Chapter 6

Modern EAI Frameworks

6.1 Introduction

We have discussed number of EAI frameworks and approaches in chapter 3. However the modern EAI frameworks are based on the concept of Web Services. Present Enterprise Application Integration solutions are usually tightly coupled, proprietary, and are not easily amenable to changes. On the other hand web services have emerged as solution to the Internet business with simple and extensible option. Usage of Web services results in an easy and scalable Enterprise Application Integration. In this chapter we discuss in detail the web services concept and its associated components viz. Simple Object Access Protocol (SOAP), Web Services Description Language (WSDL), and Universal Description, Discovery and Integration (UDDI).

6.2 Web Services

6.2.1 Purpose:

Web services provide a standard means of interoperating between different software applications, running on a variety of platforms and/or frameworks. Interoperability is the key to business automation. Hence web services provides the ability of the application to interface and communicate with other application on disparate systems with less of integration and development effort [39].

6.2.2 Definition:

Web Services are defined as “A software system identified by a URI, whose public interfaces and bindings are defined and described using XML” [40].
A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards. Accordingly, a web service is defined as the software components that employ one or more of the following technology-SOAP, WSDL or UDDI. Web Services involved following categories:

- **Agents and Services**

A Web service is an abstract notion that must be implemented by a concrete agent. (refer Figure 6.1) The agent is the concrete piece of software or hardware that sends and receives messages, while the service is the resource characterized by the abstract set of functionality that is provided.

- **Requesters and Providers**

The purpose of a Web service is to provide some functionality on behalf of its owner -- a person or organization, such as a business or an individual. The provider entity is the person or organization that provides an appropriate agent to implement a particular service. (refer Figure 6.1)

A requester entity is a person or organization that wishes to make use of a provider entity's Web service. It will use a requester agent to exchange messages with the provider entity's provider agent. In order for this message exchange to be successful, the requester entity and the provider entity must first agree on both the semantics and the mechanics of the message exchange.

- **Service Description**

The mechanics of the message exchange are documented in a Web service description (WSD). (refer Figure 6.1) The WSD is a machine-processable specification of the Web service's interface, written in WSDL. It defines the message formats, datatypes, transport protocols, and transport serialization formats that should be used between
the requester agent and the provider agent. It also specifies one or more network locations at which a provider agent can be invoked, and may provide some information about the message exchange pattern that is expected.

• Semantics

The semantics of a Web service is the shared expectation about the behavior of the service, in particular in response to messages that are sent to it. In effect, this is the "contract" between the requester entity and the provider entity regarding the purpose and consequences of the interaction. Although this contract represents the overall agreement between the requester entity and the provider entity on how and why their respective agents will interact, it is not necessarily written or explicitly negotiated. It may be explicit or implicit, oral or written, machine processable or human oriented.

6.2.3 Broad Steps to Illustrate Web Services:

![Diagram of Web Service Interaction]

1. Parties "become known" to each other

2. Agree on semantics & WSD

3. Input Semantics & WSD

4. Interact

Figure 6.1 Basic Architectural Roles

89
(1) the requester and provider entities become known to each other (or at least one becomes known to the other).

a) The discovery service somehow obtains both the Web service description and an associated functional description of the service.

The functional description is a machine-processable description of the functionality (or partial semantics) of the service that the provider entity is offering. It could be as simple as a few words of metadata or a URI, or it may be more complex, such as a TModel (in UDDI) or a collection of RDF, DAML-S or OWL-S statements.

This architecture does not specify or care how the discovery service obtains the service description or functional description. For example, if the discovery service is implemented as a search engine, then it might crawl the Web, collecting service descriptions wherever it finds them, with the provider entity having no knowledge of it. Or, if the discovery service is implemented as a registry (such as UDDI), then the provider entity may need to actively publish the service description and functional description directly to the discovery service.

b) The requester entity supplies criteria to the discovery service to select a Web service description based on its associated functional description, capabilities and potentially other characteristics. One might locate a service having certain desired functionality or semantics; however, it may be possible to specify "non-functional" criteria related to the provider agent, such as the name of the provider entity, performance or reliability criteria, or criteria related to the provider entity, such as the provider entity's vendor rating.

c) The discovery service returns one or more Web service descriptions (or references to them) that meet the specified criteria. If multiple service descriptions are returned, the requester entity selects one, perhaps using additional criteria.
(2) the requester and provider entities somehow agree on the service description and semantics that will govern the interaction between the requester and provider agents.

