CHAPTER I: WEB SERVICES BASICS

Topics covered:

What Are Web Services? Types of Web Services, Distributed computing infrastructure, overview of XML, SOAP, Building Web Services with JAX-WS, Registering and Discovering Web Services, Service Oriented Architecture, Web Services Development Life Cycle, Developing and consuming simple Web Services across platform.

What Are Web Services?:

A Web service is a self-describing, self-contained software module available via a network, such as the Internet, which completes tasks, solves problems, or conducts transactions on behalf of a user or application. Web services constitute a distributed computer infrastructure made up of many different interacting application modules trying to communicate over private or public networks (including the Internet and Web) to virtually form a single logical system.

A Web service can be: (i) a self-contained business task, such as a funds withdrawal or funds deposit service; (ii) a full-fledged business process, such as the automated purchasing of office supplies; (iii) an application, such as a life insurance application or demand forecasts and stock replenishment; or (iv) a service-enabled resource, such as access to a particular back-end database containing patient medical records. Web services can vary in function from simple requests (e.g., credit checking and authorization, pricing enquiries, inventory status checking, or a weather report) to complete business applications that access and combine information from multiple sources, such as an insurance brokering system, an insurance liability computation, an automated travel planner, or a package tracking system.

Web services address the problems of rigid implementations of predefined relationships and isolated services scattered across the Internet. The long-term goal of Web services technology is to enable distributed applications that can be dynamically assembled according to changing business needs, and customized based on device (such as personal computers, workstations, laptops, WAP-enabled cellular phones, personal digital assistants), network (such as cable, UMTS, XDSL, Bluetooth, etc.) and user access while enabling wide utilization of any given piece of business logic wherever it is needed. Once a Web service is deployed, other applications and Web services can discover and invoke it.

A more complete definition of Web services is given in section 1.3 after readers have familiarized themselves with the concept of software-as-a-service and understood the differences between Web services and Web-based applications.

Typical Web services scenarios

Web services efforts focus on reusing existing applications (including legacy code) for lightweight integration with other applications, often motivated by the desire for new forms of sharing of services across lines of business, or between business partners.

To better understand the mission of Web services, consider, as an example, an insurance company that decides to offer an on-line quoting Web service to its customers.

Rather than developing the entire application from scratch, this enterprise looks to supplement its home-grown applications with modules that perform industry standard functions.

Therefore, it may seamlessly link up with the Web service of another enterprise that specializes in insurance liability computations. The insurance liability Web service may present a quote form to the customer to collect customer information based on the type of the desired insurance. Subsequently, the Web service would present the customer with a quote including a premium estimate. If the customer selected to buy that particular insurance policy, the system would take the customer's payment information and run it through a payment Web service offered by yet another company (service provider). This payment Web service will ultimately return billing information to the customer and to the originating company.

Enterprise applications, such as the insurance quoting Web service, are among the most likely candidate applications that can benefit from the use of Web services technologies.
Enterprise applications cover a wide spectrum of Web services scenarios including interactions between the departments within an organization as well as interactions between business partners. Enterprises can typically use a single Web service to accomplish a specific business task, such as billing or inventory control, or they may compose several Web services together to create a distributed enterprise application such as customized ordering, customer support, procurement, and logistical support. These enterprise application scenarios require both the reuse and integration of existing back-end systems within an enterprise, which are the target of enterprise application integration or EAI, see section 2.9.
More sophisticated enterprise applications may focus on e-business, or cross-enterprise interactions involving transacting business partners over the Internet (also covered in section 2.9), which is typical of the way in which large companies procure, manufacture, sell, and distribute products .

Case study: Order management process

In the following we introduce an order management process that is crafted in such a way as to interweave Web services principles and concepts that will be covered in later chapters in this book. The case study is built around a simple order management scenario that manages a purchase order submitted by a customer to a specific supplier. An order management solution supports an end-to-end order fulfillment process that can be represented by means of a complex set of interacting Web services requiring a lot of synchronization and coordination. Such Web services may configure and order customized products; provide customers with accurate, real-time information on global product availability; provide interactive pricing options; offer real-time status enquiry; perform inventory and warehouse management; and so forth.
In the simple purchase order scenario examined in this book the customer or buying organization initially creates a purchase order and sends the request to fulfill that order to a supplier. The supplier offers a purchase order Web service that receives the purchase order and responds with either acceptance or rejection based on a number of criteria, including availability of the goods and the credit of the customer.

