SEMANTIC WEB

Unit-II

Syllabus

Web Services: Uses, Basics of Web Services, SOAP, UDDI, Orchestrating Web Services, Securing Web Services, Grid Enabled and Semantic Web of Web Services

Web services:-Web services are software applications that can be discovered, described, and accessed based on XML and standard Web protocols over intranets, extranets, and the Internet.

Breakdown explanation of the above definition

Web services are software applications means Web services are software applications available on the Web that perform specific functions.

Web service are Built on XML, because

1. A Standard that is supported and accepted by thousands of vendors worldwide,
2. it first focus on interoperability.
3. XML is the syntax of messages, and Hypertext Transport Protocol (HTTP), the underlying protocol, is how applications send XML messages to Web services.

Web services technologies, such as **U**niversal **D**escription, **D**iscovery, and **I**ntegration (**UDDI**) and ebXML registries, allow applications to dynamically discover information about Web services—the ***“discovered”*** part of our definition.

The message syntax for a Web service is ***described*** in WSDL, the **W**eb **S**ervice **D**efinition **L**anguage. When most technologists think of Web services, they think of SOAP, the ***“accessed”*** part of our Web services definition. SOAP, developed as the Simple Object Access Protocol, is the XML-based message protocol (or API) for communicating with Web services.

Figure 4.1 gives a graphical view of that definition, shown as layers. Relying on the foundation of XML for the technologies of Web services, and using HTTP as the underlying protocol, the world of Web services involves standard protocols to achieve the capabilities of access, description, and discovery.

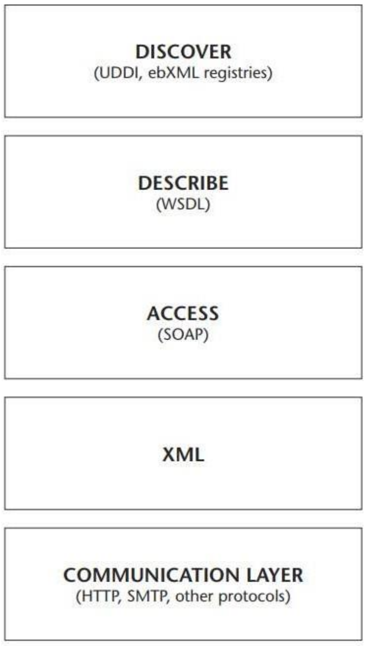


Figure 4.1 The basic layers of Web services.

Figure 4.2 shows these technologies in use in a common scenario. In Step 1, the client application discovers information about Web Service A in a UDDI registry. In Step 2, the client application gets the WSDL for Web Service A from the UDDI registry to determine Web Service A's API. Finally, in Steps 3 and 4, the client application communicates with the Web service via SOAP, using the API found in Step 2.

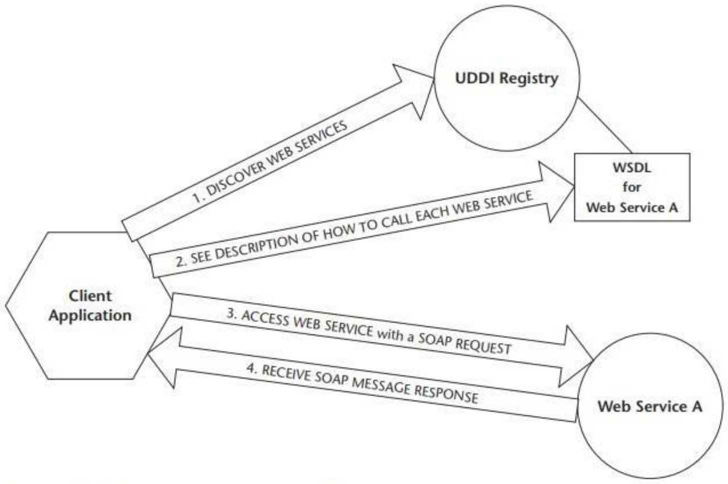

UDDI Registry
1. DISCOVER WEB SERVICES
WSDL for Web Service A
2. SEE DESCRIPTION OF HOW TO CALL EACH WEB SERVICE
Client Application
3. ACCESS WEB SERVICE with a SOAP REQUEST
4. RECEIVE SOAP MESSAGE RESPONSE
Web Service A

Figure 4.2 A common scenario of Web services in use.

Important points about web service

1. Web services can be completely independent of the presentation, or the graphical user interface (GUI) of applications.

2. Web services send data in XML format, and applications can add style and formatting when they receive the data.
3. Separating business logic from presentation is commonly known in software engineering as the Model-View-Controller (MVC) paradigm. Web services support this paradigm. Shown in Figure 4.3, The user interface details (the view) and business logic (the model) are separated in two different components, while the component layer between them (the controller) facilitates communication.

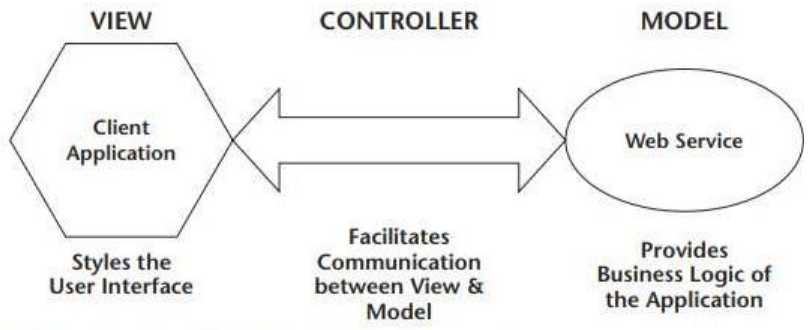

VIEW
CONTROLLER
MODEL
Client Application
Web Service
Styles the User Interface
Facilitates Communication between View & Model
Provides Business Logic of the Application

Figure 4.3 The Model-View-Controller paradigm.