(3) the service description and semantics are realized by the requester and provider agents.

(4) the requester and provider agents exchange messages, thus performing some task on behalf of the requester and provider entities [40].

6.2.4 Web Services Interoperability

Interoperability is one of the main promises of Web services. Web services are designed to be independent of the underlying operating system and programming language.

6.2.4.1 The Interoperability Challenge

Applications use SOAP toolkits (usually in the form of libraries) to prepare XML-based SOAP envelopes, send the envelopes over a transport protocol such as HTTP or SMTP, and process incoming SOAP envelopes. If two SOAP toolkits make different assumptions about how to build and interpret envelopes, interoperability problems may occur.

For example, one or both of the SOAP toolkits might implement only a subset of the full SOAP or XML specifications. In some cases, there may be a mismatch in what features are supported, where one system sends a SOAP document that the other cannot process. The SOAP specification makes certain things optional — for instance, the SOAP specification makes it optional to send type information for encoded parameters. If one implementation assumes that type information will exist on messages it receives, it may not interoperate with another implementation that
chooses not to send that information. Two implementers may interpret parts of the SOAP specification differently, where the language of the specification is ambiguous. Finally, implementers may simply have bugs in their toolkits that do not show up during stand-alone testing. Such bugs may also contribute to interoperability problems when two different toolkits attempt to hook together [41].

6.2.5 Web services characteristics

A Web Service exhibits the following definitive characteristics:
• A Web Service is accessible over the Web. Web Services communicate using platform-independent and language-neutral Web protocols. These Web protocols ensure easy integration of heterogeneous environments.
• A Web Service provides an interface that can be called from another program. This application-to-application programming interface can be invoked from any type of application client or service. The Web Service interface acts as a liaison between the Web and the actual application logic that implements the Service.
• A Web Service is registered and can be located through a Web Service Registry. The registry enables service consumers to find services that match their needs.
• Web Services support loosely coupled connections between systems. Web Services communicate by passing messages to each other. The Web Service interface adds a layer of abstraction to the environment that makes the connections flexible and adaptable [42].

6.2.6 Web Services Application

• Supply Chain Management and Logistic
The Business of Supply Chain Management And Logistic encompasses the entire process of manufacturing and distributing products, from suppliers to the customer. Logistic is that portion of supply chain that comprises distribution, transportation and inventory management. SCM comprises interaction
among different partners and collaborators indicating the unlimited possibility of web services playing crucial role in the business automation.

- **Customer Relations Management**
  CRM is all about developing customer-centric organization. In CRM three classes of problems can be identified that needs to be addressed by web technology.

  1) Application Integration
  2) Data Integration
  3) Process Integration

- **Financial Services and Banking**
  The opportunity for web services in financial services and banking institute is to create a universal approach to integration of the assets of banking business.

- **Education**
  E-learning is a system/application made up of multiple technologies. It offers an application that were bundled suites. Web services have the potential to enable atomization of what were previously bundled suites.

- **Manufacturing**
  Web services along with collaborative systems, are expected to enable manufactureres quickly and easily enforce changes in internal as well as external organizations. Hence, any enterprise can be benefited from the web services technology. The architectural components of the web services are explained in the following sections [39].

6.3 SOAP

Basic XML provides a firm foundation for the data interchange and interoperatbility. The XML specification takes care only of the data representation part. However, it does not make any assumptions on any protocol for its transport. Hence SOAP is the Simple Object Access Protocol and it is able to move XML data over the wire and specifies a set of rules for application on the target system to initiate some action, upon the delivery of XML data.
SOAP is a standard that represents a lightweight envelope containing the message payload as it moves between service producer and consumers. SOAP is fundamentally a stateless, one-way message exchange paradigm, but applications can create more complex interaction patterns (e.g., request/response, request/multiple responses, etc.) by combining such one-way exchanges with features provided by an underlying protocol and/or application-specific information. SOAP is silent on the semantics of any application-specific data it conveys, as it is on issues such as the routing of SOAP messages, reliable data transfer, firewall traversal, etc. However, SOAP provides the framework by which application-specific information may be conveyed in an extensible manner. Also, SOAP provides a full description of the required actions taken by a SOAP node on receiving a SOAP message [43].