Figure 1.1 shows how such a purchase order process can be developed at the supplier's side in terms of interacting Web services that involve purchase orders, credit checks, automated billing, stock updates, and shipping originating from various

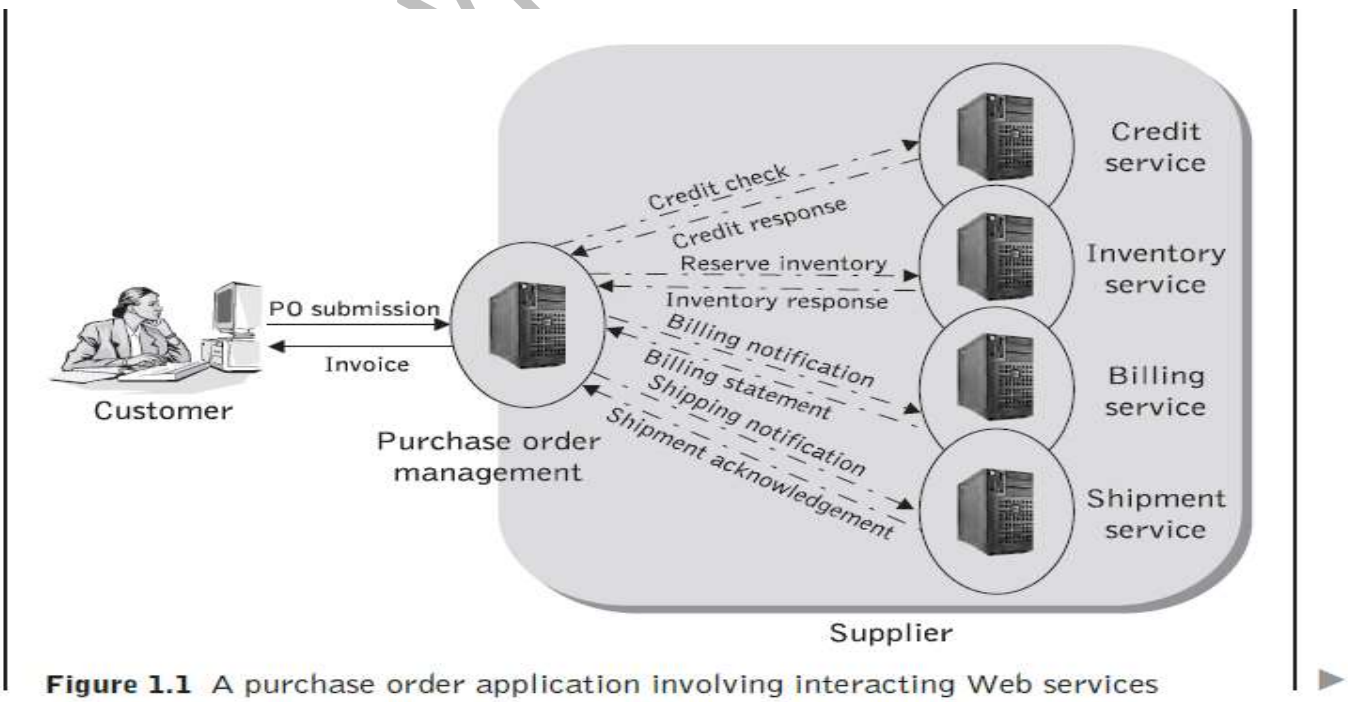

Figure 1.1 A purchase order application involving interacting Web services

Figure 1.1 A purchase order application involving interacting Web services

1.2 The concept of software as a service

Web services are very different from Web pages that also provide access to applications across the Internet and across organizational boundaries. Web pages are targeted at human users, whereas Web services are developed for access by humans as well as automated applications. As terminology is often used very loosely, it is easy to confuse someone by describing a "service" as a Web service when it is in fact not. Consequently, it is useful to examine first the concept of software-as-a-service on which Web services technology builds and then compare Web services to Web server-based functionality.

