WEB SERVICES

UNIT I

Evolution and Emergence of Web Services: Evolution of distributed computing, Core distributed computing technologies, client/server, CORBA, JAVA RMI, Micro Soft DCOM, MOM, Challenges in Distributed Computing, role of J2EE and XML in distributed computing, emergence of Web Services and Service Oriented Architecture (SOA).

Introduction to Web Services: The definition of web services, basic operational model of web services, tools and technologies enabling web services, benefits and challenges of using web services.

Web Services Architecture: Web services Architecture and its characteristics, core building blocks of web services, standards and technologies available for implementing web services, web services communication, basic steps of implementing web services.

UNIT II

Fundamentals of SOAP: SOAP Message Structure, SOAP Encoding, Encoding of different data types, SOAP message exchange models, SOAP communication and messaging, Java and Axis, Limitations SOAP.

UNIT III

Describing Web Services: WSDL, WSDL in the world of Web Services, Web Services life cycle, anatomy of WSDL definition document, WSDL bindings, WSDL Tools, limitations of WSDL.

UNIT IV

Discovering Web Services: Service discovery, role of service discovery in a SOA, service discovery mechanisms, UDDI: UDDI Registries, uses of UDDI Registry, Programming with UDDI, UDDI data structures, Publishing API, Publishing information to a UDDI Registry, searching information in a UDDI Registry, limitations of UDDI.

UNIT V

Web Services Interoperability: Means of ensuring Interoperability, Overview of .NET, Creating a .NET Client for an Axis Web Services, Creating Java Client for a web service, Challenges in Web Services Interoperability.

Services Security: XML security frame work, Goals of cryptography, Digital Signature, Digital Certificate, XML encryption.

TEXT BOOKS:

1. Developing Java Web Services, R. Nagappan, R. Skoczylas, R.P. Sriganesh, Wiley India, rp 2008.
2. Developing Enterprise Web Services, S. Chatterjee, J. Webber, Pearson Education, 2008.
3. XML, Web Services, and the Data Revolution, F.P.Coyle, Pearson Education.

REFERENCES:

1. Building Web Services with Java, Second Edition, S. Graham and others, Pearson Edn., 2008.
2. Java web services, D.A. Chappell and T.Jewell, O'Reilly,SPD.
3. Java Web Services Architecture, McGovern, Sameer Tyagi etal.., Elsevier.
4. Web Services, G. Alonso, F. Casati and others, Springer, 2005.

WEB SERVICES

UNIT-1

Evolution of Distributed Computing:-

In the early years of computing, mainframe-based applications were considered to be the best-fit solution for executing large-scale data processing applications. With the advent of personal computers (PCs), the concept of software programs running on standalone machines became much more popular in terms of the cost of ownership and the ease of application use. With the number of PC-based application programs running on independent machines growing, the communications between such application programs became extremely complex and added a growing challenge in the aspect of application-to-application interaction. Lately, network computing gained importance, and enabling remote procedure calls (RPCs) over a network protocol called Transmission Control Protocol/Internet Protocol (TCP/IP) turned out to be a widely accepted way for application software communication. Since then, software application running on a variety of hardware platforms, operating systems, and different networks faced some challenges when required to communicate with each other and share data. This demanding requirement leads to the concept of distributed computing applications. As a definition, "Distributing Computing is a type of computing in which different components and objects comprising an application can be located on different computers connected to a network distributed computing model that provides an infrastructure enabling invocations of object functions located anywhere on the network. The objects are transparent to the application and provide processing power as if they were local to the application calling them.

Importance of Distributed Computing

The distributed computing environment provides many significant advantages compared to a traditional standalone application. The following are

Some of those key advantages:

Higher performance. Applications can execute in parallel and distribute the load across multiple servers.

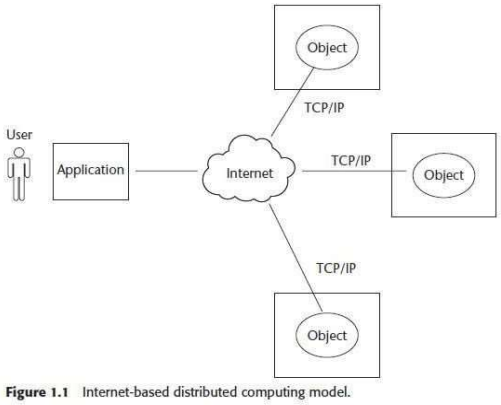


Figure 1.1 Internet-based distributed computing model.

Collaboration. Multiple applications can be connected through standard distributed computing mechanisms.

Higher reliability and availability. Applications or servers can be clustered in multiple machines.

Scalability. This can be achieved by deploying these reusable distributed components on powerful servers.

Extensibility. This can be achieved through dynamic (re)configuration of applications that are distributed across the network. Higher productivity and lower development cycle time. By breaking up large problems into smaller ones, these individual components can be enveloped by smaller development teams in isolation.

components. Reduced cost. Because this World model provide satfeuse once developed **components** that are accessible over the network, significant cost educations can be achieved.

Reuse. The distributed components may perform various se vices that can potentially be used by multiple client applications. It saves repetitive development effort and improves interoperability between

Distributed computing also has changed the way traditional network programming is done by providing a shareable object like semantics across networks using programming languages like Java, C, and C++ . The following sections briefly discuss core distributed computing technologies such as Client/Server applications, OMG CORBA, Java RMI, Microsoft COM/DCOM, and MOM.

Client-Server Applications

The early years of distributed application architecture were dominated by two-tier business applications. In a two-tier architecture model, the first (upper) tier handles the presentation and business logic of the user application (client), and the second/lower tier handles the application organization and its data storage (server). This approach is commonly called client-server applications architecture. Generally, the server in a client/server application model is a database server that is mainly responsible for the organization and retrieval of data. The application client in this model handles most of the business processing and provides the graphical user interface of the application. It is a very popular design in business applications where the user.

interface and business logic are tightly coupled with a database server for handling data retrieval and processing.

For example, the client-server model has been widely used in enterprise resource planning (ERP), billing, and Inventory application systems where a number of client business applications residing in multiple desktop systems interact with a central database server.

Figure 1.2 shows an architectural model of a typical client server system in which multiple desktop-based business client applications access a central database server.

Some of the common limitations of the client-server application model are as follows:

■ Complex business processing at the client side demands robust client systems.

■ Security is more difficult to implement because the algorithms and logic reside on the client side making it more vulnerable to hacking.

■ Increased network bandwidth is needed to accommodate many calls to the server, which can impose scalability restrictions.

■ Maintenance and upgrades of client applications are extremely difficult because each client has to be maintained separately.

■ Client-server architecture suits mostly database-oriented standalone applications and does not target robust reusable component oriented applications.

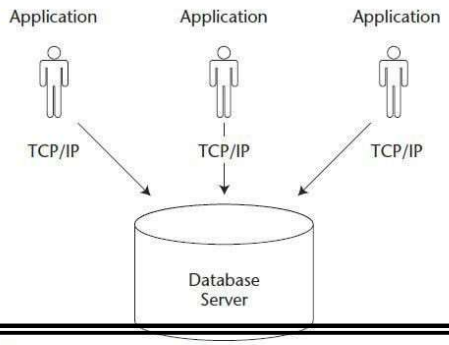


Figure 1.2 An example of a client-server application.

CORBA

The Common Object Request Broker Architecture (CORBA) is an industry wide, open standard initiative, developed by the Object Management Group (OMG) for enabling distributed computing that supports a wide range of application environments. OMG is a nonprofit consortium responsible for the production and maintenance of framework specifications for distributed and interoperable object-oriented systems.

CORBA differs from the traditional client/server model because it provides an object-oriented solution that does not enforce any proprietary protocols or any particular programming language, operating system, or hardware platform. By adopting CORBA, the applications can reside and run on any hardware platform located anywhere on the network, and can be written in any language that has mappings to a neutral interface definition called the Interface Definition Language (IDL). An IDL is a specific interface language designed to expose the services (methods/functions) of a CORBA remote object. CORBA also defines a collection of system-level services for handling low-level application services like life-cycle, persistence, transaction, naming, security, and so forth. Initially, CORBA 1.1 was focused on creating component level, portable object applications without interoperability. The introduction of CORBA 2.0 added interoperability between different ORB vendors by implementing an Internet Inter-ORB Protocol (IIOP). The IIOP defines the ORB backbone, through which other ORBs can bridge and provide interoperation with its associated services. In a CORBA-based solution, the Object Request Broker (ORB) is an object bus that provides a transparent mechanism for sending requests and receiving responses to and from objects, regardless of the environment and its location. The ORB intercepts the client's call and is responsible for finding its server object that implements the request, passes its parameters, invokes its method, and returns its results to the client. The ORB, as part of its implementation, provides interfaces to the CORBA services, which allows it to build custom-distributed application environments.

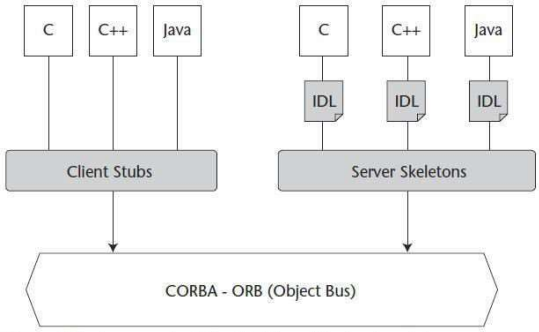


Figure 1.3 An example of the CORBA architectural model.

Figure 1.3 illustrates the architectural model of CORBA with an example representation of applications written in C, C++, and Java providing IDL bindings.

The CORBA architecture is composed of the following components:

IDL. CORBA uses IDL contracts to specify the application boundaries and to establish interfaces with its clients. The IDL provides a mechanism by which the distributed application component's interfaces, inherited classes, events, attributes, and exceptions can be specified.

ORB. It acts as the object bus or the bridge, providing the communication infrastructure to send and receive request/responses from the client and server. It establishes the foundation for the distributed application objects, achieving interoperability in a heterogeneous environment. Some of the distinct advantages of CORBA over a traditional client/server application model are as follows:

OS and programming-language independence. Interfaces between clients and servers are defined in OMG IDL, thus providing the following advantages to Internet programming: Multi-language and

multi-platform application environments, which provide a logical separation between interfaces and implementation.

Legacy and custom application integration. Using CORBA IDL, developers can encapsulate existing and custom applications as callable client applications and use them as objects on the ORB.

Rich distributed object infrastructure. CORBA offers developers a rich set of distributed object services, such as the Lifecycle, Events, Naming, Transactions, and Security services.

Location transparency. CORBA provides location transparency: An object reference is independent of the physical location and application level location. This allows developers to create CORBA-based systems where objects can be moved without modifying the underlying applications.

Java RMI

Java RMI was developed by Sun Microsystems as the standard mechanism to enable distributed Java objects-based application development using the Java environment. RMI provides a distributed Java application environment by calling remote Java objects and passing them as arguments or return values. It uses Java object serialization—a lightweight object persistence technique that allows the conversion of objects into streams. Before RMI, the only way to do inter-process communications in the Java platform was to use the standard Java network libraries. Though the java.net APIs provided sophisticated support for network functionalities, they were not intended to support or solve the distributed computing challenges.

Java RMI uses Java Remote Method Protocol (JRMP) as the inter process communication protocol, enabling Java objects living in different Java Virtual Machines (VMs) to

Transparently invoke one another's methods. Because these VMs can be running on different computers anywhere on the network, RMI enables object-oriented distributed computing. RMI also uses a reference-counting garbage collection mechanism that keeps track f external live object references to remote objects (live connections) using the virtual machine. When an object is found unreferenced, it is considered to be a weak reference and it will be garbage collected.

In RMI-based application architectures, a registry (rmiregistry) - oriented mechanism provides a simple non-persistent naming lookup service that is used to store the remote object references and to enable lookups from client applications. The RMI infrastructure based on the JRMP acts as the medium between the RMI clients and remote objects. It intercepts client requests, passes invocation arguments, delegates invocation requests to the RMI skeleton, and finally passes the return values of the method execution to the client stub. It also enables callbacks from server objects to client applications so that the asynchronous notifications can be achieved. Figure 1.4 depicts the architectural model of a Java RMI-based application solution.

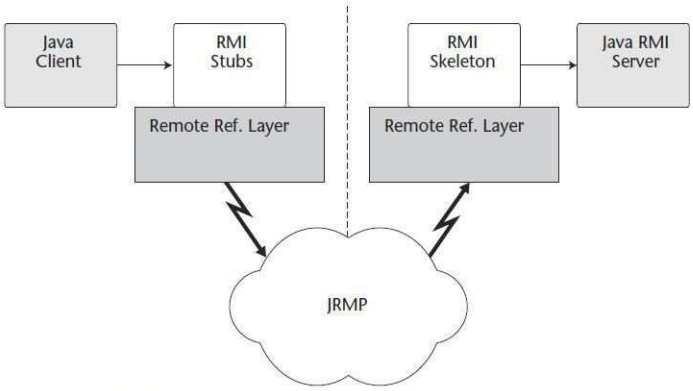


Figure 1.4 A Java RMI architectural model.

The java RMI architecture is composed of the following components:

RMI client. The RMI client, which can be a Java applet or a standalone application, performs the remote method invocations on a server object. It can pass arguments that are primitive data types or serializable objects.

RMI stub. The RMI stub is the client proxy generated by the rmi compiler (*rmic* provided along with Java developer kit—JDK) that encapsulates the network information of the server and performs the delegation of the method invocation to the server. The stub also marshals the method arguments and unmarshals the return values from the method execution.