SOAP is composed of following part:

- Message Envelope
- Encoding Rules
- RPC Conventions
- Binding with Underlying Protocol

6.3.1 SOAP message

A SOAP message is fundamentally a one-way transmission between SOAP nodes, from a SOAP sender to a SOAP receiver, but SOAP messages are expected to be combined by applications to implement more complex interaction patterns ranging from request/response to multiple, back-and-forth "conversational" exchanges.

- SOAP Messages

A SOAP header element is optional, but it has been included in the example to explain certain features of SOAP. A SOAP header is an extension mechanism that provides a way to pass information in SOAP messages that is not application payload. Such "control" information includes, for example, passing directives or contextual information related to the processing of the message. This allows a SOAP message to
be extended in an application-specific manner. The immediate child elements of the 
env:Header element are called header blocks, and represent a logical grouping of data 
which, as shown later, can individually be targeted at SOAP nodes that might be 
encountered in the path of a message from a sender to an ultimate receiver.

SOAP headers have been designed in anticipation of various uses for SOAP, many of 
which will involve the participation of other SOAP processing nodes - called SOAP 
intermediaries - along a message's path from an initial SOAP sender to an ultimate 
SOAP receiver. This allows SOAP intermediaries to provide value-added services. 
Headers, as shown later, may be inspected, inserted, deleted or forwarded by SOAP 
nodes encountered along a SOAP message path. (It should be kept in mind, though, 
that the SOAP specifications do not deal with what the contents of header elements 
are, or how SOAP messages are routed between nodes, or the manner by which the 
route is determined and so forth. These are a part of the overall application, and could 
be the subject of other specifications.)

The SOAP body is the mandatory element within the SOAP env:Envelope, which 
implies that this is where the main end-to-end information conveyed in a SOAP 
message must be carried.

The env:Body element and its associated child elements, itinerary and lodging, are 
tended for exchange of information between the initial SOAP sender and the SOAP 
node which assumes the role of the ultimate SOAP receiver in the message path, 
which is the travel service application. Therefore, the env:Body and its contents are 
implicitly targeted and are expected to be understood by the ultimate receiver. The 
means by which a SOAP node assumes such a role is not defined by the SOAP 
specification, and is determined as a part of the overall application semantics and 
associated message flow.

• SOAP Message Exchange

SOAP Version 1.2 is a simple messaging framework for transferring information 
specified in the form of an XML infoset between an initial SOAP sender and an
ultimate SOAP receiver. The more interesting scenarios typically involve multiple message exchanges between these two nodes. The simplest such exchange is a request-response pattern. Some early uses of emphasized the use of this pattern as means for conveying remote procedure calls (RPC), but it is important to note that not all SOAP request-response exchanges can or need to be modelled as RPCs. The latter is used when there is a need to model a certain programmatic behavior, with the exchanged messages conforming to a pre-defined description of the remote call and its return.

A much larger set of usage scenarios than that covered by the request-response pattern can be modeled simply as XML-based content exchanged in SOAP messages to form a back-and-forth "conversation", where the semantics are at the level of the sending and receiving applications.

- Remote Procedure Calls

One of the design goals of SOAP Version 1.2 is to encapsulate remote procedure call functionality using the extensibility and flexibility of XML. SOAP Part 2 section 4 has defined a uniform representation for RPC invocations and responses carried in SOAP messages. This section continues with the travel reservation scenario to illustrate the use of SOAP messages to convey remote procedure calls and their return.