This approach has benefits:

1. **Focus on Business Logic:** Developers can focus on building the core functionality (blueprints) without worrying about the presentation.
2. **Flexibility:** The user interface (interior design) can be tailored to different needs and platforms (browsers) without affecting the core functionality.

Why Use Web Services?

1. Do Web services solve real problems? What problems do Web services solve?

2. Is there really a future for Web services? That is, will this market continue to grow? Will Web services really play a big part in the next generation of the Web—or are we drowning in technology hype?
3. How can I use Web services? Where exactly can my business focus to take advantage of the technology?

a) **Do Web services solve real problems?**

Many businesses suffer from integration problems in our fast-paced world of ever-changing technologies, market conditions, and business relationships. Because of different database languages, different communication protocols, and different ways of expressing

problems in languages understood by computers, integrating systems is extremely difficult. So, we have integration problems, but we want to solve them quickly. The problem with solutions in the past is that integration efforts have taken too long, and we have created new stovepipes by creating inflexible, hard-tochange architectures.
We need a new technology that can solve integration problems by providing a common language that could be used in integration— both within and between enterprises. Without agreement on a common language, there would be no interoperability.

Now that major businesses have adopted SOAP(HTTP became a widely supported application-layer protocol, and SOAP was developed using HTTP as its foundation) for the communication medium between applications and servers, this ensures that everyone's applications have a chance to speak a common language. Web services are based on SOAP and represent our current state of evolution in communication agreement.

b) Is There Really a Future for Web Services?

One of the major indicators of a successful technology is its adoption by key business players, that is

1. **Technical excellence doesn't guarantee success**: Many great technologies have failed because businesses didn't adopt them.

2. **Mass adoption by major players is crucial**: When key businesses use and promote a technology, it increases its chances of success.

When the technology solves key problems, is simple to understand, and is adopted by all key businesses, its success in the future is nearly ensured.

So, while a technology's functionality is important, it's equally important for it to be commercially attractive and gain traction within the business landscape.

The success of Web services seems likely:

When giants such as Microsoft, IBM, Sun, and the ***open source community*** agree on something, it is not only a major milestone. it is a sign that whatever they have agreed on has a big future. Widespread adoption of core protocol (SOAP) further strengthens the technology's future.

c) How Can I Use Web Services?

Web services: Benefits for different groups:

1. ***Application vendors***: Creating a SOAP API is essential for wider compatibility across platforms.

2. ***Service providers:*** Web services can offer new ways to deliver services to individuals and businesses.

3. ***Organizations with legacy systems:*** Web services can potentially integrate these systems without requiring complete overhauls.

The value of Web services lies in their ability to connect different systems (interoperability). This allows businesses to:

1. ***Focus on business processes:*** By solving communication problems, businesses can focus on their core operations.

2. ***Achieve networked business goals:*** Seamless data exchange between internal departments (EAI) and external partners (B2B) becomes possible.

EAI is currently the killer app for Web services. Because we are at the stage of Web services where legacy applications can be made Web service-enabled via SOAP, EAI is doable now. Most analysts believe that organizations will adopt Web services “from the inside out.”. if your systems have SOAP interfaces, integrating them will be easier. Tying together your internal infrastructure, such as Enterprise Resource Planning, customer relationship management, project management, value chain management, and accounting, all with Web services, will eventually prepare you to interoperate with business partners on a B2B basis. Web services allow you to integrate your internal processes, saving time and money.

B2B: The future of Web services?

While **EAI** is driving the current use of Web services internally, **B2B** integration may be its future potential.

1. **Standards being developed:** Organizations like OASIS are working on standards to enable seamless communication for B2B interactions.
2. **Benefits of early adoption:** Businesses that adopt Web services now will be well-positioned for the future B2B boom.
3. **Possible intermediary step:** Before full-fledged B2B, companies may use Web services to collaborate with specific partners within private networks (extranets).In essence, while internal integration (EAI) is the present focus, B2B collaboration holds strong potential for the future of Web services.

Understanding the Basics of Web Services

What Is SOAP?

SOAP (Simple Object Access Protocol) is the “envelope” that packages the XML messages that are sent over HTTP between clients and Web services.

W3C defined SOAP as , SOAP is “a lightweight protocol for exchange of information in a decentralized, distributed environment.” It provides a standard language for tying applications and services together. An application sends a SOAP request to a Web service, and the Web service returns the response in something called a SOAP response.

SOAP has been adopted as the standard for Web services, and applications from major vendors have developed SOAP APIs for their products, thus making software systems integration easier. The syntax of SOAP, in its basic form, is fairly simple, as shown in Figure 4.4. A SOAP message contains the following elements:

