SCR = new SearchCustomerResponce;
SCR. setcustomerID(CustomerInfo.getcustomerID(SearchCustomerRequest)
SCR. setroleCode(CustomerInfor. getcustomerRole(SearchCustomerRequest)
...
...
...

          return SCR;
   }//WebMethod
}//SearchCustomer
```

Refer to the following link to see more information on annotations (https://mustang.dev.java.net/).

4. **Generate WSDL:** The annotations from the previous step indicate to the Annotation Processing Tool or the Java compiler that a WSDL is to be generated at compile-time. This description of the service is used in two ways. One, the description acts as an advertisement when it is published on the Web. The information gleaned from the WSDL file is published in UDDI registries so that queries can be executed to discover the service that is needed. Second, it provides all the information needed to invoke this service remotely.
5. **Implement Methods:** At this point in development, we want to create an implementation class that extends our abstract class. The difference that the developer must deal with is writing the code to the proper conventions. Any class that is created must have getters and setters for all member variables. These are used during invocation by the SOAP engine to serialize and deserialize the data that is in the SOAP messages into Java objects and back to SOAP.
6. **Deploy Service:** Deploying a service is accomplished using a Web application server and a SOAP engine, like Tomcat and Axis2 respectively. If using Axis2, deploying a service is as simple as dropping the .aar files, which are .jar files with a different extension, into the \WEB-INF\services directory. Directions on deployment in Axis2 can be found on the Web at http://ws.apache.org/axis2 .
7. **Test Service:** A simple Java program can be sufficient to test a service. In others it may require a more complex client. Either way the fundamentals for writing a client are the End Point Reference, which is a URL to the service, a call setting the target, and setting the transport information. All of this information is put into a call object that exists in the org.apace.soap package. The setup of this object is in Figure 10.

This code creates a call to a service named "CMRService" with an operation name "search". This operation takes a SearchCustomerType as input, thus you see an instance of this class is created and added as a parameter to the call object.

```
Response resp = call.invoke(url, "");
```

This calls the invoke method on the call object to execute the operation in the service. The results of the service are put into the Response object and can be accessed from there.

Figure 10. Partial listing of Web service client

```
Call call = new Call();
call.setSOAPMappingRegistry(smr); call.setTargetObjectURI("http://138.232.65.158/moon/ser-
vices/CRMService");
call.setMethodName("search");
call.setEncodingStyleURI(Constants.NS_URI_SOAP_ENC);
Vector params = new Vector();
SearchCustomerType sct = new SearchCustomerType();
sct.setSearchString(name);
params.addElement(new Parameter("request", SearchCustomerType.class, sct, null));
call.setParams(params);
```

8. **Publish Service:** Publishing a service requires the use of UDDI registries. Setting up a registry varies based on which registry is chosen. For our example, we used the jUDDI registry on a Tomcat server. The action of publishing a service is similar to advertising a business. After deployment and testing, the service is open to the world and ready to accept request, but until it is published, it is unlikely that anyone will know about your service. Tools that simplify this process are Radiant and Lumina (Li, 2005), both from the METEOR-S tool suite.

CONCLUSION

The service oriented architecture (SOA) is currently a "hot" topic. It is an evolution of the distributed systems technology of the 1990s, such as DCOM, CORBA, and Java RMI. This type of architecture requires the existence of main components and concepts such as services, service descriptions, service security parameters and constraints, advertising and discovery, and service contracts in order to implement distributed systems. In contrast to the Event-Driven Architecture, in which the services are independent, the SOA-based approach requires services to be loosely coupled.

SOA are often associated with Web services and sometimes, SOA are even confused with Web services, but, SOA does not specifically mean Web services. Instead, Web services can be seen as a specialized SOA implementation that embodies the core aspects of a service-oriented approach to architecture. Web service technology has come a long way toward achieving the goal of the SOA. With Web services, developers do not need to know how a remote program works, only the input that it requires, the output it provides and how to invoke it for execution. Web services provide standards and specifications that create an environment where services can be designed, executed, and composed into processes to achieve very complicated tasks.