To that end, the next example shows the payment for the trip using a credit card. (It is assumed that the conversational exchanges described in section 2.2.1 have resulted in a confirmed itinerary.) Here, it is further assumed that the payment happens in the context of an overall transaction where the credit card is charged only when the travel and the lodging (not shown in any example, but presumably reserved in a similar manner) are both confirmed. The travel reservation application provides credit card information and the successful completion of the different activities results in the card being charged and a reservation code returned. This reserve-and-charge interaction between the travel reservation application and the travel service application is modeled as a SOAP RPC.
To invoke a SOAP RPC, the following information is needed:

1. The address of the target SOAP node.
2. The procedure or method name.
3. The identities and values of any arguments to be passed to the procedure or method together with any output parameters and return value.
4. A clear separation of the arguments used to identify the Web resource which is the actual target for the RPC, as contrasted with those that convey data or control information used for processing the call by the target resource.
5. The message exchange pattern which will be employed to convey the RPC, together with an identification of the so-called "Web Method" (on which more later) to be used.
6. Optionally, data which may be carried as a part of SOAP header blocks

Such information may be expressed by a variety of means, including formal Interface Definition Languages (IDL). Note that SOAP does not provide any IDL, formal or informal. Note also that the above information differs in subtle ways from information generally needed to invoke other, non-SOAP RPCs.

- **Fault Scenarios**

SOAP provides a model for handling situations when faults arise in the processing of a message. SOAP distinguishes between the conditions that result in a fault, and the ability to signal that fault to the originator of the faulty message or another node. The ability to signal the fault depends on the message transfer mechanism used, and one aspect of the binding specification of SOAP onto an underlying protocol is to specify how faults are signalled, if at all. The remainder of this section assumes that a transfer mechanism is available for signalling faults generated while processing received messages, and concentrates on the structure of the SOAP fault message [44].
6.3.2 SOAP Processing Model

Having established the various syntactical aspects of a SOAP message as well as some basic message exchange patterns, this section provides a general overview of the SOAP processing model. The SOAP processing model describes the actions taken by a SOAP node on receiving a SOAP message.

1) The "role" Attribute
Further processing of header blocks and the body depend on the role(s) assumed by the SOAP node for the processing of a given message. SOAP defines the (optional) env:role attribute - syntactically, xs:anyURI - that may be present in a header block, which identifies the role played by the intended target of that header block. A SOAP node is required to process a header block if it assumes the role identified by the value of the URI. How a SOAP node assumes a particular role is not a part of the SOAP specifications.

2) The "mustUnderstand" Attribute

After a SOAP node has correctly identified the header blocks (and possibly the body) targeted at itself using the env:role attribute, the additional attribute, env:mustUnderstand, in the header elements determines further processing actions that have to be taken. In order to ensure that SOAP nodes do not ignore header blocks which are important to the overall purpose of the application, SOAP header blocks also provide for the additional optional attribute, env:mustUnderstand, which, if "true", means that the targeted SOAP node must process the block according to the specification of that block. Such a block is colloquially referred to as a mandatory header block. In fact, processing of the SOAP message must not even start until the node has identified all the mandatory header blocks targeted at itself, and "understood" them. Understanding a header means that the node must be prepared to do whatever is described in the specification of that block.

3) The "relay" Attribute

SOAP defines another optional attribute for header blocks, env:relay of type xs:boolean, which indicates if a header block targeted at a SOAP intermediary must be relayed if it is not processed.

Note that if a header block is processed, the SOAP processing rules requires that it be removed from the outbound message. (It may, however, be reinserted, either unchanged or with its contents altered, if the processing of other header blocks determines that the header block be retained in the forwarded
message.) The default behavior for *an unprocessed* header block targeted at a role played by a SOAP intermediary is that it must be removed before the message is relayed.

The reason for this choice of default is to lean on the side of safety by ensuring that a SOAP intermediary make no assumptions about the survivability past itself of a header block targeted at a role it assumes, and representing some value-added feature, particularly if it chooses not to process the header block, very likely because it does not "understand" it. That is because certain header blocks represent hop-by-hop features, and it may not make sense to unknowingly propagate it end-to-end. As an intermediary may not be in a position to make this determination, it was thought that it would be safer if unprocessed header blocks were removed before the message was relayed.

### 6.3.3 SOAP Encoding

In order to build SOAP message it is essential to serialize the application related data into SOAP message format. Rules that are applied to serialize the data into SOAP message is called SOAP encoding.