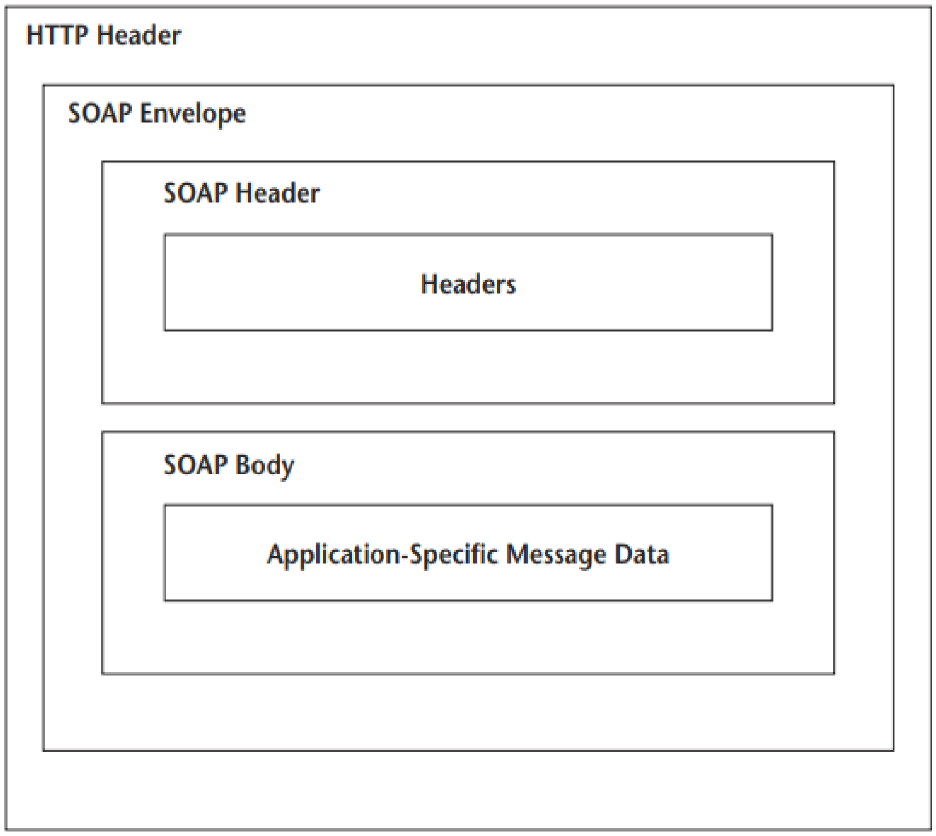
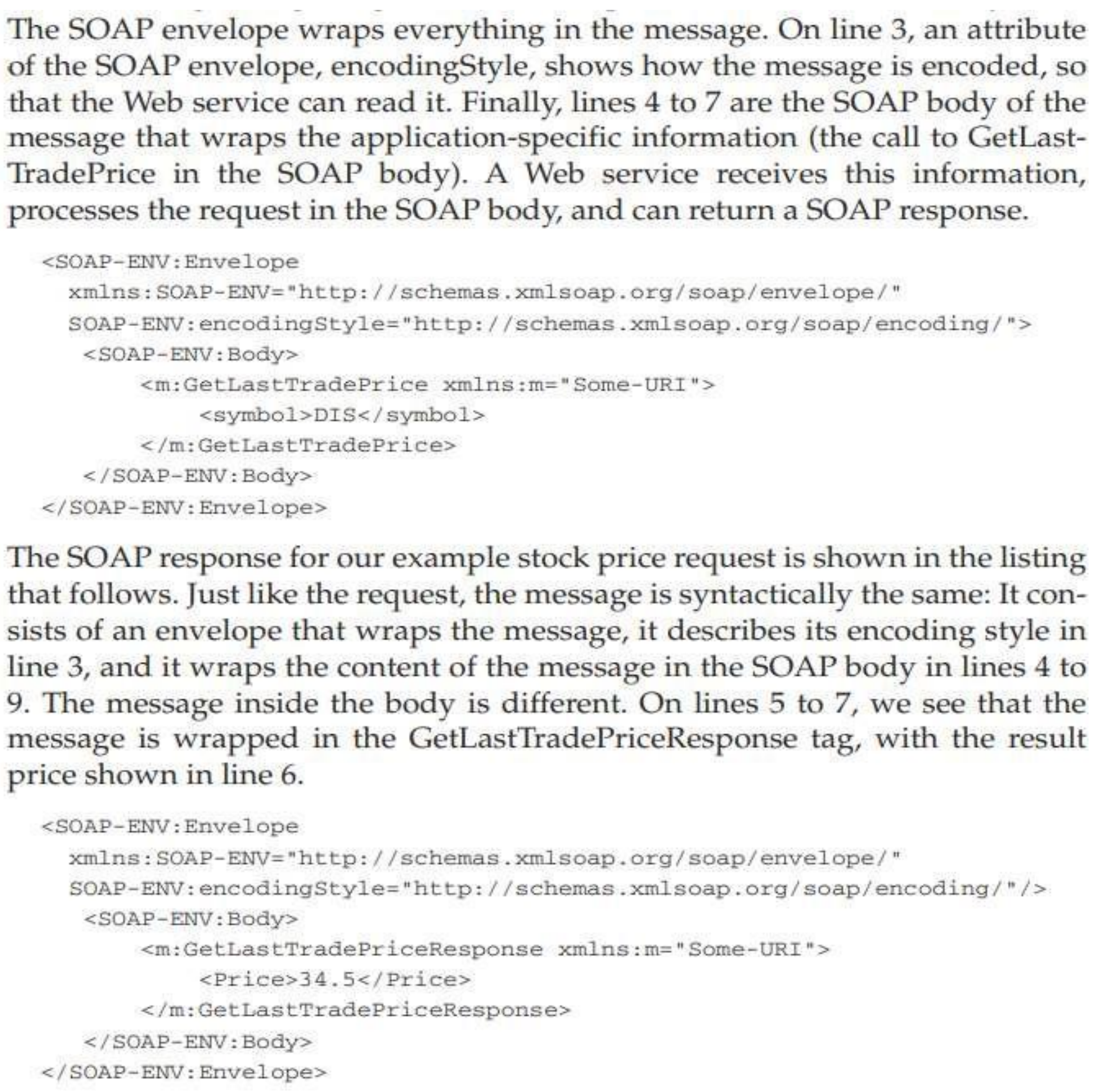


Figure 4.4 Structure of a Web-based SOAP message.