For some years now, Web services define a set of standards (such as WSDL, SOAP, and UDDI) to allow the interoperation and interoperability of services on the Internet. Recently, security and transactional stability have become priority areas of research to make Web services more accepted in the world of industry. The work done has lead to the development of a set of new specifications (such as WS-Security, WS-Policy, WS-Trust, WS-Privacy, WS-Transaction, etc.) that describe how Web services can establish secure communications, define policies services' interactions, and define rules of trust between services.

REFERENCES

Arjuna Technologies Limited (2005). Arjuna transaction service suite. Retrieved October 18, 2006, from http://www.arjuna.com/products/arjunats/index.html

Axis Development Team (2006) . *Webservices – Axis*. Retrieved October 18, 2006, from http://ws.apache.org/axis/

Bellwood, T. (2002) *UDDI Version 2.04 Api specification*. Retrieved February 20, 2007 from http://uddi.org/pubs/ProgrammersAPI-V2.04-Published-20020719.htm

Birrell, A.D. & Nelson, B.J. (1984). Implementing remote procedure calls. *ACM Transactions on Computer Systems, 2*(1), 39-54.

Booth, D., Hass, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., & Orchard, D. (2004) *Web services architecture*, W3C Working Group Note. Retrieved October 18, 2006, from http://www.w3.org/TR/ws-arch/

Brewer, D., LSDIS Lab, University of Georgia (2005). Radiant. Retrieved October 18, 2006, from http://lsdis.cs.uga.edu/projects/meteor-s/downloads/index.php?page=1

Brown, N., & Kindel. C. (1996). *Distributed component object model protocol*, DCOM/1.0. Redmond, WA: Microsoft Corporation.

Cabrera, L. F., Copeland, G., Feingold, M., Freund, T., Johnson, J., & Joyce, S., et al. (2005) *Web services atomic transaction (WS-Atomic Transaction)*. retrieved February 20, 2007 from http://www128.ibm.com/developerworks/library specification/ws-tx/#atom

Cabrera, L. F., Copeland, G., Feingold, M., Freund R. W., Freund, T., & Joyce, S., et al. (2005). *Web services business activity framework (WS-BusinessActivity)*. Retrieved February 20, 2006 from http://schemas.xmlsoap.org/ws/2004/10/wsba/

Cabrera, L. F., Copeland, G., Feingold, M., Freund, T., Freund, R. W., Johnson, J. (2005) *Web service coordination (WS-Coordination)*. Retrieved February 20, 2006 from http://specs.xmlsoap.org/ws/2004/10/wscoor/wscoor.pdf

Cardoso, J., Curbera, F., & Sheth, A. (2004, May 17-22). Tutorial: Service oriented architectures and Semantic Web processes. In *Proceedings of the Thirteenth International World Wide Web Conference (WWW2004),* New York.

Channabasavaiah, K., Holley, K., & Tuggle, E. (2003) Migrating to a service-oriented architecture, Part 1. Retrieved October 18, 2006, from http://www128.ibm.com/developerworks/Webservices/library/ws-migratesoa/

Christensen, E., Curbera, F., Meredith, G., Weerawarana, S. (2001) *W3C Web Services Description Language (WSDL)*. Retrieved October 18, 2006, from http://www.w3.org/TR/wsdl

Curbera, F., Nagy, W., Weerawarana, S. (2001). *Web services: Why and how.* Paper presented at the Workshop on Object-Oriented Web Services - OOPSLA 2001, Tampa, Florida.

Dwoning, T. (1998). *Java RMI.* Boston: IDG Books Worldwide.