The concept of *software-as-a-service* is revolutionary and appeared first with the applications service provider software model. *Application service providers* (ASPs) are service providers whose offerings are packaged to create turnkey products. On receiving the purchase order from a customer, the purchase order process may initiate several tasks concurrently: checking the creditworthiness of the user, determining whether or not an ordered part is available in the product inventory, calculating the

final price for the order and billing the customer, selecting a shipper, and scheduling the production and shipment for the order. While some of the processing can proceed concurrently, there are synchronization dependencies between these tasks. For instance, the customer's creditworthiness must be ascertained first before accepting the order, the shipping price is required to finalize the price calculation, and the shipping date is required for the complete fulfillment schedule. When these tasks are completed successfully, invoice processing can proceed and the invoice will be sent to the customer.

A purchase order Web service, like the one we described earlier, can also handle more elaborate tasks. For instance, it could provide tracking and adjusting facilities that track and adjust purchase orders due to unexpected events such as the customer initiating a purchase order change or cancellation. These tasks involve a lot of coordination work and call for the use of reactive Web services. If a single event in the purchase order needs to change or is cancelled, the entire purchase order process can unravel instantly. Employing a collection of Web services that work together to adjust purchase orders for such situations creates an automated solution to this problem. In the case of a purchase order cancellation the purchase order Web service can automatically reserve a suitable replacement product and notify the billing and inventory services of the changes. When all of these Web services interactions have been completed and the new adjusted schedule is available, the purchase order Web service notifies the customer, sending the customer an updated invoice .The order management example will help set the stage for the following chapters.

Types of Web services:

Topologically, Web services can come in two flavors, see Figure 1.2. Informational, or type I, Web services, which support only simple request/response operations and always wait for a request; they process it and respond. Complex, or type II Web services implement some form of coordination between inbound and outbound operations. Each of these two models exhibits several important characteristics and is in turn subdivided in more specialized subcategories.

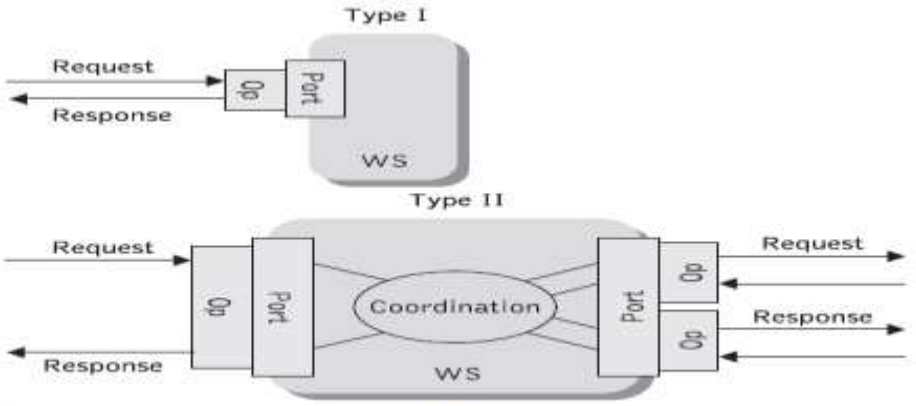

Figure 1.2 High-level view of informational and complex services

Simple or informational services

Informational services are services of relatively simple nature. They either provide access to content interacting with an end user by means of simple request/response sequences, or alternatively may expose back-end business applications to other applications. Web services that typically expose the business functionality of the applications and components that underlie them are known as *programmatic services*. For instance, they may expose function calls, typically written in programming languages such as Java/EJB, Visual Basic, or C++. The exposed programmatic simple services perform a request/response type of business task and can be viewed as "atomic" (or singular) operations. Applications access these function calls by executing a Web service through a standard programmatic interface specified in the Web Services Description Language or WSDL (see Chapter 5).
Informational services can be subdivided into three subcategories according to the business problems they solve:
1. **Pure *content services***, which give programmatic access to content such as weather report information, simple financial information, stock quote information, design information, news items, and so on.

2. ***Simple trading services***, which are more complicated forms of informational services that can provide a seamless aggregation of information across disparate systems and information sources, including back-end systems, giving programmatic access to a business information system so that the requestor can make informed decisions.

Such service requests may have complicated realizations. Consider, for example, "pure" business services, such as logistic services, where automated services are the actual front-ends to fairly complex physical organizational information systems.