The SOAP envelope wraps everything in the message. On line 3, an attribute of the SOAP envelope, encodingStyle, shows how the message is encoded, so that the Web service can read it. Finally, lines 4 to 7 are the SOAP body of the message that wraps the application-specific information (the call to GetLast-TradePrice in the SOAP body). A Web service receives this information, processes the request in the SOAP body, and can return a SOAP response.

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <SOAP-ENV:Body>
       <m:GetLastTradePrice xmlns:m="Some-URI">
           <symbol>DIS</symbol>
       </m:GetLastTradePrice>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The SOAP response for our example stock price request is shown in the listing that follows. Just like the request, the message is syntactically the same: It consists of an envelope that wraps the message, it describes its encoding style in line 3, and it wraps the content of the message in the SOAP body in lines 4 to 9. The message inside the body is different. On lines 5 to 7, we see that the message is wrapped in the GetLastTradePriceResponse tag, with the result price shown in line 6.

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
   <SOAP-ENV:Body>
       <m:GetLastTradePriceResponse xmlns:m="Some-URI">
           <Price>34.5</Price>
       </m:GetLastTradePriceResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

How to Describe Basic Web Services

SOAP is the communication language of Web services, Web Service Definition Language (WSDL) is the way we describe the communication details and the application-specific messages that can be sent in SOAP. WSDL, like SOAP, is an XML grammar. The W3C defines WSDL as "***an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information***." *To know how to send messages to a particular Web service, an application can look at the WSDL and dynamically construct SOAP messages.*

WSDL describes the operational information—where the service is located, what the service does, and how to talk to (or invoke) the service. It can be thought of as an XML form of CORBA's Interface Definition Language (IDL).

Developers and integrators do not have to understand WSDL and SOAP to create Web services. When you create a Web service from your enterprise applications, most toolkits create WSDL for you. Figure 4.5 shows an example of how this process works. In Figure 4.5, the Web service developer creates a WSDL description with developer tools that inspect the Web service's SOAP interface layer in Step 1. In Step 2, the client application generates the code for handling the Web service (its SOAP handler) by looking at the WSDL. Finally, in Step 3, the client application and the Web service can communicate.

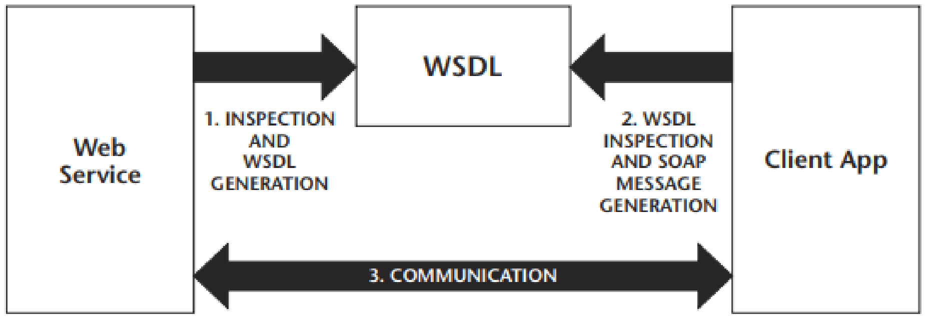

WSDL
Web Service
1. INSPECTION AND WSDL GENERATION
2. WSDL INSPECTION AND SOAP MESSAGE GENERATION
Client App
3. COMMUNICATION

Figure 4.5 Dynamic communication by inspecting WSDL.

How to Discover Web Services.

When we would like to search for Web services based on the features they provide and then dynamically connect to them and use them, we need a Web service registry. Finding Web services based on what they provide introduces two key registry technologies:

1. UDDI (Universal Description, Discovery, and Integration) and
2. ebXML(electronic business XML) registries.

What Is UDDI?

UDDI, introduced in 2000 by Ariba, Microsoft, and IBM, was created to facilitate the discovery of business processes. UDDI is an evolving technology and is not yet a standard, but it is being implemented and embraced by major vendors. Simply put, UDDI is a phone book for Web services.

Two functions of UDDI registry are

1. Organizations can register public information about their Web services and types of services with UDDI, and
2. Applications can view information about these Web services with UDDI. UDDI allows you to discover Web services just like network discovery tools (such as "My Network Places") can discover nodes on a network.

The information provided in a UDDI business registration consists of three components:

1. white pages of company contact information.
2. yellow pages that categorize businesses by standard taxonomies, and
3. green pages that document the technical information about services that are exposed.

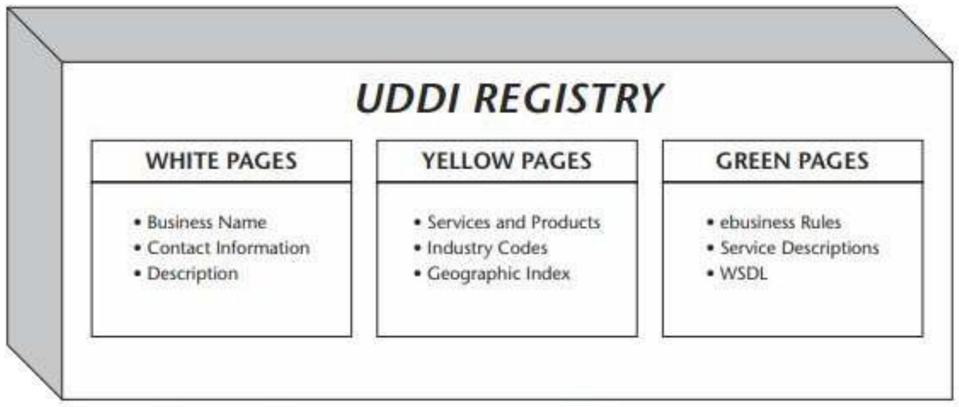


Figure 4.6 A UDDI Registry as a conceptual phone book.