RMI infrastructure. The RMI infrastructure consists of two layers: the remote reference layer and the transport layer. The remote reference layer separates out the specific remote reference behavior from the client stub. It handles certain reference semantics like connection entries, which are unicast/multicast of the invocation requests. The transport layer actually provides the networking infrastructure, which facilitates the actual data transfer during method invocations, the passing of formal arguments, and the return of back execution results. **RMI skeleton.** The RMI skeleton, which also is generated using the RMI compiler (rmic) receives the invocation requests from the stub and processes the arguments (unmarshalling) and delegates them to the RMI server. Upon successful method execution, it marshals the return values and then passes them back to the RMI stub via the RMI infrastructure.

RMI server. The server is the Java remote object that implements the exposed interfaces and executes the client requests. It receives incoming remote method invocations from the respective skeleton, which

passes the parameters after unmarshalling. Upon successful method execution, return values are sent back to the skeleton, which passes them back to the client via the RMI infrastructure.

Microsoft DCOM

The Microsoft Component Object Model (COM) provides a way for Windows-based software components to communicate with each other by defining a binary and network standard in a Windows operating environment. COM evolved from OLE (Object Linking and Embedding), which employed a Windows registry-based object organization mechanism. COM provides a distributed application model for ActiveX components. As a next step, Microsoft developed the Distributed Common Object

Model (DCOM) as its answer to the distributed computing problem in the Microsoft Windows platform. DCOM enables COM applications to communicate with each other using an RPC mechanism, which employs a DCOM protocol on the wire.

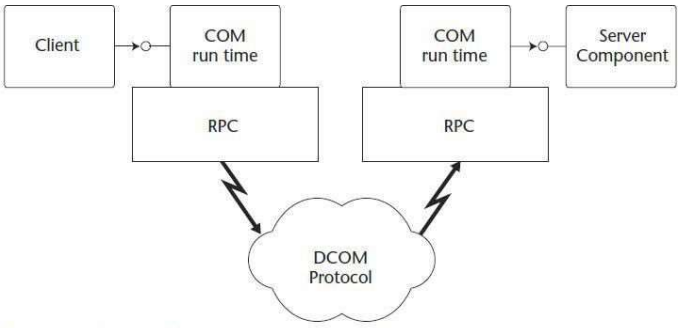


Figure 1.5 Basic architectural model of Microsoft DCOM.

Figure 1.5 shows an architectural model of DCOM. DCOM applies a skeleton and stub approach whereby a defined interface that exposes the methods of a COM object can be invoked remotely over a network. The client application will invoke methods on such a remote COM object in the same fashion that it would with a local COM object. The stub encapsulates the network location information of the COM server object and acts as a proxy on the client side. The servers can potentially host multiple COM objects, and when they register themselves against a registry, they become available for all the clients, who then discover them using a lookup mechanism.

DCOM is quite successful in providing distributed computing support on the Windows platform. But, it is limited to Microsoft application environments. The following are some of the common limitations of DCOM:

- Platform lock-in
- State management
- Scalability
- Complex session management issues

Message-Oriented Middleware

Although CORBA, RMI, and DCOM differ in their basic architecture and approach, they adopted a tightly coupled mechanism of a synchronous communication model (request/response). All these technologies are based upon binary communication protocols and adopt tight integration across their logical tiers, which is susceptible to scalability issues. Message-Oriented Middleware (MOM) is based upon a loosely coupled asynchronous communication model where the application client does not

need to know its application recipients or its method arguments. MOM enables applications to communicate indirectly using a messaging provider queue. The application client sends messages to the message queue (a message holding

area), and the receiving application picks up the message from the queue. In this operation model, the application sending messages to another application continues to operate without waiting for the response from that application.

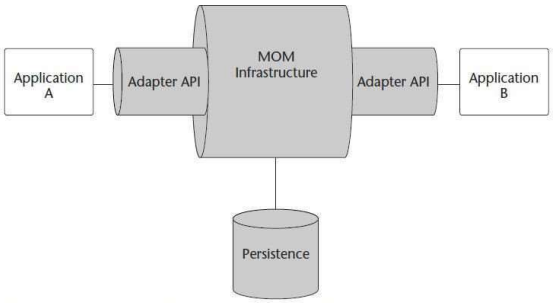


Figure 1.6 A typical MOM-based architectural model.

MS provides Point-to-Point and Publish/Subscribe messaging models with the following features:

- Complete transactional capabilities
- Reliable message delivery
- Security

Some of the common challenges while implementing a MOM-based application environment have been the following:

- Most of the standard MOM implementations have provided native APIs for communication with their core infrastructure. This has affected the portability of applications across such implementations and has led to a specific vendor lock-in.
- The MOM messages used for integrating applications are usually based upon a proprietary message format without any standard compliance.

Challenges in Distributed Computing

Distributed computing technologies like CORBA, RMI, and DCOM have been quite successful in integrating applications within a homogenous environment inside a social area network. As the Internet becomes a logical solution that spans and connects the boundaries of businesses, it also demands the interoperability of applications across networks. This section discusses some of the common challenges noticed in the CORBA-, RMI-, and DCOM-based distributed computing solutions:

- Maintenance of various versions of stubs/skeletons in the client and server environments is extremely complex in a heterogeneous network environment.
- Quality of Service (QoS) goals like Scalability, Performance, and Availability in a distributed environment consume a major portion f the application's development time.
- Interoperability of applications implementing different protocols on heterogeneous platforms almost becomes impossible. For example, a DCOM client communicating to an RMI server or an RMI client
communicating to a DCOM server.
- Most of these protocols are designed to work well within local networks. They are not very firewall friendly or able to be accessed over the Internet.

The Role of J2EE and XML in Distributed Computing

The emergence of the Internet has helped enterprise applications to be easily accessible over the Web without having specific client-side software installations. In the Internet-based enterprise application model, the focus was to move the complex business processing toward centralized servers in the back end. The first generation of Internet servers was based upon Web servers that hosted static Web pages and provided content to the clients via H P (Hyper ext Transfer Protocol). HTTP is a stateless protocol that connects Web browsers to Web servers, enabling the transportation of HTML content to the user.

With the high popularity and potential of this infrastructure, the push for a more dynamic technology was inevitable. This was the beginning of server-side scripting using technologies like CGI, NSAPI, and ISAPI. With many organizations moving their businesses to the Internet, a whole new category of business models like business-to-business (B2B) and business-to-consumer (B2C) came into existence.

This evolution lead to the specification of J2EE architecture, which promoted a much more efficient platform for hosting Web-based applications. J2EE provides a programming model based upon Web and business components that are managed by the J2EE application server.

The application server consists of many APIs and low-level services available to the components. These low-level services provide security, transactions, connections and instance pooling, and concurrency services, which enable a J2EE developer to focus primarily on business logic rather than plumbing. The power of Java and its rich collection of APIs provided the perfect solution for developing highly transactional, highly available and scalable enterprise applications. Based on many standardized industry specifications, it provides the interfaces to connect with various back-end legacy and information systems. J2EE also provides excellent client connectivity capabilities, ranging from PDA to Web browsers to Rich Clients (Applets, CORBA applications, and Standard Java Applications). Figure 1.7 shows various components of the J2EE architecture. A typical J2EE architecture is physically divided in to three logical tiers, which enables clear separation of the various application components with defined roles and responsibilities. The following is a breakdown of functionalities of those logical tiers:

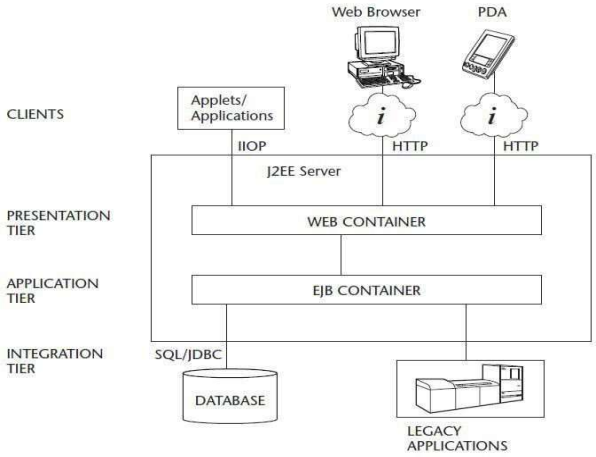


Figure 1.7 J2EE application architecture.

Presentation tier. The Presentation tier is composed of Web components, which handle HTTP quests/responses, Session management, Device independent content delivery, and the invocation of business tier components.

Application tier. The Application tier (also known as the Business tier) deals with the core business logic processing, which may typically deal with workflow and automation. The business components

retrieve data from the information systems with well-defined APIs provided by the application server.

Integration tier. The Integration tier deals with connecting and communicating to back-end Enterprise Information Systems (EIS), database applications and legacy applications, or mainframe applications.

UNIT 2

Emergence of Web Services

Today, the adoption of the Internet and enabling Internet-based applications has created a world of discrete business applications, which co-exist in the same technology space but without interacting with each other. The increasing demands of the industry for enabling B2B, application-to application

(A2A), and inter-process application communication has led to a growing requirement for service-oriented architectures. Enabling service- oriented applications facilitates the exposure of business applications as service components enable business applications from other organizations

to link with these services for application interaction and data sharing without human intervention. By leveraging this architecture, it also enables interoperability between business applications and processes.

By adopting Web technologies, the service-oriented architecture model facilitates the delivery of services over the Internet by leveraging standard technologies such as XML. It uses platform-neutral standards by exposing the underlying application components and making them available to any application, any platform, or any device, and at any location. Today, this phenomenon is well adopted for implementation and is commonly referred to as Web services. Although this technique enables

communication between applications with the addition of service activation technologies and open technology standards, it can be leveraged to publish the services in a register of yellow pages available on the Internet. This will further redefine and transform the way businesses communicate over the Internet. This promising new technology sets the strategic vision of the next generation of virtual business models and the unlimited potential for organizations doing business collaboration and business process management over the Internet.

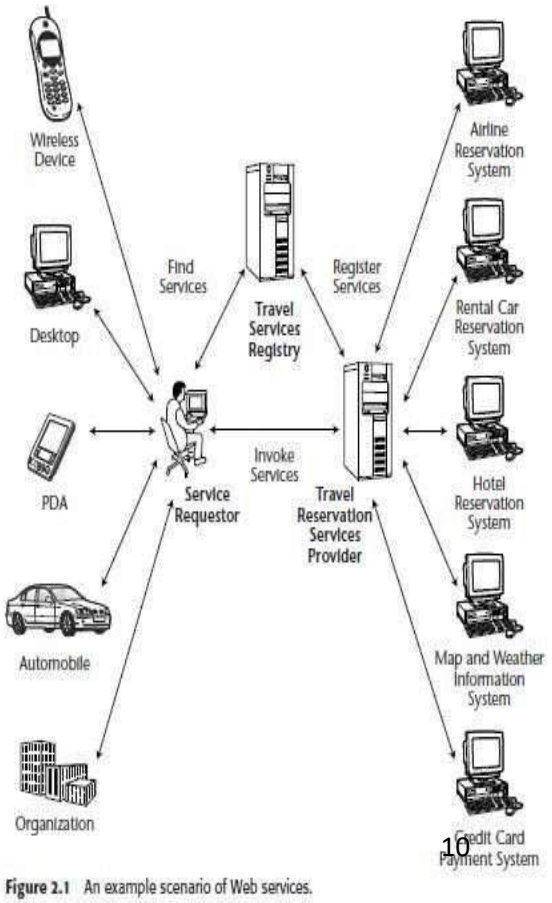


Figure 2.1 An example scenario of Web services.

What Are Web Services

Web services are based on the concept of service-oriented architecture (SOA). SOA is the latest evolution of distributed computing, which enables software components, including application functions, objects, and processes from different systems, to be exposed as services. According to Gartner research

(June 15, 2001), "Web services are loosely coupled software components delivered over Internet standard technologies." In short, Web services are self-describing and modular business applications that expose the business logic as services over the Internet through programmable interfaces and using Internet protocols for the purpose of providing ways to find, subscribe, and invoke those services. Based on XML standards, Web services can be developed as loosely coupled application components using any programming language, any protocol, or any platform. This facilitates delivering business applications as a service accessible to anyone, anytime, at any location, and using any platform. Consider the simple example shown in Figure 2.1 where a travel reservation services provider exposes its business applications as Web services supporting a variety of customers and application clients. These business applications are provided by different travel organizations residing at different networks and geographical locations.

Motivation and Characteristics

Web-based B2B communication has been around f r quite some time. These Web-based B2B solutions are usually based on custom and proprietary technologies and are meant for exchanging data and doing transactions over the Web. However, B2B has its own challenges. For example, in B2B communication, connecting new or existing applications and adding new business partners have always been a challenge. Due to this fact, in some cases the scalability of the underlying business applications is affected. Ideally, the business applications and information from a partner organization should be able to interact with the application of the potential partners seamlessly without redefining the system or its resources. To meet these challenges, it is clearly evident that there is a need for standard protocols and data formatting for enabling seamless and scalable B2B applications and services. Web services provide the solution to resolve these issues by adopting open standards. Figure 2.2 shows a typical B2B infrastructure (e-marketplace) using XML for encoding data between applications across the Internet.