**Encoding Details**

SOAP Encoding enables to represent data in SOAP required format. The data type that can be used in SOAP can be divided into simple types and compound types

- **Simple Types**

  A simple type data can be either build in data type or derived data type. A build in data type is a data type that is defined in XML schema. Derived data type is a data type that is derived from built in data type.

- **Compound Types**

  Compound type data can be of struct or array data types [39].

100
6.3.4 Using Various Protocol Bindings

SOAP messages may be exchanged using a variety of "underlying" protocols, including other application layer protocols. The specification of how SOAP messages may be passed from one SOAP node to another using an underlying protocol is called a SOAP binding. In terms of element and attribute information items of an abstract "document" called the env:Envelope. Any SOAP env:Envelope infoset representation will be made concrete through a protocol binding, whose task, among other things, it is to provide a serialized representation of the infoset that can be conveyed to the next SOAP node in the message path in a manner such that the infoset can be reconstructed without loss of information. In typical examples of SOAP messages, and certainly in all the examples in this primer, the serialization shown is that of a well-formed document. However, there may be other protocol bindings - for example a protocol binding between two SOAP nodes over a limited bandwidth interface - where an alternative, compressed serialization of the same infoset may be chosen. Another binding, chosen for a different purpose, may provide a serialization which is an encrypted structure representing the same infoset.

1) The SOAP HTTP Binding

HTTP has a well-known connection model and a message exchange pattern. The client identifies the server via a URI, connects to it using the underlying TCP/IP network, issues a HTTP request message and receives a HTTP response message over the same TCP connection. HTTP implicitly correlates its request message with its response message; therefore, an application using this binding can chose to infer a correlation between a SOAP message sent in the body of a HTTP request message and a SOAP message returned in the HTTP response. Similarly, HTTP identifies the server endpoint via a URI, the Request-URI, which can also serve as the identification of a SOAP node at the server.

HTTP allows for multiple intermediaries between the initial client and the origin server identified by the Request-URI, in which case the request/response model is a
series of such pairs. Note, however, that HTTP intermediaries are distinct from SOAP intermediaries.

2) SOAP Over Email

Application developers can use the Internet email infrastructure to move SOAP messages as either email text or attachments. The examples shown below offer one way to carry SOAP messages, and should not be construed as being the standard way of doing so. There is a non-normative W3C describing an email binding for SOAP, its main purpose being to demonstrate the application of the general SOAP Protocol Binding Framework.

Hence SOAP is a simple but powerful protocol that is designed to interchange the data in interoperable and extensible form [45].

6.3.5 Use of URIs in SOAP

SOAP uses URIs for some identifiers including, but not limited to, values of the encodingStyle and role attribute information items. To SOAP, a URI is simply a formatted string that identifies a web resource via its name, location, or any other characteristics. Although this section only applies to URIs directly used by information items defined by this specification, it is RECOMMENDED that application-defined data carried within a SOAP envelope use the same [46].

6.3.6 Characteristics of Service-Oriented Architectures

- Legacy applications – a wide range of them, written in a variety of procedural languages that typically do not have well-defined interfaces for collaboration with other applications.
- Object-oriented applications – many different applications developed as components for other applications, such as Enterprise JavaBeans™, COM, or CORBA objects.
- Transaction systems – custom-developed applications that run under the control of transaction processing monitors such as customer information control system
(CICS), IMS Transaction Manager (IMS/TM), BEA Tuxedo, and other software; and whose individual transactions are well suited to incorporation into collaborative business processes.

- Packaged application systems – vendor-supplied, proprietary systems with well-defined, often completely proprietary, collaborative interfaces.
- Databases and files – vendor-supplied, proprietary relational database management systems (RDBMSs), legacy database management systems (DBMSs), and file systems with varying degrees of standard and proprietary interfaces.
- Communication transports and message formats – industry-standard transports such as hypertext transport protocol (HTTP), file transfer protocol (FTP), and simple mail transfer protocol (SMTP); vendor-supplied proprietary messaging and queuing systems from companies such as IBM and TIBCO; and a variety of formats for exchanging data between enterprises, such as electronic data interchange (EDI), Society for the Worldwide Interbank Financial Telecommunication (SWIFT), Health Insurance Portability and Accountability Act (HIPAA), and others.
- e-Business exchanges – proprietary and standards-based public and private exchanges through which businesses collaborate with other businesses.
- Application server platforms – application and integration servers that host all manner of applications and facilitate electronic collaboration within and between enterprises [47].