A business's ***white pages*** may include basic business information, such as a description of the business in different languages, points of contact with email addresses and phone numbers, and links to external documents that describe the business in more detail.

The ***yellow pages*** describe taxonomies of what kinds of information the services provide.

Finally, the ***green pages*** show information on how to do business with the Web service, listing business rules and specifying how to invoke Web services (the WSDL).

The business can decide what it would like to register about itself in a registry, and once the information is in there, applications can discover and browse the information, getting information on how to do business with the organization.

Obviously, there may be security concerns about placing information in a public registry. Although it can be seen that such a technology could be very powerful in a dynamic e-business environment, placing information about all of your assets in a public registry may be risky.

For internal integration, the use of ***<u>UDDI private registries</u>*** may be where much value is today. Within a large organization, where several large enterprise applications may need to interoperate in the future, the use of UDDI registries within the organization can be helpful for discovering how to do so. The use of such a private registry, where Web services are described with WSDL and other information, could potentially minimize the use of interoperation documentation. Using such a registry, in addition to WSDL and SOAP, could reduce integration and development time for legacy enterprise applications. Once applications have been SOAP-enabled, and once the interfaces have been described in WSDL and published in a private UDDI registry, programs and projects within your organization can dynamically connect and begin to interoperate.

Although UDDI has been embraced by major vendors, such as Microsoft and IBM, it is evolving and changing. It is not yet a standard, but it will be.

What Are ebXML Registries?

The ebXML standard was created by OASIS to link traditional data exchanges to business applications to enable intelligent business processes using XML. Because XML by itself does not provide semantics to solve interoperability problems, **ebXML** was developed as a mechanism for XML-based business vocabularies. In short, ebXML provides a common way for businesses to quickly and dynamically perform business transactions based on common business practices. Figure 4.7 shows an example of an ebXML architecture in use. In the diagram, company business process information and implementation details are found in the ebXML registry, and businesses can do business transactions after they agree on trading arrangements. Information that can be described and discovered in an ebXML architecture includes the following:

1. Business processes and components described in XML
2. Capabilities of a trading partner
3. Trading partner agreements between companies

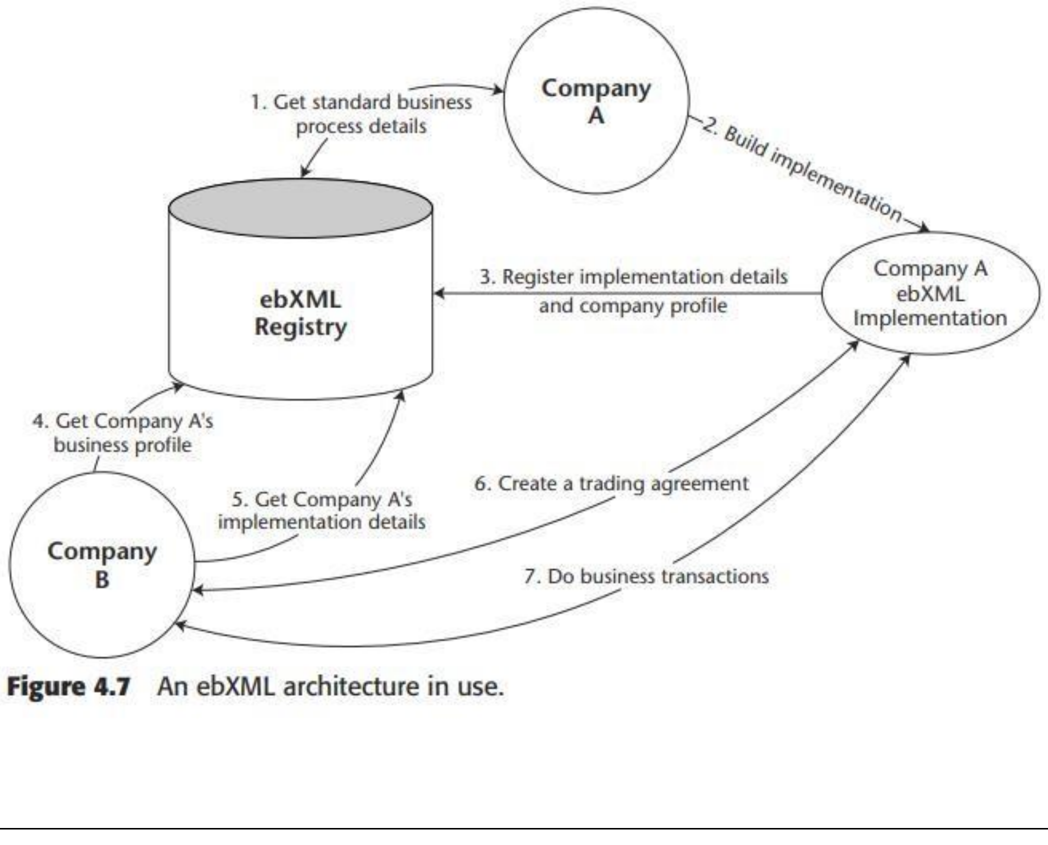

1. Get standard business process details
Company A
2. Build implementation
ebXML Registry
3. Register implementation details and company profile
Company A ebXML Implementation
4. Get Company A's business profile
5. Get Company A's implementation details
6. Create a trading agreement
7. Do business transactions
Company B

Figure 4.7 An ebXML architecture in use.

The heart of the ebXML architecture is the ebXML registry, which is the mechanism that is used to store and discover this information. Although it seems similar in purpose to UDDI, the ebXML registry contains domain-specific semantics for B2B. These domain-specific semantics are the product of agreement on many business technologies and protocols, such as EDI, SAP, and RosettaNet. Simply put, ebXML could be described as the start of a domain specific Semantic Web.