3. ***Information syndication services***, which are value-added information Web services that purport to "plug into" commerce sites of various types, such as e-marketplaces,

Figure 1.2 High-level view of informational and complex services

or sell-sites. Generally speaking, these services are offered by a third party and run the whole range from commerce-enabling services, such as logistics, payment, fulfillment, and tracking services, to other value-added commerce services, such as rating services. Typical examples of syndicated services might include reservation services on a travel site or rate quote services on an insurance site.

Informational services are singular in nature in that they perform a complete unit of work that leaves its underlying datastores in a consistent state. However, they are not transactional in nature (although their back-end realizations may be). An informational service does not keep any memory of what happens to it between requests. In that respect this type of service is known as a *stateless Web service*.

Informational and simple trading services require support by the three evolving standards:

(i) communication protocol (Simple Object Access Protocol), (ii) service description

Web Service Description Language (WSDL), (iii) service publication and discovery

(Universal Description, Discovery, and Integration infrastructure).

Complex services or business processes

Enterprises can use a singular (discrete) service to accomplish a specific business task, such as billing or inventory control. However, for enterprises to obtain the full benefit of Web services, business process and transactional-like Web services functionality is required that is well beyond that found in informational Web services. When enterprises need to compose several services together to create a business process such as customized ordering, customer support, procurement, and logistical support, they need to use complex

Web services. Complex (or *composite*) services typically involve the assembly and invocation of many pre-existing services possibly found in diverse enterprises to complete a multi-step business interaction. Consider for instance a supply-chain application involving order taking, stocking orders, sourcing, inventory control, financials, and logistics. Numerous document exchanges will occur in this process including requests for quotes, returned quotes, purchase order requests, purchase order confirmations, delivery information, and so on. Long-running transactions and asynchronous messaging will also occur, and business "conversation" and even negotiations may occur before the final agreements are reached.

This functionality is a typical characteristic of business processes (or complex services).

Complex Web services can in turn be categorized according to the way that they compose simple services. Some complex Web services compose simple services that exhibit programmatic behavior whereas others compose services that exhibit mainly interactive behavior where input has to be supplied by the user. This makes it natural to distinguish between the following two types of complex Web services:

1. ***Complex services that compose programmatic Web services****:* The clients of these Web services can assemble them to build complex services. An example typical of a simple service exhibiting programmatic behavior could be an inventory checking service that comprises part of an inventory management process.

2. ***Complex services that compose interactive Web services****:* These services expose the functionality of a Web application's presentation (browser) layer. They frequently expose a multi-step Web application behavior that combines a Web server, an application server, and underlying database systems and typically deliver the application directly to a browser and eventually to a human user for interaction. Clients of these Web services can incorporate interactive business processes into their Web applications, presenting integrated (aggregated) applications from external service providers. Obviously interactive services can be combined with programmatic services thus delivering business processes that combine typical business logic functionality with Web browser interactivity.

Complex services exhibit *coarse-grained* functionality and are stateful. A *stateful Web service* maintains some state between different operation invocations issued by the same or different Web service clients.

The complex Web services standards are still evolving and are converging on the communication protocol (Simple Object Access Protocol), WSDL, Universal Description, Discovery, and Integration infrastructure, WS-MetaDataExchange (which allows service endpoints to provide metadata information to requestors and support the bootstrapping of Web service interactions) and the Web services Business Process Execution Language or BPEL.

Functional and non-functional properties

Services are described in terms of a description language. A service description has two major interrelated components: its functional and non-functional characteristics. The *functional description* details the operational characteristics that define the overall behavior of the service, i.e., defines details of how the service is invoked, the location where it is invoked, and so on. This description focuses on details regarding the syntax of messages and how to configure the network protocols to deliver messages. The *non-functional*

description concentrates on service quality attributes, such as service metering and cost, performance metrics, e.g., response time or accuracy, security attributes, authorization, authentication, (transactional) integrity, reliability, scalability, and availability. Nonfunctional descriptions force the service requestor's run-time environment to include, for instance, SOAP headers that specify non-functional requirements that may influence which service provider a service requestor may choose. An example of this may be a security policy statement (refer to Chapter 12 for details regarding service security policies).
Functional properties of services are examined in Chapter 5, while non-functional ones are examined in section 1.8 of this chapter.