6.4 Web Services Description Language (WSDL)

WSDL is a language for describing the capabilities of Web services. WSDL is based on XML and is a key part of the UDDI initiative. The WSDL document specification helps improve interoperability between applications, regardless of the protocol or the encoding scheme. The WSDL 1.1 specification defines WSDL as “an XML grammar for describing network services as collections of communication endpoints capable of exchanging messages.”
Essentially, a WSDL document describes how to invoke a service and provides information on the data being exchanged, the sequence of messages for an operation, protocol bindings, and the location of the service. A WSDL document defines services as a collection of endpoints, but separates the abstract definition from the concrete implementation [48].

Messages and port types provide abstract definitions for the data being exchanged and the operations being performed by a service. A binding is provided to map to a concrete set of ports, usually consisting of a URL location and a SOAP binding. Figure 6.3 illustrates the various components in a WSDL document.

6.4.1 Web Service Invocation And WSDL

There are three participants for web service invocation. They are Web service provider or Server Application, Service Requestor or Client Application, Service Broker or Registry.

Following describes some of the basic steps in detail:

a) Service Creation

Service creation is a step that will enable the provider of the service to develop and deploy the service which should be accessible in a distributes enviorment. This service is entierly optional ,an application that are presently available on the enterprise could be considerd as candidate for web services.

b) Service Description

Software tools and IDEs could be used to generate the WSDL document for a given service. WSDL takes care of three important things location of the available service, exposition of appropriate service(s) and the semantics of the service invocation.

c) Service Registration

Service Provider now registers himself with a registry, public or private, and stores the location information of the WSDL document in the registry. It is done only once for a given service if updated the client has to pick up the updated information. The process of service registration is done by SOAP request.
d) Service Discovery
In this step a client who is on the look out for a service browses the registry for services using a set of keywords. Automatic scanning of a registry by application is a SOAP RPC request to the registry or standard API's are available that help applications.

e) Service Invocation
There are two primary strategies for service invocation.
   i. Cinets use low level API's to get connected the services.
   ii. Clients can generate proxy locally and use the same for service invocation.

Figure 6.3 WSDL Components
6.4.2 Web Service Description Details (WSDL)

a) Elements
   The definition is the root element that contains seven elements which describe
   web services:
   
   - **Types**—a container for data type definitions using some type system (such as
     XSD).
   - **Message**—an abstract, typed definition of the data being communicated.
   - **Operation**—an abstract description of an action supported by the service.
   - **Port Type**—an abstract set of operations supported by one or more endpoints.
   - **Binding**—a concrete protocol and data format specification for a particular
     port type.
   - **Port**—a single endpoint defined as a combination of a binding and a network
     address.
   - **Service**—a collection of related endpoints [49].

b) Description Of Elements
   
   - **The <type> Element** which is container for data type definitions using some
     type definition using some type system such as XML Schema definition. They
     define the data type of the information used in the element <message> it
     support one or more <schema> as a child element.
   - **The <message> Element** will be one or more in number, which is an
     abstract,"typed" definition of data being communicated. It refers to the type
     defined in <types> element. It supports <part> elements.
   - **The <operation> Elements** is an abstract description of an action supported
     by the <service> element. There are four types of behavior that are considered
     important:
     
     - **Request-Response** corresponds to synchronous behavior. The client
       sends the request and blocks the call until the server responds.
- **Solicit-Response** This behavior corresponds to reverse in order of the Request-Response behavior. The server sends the Solicite and expects the response from the client.

- **One Way** This is the behavior that corresponds to the client sending a Request to the web server, without waiting or blocking for Response.

- **Notification** This behavior offers the reverse of the One-Way.

- **The `<protoType>` Element** describes one or more abstract set of operations supported by one or more end points.

- **The `<port>` Element** provides a simple end point defined as a combination of a `<binding>` and network address. The `<port>` element models single web service [50].