Orchestrating Web Services

Orchestration is the process of combining simple Web services to create complex, sequence-driven tasks. This process, sometimes called flow composition or Web service choreography, involves creating business logic to maintain conversations between multiple Web services. Orchestration can occur between an application and multiple Web services, or multiple Web services can be chained into a workflow, so that they can communicate with one another.

A Simple Example

For our example, we'll list five separate Web services within a fictional organization: a hotel finder Web service, a driving directions finder, an airline ticket booker, a car rental service, and an expense report creator:

Hotel finder Web service: This Web service provides the ability to search for a hotel in a given city, list room rates, check room availability, list hotel amenities, and make room reservations.

Driving directions finder: This Web service gives driving directions and distance information between two addresses.

Airline ticket booker: This Web service searches for flights between two cities in a certain timeframe, lists all available flights and their prices, and provides the capability to make flight reservations.

Car rental Web service: This provides the capability to search for available cars on a certain date, lists rental rates, and allows an application to make a reservation for a car.

Expense report creator**:** This Web service automatically creates expense reports, based on the expense information sent.

By themselves, these Web services provide simple functionality. By using them together, however, a client application can solve complex problems.

Consider the following scenario:

After your first week on the job, your new boss has requested that you go to Wailea, Maui, on a business trip, where you will go to an important conference at Big Makena Beach. (We can dream, can't we?) Given a limited budget, you are to find the cheapest airline ticket, a hotel room less than $150 a night, and the cheapest rental car, and you need to provide this documentation to your internal accounting department. For your trip, you want to find a hotel that has a nonsmoking room and a gym, and you would like to use your frequent flyer account on Party Airlines. Because you don't like to drive, you would like to reduce your car driving time to a minimum.

Luckily, the software integrators at your organization were able to compose the existing Web services into an application that accomplishes these tasks. Going to your organization's internal Web site, you fill in the required information for your trip and answer a few questions online. Because the internal application resides in your organization, you have assurance of trust and can provide your credit card to the application.

Figure 4.8 shows a high-level diagram of our application's solution. The following steps took place in this example:

1. The client application sent a message to the hotel finder Web service, looking for the name, address, and the rates of hotels (with nonsmoking rooms, local gyms, and rates below $150 a night) available in the Wailea, Maui, area during the duration of your trip.

2. The client application sent a message to the driving directions finder Web service. For the addresses returned in Step 1, the client application requests the distance to Big Makena Beach. Based on the distance returned for the requests to this Web service, the client application finds the four closest hotels.

3. After finding the four closest hotels, the client application requested the user to make a choice. Once that choice was selected, the application booked a room at the desired hotel by sending another message to the hotel finder Web service.

4. Based on the user's frequent flyer information on Party Airlines and the date of the trip to Maui, the client application sent a message to the airline ticket booker Web service, requesting the cheapest ticket on Party Airlines, as well as the cheapest ticket in general. Luckily, Party Airlines had the cheapest ticket, so after receiving user confirmation on the flight, the application booked this flight reservation.

5. The client application sent a message to the car rental Web service, requesting the cheapest rental car during the dates of the trip. Because multiple car types were available for the cheapest price, the client application prompted the user for a choice. After the user selected a car model, the client application reserved the rental car for a pickup at the airport arrival time found in Step 4, and the drop-off time at a time two hours prior to the airport departure time.
6. Sending all necessary receipt information found in Steps 1 to 5, the client application requested an expense report generated from the expense report creator Web service. The client application then emailed the resulting expense report, in the corporate format, to the end user.

Our travel example shows important concepts in orchestration. The client application must make decisions based on business logic and may need to interact with the end user. In the example, the Web services were developed internally, so the client application may know all of the Web service-specific calls. In another situation, however, the technologies of Web services provide the possibility that the client application could “discover” the available services via UDDI, download the WSDL for creating the SOAP for querying the services, and dynamically create those messages on the fly. If the client application understands the semantics of how the business process works, this is doable.

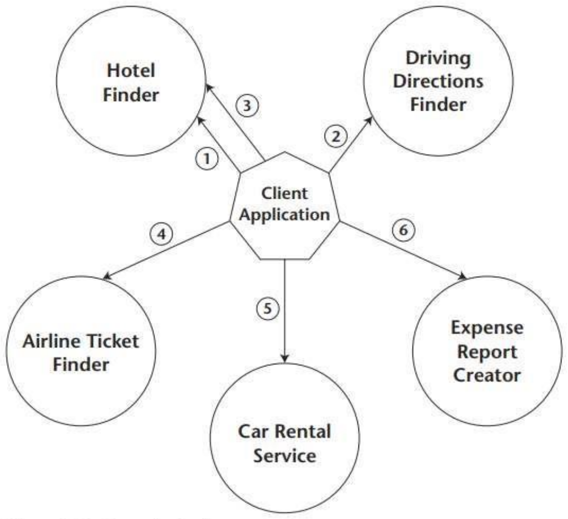


Figure 4.8 An orchestration example.

Securing Web Services

One of the biggest concerns in the deployment of Web services today is security. In a distributed Internet environment where portals may talk to other Web services, which in turn talk to other Web services. in this scenario some of the issues are

1. How can we know the identity of who’s getting the information?
2. How can we know what information that user is allowed to see?

3. With online transactions, how can we have some assurance that the transaction is valid?
4. How can we keep sensitive information transfers confidential?
5. How can we prove, in a court of law, that someone accessed information?
6. How can we know that a user's transmission hasn't been intercepted and changed?