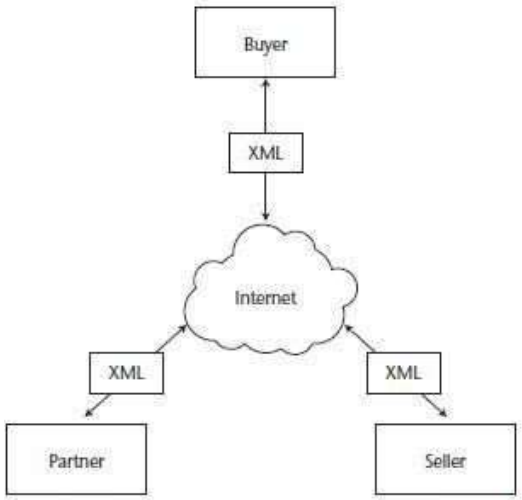


Figure 2.2 Using XML for encoding data in a B2B communication.

Web services enable businesses to communicate, collaborate, conduct business transactions using a lightweight infrastructure by adopting an XML-based data exchange format and industry standard delivery protocols.

The basic characteristics of a Web services application model are as follows:

- Web services are based on XML messaging, which means that the data exchanged between the Web service provider and the user are

defined in XML.

- Web services provide a cross-platform integration of business applications over the Internet.
- To build Web services, developers can use any common programming

language, such as Java, C, C++, Perl, Python, C#, and/or Visual Basic, and its existing application components.

- Web services are not meant for handling presentations like HTML context—it is developed to generate XML for uniform accessibility through any software application, any platform, or device.

- Because Web services are based on loosely coupled application components, each component is exposed as a service with its unique functionality.
- Web services use industry-standard protocols like HTTP, and they can be easily accessible through corporate firewalls.
- Web services can be used by many types of clients.
- Web services vary in functionality from a simple request to a complex business transaction involving multiple resources.
- All platforms including J2EE, CORBA, and Microsoft .NET provide

extensive support for creating and deploying eb services.

- Web services are dynamically located and invoked from public and private registries based on industry standards such as UDDI and ebXML.

Why Use Web Services

Traditionally, Web applications enable interaction between an end user and a Web site, while Web services are service-oriented and enable application to- application communication over the Internet and easy accessibility to heterogeneous applications and devices. The following are the major technical reasons for choosing Web services over Web applications:

- Web services can be invoked through XML-based RPC mechanisms across firewalls.
- Web services provide a cross-platform, cross-language solution based on XML messaging.
- Web services facilitate ease of application integration using a lightweight infrastructure without affecting scalability.
- Web services enable interoperability among heterogeneous applications.

Web Services Architecture and Its Core Building Blocks

The basic principles behind the Web services architecture are based on SOA and the Internet protocols. It represents a composable application solution based on standards and

standards-based technologies. This ensures that the implementations of Web services applications are compliant to standard

specifications, thus enabling interoperability with those compliant applications.

Some of the key design requirements of the Web services architecture are the following:

■ To provide a universal interface and a consistent solution model to define the application as modular components, thus enabling them as exposable services

■ To define a framework with a standards-based infrastructure mo el and protocols to support services-based applications over the Internet

■ To address a variety of service delivery scenarios ranging from e-business (B2C), business-to-business (B2B), peer-to-peer (P2P), and enterprise application integration (EAI)-based application communication

■ To enable distributable modular applications as a centralized and decentralized application environment that supports boundary-less application communication for inter-enterprise and intra-enterprise application connectivity

■ To enable the publishing of services to one or more public or private directories, thus enabling potential users to locate the published services using standard-based mechanisms that are defined by standards organizations

■ To enable the invocation of those services when it is required, subject to authentication, authorization, and other security measures

Web Services Description Language (WSDL)

The Web Services Description Language, or WDDL, is an XML schema based specification for describing Web services as a collection of operations and data input/output parameters as messages. WSDL also defines the communication model with a binding mechanism to attach any transport

protocol, data format, or structure to an abstract message, operation, or endpoint. Listing 3.2 shows a WSDL example that describes a Web service meant for obtaining a price of a book using a GetBookPrice operation.

```
<?xml version="1.0"?>
<definitions name="BookPrice"
targetNamespace="http://www.wiley.com/bookprice.wsdl
" xmlns:tns=http://www.wiley.com/bookprice.wsdl
```

Web Services Communication Models

In Web services architecture, depending upon the functional requirements, it is possible to implement the models with RPC-based synchronous or messaging-based synchronous/ asynchronous communication models. These communication models need to be understood before Web services are designed and implemented.

RPC-Based Communication Model

The RPC-based communication model defines a request/response-based,synchronous communication. When the client sends a request, the client waits until a response is sent back from the server before continuing any operation. Typical to implementing CORBA or RMI communication, the RPC-based Web services are tightly coupled and are implemented with remote objects to the client application. Figure 3.3 represents an RPC-based communication model in Web services architecture. The clients have the capability to provide parameters in method calls to the Web service provider. Then, clients invoke the Web services by sending parameter values to the Web service provider that executes the required methods, and then sends back the return values. Additionally, using RPC based communication, both the service provider and requester can register and discover services, respectively.

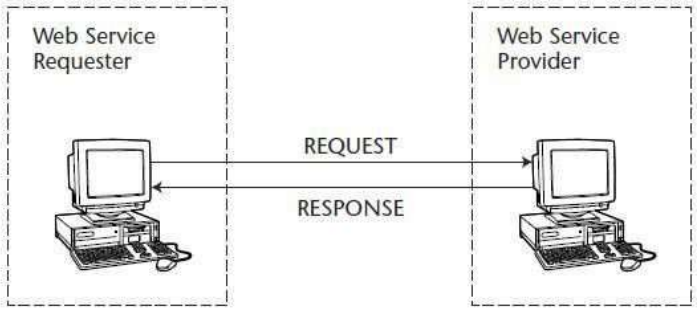


Figure 3.3 RPC-based communication model in Web services.

Implementing Web Services

The process of implementing Web services is quite similar to implementing any distributed application using CORBA or RMI. However, in web services, all the components are bound dynamically only at its runtime using standard protocols. Figure 3.5 illustrates the process highlights of implementing Web services. As illustrated in Figure 3.5, the basic steps of implementing Web services are as follows:

1.The service provider creates the Web service typically as SOAPbased service interfaces for exposed business applications. he provider then deploys them in a service container or using a SOAP runtime environment, and then makes them available for invocation over a network. The service provider also describes the Web service as a WSDL-based service description, which defines the clients and the service container with a consistent

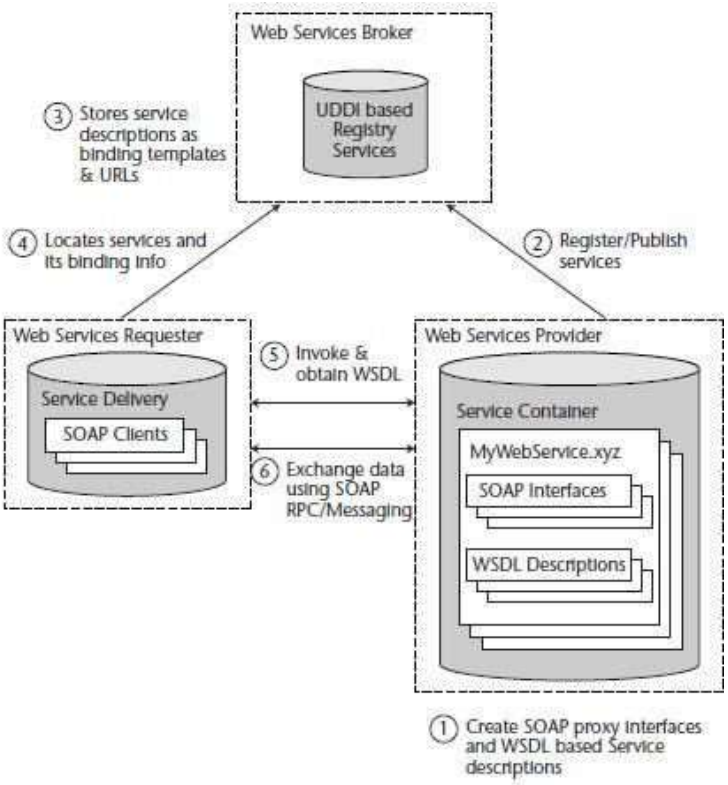


Figure 3.5 Process steps involved in implementing Web services.

way of identifying the service location, operations, and its communication model.

2. The service provider then registers the WSDL-based service description with a service broker, which is typically a UDDI registry.

3. The UDDI registry then stores the service description as binding templates and URLs to WSDLs located in the service provider environment.

4. The service requester then locates the required services by querying the UDDI registry. The service requester obtains the binding information and the URLs to identify the service provider.

5. Using the binding information, the service requester then invokes the service provider and then retrieves the WSDL Service description for those registered services. Then, the service requester creates

a client proxy application and establishes communication with the service provider using SOAP.

6. Finally, the service requester communicates with the service provider and exchanges data or messages by invoking the available services in the service container.

In the case of an ebXML-based environment, the steps just shown are the same, except ebXML registry and repository, ebXML Messaging, and ebXML CPP/CPA are used instead of UDDI, SOAP, and WSDL, respectively. The basic steps just shown also do not include the implementation of security and quality of service (QoS) tasks. Web Services Security." So far we have explored the Web services architecture and technologies. Let's now move forward to learn how to develop web services-enabled applications as services using the Web services architecture.

WSDL Limitations

There are some limitations to consider when using the WSDL-first approach and svcutil to create Contract files.

Declared Faults

When the WSDL contains declared faults:

- Specify the /UseSerializerForFaults argument during proxy code generation. For example: svcutil /UseSerializerForFaults *.wsdl *.xsd.
- If a port type of an operation includes Fault child node, the operation must use Document style.
- The fault part should refer to element but not type. For example:

Supported

```
<message name="SimpleTypeFault">
  <part name="SimpleTypeFault" element="ns2:StringFaultElement" />
</message>
```

The following is *incorrect* for faults:

Not Supported

```
<message name="SimpleTypeFault">
  <part name="SimpleTypeFault" type="xs:string" />
```

</message>

Removing OperationFormatStyle.Rpc Attribute

The OperationFormatStyle.Rpc attribute is not supported if the operation also has the fault contract attribute.

If the generated proxy code contains an attribute OperationFormatStyle.Rpc, then you must regenerate the WSDL from the code after deleting the attribute.

Identical part Elements

The part elements of messages cannot be same. If the elements are identical, svcutil throws an error. For example, this definition is allowed:

Supported

```
<message name="MultipartInputElement">
 <part name="Fortune" element="ns2:PersonDetailsElementsOne" />
 <part name="Person" element="ns2:PersonDetailsElementsTwo" />
</message>
```

This definition, where the parts refer to same element, is incorrect:

Not Supported

```
<message name="MultipartInputElement">
 <part name="Fortune" element="ns2:PersonNestedElements" />
 <part name="Person" element="ns2:PersonNestedElements" />
</message>
```

Mixed Type Messages

Mixed type messages are not supported. All message parts must refer to either element or type.

For example, the following definition is not permitted:

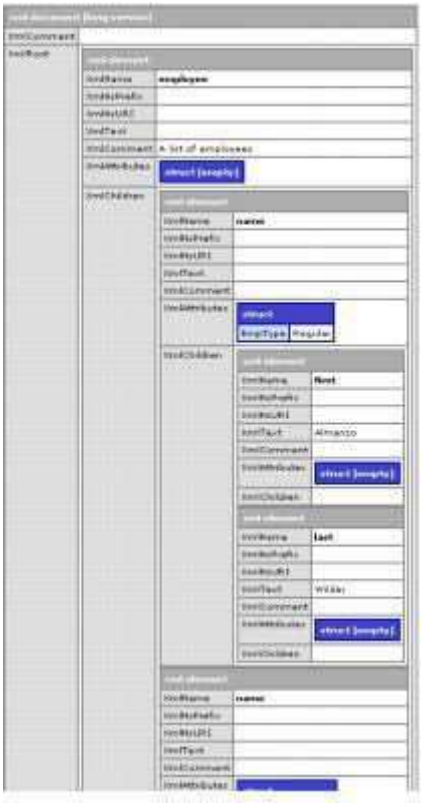
Not Supported

```
<message name="MultipartInputElement">
 <part name="Fortune" element="ns2:PersonDetailsElementsOne" />
 <part name="Person" type="xs:string" />
</message>
```

UNIT-3

XML document structures

An XML document object is a structure that contains a set of nested XML element structures. The following image shows a section of the cfdump tag output for the document object for the XML in A simple XML document. This image shows the long version of the dump, which provides complete details about the document object. Initially, ColdFusion displays a short version, with basic information. Click the dump header to change between short, long, and collapsed versions of the dump.



The following code displays this output. It assumes that y u save the code in a file under your web root, such as C:\Inetpub\ wwwroot\testdocs\employeesimple.xml

```
<cffile action="read" file="C:\Inetpub\wwwroot\testdocs\
employeesimple.xml"
   variable="xmldoc">
<cfset mydoc = XmlParse(xmldoc)>
<cfdump var="#mydoc#">
```

The document object structure

At the top level, the XML document object has the following three entries:

Entry name	Type	Description
XmlRoot	Element	The root element of the document.
XmlComment	String	A string made of the concatenation of all comments on the document, that is, comments in the document prologue and epilog. This string does not include comments inside document elements.
XmlDocType	XmlNode	The DocType attribute of the document. This entry only exists if the document specifies a DocType. This value is read-only; you cannot set it after the document object has

		been created This entry does not appear when the cfdump tag displays an XML element structure.