Graham, S., Simenov, S., Davis, D., Daniels, G., Brittenham, P., Nakamura, Y., Fremantle, P., Koeing, D., & Zentner, C. (2002). *Building Web services with Java: Making sense of XML, SOAP, WSDL, and UDDI, SAMS*. Indianapolis, Indiana.

IBM, BEA Systems, Microsoft, SAP AG, Sonic Software, VeriSign (2006).

Web service policy attachment. Retrieved October 18, 2006, from http://www-128.ibm.com/developerworks/library/specification/ws-polatt/index.html

IBM, BEA Systems, Microsoft, Layer 7 Technologies, Oblix, VeriSign, Actional, Computer Associates, OpenNetwork Technologies, Ping Identity, Reactivity, RSA Security (2005). Web services trust language. Retrieved October 18, 2006, from http://www-128.ibm.com/developerworks/library/specification/ws-trust/

IBM, BEA Systems, Microsoft, Computer Associates, Actional, VeriSign, Layer 7 Technologies, Oblix, OpenNetwork Technologies, Ping Identity, Reactivity, RSA Security (2005). Web service secure conversation language specification. Retrieved October 18, 2006 from http://www-128.ibm.com/developerworks/library/specification/ws-secon/

Li, K. (2005). *Lumina: Using WSDL-S for Web service discovery*. Masters Thesis, University of Georgia.

Microsoft, BEA & IBM. (2005). *Web Services Atomic Transaction*

Microsoft, BEA & IBM. (2005). *Web Services Business Activity*

Microsoft, BEA & IBM. (2005). *Web Services Coordination*.

Mikalsen, T., Rouvellou, I., & Tai. S. (2003). *Advanced enterprise middleware: Transaction* *processing*. Retrieved October 18, 2006, from http://www.research.ibm.com/AEM/txa.html

Mikalsen, T., Tai, S., & Rouvellou, I. (2002). *Transactional attitudes. Reliable composition of autonomous Web services.* Paper presented at the International Conference on Dependable Systems and Networks.

Object Management Group. (1995, July). *CORBA: The Common Object Request: Architecture and Specification*, Release 2.0. Retrieved February 20, 2007 from http://www.omg.org/cgi-bin/apps/doc?formal/99-10-07.pdf

Orfali, R., & Herkey, D. (1998). *Client/Server programming with Java and CORBA* (2nd ed.). Hoboken NJ: John Wiley & Sons.

Semantic Web Services Challenge (2006). Main page. Retrieved October 18, 2006, from http://www.sws-challenge.org/

SOAP (2003). *Simple object access protocol 1.2*. Retrieved October 18, 2006, from http://www.w3.org/TR/soap/

Trainotti, M., Pistore, M., Calabrese, G., Zacco, G., Lucchese, G., Barbon F., Bertoli, P., Traverso P., & ASTRO. (2005). *Supporting composition and execution of Web services*. Paper presented at the International Conference on Service Oriented Computing.

UDDI (2002). *Universal Description, Discovery, and Integration.*

Vasquez, I., Miller, J., Verma, A., & Sheth, A. (2005). *OpenWS-Transaction: Enabling reliable Web service transactions*. Paper presented at the International Conference on Service Oriented Computing.

Verma, K., Akkiraju, R., Goodwin, R. (2005). *Semantic matching of Web service policies.* Paper presented at the Second International Workshop on Semantic and Dynamic Web Processes (SDWP 2005), Part of the 3rd International Conference on Web Services (ICWS'05).

Waldo, J. (1999, October). The Jini architecture for network-centric computing. *Communications of the ACM, 42*(10), 76-82.

Weeratunge, D., Weerawarana, S., & Gunarathne, T. (2004) *Kandula - Apache Kandula.* Retrieved October 18, 2006, from http://ws.apache.org/kandula/

XML (2005). Extensible Markup Language (XML) 1.0 (3rd ed.). W3C Recommendation 04 February 2004. Retrieved October 18, 2006, from http://www.w3.org/TR/REC-xml/