State properties

Services could be stateless or stateful. If services can be invoked repeatedly without

having to maintain context or state they are called stateless, while services that may require

their context to be preserved from one invocation to the next are called stateful. The services access protocol is always connectionless. A *connectionless protocol* means that the protocol has no concept of a job or a session and does not make any assumptions about eventual delivery (see section 2.1.1.2).

A Web service in its simplest form, e.g., an informational weather report service, does not keep any "memory" of what happens to it between requests. Such Web services are known as *stateless* Web services. The concept of statelessness means that each time a consumer interacts with a Web service, an action is performed. After the results of the service invocation have been returned, the action is finished. There is no assumption that subsequent invocations are associated with prior ones. Consequently, all the information

required to perform the service is either passed with the request message or can be retrieved from a data repository based on some information provided with the request.

In contrast to a stateless Web service, a *stateful* Web service maintains some state between different operation invocations issued by the same or different Web service clients. If a particular "session" or "conversation" involves composite Web services then transient data between operation invocations is stateful. A message sent to a Web service stateful instance would be interpreted in relation to that instance-specific state. Typically, business processes specify stateful interactions involving the exchange of messages

between partners, where the state of a business process includes the messages that are exchanged as well as intermediate data used in business logic and in composing messages sent to partners. Consider, for instance, an order management application, where a seller's business process might offer a service that begins an interaction by accepting a purchase order through an input message, and then returns an acknowledgement to the customer if the order can be fulfilled. The application might later send further messages to the customer, such as shipping notices and invoices. The seller's business process must "remember" the state of each such purchase order interaction separately from other similar interactions. This is necessary when a customer has many purchase processes with the same seller that are executed simultaneously.

Loose coupling

Web services interact with one another dynamically and use Internet standard technologies, making it possible to build bridges between systems that otherwise would require extensive development efforts. The term *coupling* indicates the degree of dependency any two systems have on each other.
In a *tightly coupled* exchange, applications need to know how their partner applications behave. They also need to know intimate details of how their partner requires to be communicated with – the number of methods it exposes and the details of the parameters that each method accepts, and the type of results it returns. In addition, tightly coupled applications need to know the location of the applications with which they work (and this implies a certain "security" guarantee). Traditional application design depends upon a tight coupling of all subsidiary elements, often running in the same process. Consequently, a key design

pattern in tightly coupled environments is synchronous interactions. Tight coupling requires that the interfaces between the different components of an application are tightly interrelated in function and form, thus making them brittle when any form of change or replacement is required to parts or the whole application. It is often quite difficult and cumbersome to build tightly coupled applications because when the number of applications and services increases, the number of interfaces that need to be created and maintained quickly becomes unwieldy. This requires that a lot of time needs to be spent in defining the connections and relationships between any two cooperating applications.

As opposed to tight coupling principles that require agreement and shared context between communicating systems as well as sensitivity to change, loose coupling allows systems to connect and interact more freely (possibly across the Internet). In a loosely coupled exchange, applications need not know how their partner applications behave or are implemented. The benefit of a loosely coupled system lies in its agility and the ability to survive evolutionary changes in the structure and implementation of the internals of each service, which make up the whole application. Systems that are loosely coupled in time have an asynchronous or event-driven model rather than a synchronous model of interaction (see section 2.4). Loose coupling of applications provides a level of flexibility and interoperability that cannot be matched using traditional approaches to building highly integrated, cross-platform, program-to-program communications environments.

With the Web services approach, the binding from a service requestor to a service provider is loosely coupled. This means that the service requestor has no knowledge of the technical details of the provider's implementation, such as the programming language, deployment platform, and so forth. The service requestor typically invokes operations by way of messages – a request message and the response – rather than through the use of application programming interfaces or file formats.

Table 1.1 Tight versus loose coupling

	Tight coupling	*Loose coupling*
Interaction pattern	Synchronous	Asynchronous
Messaging style	RPC style	Document style
Message path	Hard coded	Routed
Underlying platform	Homogeneous	Heterogeneous
Binding protocol	Static	Dynamic – late binding
Objective	Reuse	Flexibility, broad applicability

Table 1.1 summarizes the differences between loose and tight coupling.