6.5 **Universal Description, Discovery and Integration (UDDI)**

Universal Description, Discovery and Integration (UDDI) is a specification for distributed Web-based information registries of Web services. UDDI is also a publicly accessible set of implementations of the specification that allow businesses to register information about the Web services they offer so that other businesses can find them. Web services are the next step in the evolution of the World Wide Web (WWW) and allow programmable elements to be placed on Web sites where others can access distributed behaviors. UDDI registries are used to promote and discover these distributed Web services.

The Universal Description, Discovery and Integration (UDDI) specifications define a way to publish and discover information about Web services. The term "Web service" describes specific business functionality exposed by a company, usually through an Internet connection, for the purpose of providing a way for another company or software program to use the service. Web services are becoming the programmatic backbone for electronic commerce. For example, one company calls another's service to send a purchase order directly via an Internet connection.
Another example is a service that calculates the cost of shipping a package of a certain size or weight, so many miles via a specific carrier.

6.5.1 UDDI Business Registrations and the UDDI Business Registry

The core component of the UDDI project is the UDDI business registration, an XML file used to describe a business entity and its Web services. Conceptually, the information provided in a UDDI business registration consists of three components: "white pages" including address, contact, and known identifiers; "yellow pages" including industrial categorizations based on standard taxonomies; and "green pages", the technical information about services that are exposed by the business. Green pages include references to specifications for Web services, as well as support for pointers to various file and URL based discovery mechanisms if required.

Using UDDI

The number of ways that companies are using the World Wide Web varies considerably. Many companies are starting to define ways to allow their internal applications to interact with the business systems at other companies using the emerging Web infrastructure. Left alone, each company invents a unique approach based on the experiences of designers, available technologies, and project budgets. The proliferations of integration approaches and unique solutions have spawned an entire sub-industry focused on bridging incompatible service layers within and across company boundaries. Recent work within the W3C starts to raise hopes that Extensible Markup Language (XML) will play a role in simplifying the exchange of business data between companies. Further, collaboration between computer industry giants and small companies alike have outlined a framework called SOAP that allows one program to invoke service interfaces across the Internet, without the need to share a common programming language or distributed object infrastructure. All of this is good news for companies feeling the cost pressures associated with electronic commerce because the foundations for common interoperability standards are being laid. Because of these foundation technologies and emerging standards, some of the intractable problems of the past are becoming easier to approach.
The Universal Description, Discovery and Integration (UDDI) specification describes a conceptual cloud of Web services and a programmatic interface that define a simple framework for describing any kind of Web service. The specification consists of several related documents and an XML schema that defines a SOAP-based programming protocol for registering and discovering Web services. These specifications were defined over a series of months by technicians and managers from several leading companies. Together, these companies have undertaken the task of building the first implementation of the UDDI services and running these services as a publicly accessible, multi-site partnership that shares all registered information.

Figure 6.4 shows the relationship between the specifications, the XML schema and the UDDI business registry cloud that provides “register once, published everywhere” access to information about Web services.

Using the UDDI discovery services, businesses individually register information about the Web services that they expose for use by other businesses. This information can be added to the UDDI business registry either via a Web site or by using tools that make use of the programmatic service interfaces described in the UDDI Programmer’s API Specification. The UDDI business registry is a logically centralized, physically distributed service with multiple root nodes that replicate data with each other on a regular basis. Once a business registers with a single instance of the business registry service, the data is automatically shared with other UDDI root
nodes and becomes freely available to anyone who needs to discover what Web services are exposed by a given business [51].

6.5.2 Business Discovery and UDDI

UDDI is designed to complement existing online marketplaces and search engines by providing them with standardized formats for programmatic business and service discovery. The ability to locate parties that can provide a specific product or service at a given price or within a specific geographic boundary in a given timeframe is not directly covered by the UDDI specifications. These kinds of advanced discovery features require further collaboration and design work between buyer and sellers. Instead, UDDI forms the basis for defining these services in a higher layer.