The element structure

Each XML element has the following entries:

Entry name	Type	Description
XmlName	String	The name of the element; includes the namespace prefix.
XmlNsPrefix	String	The prefix of the namespace.
XmlNsURI	String	The URI of the namespace.
XmlText or XmlCdata	String	A string made of the concatenation f all text and CData text in the element, but n t inside any child elements. When you assign a value to the XmlCdata element, ColdFusion puts the text inside a CDATA information item. then you retrieve information from document object, these element names return identical values.
XmlComment	String	A string made of the concatenation of all comments inside the XML element, but not inside any child elements.
XmlAttributes	Structure	All of this element's attributes, as name-value pairs.
XmlChildren	Array	All this element's children elements.
XmlParent	XmlNode	The parent DOM node of this element. This entry does not appear when the cfdump tag displays an XML element structure.
XmlNodes	Array	An array of all the XmlNode DOM nodes contained in this element. This entry does not appear the cfdump tag when displays an XML element structure.

XML DOM node structure

The following table lists the contents of an XML DOM node structure:

Entry name	Type	Description
XmlName	String	The node name. For nodes such as Element or Attribute, the no e name is the element attribute name.
XmlType	String	The node XML DOM type, such as Element or Text.
XmlValue	String	The node value. This entry is used only for Attribute, CDATA, Comment, and Text type nodes.

Note: *The tag does not display XmlNode structures. If you try to dump an Xm Node structure, the cfdump tag displays "Empty Structure."*

The following table lists the contents of the XmlName and XmlValue fields for each node type that is valid in the XmlType entry. The node types correspond to the object types in the XML DOM hierarchy.

Node type	XmlName	xmlValue
CDATA	#cdata- section	Content of the CDATA section
COMMENT	#comment	Content of the comment
ELEMENT	Tag name	Empty string
ENTITYREF	Name of entity referenced	Empty string
PI (processing instruction)	Target entire content excluding the target	Empty string
TEXT	#text	Content of the text node
EN I Y	Entity name	Empty string
O A ION	Notation name	Empty string
DOCUME	#document	Empty string
FRAGME	#document-fragment	Empty string
DOCTYPE	Document type name	Empty string

Note:*AlthoughXMLattributesarenodes on the DOM tree, ColdFusion does not expose them as XML DOM node data structures. To view an element's attributes, use the element structure's XMLAttributes structure.*

The XML document object and all its elements are exposed as DOM node structures. For example, you can use the following variable names to reference nodes in the DOM tree that you created from the XML example in A simple XML document:

```
mydoc.XmlName mydoc.XmlValue mydoc.XmlRoot.XmlName mydoc.employee.XmlType
mydoc.employee.XmlNodes[1].XmlType
```

XML namespace:-

XML namespaces are used for providing uniquely named elements and attributes in an XML document. They are defined in a W3C recommendation. An XML instance may contain element or attribute names from more than one XML vocabulary. If each vocabulary is given a namespace, the ambiguity between identically named elements or attributes can be resolved.

A simple example would be to consider an XML instance that contained references to a customer and an ordered product. Both the customer element and the product element could have a child element named **id** . References to the **id** element would therefore be ambiguous; placing them in different namespaces would remove the ambiguity.

A namespace name is a uniform resource identifier (URI). Typically, the URI chosen for the namespace of a given XML vocabulary describes a resource under the control of the author or organization defining the vocabulary, such as a URL for the author's Web server. However, the namespace specification does not require nor suggest that the namespace URI be used to retrieve information; it is simply treated by an XML parser as a string. For example, the document at http://www.w3.org/1999/xhtml itself does not contain any code. It simply describes the XHTML namespace to human readers. Using a URI (such as "http://www.w3.org/1999/xhtml") to identify a namespace, rather than a simple string (such as "xhtml"), reduces the probability of different namespaces using duplicate identifiers.

Although the term namespace URI is widespread, the W3C Recommendation refers to it as the namespace name. The specification is not entirely prescriptive about the precise rules for namespace names (it does not explicitly say that parsers must reject documents where the namespace name is not a valid Uniform Resource Identifier), and many XML parsers

The XML document object and all its elements are exposed as DOM node structures. For example, you can use the following variable names to reference nodes in the DOM tree that you created from the XML example in A simple XML document:

```
mydoc.XmlName
mydoc.XmlValue
mydoc.XmlRoot.XmlName
mydoc.employee.XmlType
mydoc.employee.XmlNodes[1].XmlType
```

XML namespace:-

XML namespaces are used for providing uniquely named elements and attributes in an XML document. They are defined in a W3C recommendation. An XML instance may contain element or attribute names from more than one XML vocabulary. If each vocabulary is given a namespace, the ambiguity between identically named elements or attributes can be resolved.

A simple example would be to consider an XML instance that contained references to a customer and an ordered product. Both the customer element and the product element could have a child element named **id** . References to the **id** element would therefore be ambiguous; placing them in different namespaces would remove the ambiguity.

A namespace name is a uniform resource identifier (URI). Typically, the URI chosen for the namespace of a given XML vocabulary describes a resource under the control of the author or organization defining the vocabulary, such as a URL for the author's Web server. However, the namespace specification does not require nor suggest that the namespace URI be used to retrieve information; it is simply treated by an XML parser as a string. For example, the document at http://www.w3.org/1999/xhtml itself does not contain any code. It simply describes the XHTML namespace to human readers. Using a URI (such as "http://www.w3.org/1999/xhtml") to identify a namespace, rather than a simple string (such as "xhtml"), reduces the probability of different namespaces using duplicate identifiers.

Although the term namespace URI is widespread, the W3C Recommendation refers to it as the namespace name. The specification is not entirely prescriptive about the precise rules for namespace names (it does not explicitly say that parsers must reject documents where the namespace name is not a valid Uniform Resource Identifier), and many XML parsers

allow any character string to be used. In version 1.1 of the recommendation, the namespace name becomes an Internationalized Resource Identifier, which licenses the use of non-ASCII characters that in practice were already accepted by nearly all XML software. The term namespace URI persists, however, not only in popular usage, but also in many other specifications from W3C and elsewhere World.

Following publication of the Namespaces recommendation, there was an intensive elaborate about how a relative URI should be handled, with some intensely arguing that it should simply be treated as a character string, and others arguing with conviction that it should be

turned into an absolute URI by resolving it against the base URI of the document The result of the debate was a ruling from W3C that relative URIs we e deprecated

The use of URIs taking the form of URLs in the http scheme (such as http://www.w3.org/1999/xhtml) is common, despite the absence of any formal relationship with the HTTP protocol. The Namespaces specification does not say what should happen if such a URL is dereferenced (that is, if software attempts to retrieve a document from this location). One convention adapted by s me users is to place an RDDL document at the location. In general, however, users should assume that the namespace URI is simply a name, not the address of a document n the Web.

The Emergence of SOAP

SOAP initially was developed by Develop Mentor, Inc., as a platform independent protocol for accessing services, objects between applications, and servers using HTTP-based communication. SOAP used an XML - based vocabulary for representing RPC calls and its parameters and return values. In 1999, the SOAP 1.0 specification was made publicly available as a joint effort supported by vendors like ogue Wave, IONA, Object Space, Digital Creations, UserLand, Microsoft, and DevelopMentor. Later, the SOAP 1.1 specification was released as a W3C Note, with additional contributions from IBM and the Lotus Corporation supporting a wide range of systems and communication models like RPC and messaging.

Nowadays, the current version of SOAP 1.2 is part of the W3C XML Protocol Working Group effort led by vendors such as Sun Microsystems, IBM, HP, BEA, Microsoft, and Oracle. At the time of this book's writing, SOAP 1.2 is available as a public W3C working draft. To find out the current status of the SOAP specifications produced by the XML Protocol Working Group, refer to the W3C Web site at www.w3c.org.

Understanding SOAP Specifications

The SOAP 1.1 specifications define the following:

■■ Syntax and semantics for representing XML documents as structured SOAP messages

■■ Encoding standards for representing data in SOAP messages

■■ A communication model for exchanging SOAP messages

■■ Bindings for the underlying transport protocols such as SOAP transport ■■

Conventions for sending and receiving messages using RPC and messaging

Note that SOAP is not a programming language a business application component for building business applications. SOAP is intended for use as a portable communication protocol to deliver SOAP messages, which have to be created and processed by an application. In general, SOAP is simple and extensible by design, but unlike other distributed computing protocols, the following features are n t supported by SOAP:

■ Garbage collection

■ Object by reference

■ Object activation

■ Message batching

SOAP and ebXML are complementary to each other. In fact, SOAP is leveraged by an ebXML Messaging service as a communication protocol with an extension that provides added security and reliability for handling business transactions in e-business and B2B frameworks. More importantly, SOAP adopts XML syntax and standards like XML Schema and namespaces as part of its message structure. To understand the concepts of XML notations, XML Schema, and namespaces, refer to Chapter 8, “XML Processing and Data Binding with Java APIs.” Now, let’s take a closer look at the SOAP messages, standards, conventions, and other related technologies, and how they are represented in a development process.

Structure of SOAP messages:-

Usually a SOAP message requires defining two basic namespaces: SOAP

Envelope and SOAP Encoding. The following list their forms in both versions 1.1 and 1.2 of SOAP.

SOAP ENVELOPE

- http://schemas.xmlsoap.org/soap/envelope/ (SOAP 1.1)
- http://www.w3.org/2001/06/soap-envelope (SOAP 1.2)

SOAP ENCODING

- http://schemas.xmlsoap.org/soap/encoding/ (SOAP 1.1)
- http://www.w3.org/2001/06/soap-encoding (SOAP 1.2)

Additionally, SOAP also can use attributes and values defined in W3C XML Schema instances or XML Schemas and can use the elements based on custom XML conforming to W3C XML Schema specifications. SOAP does not support or use DTD-based element or attribute declarations. To

understand the fundamentals of XML namespaces, refer to Chapter 8, "XML Processing and Data Binding with Java APIs." Typical to the previous example message, the structural format of a

SOAP message (as per SOAP version 1.1 with attachments) contains the following elements:

- Envelope
- Header (optional)
- Body
- Attachments (optional)

Figure 4.1 represents the structure of a SOAP message with attachments.Typically, a SOAP message is represented by a SOAP envelope with zero or more attachments. The SOAP message envelope contains the header and body of the message, and the SOAP message attachments enable the message to contain data, which include XML and non-XML data

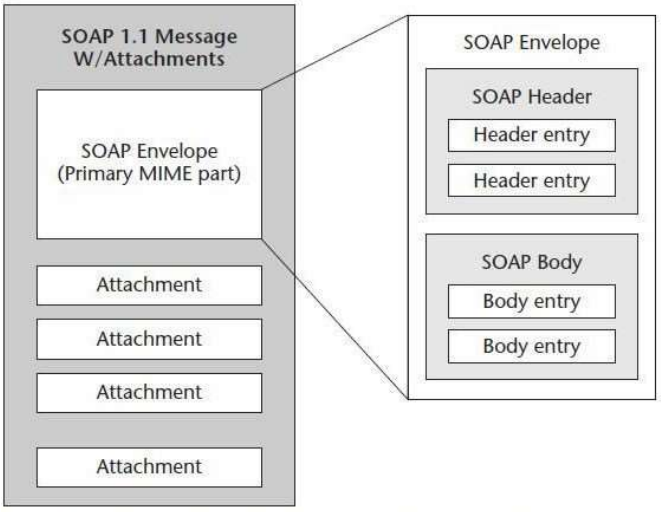
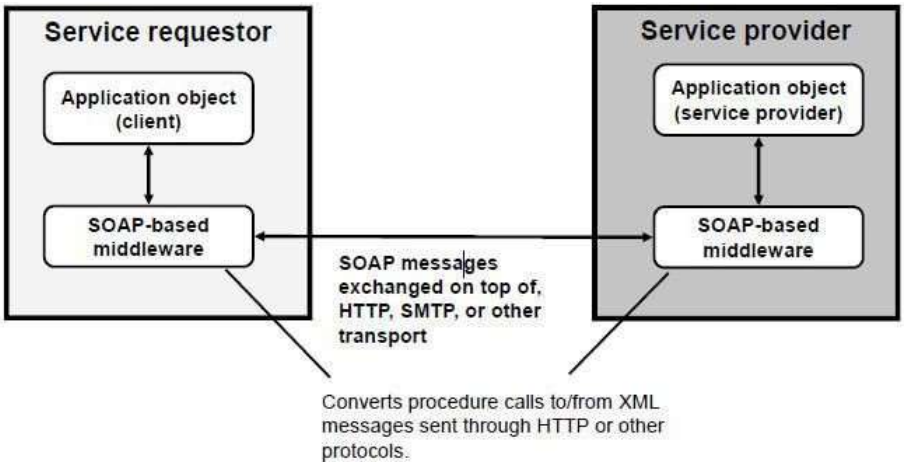


Figure 4.1 Structure of a SOAP message with attachments.

(like text/binary files). In fact, a SOAP message package is constructed

using the MIME Multipart/ Related structure approaches to separate and identify the different parts of the message. Now, let's explore the details and characteristics of the parts of a SOAP message.



What is SOAP:-

SOAP is the standard messaging protocol used by Web services. SOAP's primary application is inter application communication. SOAP codifies the use of XML as an encoding scheme for request and response parameters using HTTP as a means for transport.

SOAP covers the following four main areas:

– A message format for one-way communication describing how a message can be packed into an XML document.