Service granularity

Web services may vary in function from simple requests to complex systems that access and combine information from multiple sources. Even simple service requests may have complicated realizations. Simple services are discrete in nature, exhibit normally a request/ reply mode of operation, and are of fine granularity, i.e., they are atomic in nature. In contrast, complex services are coarse grained, for instance the SubmitPurchaseOrder process, and involve interactions with other services and possibly end users in a single or multiple sessions. Coarse-grained communication implies larger and richer data structures, i.e., those supported by XML schema, and enables looser coupling, which in turn enables asynchronous communication where the information exchange is the minimal required to complete the task.

Synchronicity

We may distinguish between two programming styles for services: synchronous or remote procedure call (RPC) style versus asynchronous or message (document) style, see sections 2.4 and 2.5:

***Synchronous services**:* Clients of synchronous services express their request as a method call with a set of arguments, which returns a response containing a return value. This implies that when a client sends a request message, it expects a response message before continuing with its computation. This makes the whole invocation an all-or-nothing proposition. If one operation is unable to complete for any reason, all other dependent operations will fail. Because of this type of bilateral communication between the client and service, RPC-style services require a tightly coupled model of communication between the client and service provider. RPCstyle Web services are normally used when an application exhibits the following characteristics:

- ◆ The client invoking the service requires an immediate response.
- ◆ The client and service work in a back-and-forth conversational way.

Examples of typical simple synchronous services with an RPC-style include returning the current price for a given stock; providing the current weather conditions in a particular location; or checking the credit rating of a potential trading partner prior to the completion of a business transaction.

Asynchronous services: Asynchronous services are document-style or message driven services. When a client invokes a message-style service, the client typically sends it an entire document, such as a purchase order, rather than a discrete set of parameters. The service accepts the entire document, it processes it and may or may not return a result message. A client that invokes an asynchronous service does not need to wait for a response before it continues with the remainder of its application. The response from the service, if any, can appear hours or even days later.

Asynchronous interactions (messaging) are a key design pattern in loosely coupled environments. Messaging enables a loosely coupled environment in which an application does not need to know the intimate details of how to reach and interface with other applications. This allows a communication operation between any two processes to be a self-contained, standalone unit of work. Document style Web services are normally used when an application exhibits the following characteristics:

- ◆ The client does not require (or expect) an immediate response.
- ◆ The service is document oriented (the client typically sends an entire document,

e.g., a purchase order, rather discrete parameters).

Examples of document-style Web services include processing a purchase order; responding to a request for quote order from a customer; or responding to an order placement by a particular customer. In all these cases, the client sends an entire document, such as a purchase order, to the Web service and assumes that the Web service is processing it in some way, but the client does not require an immediate answer.

Well-definedness

The service interaction must be well defined. WSDL allows applications to describe to other applications the rules for interfacing and interacting. WSDL provides a uniform mechanism for describing abstract service interfaces and specific protocol bindings that support the service. As such it is a widely supported way of describing the details required by a service requestor for binding to a service provider. The service descriptions focus on how operations interact with a service; how messages invoke operations; details of constructing such messages; and details of where to send messages for processing, i.e., determining service access points.

WSDL does not include any technology details of the implementation of a Web service.

The service requestor neither knows nor cares whether the service is implemented in a programming language such as Java, C#, C, and so on. As long as the Web service can handle SOAP messages, it does not matter which platform the service is developed and implemented on.

These issues are described further in section 1.5 where we distinguish between the service interface and service implementation sections of a service definition.

Service usage context

In addition to the types and characteristics of Web services mentioned above it also useful to divide information services into different categories based on the Web service requestor's perspective. Here, we may distinguish between replaceable and mission-critical services.

A *replaceable Web service* is a service provided by several providers and replacing one provider with another does not affect application functionality as long as the service interfaces are identical. The productivity is not impacted severely if the service is unavailable for a short period of time as another provider may be chosen. A discrete (enumerated) discovery process involving several alternative possibilities may be pursued here. An example of this kind of service is a car rental service. Here we may pursue different car rental agencies, e.g., Avis, Hertz, Budget, and choose the first service response that arrives and satisfies our needs. This type of service is usually well integrated with the consumer processes (e.g., rent-a-car activity) and does not exchange any critical business data.

A *mission-critical Web service* is a service possibly provided by a single specific provider, which if replaced compromises severely the functionality of an entire application.

If the service is unavailable for a period of time it would drastically reduce the productivity of the application. This type of service would typically hold some critical business data and be integrated at the process level.