For the purpose of simplicity, lets understand some common vocabulary of security concerns and explain how they are related to Web services security

Authentication

This means validating identity. In a Web services environment, it may be important to initially validate a user's identity in certain transactions. Usually, an organization's infrastructure provides mechanisms for proving a user's identity. Mutual authentication means proving the identity of both parties involved in communication, and this is done using special security protocols. Message origin authentication is used to make certain that the message was sent by the expected sender and that it was not "replayed."

Authorization.

Once a user's identity is validated, it is important to know what the user has permission to do. Authorization means determining a user's permissions. Usually, an organization's infrastructure provides mechanisms (such as access control lists and directories) for finding a user's permissions and roles.

Single sign-on (SSO).

Although this term may not fit with the other security terms in this list, it is a popular feature that should be discussed. SSO is a concept, or a technical mechanism, that allows the user to only authenticate once to her client, so that she does not have to memorize many usernames and passwords for other Web sites, Web services, and server applications. SSO blends the concepts of authentication and authorization enabling other servers to validate a user's identity and what the user is allowed to do. There are many technology enablers for SSO, including Kerberos, Secure Assertion Markup Language (SAML), and other cryptographic protocols.

Confidentiality

When sensitive information is transmitted, keeping it secret is important. It is common practice to satisfy confidentiality requirements with encryption.

Integrity

In a network, making sure data has not been altered in transit is imperative. Validating a message's integrity means using techniques that prove that data has not been altered in transit. Usually, techniques such as hash codes and MAC (Message Authentication Codes) are used for this purpose.

Nonrepudiation

The process of proving legally that a user has performed a transaction is called nonrepudiation. Using digital signatures provides this capability.

In many environments, satisfying these security concerns is vital. We defined the preceding terms from the Web service's perspective, but it is important to know that these security basics may need to be satisfied between every point. That is, ***a user may want assurance that he's talking to the right Web service, and the Web service may want assurance that it is talking to the right user. In addition, every point in between the user and the Web service (a portal, middleware, etc.) may want to satisfy concerns of authentication, authorization, confidentiality, integrity, and nonrepudiation***. Figure 4.9 shows a good depiction of the distributed nature of Web services and its impact on security.

In the figure, if the user authenticates to the portal, how do the next two Web services and the back-end legacy application know the user's identity? If there is any sort of SSO solution, you wouldn't want the user to authenticate four times. Also, between the points in the figure, do the back-end applications have to authenticate, validate integrity, or encrypt data to each other to maintain confidentiality? If messages pass through multiple points, how does auditing work? It is possible that certain organizations may have security policies that address these issues, and if the security policies exist, the ability to address them with solutions for Web services is important.

Fortunately, technologies for Web services security and XML security have been evolving over the past few years. Some of these technologies are ***XML Signature, XML Encryption, XKMS, SAML, XACML, and WS-Security.***

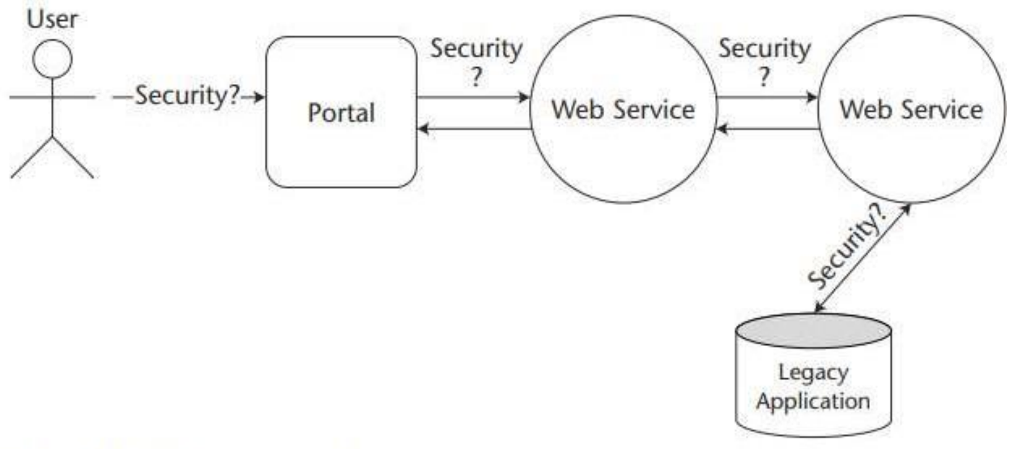


Figure 4.9 Protection at every point?

XML Signature

XML Signature is a W3C Recommendation that provides a means to validate ***message integrity and nonrepudiation***. With XML Signature, any part of an XML document can be digitally signed. In fact, multiple parts of an XML document can be signed by different people or applications. XML Signature, sometimes called XML-DSIG or XML-SIG, relies on public key technology in which the hash (or message digest) of a message is cryptographically signed. Because of the nature of public key signatures, anyone with the signer's digital certificate (or public key) can validate that the signer indeed signed the message, providing legal proof that the signer cannot refute. A trusted third party can look at the message and validate that the message was indeed digitally signed by the sender. A digitally signed message can verify the message's origin and content, which can also serve as authentication of SOAP messages. For example, in Figure 4.9, the user could sign part of the message that is initially sent to the portal and that initially needs to be read by the last Web service. When that part of the message gets to the final Web service, it can validate that the user indeed sent the message. XML digital signatures will play an important role in Web services security. If a Web service sells widgets, and a purchase order for 500 widgets is made, making sure that the message wasn't altered (for example, someone changing the purchase to 5000 widgets) will be important, as will be ensuring that the purchaser digitally signed the purchase order to provide the legal proof.