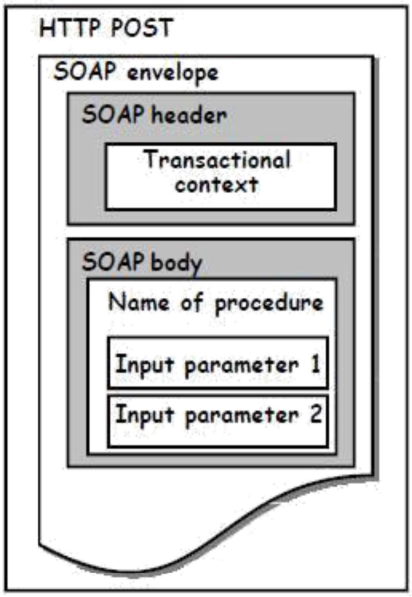
– A description of how a SOAP message should be transported using HTTP (for Web- based interaction) or SMTP (for e-mail-based interaction).

– A set of rules that must be followed when processing a SOAP message and a simple classification of the entities involved in processing a SOAP message.

– A set of conventions on how to turn an RPC call into a SOAP message and back.

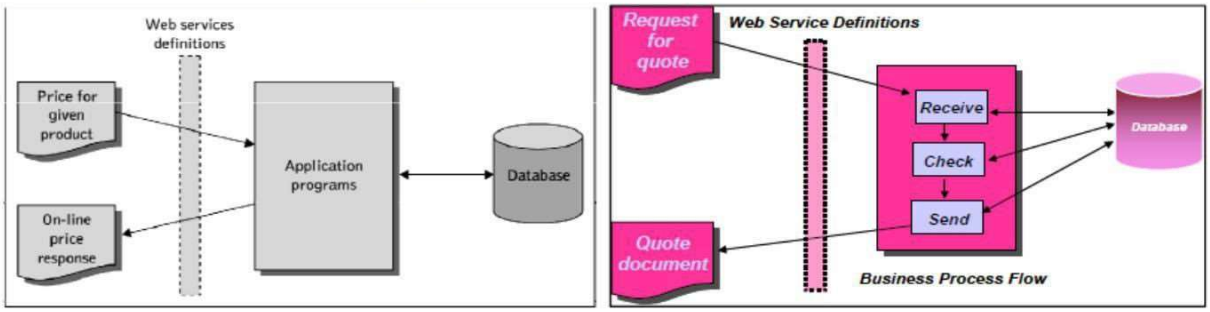
SOAP and HTTP

- A binding of SOAP to a transport protocol is a description of how a SOAP message is to be sent using that transport protocol.
- The typical binding for SOAP is HTTP.
- SOAP can use GET or POST. With GET, the request is not a SOAP message but the response is a SOAP message, with POST both request and response are SOAP messages (in version 1.2, version 1.1 mainly considers the use of POST).
- SOAP uses the same error and status codes as those used in HTTP so that HTTP responses can be directly interpreted by a SOAP module.



The SOAP Communication Model

- SOAP communication model is defined by its encoding style and its communication style
- Encoding style are about how applications on different platforms share and exchange data, although they may have different data types
- SOAP encoding rules are identified by the URI "http://www.w3.org/2003/05/soap-encoding"
- SOAP supports two possible communication styles:
  - remote procedure call (RPC) and
  - document (or message).



RPC-style (synchronous) interaction

Document-style (asynchronous) interaction

The SOAP Body

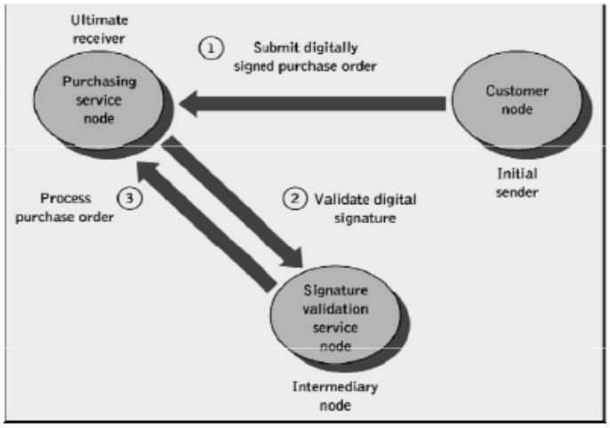
- The SOAP body is the area of the SOAP message, where the application specific XML data (payload) being exchanged in the message is placed.

- The **<Body> element** must be present and is an immediate child of the envelope. It may contain a number of child elements, called body entries, but it may also be empty. The **<Body> element** contains either of the following:
  - **Application-specific data** is the information that is exchanged with a Web service. The SOAP <Body> is where the method call information and its related arguments are encoded. It is where the response to a method call is placed, and where error information can be stored.
  - A **fault message** is used only when an error occurs.

- A SOAP message may carry either application-specific data or a fault, but not both.

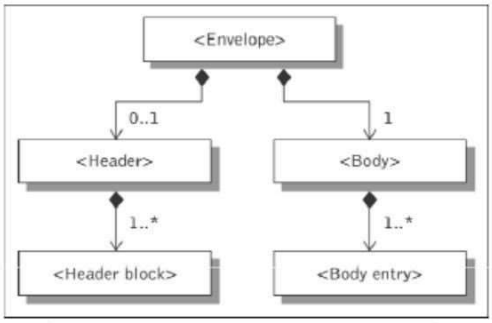
SOAP Intermediaries

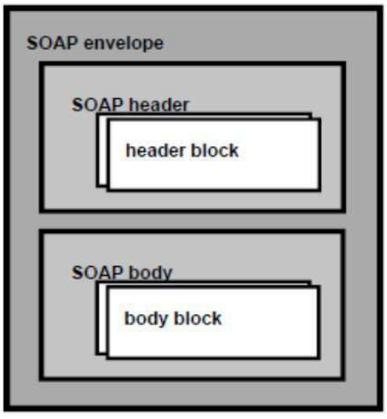
- SOAP headers have been designed in anticipation of participation of other SOAP processing nodes – called **SOAP *intermediaries*** – along a **message's path** from an initial SOAP sender to an ultimate SOAP receiver.
- A SOAP message travels along the message path from a sender to a receiver.
- All SOAP messages start with an initial sender, which creates the SOAP message, and end with an ultimate receiver.



SOAP messages

- SOAP is based on **message exchanges.**
- Messages are seen as **envelopes** where the application encloses the data to be sent.
- A SOAP message consists of a SOAP of an <Envelope> element containing an optional <Header> and a mandatory <Body> element.
- The contents of these elements are application defined and not a part of the SOAP specifications.
- A SOAP <Header> contains blocks of information relevant to how the message is to be processed. This helps pass information in SOAP messages that is not application payload.
- The SOAP <Body> is where the main end-to-end information conveyed in a SOAP message must be carried.





SOAP Envelope

The SOAP envelope is the primary container of a SOAP message's structure and is the mandatory element of a SOAP message. It is represented as the root element of the message as Envelope. As we discussed earlier, it is usually declared as an element using the XML namespace ttp://schemas .xmlsoap.org/soap/envelope/. As per SOAP 1.1 specifications, SOAP messages that do not follow this namespace declaration are not processed and are considered to be invalid. Encoding styles also can be defined using a namespace under Envelope to represent the data types used in the message. Listing 4.3 shows the SOAP envelope element in a SOAP message.

```
<SOAP-ENV:Envelope
xmlns:SOAP-ENV=http://schemas.xmlsoap.org/soap/envelope/
xmlns:xsi="http://www.w3c.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
SOAP-ENV:
encodingStyle="http://schemas.xmlsoap.org/soap/enc ding/"/>
<!--SOAP Header elements - -/>

<!--SOAP Body element - -/>
</SOAP-ENV:Envelope>
```

SOAP Header

The SOAP header is represented as the first immediate child element of a SOAP envelope, and it has to be namespace qualified. In addition, it also may contain zero or more optional child elements, which are referred to as SOAP header entries. The SOAP encoding Style attribute will be used to

define the encoding of the data types used in header element entries. The SOAP actor attribute and SOAP must Understand attribute can be used to indicate the target SOAP application node sender/Receiver/Intermediary) and to process the Header entries. Listing 4.4 shows the sample representation of a SOAP header element in a SOAP message.

```
<SOAP-E V:Header>
<wiley:Transaction
xmlns:wiley="http://jws.wiley.com/2002/booktx"
SOAP-E V:mustUnderstand="1">
<keyValue> 5 </keyValue>
</wiley:Transaction>
</SOAP-ENV:Header>
```

SOAP Body

A SOAP envelope contains a SOAP body as its child element, and it may contain one or more optional SOAP body block entries. The Body represents the mandatory processing information or the payload intended for the receiver of the message. The SOAP 1.1 specification mandates that there must be one or more optional SOAP Body entries in a message. A Body block of a SOAP message can contain any of the following:

- RPC method and its parameters
- Target application (receiver) specific data
- SOAP fault for reporting errors and status information

Listing 4.5 illustrates a SOAP body representing an RPC call for getting the book price information from www.wiley.com for the book name *Developing Java Web Services*.

```
<SOAP-ENV:Body>
<m:GetBookPrice
xmlns:m="http://www.wiley.com/jws.book.priceList/">
<bookname xsi:type='xsd:string'>
Developing Java Web services</bookname>
</m:GetBookPrice>
</SOAP-ENV:Body>
```

SOAP Encoding

SOAP 1.1 specifications stated that SOAP- based applications can represent their data either as literals or as encoded values defined by the "XML Schema, Part -2" specification (see ww.w3.org/TR/xmlschema-2/). Literals refer to message contents that are encoded according to the W3C XML Schema. Encoded values refer to the messages encoded based on SOAP encoding styles specified in SOAP Section 5 of the SOAP 1.1 specification. The namespace identifiers for these SOAP encoding styles are defined in http://schemas.xmlsoap.org/soap/encoding/(SOAP1.1)and http://www.w3.org/2001/06/soap-encoding (SOAP 1.2). The SOAP encoding defines a set of rules for expressing its data types. It is a generalized set of data types that are represented by the programming languages, databases, and semi-structured data required for an application. SOAP encoding also defines serialization rules for its data model using an encoding Style attribute under the SOAP-ENV namespace that specifies the serialization rules for a specific element or a group of elements. SOAP encoding supports both simple- and compound-type values.

SOAP Messaging

SOAP Messaging represents a loosely coupled communication model based on message notification and the exchange of XML documents. The SOAP message body is represented by XML documents or literals encoded according to a specific W3C XML schema, and it is produced and consumed by sending or receiving SOAP node(s). The SOAP sender node sends a message with an XML document as its body message and the SOAP receiver node

processes it.4.26 represents a SOAP message and a SOAP messaging-based communication. The message contains a header block Inventory Notice and the body product, both of which are application-defined and not defined by SOAP. The header contains information required by the receiver node and the body contains the actual message to be delivered.

Advantages and disadvantages of SOAP

- Advantages of SOAP are:
  - Simplicity: Based on highly-structured format of XML
  - Portability: No dependencies on underlying platform
  - Firewall friendliness: By posting data over HTTP
  - Use of open standards: text-based XML standard
  - Interoperability: Built on open technologies (XML and HTTP)
  - Universal acceptance. Most widely accepted message communication standard
- Disadvantages of SOAP are:
  - Too much reliance on HTTP: limited only to request/response model and HTTP's slow protocol causes bad performance
  - Statelessness: difficult for transactional and business processing applications
  - Serialization by value and not by reference: impossible to refer or point to external data source

UNIT-4

Universal Description, Discovery and Integration (UDDI)

is a platform-independent, extensible world markup language(XML)-based registry by which businesses worldwide can list themselves on the Internet, and a mechanism to register and locate web service applications. UDDI is an open industry initiative, sponsored by the Organization for the Advancement of Structured Information Standards (OASIS), for enabling businesses to publish service listings and discover each other, and to define how the services or software applications interact over the Internet.

UDDI was originally proposed as a core Web service standard .[1] It is designed to be interrogated by SOAP messages and to provide access to Web Services Description Language (WSDL) documents describing the protocol bindings and message formats required to interact with the web services listed in its directory.

A UDDI business registration consists of three components:

- White Pages — address, contact, and known identifiers;
- Yellow Pages — industrial categorizations based on standard taxonomies;
- Green Pages — technical information about services exposed by the business.

White Pages

White pages give information about the business supplying the service. This includes the name of the business and a description of the business - potentially in multiple languages. Using this information, it is possible to find a service about which some information is already known (for example, locating a service based on the provider's name).[6]

Contact information for the business is also provided - for example the businesses address and phone number; and other information such as the Dun & Bradstreet Universal umbering System number.

Yellow Pages

Yellow pages provide a classification of the service or business, based on standard taxonomies. These include the Standard Industrial Classification (SIC), the North American Industry Classification System (NAICS),[6] or the United Nations Standard Products and Services Code (UNSPSC) and geographic taxonomies.

Because a single business may provide a number of services, there may be several Yellow Pages (each describing a service) associated with one White Page (giving general information about the business).

Green Pages

Green pages are used to describe how to access a Web Service, with information on the service bindings. Some of the information is related to the Web Service - such as the address of the service and the parameters, and references to specifications of interfaces.[6] Other information is not related directly to the Web Service - this includes e-mail, FTP,CORBA and telephone details for the service. Because a Web Service may have multiple bindings (as defined in its WSDL description), a service may have multiple Green Pages, as each binding will need to be accessed diferently.