![Figure 6.5 UDDI Registries, aggregation, and Search Capabilities](image)

In Figure 6.5 we can see the relationship between the technical discovery layers defined by UDDI and the role of aggregation and specialized search capabilities that address business level searches. Currently, marketplaces and search portals fill this need, and can be integrated or populated using information published in the UDDI distributed registries.
Technical Overview

The Universal Description, Discovery and Integration (UDDI) specifications consist of an XML schema for SOAP messages, and a description of the UDDI API specification. Together, these form a base information model and interaction framework that provides the ability to publish information about a broad array of Web services [52].

6.5.3 Four Information Types

The core information model used by the UDDI registries is defined in an XML schema. XML was chosen because it offers a platform-neutral view of data and allows hierarchical relationships to be described in a natural way. The emerging XML schema standard was chosen because of its support for rich data types as well as its ability to easily describe and validate information based on information models represented in schemas.

The UDDI XML schema defines four core types of information that provide the kinds of information that a technical person would need to know in order to use a partner's Web services. These are: business information; service information, binding information; and information about specifications for services.

Business Information: the Business Element

Many partners will need to be able to locate information about your services and will have as starting information a small set of facts about your business. Technical staff, programmers or application programs themselves will know either your business name or perhaps your business name and some key identifiers, as well as optional categorization and contact information. The core XML elements for supporting publishing and discovering information about a business – the UDDI Business Registration -- are contained in a structure named “businessEntity”. This structure serves as the top-level information manager for all of the information about a particular set of information related to a business summit.

The overall businessEntity information includes support for “yellow pages” taxonomies so that searches can be performed to locate businesses who service a
particular industry or product category, or who are located within a specific geographic region.

Service Information: the Business services and Binding Template

Technical and business descriptions of Web services – the “green pages” data -- live within substructures of the businessEntity information. Two structures are defined: businessService and bindingTemplate. The businessService structure is a descriptive container that is used to group a series of related Web services related to either a business process or category of services. Examples of business processes that would include related Web service information include purchasing services, shipping services, and other high-level business processes. These businessService information sets can each be further categorized – allowing Web service descriptions to be segmented along combinations of industry, product and service or geographic category boundary [53].

6.5.4 UDDI Standard

The UDDI Project is an industry initiative that is working to enable businesses to quickly, easily, and dynamically find and transact with one another. UDDI will benefit businesses of all sizes by creating a global, platform-independent, open architecture for describing businesses and services, discovering those businesses and services, and integrating businesses using the Internet. Any kind of service can be registered in the UDDI Business Registry, such as manual services and electronic services, but the primary intent behind UDDI is to provide a global registry for Web Services.

UDDI enables a business to

1. Describe its business and its services: UDDI can give a business visibility on a global scale by providing a means for an organization to advertise its business and services in a global registry. The UDDI Business Registry provides a
place for a company to programmatically describe their services and business processes and their preferred methods for conducting business.

2. Discover other businesses that offer desired services: Today organizations find it difficult to find a business that offers services that best fit their needs. UDDI makes it possible for organizations to quickly discover the right business out of the millions that are currently online.

3. Integrate with these other businesses: Once an organization finds a potential business partner, there's no standard mechanism to figure out how to conduct electronic business with this partner. UDDI can simplify the effort of integrating disparate business processes, so that partners can quickly and easily begin trading.

The UDDI Business Registry contains information about businesses and the services they offer. The information is organized as follows:

- **Business Entity**: A business entity represents information about a business. Each business entity contains a unique identifier, the business name, a short description of the business, some basic contact information, a list of categories and identifiers that describe the business, and a URL pointing to more information about the business.

- **Business Service**: Associated with the business entity is a list of business services offered by the business entity. Each business service entry contains a business description of the service, a list of categories that describe the service, and a list of pointers to references and information related to the service.

- **Specification Pointers**: Associated with each business service entry is a list of binding templates that point to specifications and other technical information about the service. For example, a binding template might point to a URL that supplies information on how to invoke the service. The specification pointers also associate the service with a service type.
Service Types: A service type is defined by a tModel. Multiple businesses can offer the same type of service, as defined by the tModel. A tModel specifies information such as the tModel name, the name of the organization that published the tModel, a list of categories that describe the service type, and pointers to technical specifications for the service type such as interface definitions, message formats, message protocols, and security protocols [54].