XML Encryption

XML Encryption is a technology and W3C Candidate Recommendation that handles confidentiality; it can hide sensitive content, so that only the intended recipient can read the sensitive information. In an XML file, different parts of the document can be encrypted, while other parts can remain unencrypted. This can be helpful with Web services, when messages may be sent to multiple points before the receiver gets the message. Different ciphers (encryption mechanisms) can be used, including symmetric (secret key) and public key encryption. If confidentiality is a factor in Web services, a part of the applicationspecific SOAP message may be encrypted to the intended recipient. For example, in Figure 4.9, one of the back-end Web services may encrypt a piece of information so that only the intended user can see the contents of the message. Although the message may travel through many servers, only the intended user should be able to read the message.

XML encryption will also play an important role in Web services security. In the purchasing scenario that is in XML Signature, we provided an example of a Web service that sells widgets. While the purchase request itself may be signed, it may be important to encrypt confidential information, such as the credit card number.

XKMS

XML Key Management Specification (XKMS) is a W3C Note that was developed jointly by the W3C and the IETF, and it specifies protocols for registering and distributing public keys. It is something that is intended for use in conjunction with XML Signature and XML Encryption. XKMS is composed of the XML Key Information Service Specification (X-KISS) and the XML Key Registration Service Specification (X-KRSS). These protocols can be used with SOAP for securely distributing and finding key information.

XACML

Extensible Access Control Markup Language (XACML) is an initiative driven by OASIS that expresses access control policy (authentication and authorization information) for XML documents and data sources. It is currently under development. In simple terms, it relates to SAML in the sense that SAML provides the mechanism of propagating authentication and authorization information between services and servers, and XACML is the authentication and authorization information. The idea of XACML is that XML documents (or SOAP messages themselves) can describe the policy of who can access them, which has interesting potential.

WS-Security

The WS-Security specification was released in April 2002 by Microsoft, IBM, and VeriSign, and is a specification that describes enhancements to SOAP messaging to provide protection through integrity, confidentiality, and message authentication. It combines SOAP with XML Encryption and XML Signature, and is intended to complement other security models and other security technologies. WS-Security also includes a family of specifications, including specifications unveiled in December 2002: WS-Policy, WS-Trust, and WS-SecureConversation.

SAML

Security Assertion Markup Language (SAML) is an OASIS standard that has received industry wide support and acceptance, and it promises to be key in the achievement of SSO in Web services. An initiative driven by OASIS that is used for passing authentication and authorization information between parties. SAML provides "assertions" of trust. That is, an application can assert that it authenticated a user, and that the user has certain privileges. A SAML document can be digitally signed using XML Signature, providing nonrepudiation of a user's original authentication, identity, and authorization credentials. Because SAML is used to distribute information between platforms and organizations, regardless of how many points it crosses, it can solve tough challenges in Web services security.

In Figure 4.9, for example, if the portal authenticates the user “Alice” and knows that Alice has the “Producer” role, the portal application will attach this assertion to a SOAP message with the request to the next Web service. The next Web service, seeing that it can validate the portal’s identity by validating its digital signature, can then grant or deny access to the user based on the user’s role. SAML is an OASIS standard, and it has industry wide support. It is a key technology enabler in SSO initiatives such as the Liberty Alliance Project, and a working draft of a WS-Security profile of SAML has been recently released.

Liberty Alliance Project

The Liberty Alliance Project was established by a group of corporations with the purpose of protecting consumer privacy and establishing an open standard for achieving “federated network identity” for SSO across multiple networks, domains, and organizations. Using the specifications of this project, organizations have the potential to partner in a “federation” so that the credentials of users can be trusted by a group. Federated SSO enables users to sign on once to one site and subsequently use other sites within a group without having to sign on again. The Liberty Alliance released specifications in the summer of 2002, and these specifications include protocols that use XML Encryption, XML Signature, and SAML.

Grid-Enabled Web Services

Grid computing is a technology concept that can achieve flexible, secure, and coordinated resource sharing among dynamic collections of individuals, institutions, and resources. One popular analogy of grid computing is the electric utility grid, which makes power available in our homes and businesses. A user connects to this system with a power outlet, without having to know where the power is coming from and without scheduling an appointment to receive power at any given instant. The power amount that the user requires is automatically provided, the power meter records the power consumed by the user, and the user is charged for the power that is used. In a grid-computing environment, a user or application can connect to a computational grid with a simple interface (a Web portal or client application) and obtain resources without having to know where the resources are. Like the electricity grid, these resources are provided automatically.

A computational grid is a collection of distributed systems that can perform operations. Each individual system may have limitations, but when hundreds, thousands, or millions of systems work together in a distributed environment, much computing power can be unleashed. In a Web services environment, such a concept brings more distributed power to the network. If you want an online production system based on Web services that serves millions of customers, you will need load balancing and fault tolerance on a massive scale.

A Semantic Web of Web Services

The Semantic Web and Web services go hand in hand. XML, a self-describing language, is not enough. WSDL, a language that describes the SOAP interfaces to Web services, is not enough. Automated support is needed in dealing with numerous specialized data formats. In the next 10 years, we will see semantics to describe problems and business processes in specialized domains. Ontologies will be this key enabling concept for the Semantic Web, interweaving human understanding of symbols with machine processibility.

Much effort is going into ontologies in Web services. DARPA Agent Markup Language Services (DAML-S) is an effort that is specifically addressing this area. Built on the foundation of Resource Description Framework (RDF), RDF Schema, and DAML+OIL, DAML-S provides an upper ontology for describing properties and capabilities of Web services in an unambiguous, computer interpretable markup language.5 Simply put, DAML-S is an ontology for Web services. In addition, Semantic Web Enabled Web Services (SWWS) was developed in August 2002 to provide a comprehensive Web service description framework and discovery framework, and to provide scalable Web service mediation. Together, both of these technologies have the potential to increase automated usability of Web services.