UDDI Nodes & Registry

UDDI nodes are servers which support the UDDI specification and belong to a UDDI registry while UDDI registries are collections of one or more nodes.

SOAP is an XML-based protocol to exchange messages between a requester and a provider of a Web Service. The provider publishes the WSDL to UDDI and the requester can join to it using SOAP.

UDDI Technical Architecture: -

The UDDI technical architecture consists of three parts:

UDDI data model:

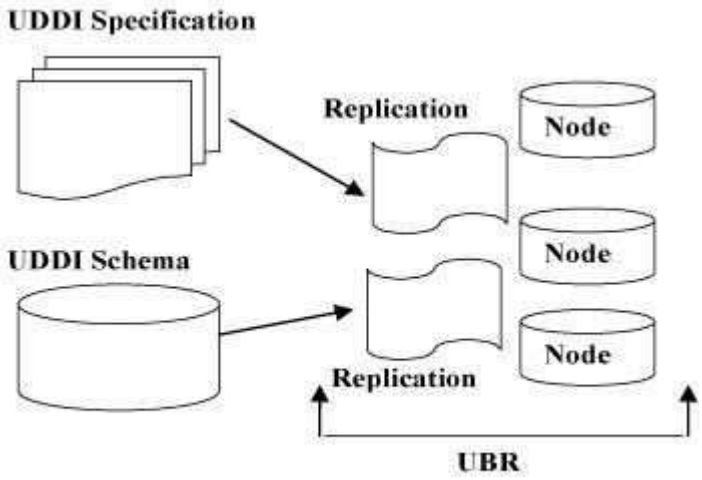
An XML Schema for describing businesses and web services. The data model is described in detail in the "UDDI Data Model" section.

UDDI API Specification:

A Specification of API for searching and publishing UDDI data.

UDDI cloud services:

This is operator sites that provide implementations of the UDDI specification and synchronize all data on a scheduled basis.



UDDI Technical Architecture

The UDDI Business Registry (UBR), also known as the Public Cloud, is a conceptually single system built from multiple nodes that has their data synchronized through replication.

The current cloud services provide a logically centralized, but physically distributed, directory. This means that data submitted to one root node will automatically be replicated across all the other root nodes. Currently, data replication occurs every 24 hours.

UDDI cloud services are currently provided by Microsoft and IBM. Ariba had originally planned to offer an operator as well, but has since backed away from the commitment. Additional operators from other companies, including Hewlett-Packard, are planned for the near future. It is also possible to set up private UDDI registries. For example, a large company may set up its own private UDDI registry for registering all internal web services. As these registries are not automatically synchronized with the root UDDI nodes, they are not considered part of the UDDI cloud.

UDDI Data Model

UDDI includes an XML Schema that describes four five data structures:

- businessEntity
- businessService
- bindingTemplate
- tModel
- publisherAssertion

- **businessEntity data structure:**

The business entity structure represents the provider of web services. Within the UDDI registry, this structure contains information about the company itself, including contact information, industry categories, business identifiers, and a list of services provided.
Here is an example of a fictitious business's UDDI registry entry:

businessService data structure:

The business service structure represents an individual web service provided by the business entity. Its description includes information on how to bind to the web service, what type of web service it is, and what taxonomical categories it belongs to:

Here is an example of a business service structure for the Hello World web service.

Notice the use of the Universally Unique Identifiers (UUIDs) in the *businessKey* and *serviceKey*attributes. Every business entity and business service is uniquely identified in all UDDI registries through the UUID assigned by the registry when the information is first entered.

bindingTemplate data structure:

Binding templates are the technical descriptions of the web services represented by the business service structure. A single business service may have multiple binding templates. The binding template represents the actual implementation of the web service.

Here is an example of a binding template for Hello World.

Because a business service may have multiple binding templates, the service may specify different implementations of the same service, each bound to a different set of protocols or a different network address.

tModel data structure:

The tModel is the last core data type, but potentially the most difficult to grasp. tModel stands for technical model.

A tModel is a way of describing the various business, service, and template structures stored within the DDI registry. Any abstract concept can be registered within UDDI as a tModel. For instance, if you define a new WSDL port type, you can define a tModel that represents that port type within UDDI. Then, you can specify that a given business service implements that port type by associating the tModel with one of that business service's binding templates.

Here is an example of A tModel representing the HelloWorldInterface port type

publisherAssertion data structure:

This is a relationship structure putting into association two or more businessEntity structures according to a specific type of relationship, such as subsidiary or department.

The publisherAssertion structure consists of the three elements fromKey (the first businessKey), toKey (the second businessKey) and keyedReference.

The keyedReference designates the asserted relationship type in terms of a keyName keyValue pair within a tModel, uniquely referenced by a tModelKey.

Service registries

- To discover Web services, a service registry is needed. This requires describing and registering the Web service
  - Compare/Contrast with URLs
- Publication of a service requires proper description of a Web service in terms of business, service, and technical information.
- Registration deals with persistently storing the Web service descriptions in the Web services registry.
- Two types of registries can be used:
  - The document-based registry: enables its clients to publish information, by storing XML-based service documents such as business profiles or technical specifications (including WSDL descriptions of the service).
  - The metadata-based service registry: captures the essence of the submitted document.

Service discovery

- Service discovery is the process of locating Web service providers, and retrieving Web services descriptions that have been previously published.
- Interrogating services involve querying the service registry for Web services matching the needs of a service requestor.
  - A query consists of search criteria such as:
    - the type of the desired service, preferred price and maximum number of returned results, and is executed against service information published by service provider.
  - Discovering Web services is a process that is also dependent on the architecture of the service registry.
- After the discovery process is complete, the service developer or client application should know the exact location of a Web service (URI), its capabilities, and how to interface with it.

Types of service discovery

Static

- The service implementation details are bound at design time and a service retrieval is performed on a service registry.
- The results of the retrieval operation are examined usually by a human designer and the service description returned by the retrieval operation is incorporated into the application logic.

Dynamic

- The service implementation details are left unbound at design time so that they can be determined at run-time.
- The Web service requestor has to specify preferences to enable the application to **infer/reason** which Web service(s) the requester is most likely to want to invoke.
- Based on application logic quality of service considerations such as best price, performance, security certificates, and so on, the application chooses the most appropriate service, binds to it, and invokes it.

What is UDDI?

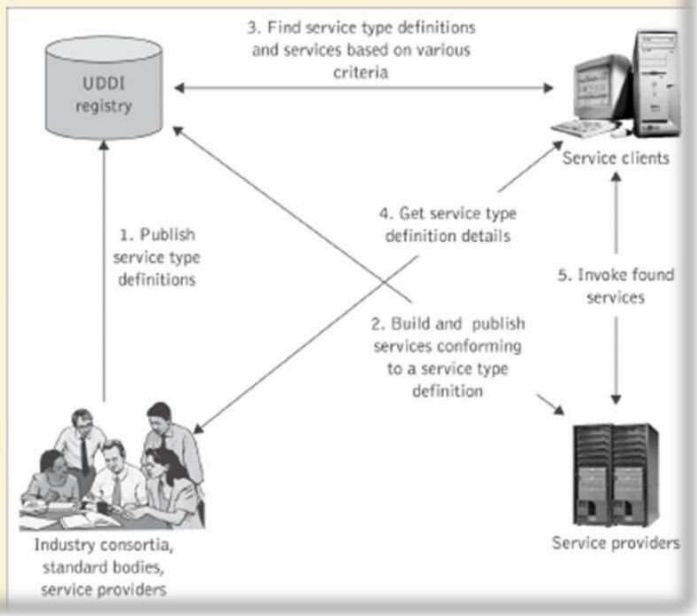
- The universal description, discovery, and integration is a registry standard for Web service description and discovery together with a registry facility that supports the WS publishing and discovery processes.
- UDDI enables a business to:
  - **describe** its business and its services;
  - **discover** other businesses that offer desired services;
  - **integrate (interoperate)** with these other businesses.
- Conceptually, a UDDI business registration consists of three inter-related components:
  - "white pages" (address, contact, and other key points of contact);
  - "yellow pages" classification info. based on standard industry taxonomies; and
  - "green pages", the technical capabilities and information about services.

UDDI – main characteristics

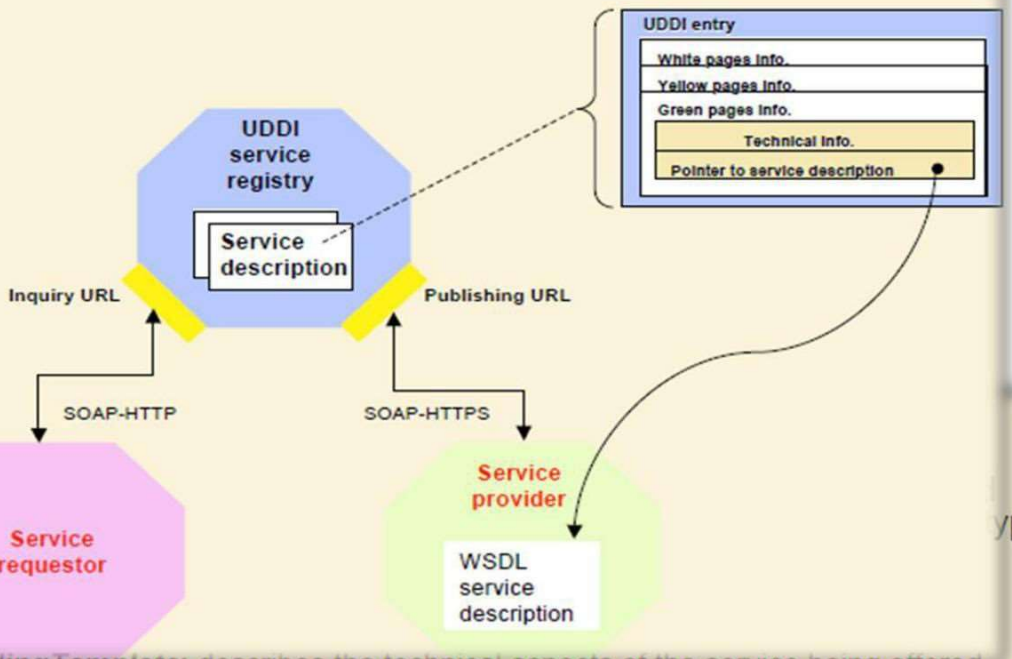
- UDDI provides a mechanism to categorize businesses and services using **taxonomies**.
  - Service providers can use a taxonomy to indicate that a service implements a specific domain standard, or that it provides services to a specific geographic area
  - UDDI uses standard taxonomies so that information can be discovered on the basis of categorization.
- **UDDI business registration: an XML document** used to describe a business entity and its Web services.
- UDDI is not bound to any technology. In other words,
  - An entry in the UDDI registry can contain any type of resource, independently of whether the resource is XML based or not, e.g., the UDDI registry could contain information about an enterprise's electronic document interchange (**EDI**) system or even a service that, uses the **fax machine** as its primary communication channel.
  - While UDDI itself uses XML to represent the data it stores, it allows for other kinds of technology to be registered.

The UDDI usage model

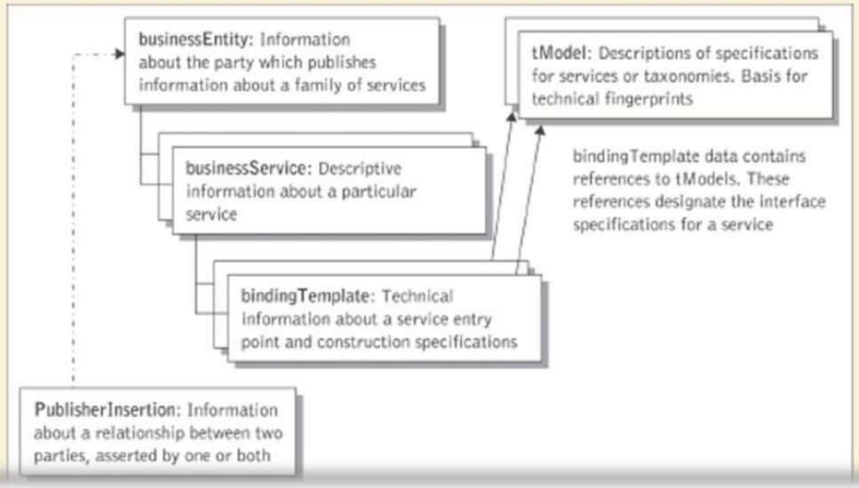
- An enterprise may set up multiple *private* UDDI registries in-house to support intranet and e-Business operations.
- Public UDDI registries can be set up by customers and business partners of an enterprise.
  - Services must be published in a public UDDI registry so that potential clients and service developers can discover them.



UDDI and WSDL



- ***bindingTemplate***: describes the technical aspects of the service being offered.
- ***tModel***: ("technical model")is a generic element that can be used to store technical information on how to use the service, conditions for use, guarantees, etc.



UDDI—A Global Registry of Web Services

UDDI is a public registry designed to house information about businesses and their services in a structured way. Through UDDI, one can publish and discover information about a business and its Web Services. This data can be classified using standard taxonomies so that information can be found based on categorization. Most importantly, UDDI contains information about the technical interfaces of a business's services. Through a set of SOAP-based XML API calls, one can interact with UDDI at both design time and run time to discover technical data, such that those services can be invoked and used. In this way, UDDI serves as infrastructure for a software landscape based on Web Services.

Why UDDI? What is the need for such a registry? As we look towards a software landscape of thousands—perhaps millions—of Web Services, s me t ugh challenges emerge:

- How are Web Services discovered?
- How is this information categorized in a meaningful way?
- What implications are there for localization?
- What implications are there around proprietary technologies? How can I guarantee interoperability in the discovery mechanism?
- How can I interact with such a discovery mechanism at run time once my application is dependent upon a web Service?

In response to these challenges, the UDDI initiative emerged. A number of companies, including Microsoft, IBM, Sun, Oracle, Compaq, Hewlett Packard, Intel, SAP, and over three hundred other companies (see DDI: Community for a complete list), came together to develop a specification based on open standards and non-proprietary technologies to solve these challenges. he result, initially launched in beta December 2000 and in production by May 2001, was a global business registry hosted by multiple operator nodes that users could—at no cost—both search and publish to.

With such an infrastructure for Web Services in place, data about Web Services can now be found consistently and reliably in a universal, completely vendor-neutral capacity. Precise categorical searches can be performed using extensible taxonomy systems and identification. Run-time UDDI integration can be incorporated into applications. As a result, a Web Services software environment can flourish.

WSDL and UDDI

WSDL has emerged as an important piece of the Web Services protocol stack. As such, it is important to grasp how UDDI and WSDL work together and how the notion of interfaces vs. implementations is part of each protocol. Both WSDL and UDDI were designed to clearly delineate between abstract meta-data and concrete implementations, and understanding the implications of the division is essential to understanding WSDL and UDDI.

For example, WSDL makes a clear distinction between messages and ports: Messages, the required syntax and semantics of a Web Service, are always abstract, while ports, the network address where the Web Service can be invoked, are always concrete. One is not required to provide port information in a WSDL file. A WSDL can contain solely abstract interface information and not provide any concrete implementation data. Such a WSDL file is considered valid. In this way, WSDL files are decoupled from implementations.

One of the most exciting implications of this is that there can be multiple implementations of a single WSDL interface. This design allows disparate systems to write implementations of the same interface, thus guaranteeing that the systems can talk to one another. If three different companies have implemented the same WSDL file, and a piece of client software has created the proxy/stub code from that WSDL interface, then the client software can communicate with all three of those implementations with the same code base by simply changing the access point.

UDDI draws a similar distinction between abstraction and implementation with its concept of tModels. The tModel structure, short for "Technology Model", represents technical fingerprints, interfaces and abstract types of meta-data. Corollary with tModels are binding templates, which are the concrete implementation of one or more tModels. Inside a binding template, one registers the access point for a particular implementation of a tModel. Just as the schema for WSDL allows one to decouple interface and implementation, UDDI provides a similar mechanism, because tModels can be published separately from binding templates that reference them. For example, a standards body or industry group might publish the canonical interface for a particular industry, and then multiple businesses could write implementations to this interface. Accordingly, each of those businesses' implementations would refer to that same t

Model. WSDL files are perfect examples of a UDDI tModel.

Registering with UDDI

Publishing to UDDI is a relatively straightforward process. The first step is to determine some basic information about how to model your ompany and its services in UDDI. Once that is determined, the next step is to actually perform the registration, which can be done either through a Web-based user interface or programmatically. The final step is to test your entry to insure that it was correctly registered and appears as expected in different types of searches and tools.

Step 1: Modeling Your UDDI Entry

Considering the data model outlined above, several key pieces of data need to be collected before establishing a UDDI entry.

1. Determine the tModels (WSDL files) that your Web Service implementations use.

   Simliar to developing a COM component, your Web Service has been developed either based on an existing interface, or using an interface you designed yourself. In the case of a Web Service based on an existing WSDL, you will need to determine if that WSDL file has been registered in UDDI. If it has, you will need to note its name and tModelKey, which is the GUID generated by UDDI when that WSDL file was registered.

2. Determine the categories appropriate to your services.

   Just as a company can be categorized, Web Services can also be categorized. As such, a company might be categorized at the business level as **NAICS: Software Publisher (51121)**, but its hotel booking Web Se vice might be catego ized at the service level as **NAICS: Hotels and Motels (72111)**.

Step Two: Registering Your UDDI Entry

Upon completion of the modeling exercise, the next step is to register your company. You will need to obtain an account with a UDDI registry, which cannot be done programmatically, as a Terms of Use statement must be agreed to. The Microsoft Node uses Passport for its authentication, so you will need to have acquired a Passport (http://www.passport.com/Consumer/default.asp) in order to proceed.

There are two options at this point: You can either use the Web user interface provided by the Microsoft node, or you can register programmatically by issuing the SOAP API calls to the node itself. If you don't expect to be making many changes to your entry, or if your entry is relatively simple, using the Web user interface is sufficient. However, if you expect to be making frequent updates, or your entry is more complex, scripting the registration process using the Microsoft DDI SDK makes sense. Also, the Microsoft User Interface is not localized for other languages, so if you want to take advantage of that feature of the UDDI API, you will need to register programmatically.

Step Three: Searching UDDI For Your Entry

Three checks are worth performing once your entry is registered in UDDI. First, using the Microsoft Web User Interface, search for your business based on its name and categorizations to see it returned in the result sets. Second, open Visual Studio .NET and ensure that it appears through the "Add Web

Reference" dialog. If it does not appear, it is likely that your tModel was not categorized correctly using the uddi-org:types taxonomy explained above. You should be able to add the Web Service to your project and generate the proxy code based on the WSDL file. Lastly, after 24 hours, your entry will have replicated to the IBM node, which can be searched from their UI at https://www-3.ibm.com/services/uddi/protect/find.

Web Services Notification (WSN):-

The Web Services Notification (WSN) defines a set of specifications that standardize the way Web Services interact using the notification pattern. In the notification pattern, a Web Service disseminates information to a set of other Web Services, without having to have prior knowledge of these other Web Services. Characteristics of this pattern include:

- The Web Services that wish to consume information are registered with the Web Service that is capable of distributing it. As part of this registration process they may provide some indication of the nature of the information that they wish to receive.
- The distributing Web Service disseminates information by sending one-way messages to the Web Services that are registered to receive it. It is possible that more than one Web Service is registered to consume the same information. In such cases, each Web Service that is registered receives a separate copy of the information.
- The distributing Web Service may send any number of messages to each registered Web Service; it is not limited to sending just a single message.

UNIT -5

A DESCRIPTION OF WEB SERVICES :-

Each Web service has a machine processable description written in Web Services Description Language (WSDL), which is "*an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information*"8. This WSDL file can be sent directly to perspective users, or published in the UDDI registries. Upon a successful inquiry to a UDDI registry, the WSDL link about the target Web service will be returned to the requested , describing core information about the contents and providing information on how to communicate (or bind) with the target Web service.

SOA Architecture

Service-oriented architecture (SOA) allows different ways to develop applications by combining services. The main premise of SOA is to erase application boundaries and technlogy differences. As applications are opened up, how we can combine these services securely becomes an issue. Traditionally, security models have been hardcoded into applications and when capabilities of an application are opened up for use by other applications, the security models built into each application may not be good enough.

Several emerging technologies and standards address different aspects of the problem of security in SOA. Standards such as WS-Security, SAML, S-Trust, S-SecureConversation and WS-SecurityPolicy focus on the security and identity management aspects of SOA implementations that use Web services. Technologies such as virtual organization in grid computing, application-oriented networking (AON) and XML gateways are addressing the problem of SOA security in the larger context.

XML gateways are hardware or software based solutions for enforcing identity and security for SOAP, XML, and REST based web services, usually at the network perimeter. An XML gateway is a dedicated application which allows for a more centralized approach to security and identity enforcement, similar to how a protocol firewall is deployed at the perimeter of a network for centralized access control at the connection and port level.

XML Gateway SOA Security features include PKI, Digital Signature, encryption, XML Schema validation, antivirus, and pattern recognition. Regulatory certification for XML gateway security features are provided by FIPS and United States Department of Defense.

Web Services Security (WS-Security, WSS):-

It is an extension to SOAP to apply security to Web services. It is a member of the Web service specifications and was published byOASIS.

The protocol specifies how integrity and confidentiality can be enforced on messages and allows the communication of various security token formats, such as Security Assertion Markup Language (SAML), Kerberos, and X.509. Its main focus is the use of XML Signature and XML Encryption to provide end-to-end security.

USE Case

End-to-end Security

If a SOAP intermediary is required, and the intermediary is not or is less trusted, messages need to be signed and optionally encrypted. This might be the case of an application-level proxy at a network perimeter that will terminate TCP connections.

Non-repudiation

The standard method for non-repudiation is to write transactions to an audit trail that is subject to specific security safeguards. However, if the audit trail is not sufficient, digital signatures may provide a better method to enforce non-repudiation. WS-Security can provide this.

Alternative transport bindings

Although almost all SOAP services implement HTTP bindings, in theory other bindings such as JMS or SMTP could be used; in this case end-to-end security would be required.

Reverse proxy/common security token

Even if the web service relies upon transport layer security, it might be required for the service to know about the end user, if the service is relayed by a (HTTP-) reverse proxy. A WSS header could be used to convey the end user's token, vouched for by the reverse proxy.

Security Topologies:-

One of the most essential portions of information security is the design and topology of secure networks. What exactly do we mean by “topology?” Usually, a geographic diagram of a network comes to mind. However, in networking, topologies are not related to the physical arrangement of equipment, but rather, to the logical connections that act between the different gateways, routers, and servers. We will take a closer look at some common security topologies.

With network security becoming such a hot topic, you may have come under the microscope about your firewall and network security configuration. You may have even been assigned to implement or reassess a firewall design. In either case, you need to be familiar with the most common firewall configurations and how they can increase security. In this article, I will introduce you to some common firewall configurations and some best practices for designing a secure network topology.

Setting up a firewall security strategy

At its most basic level, a firewall is some sort of hardware or software that filters traffic between your company's network and the Internet. With the large number of hackers roaming the Internet today and the ease of downloading hacking tools, every network should have a security policy that includes a firewall design.

If your manager is pressuring you to make sure that you have a strong firewall in place and to generally beef up network security, what is your next move? Your strategy should be two fold:

- Examine your network and take account of existing security mechanisms (routers with access lists, intrusion detection, etc.) as part of a firewall and security plan.
- Make sure that you have a dedicated firewall solution by purchasing new equipment and/or software or upgrading your current systems.

Keep in mind that a good firewall topology involves more than simply filtering network traffic. It should include:

- A solid security policy.
- Traffic checkpoints.
- Activity logging.
- Limiting exposure to your internal network.

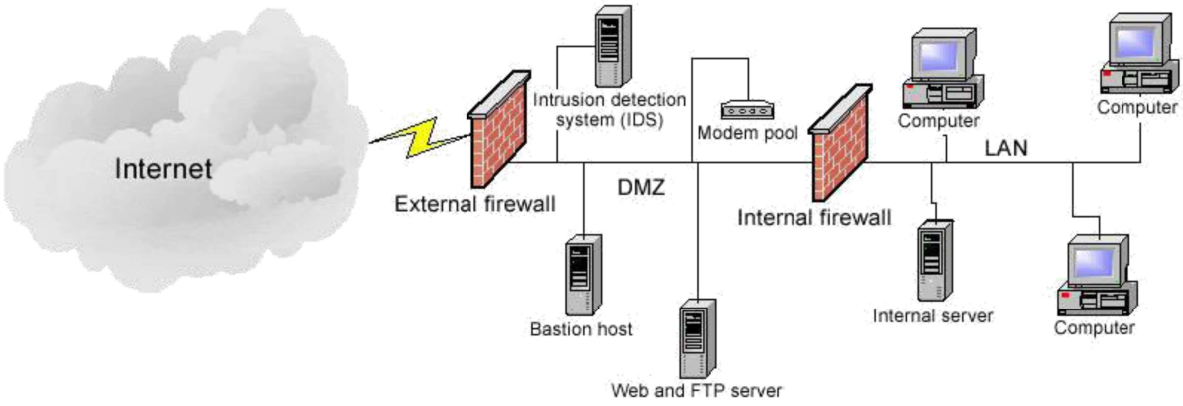
Before purchasing or upgrading your dedicated firewall, **you should have a solid security policy in place**. A firewall will enforce your security policy, and by having it documented, there will be fewer questions when configuring your firewall to reflect that policy. Any changes made to the firewall should be amended in the security policy.

One of the best features of a well-designed firewall is the ability to funnel traffic through checkpoints. When you configure your firewall to force traffic (outbound and inbound) through specific points in your firewall, you can easily monitor your logs for normal and suspicious activity.

How do you monitor your firewall once you have a security policy and checkpoints configured? By using alarms and enabling logging on your firewall, you can easily monitor all authorized and unauthorized access to your network. You can even purchase third-party utilities to help filter out the messages you don't need. It's also a good practice to hide your internal network address scheme from the outside world. It is never wise to let the outside world know the layout of your network.

Demilitarizedzone (DMZ) topology

A DMZ is the most common and secure firewall topology. It is often referred to as a screened subnet. A DMZ creates a secure space between your Internet and your network, as shown in Figure D.



A DMZ will typically contain the following:

- Web server
- Mail server
- Application gateway
- E-commerce systems (It should contain only your front-end systems. Your back-end systems should be on your internal network.)

XML and Web Services Security Standards:-

XML and Web services are widely used in current distributed systems. The security of the XML based communication, and the Web services themselves, is of great importance to the overall security of these systems. Furthermore, in order to facilitate interoperability, the security mechanisms should preferably be based on established standards. In this paper we provide a tutorial on current security standards for XML and Web services. The discussed standards include XML Signature, XML Encryption, the XML Key Management Specification (XKMS), WS-Security, WS-Trust, WS-SecureConversation, Web Services Policy, WS-SecurityPolicy, the eXtensible Access Control Markup Language (XACML), and the Security Assertion Markup Language (SAML).

What Is an XML Web Service?

XML Web services are the fundamental building blocks in the move to distributed computing on the Internet. Open standards and the focus on communication and collaboration among people and applications have created an environment where XML

Web services are becoming the platform for application integration. Applications are constructed using multiple XML Web services from various sources that work together regardless of where they reside or how they were implemented.

There are probably as many definitions of XML Web Service as there are companies building them, but almost all definitions have these things in common:

- XML Web Services expose useful functionality to Web users through a standard Web protocol. In most cases, the protocol used is SOAP.
- □ XML Web services provide a way to describe their interfaces in enough detail to allow a user to build a client application to talk to them. This description is usually provided in an XML document called a Web Services Description Language (WSDL) document.
- □ XML Web services are registered so that potential users can find them easily. This is done with Universal Discovery Description and Integration (UDDI).

I'll cover all three of these technologies in this article but first I want to explain why you should care about XML Web services.

One of the primary advantages of the XML Web services architecture is that it allows programs written in different languages on different platforms to communicate with each other in a standards-based way. Those of you who have been around the industry a while are now saying, "Wait a minute! Didn't I hear those same promises from CORBA and before that DCE? How is this any different?" The first difference is that SOAP is significantly less

complex than earlier approaches, so the barrier to entry for a standards-compliant SOAP implementation is significantly lower. Paul Kulchenko maintains a list of SOAP implementations which at last count contained 79 entries. You'll find SOAP implementations from most of the big software companies, as you would expect, but you will also find many implementations that are built and maintained by a single developer. The other significant advantage that XML Web services have over previous efforts is that they work with standard Web protocols—XML, HTTP and TCP/IP. A significant number of companies already have a Web infrastructure, and people with knowledge and experience in maintaining it, so again, the cost of entry for XML Web services is significantly less than previous technologies. We've defined an XML Web service as a software service exposed on the Web through SOAP, described with a WSDL file and registered in UDDI. The next logical question is.

services that would be difficult to implement using RPC.

The last optional part of the SOAP specification defines what an HTTP message that

contains a SOAP message looks like. This HTTP binding is important because HTTP is supported by almost all current OS's (and many not-so-current OS's). The HTTP binding is optional, but almost all SOAP implementations support it because it's the only standardized protocol for SOAP. For this reason, there's a common misconception that SOAP requires HTTP. Some implementations support MSMQ, MQ Series, SMTP, r TCP/IP transports, but almost all current XML Web services use HTTP because it is ubiquitious. Since HTTP is a core protocol of the Web, most organizations have a network infrastructure that supports HTTP and people who understand how to manage it already. The security, monitoring, and load-balancing infrastructure for HTTP are readily available today.

A major source of confusion when getting started with SOAP is the difference between the SOAP specification and the many implementations of the SOAP specification. Most people who use SOAP don't write SOAP messages directly but use a SOAP toolkit to create and parse the SOAP messages. These toolkits generally translate function calls from some kind of language to a SOAP message. For example, the Microsoft SOAP Toolkit 2.0 translates COM function calls to SOAP and the Apache Toolkit translates JAVA function calls to SOAP. The types of function calls and the data types of the parameters supported vary with each SOAP implementation so a function that works with one toolkit may not work with another. This isn't a limitation of SOAP but rather of the particular implementation you are using.

By far the most compelling feature of SOAP is that it has been implemented on many different hardware and software platforms. This means that SOAP can be used to link disparate systems within and without your organization. Many attempts have been made in the past to come up with a common communications protocol that could be used for systems integration, but none of them have had the widespread adoption that SOAP has. Why is this? Because SOAP is much smaller and simpler to implement than many of the previous protocols. DCE and CORBA for example took years to implement, so only a few implementations were ever released. SOAP, however, can use existing XML Parsers and HTTP libraries to do most of the hard work, so a SOAP implementation can be completed in a matter of months. This is why there are more than 70 SOAP implementations available. SOAP obviously doesn't do everything that DCE or CORBA do, but the lack of complexity in exchange for features is what makes SOAP so readily available. The ubiquity of HTTP and the simplicity of SOAP make them an ideal basis for implementing XML Web services that can be called from almost any environment. For more information on SOAP.

What About Security?

One of the first questions newcomers to SOAP ask is how does SOAP deal with security. Early in its development, SOAP was seen as an HTTP-based protocol so the assumption was made that HTTP security would be adequate for SOAP. After all, there are thousands of Web applications running today using HTTP security so surely this is a equate for SOAP. For this reason, the current SOAP standard assumes security is a transport issue and is silent on security issues.

When SOAP expanded to become a more general-purpose protocol running on top of a number of transports, security became a bigger issue. For example, HTTP provides several ways to authenticate which user is making a SOAP call, but how does that identity get propagated when the message is routed from HTTP to an SMTP transport? SOAP was designed as a building-block protocol, so fortunately, there are already specifications in the

works to build on SOAP to provide additional security features for Web services. The WS-Security specification defines a complete encryption system.

WSDL

WSDL (often pronounced whiz -dull) stands for Web Services Description Language. For our purposes, we can say that a WSDL file is an XML document that describes a set of SOAP messages and how the messages are exchanged. In other words, WSDL is to SOAP what IDL is to CORBA or COM. Since WSDL is XML, it is readable and editable but in most cases, it is generated and consumed by software.

To see the value of WSDL, imagine you want to start calling a SOAP method provided by one of your business partners. You could ask him for some sample SOAP messages and write your application to produce and consume messages that look like the samples, but this can be error-prone. For example, you might see a customer ID of 2837 and assume it's an integer when in fact it's a string. WSDL specifies what a request message must contain and what the response message will look like in unambiguous notation.

The notation that a WSDL file uses to describe message formats is based on the XML Schema standard which means it is both programming-language neutral and standards-based which makes it suitable for describing XML Web services interfaces that are accessible from a wide variety of platforms and programming languages. In addition to describing message contents, WSDL defines where the service is available and what communications protocol is used to talk to the service. This means that the WSDL file defines everything required to write a program to work with an XML Web service. There are several tools available to read a WSDL file and generate the code required to communicate with an XML Web service. Some of the most capable of these tools are in Microsoft Visual Studio® .NET.

Many current SOAP toolkits include tools to generate WSDL files from existing program interfaces, but there are few tools for writing WSDL directly, and tool support for WSDL isn't as complete as it should be. It shouldn't be long before tools to author WSDL files, and then generate proxies and stubs much like COM IDL tools, will be part of most SOAP implementations. At that point, WSDL will become the prefered way to author SOAP interfaces for XML Web services.

UDDI

Universal Discovery Description and Integration is the yellow pages of Web services. As with traditional yellow pages, you can search for a company that offers the services you need, read about the service offered and contact someone for more information. You can, of course, offer a Web service without registering it in UDDI, just as you can open a business in your basement and rely on word -of- mouth advertising but if you want to reach a significant market, you need UDDI so your customers can find you.

A UDDI directory entry is an XML file that describes a business and the services it offers. There are three parts to an entry in the UDDI directory.

The "white pages" describe thecompany offering the service: name, address, contacts, etc. The "yellow pages" include industrial categories based on standard taxonomies such as the North American Industry Classification System and the Standard Industrial Classification. The "green pages" describe the interface to the service in enough detail for someone to write an application to use the Web service. The way services are defined is through a UDDI document called a Type Model or tModel. In many cases, the tModel contains a WSDL file that describes a SOAP interface to an XML Web service, but the tModel is flexible enough to describe almost any kind of service.

The UDDI directory also includes several ways to search for the services you need to build your applications. For example, you can search for providers of a service in a specified geographic location or for business of a specified type. The UDDI directory will then supply information, contacts, links, and technical data to allow you to evaluate which services meet your requirements.

UDDI allows you to find businesses you might want to obtain Web services from. What if you already know whom you want to do business with but you don't know what services are offered? The WS-Inspection specification allows you to browse through a collection of XML Web services offered on a specific server to find which ones might meet your needs.

Semantic interpolation:-

he problem of interpolation is a classical problem in logic. Given a consequence relation |~ and two formulas and ψ with |~ ψ we try to find a “simple" formula α such that |~ α |~ ψ. “Simple" is defined here as “expressed in the common language of and ψ". Non-monotonic logics like preferential logics are often a mixture of a non-monotonic part with classical logic. In such cases, it is natural examine also variants of the interpolation problem, like: is there “simple" α such that α |~ ψ where is classical consequence? We translate the interpolation problem from the syntactic level to the semantic level. For example, the classical interpolation problem is now the question whether there is some “simple" model set X such that M() X M(ψ). We can show that such X always exist f monotonic and anti tonic logics. The case of non-monotonic logics is more complicated, there are several variants to consider, and we mostly have only partial results.

Service-Oriented Architecture (SOA):-

A service-oriented architecture is essentially a collection of services. These services communicate with each other. The communication can involve either simple data passing or it could involve two or more services coordinating some activity. Some means of connecting services to each other is needed.

Service-oriented architectures are not a new thing. The first service-oriented architecture for many people in the past was with the use DCOM or Object Request Brokers (ORBs) based on the CORBA specification. For more on DCOM and CORBA

Services

If a service-oriented architecture is to be effective, we need a clear understanding of the term service. A service is a function that is well-defined, self-contained, and does not depend on the context or state of other services. See Service.

Connections

The technology of Web Services is the most likely connection technology of service-oriented architectures. The following figure illustrates a basic service-oriented architecture. It shows a service consumer at the right sending a service request message to a service provider at the left. The service provider returns a response message to theservice consumer. The request and subsequent response connections are defined in some way that is understandable to both the service consumer and service provider. How those connections are defined is explained in Web Services Explained. A service provider can also be a service consumer.

Metadata

Metadata can be defined as a set of assertions about things in our domain of discourse. Metadata is a component of data, which describes the data. It is " data about data". Often there is more than that, involving information about data as they is stored managed ,and revealing partial semantics such as intended use (i.e., application) of data. This information can be of broad variety, meeting if not surpassing the variety in the data themselves. They may describe, or be a summary of the information content of the individual databases in an intentional manner. Some metadata may also capture content independent information like location and time of creation.

Metadata descriptions present two advantages [2]:

- They enable the abstraction of representational details such as the format and organization of data, and capture the information content of the underlying data independent of representational details. This represents the first step in reduction of information overload, as intentional metadata descriptions are in general an order of magnitude smaller than the underlying data.
- They enable representation of domain knowledge describing the information domain to which the underlying data belong. This knowledge may then be used to make inferences about the underlying data. This helps in reducing information overload as the inferences may be used to determine the relevance of the underlying data without accessing the data.

Metadata can be classified based on different criteria. Based on the level of abstraction in which a metadata describes content, the metadata can be classified as follows [9]:

Syntactic Metadata focuses on details of the data source (document) providing little insight into the data. This kind of metadata is useful mainly for categorizing or cataloguing the data source.
Examples if syntactic metadata include language of the data source, creation date, title, size, format etc.
Structural Metadata focuses on the structure of the document data, which facilitates data storage, processing and presentation such as navigation, eases Book Chapter, Datenbanken und Informationssysteme, Festschrift zum 60. Geburtstag von Gunter Schlageter, Publication Hagen, October 2003-09-26 3. information retrieval, and improves display. E.g. XML schema, the physical structure of the document like page images etc.
Semantic Metadata describes contextually relevant information focusing on domainspecific elements based on ontology, which a user familiar with the domain is likely to know or understand easily. Using semantic metadata, meaningful interpretation of data is possible and interoperability will then be supported at high-level (hence easier to use), providing meaning to the underlying syntax and structure.

Metadata in WSDL and UDDI standards:- The standards such as WSDL [14] and UDDI are used to share the metadata about a web service. Each standard provides metadata about services at a certain level of abstraction. WSDL describes the service using the implementation details and hence it can be considered as a standard to represent the metadata of the invocation details of service. As the purpose of UDDI is to locate WSDL descriptions, it can be thought of as a standard for publishing and discovering metadata of web services. Considering the details in WSDL and UDDI as metadata of a Web service, the different kind of metadata of Web services available in different standards can be categorized

Semantics for Web Services:-

In the previous section, we discussed different kinds of metadata available in WSDL and UDDI. Section 2 discussed the power of semantic metadata. In Web services domain, semantics represented by the semantic metadata can be classified into the following types [21], namely

- Functional Semantics
- Data Semantics
- QoS Semantics and
- Execution Semantics

These different types of semantics can be used to represent the capabilities, requirements, effects and execution pattern of a Webservice. The semantic Web research focuses to date as focused on the data semantics that helps in semantic tagging of static information available on the Web from all kind of sources. Research on semantic Web services on the other hand is based on the findings and results from the semantic Web research to apply for services that perform some action producing an effect. Unlike information retrieval,

Enterprise Management Framework (EMF)

The Hirsch Enterprise Management Framework, or EMF, is an IIS-based application that provides a browser interface to portions of the Hirsch Velocity application for user convenience and enterprise system consolidation.

EMF allows operators with occasional need to access the Velocity application to add delete a user, view events or run reports from a browser, rather than the full Velocity client. It allows
allows users with multiple Velocity servers to manage users across one more servers, view events from one or more servers, and run activity reports across one or more servers.