This presentation will focus on the general principals and technologies that are used in Web Services.

Goals

- Provide a description of Web Services base technologies
  - Web Services Description Language (WSDL)
  - Simple Object Access Protocol (SOAP)
  - Universal Description, Discovery and Integration (UDDI)

This presentation will cover the core concepts that make up Web Services. It will begin by discussing the basic technologies, like XML, SOAP and WSDL that Web Services are based on.

Agenda

- Overview
- Web Services – Core technologies overview
  - Web Services Description Language (WSDL)
  - Simple Object Access Protocol (SOAP)
  - Universal Description, Discovery and Integration (UDDI)

This presentations begins by discussing the concepts behind Web Services. Next it covers the core technologies that Web Services utilize; WSDL, SOAP and UDDI.

Section

Overview:

Web Services

Now for a brief overview of Web Services.

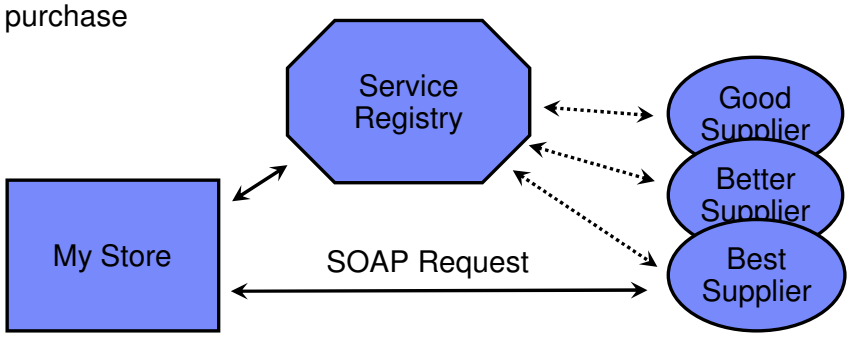
Web Services: Overview

- Web Services are modular applications that can be described, and invoked over the internet
  - Standards based, interoperable, self-contained
  - Ability to publish and search for specific services
- Web Services can be new applications or just wrapped around existing legacy systems to make them internet-enabled
- Services can rely on other services to achieve the goals and participate in a Business Process
- Web Services are called using XML-based SOAP requests
  - Allows connecting heterogeneous systems

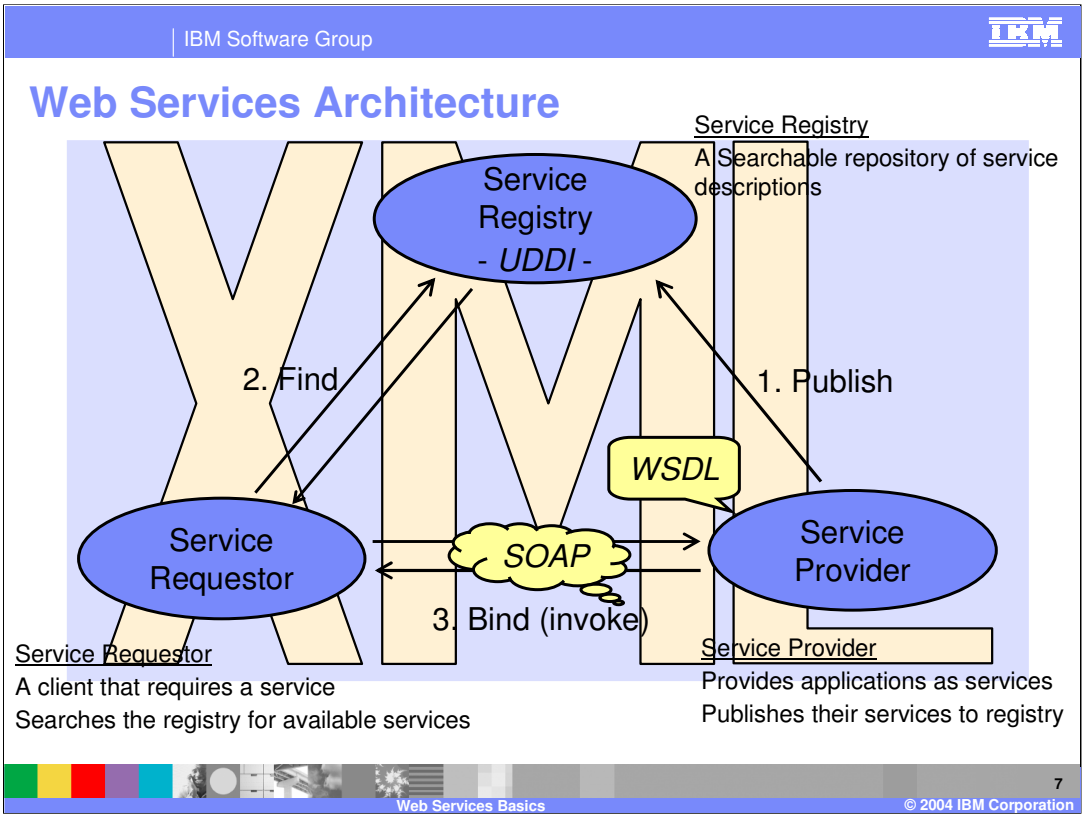
Web Services offer the ability to invoke methods and applications over the internet, using a standardized approach to describe and call methods using XML. XML is platform independent and allows Web Services to communicate in a heterogeneous environment. Web Services are described using the Web Services Description Language or WSDL, which describes the various components of a Web Service in an XML format that can be used to build a client to access that service. Web Services can be used to wrap existing applications as a way to include them into a Service Oriented Architecture or process flow. Web Services can be combined to create a Business Process.

Web Services: Example

- In a typical Web Services scenario, a business application sends a request to a service at a given URL using the SOAP protocol over HTTP
  - The service receives the request, processes it, and returns a response
- A common example of a Web Service is that of a store purchasing service, in which the requestor queries a registry for available suppliers, picks a supplier based on some criteria, then sends a request to that supplier for some goods, the supplier would then send a response for the purchase



In a basic example of a Web Service, an application sends a SOAP request to a target service using HTTP. The target service will receive and process the message, possibly returning a result to the requestor. A common example of a Web Service focuses on their use in a Business to Business transaction. A store that needs to make a purchase can query a registry of available Web Services. The registry holds information on Web Services, how to call them, and quality of service information. Based on that information the store application can make a decision on which Web Service to use, and then send a request to that target Web Service. The Web Service would consume the request, possibly returning a result to the store application.

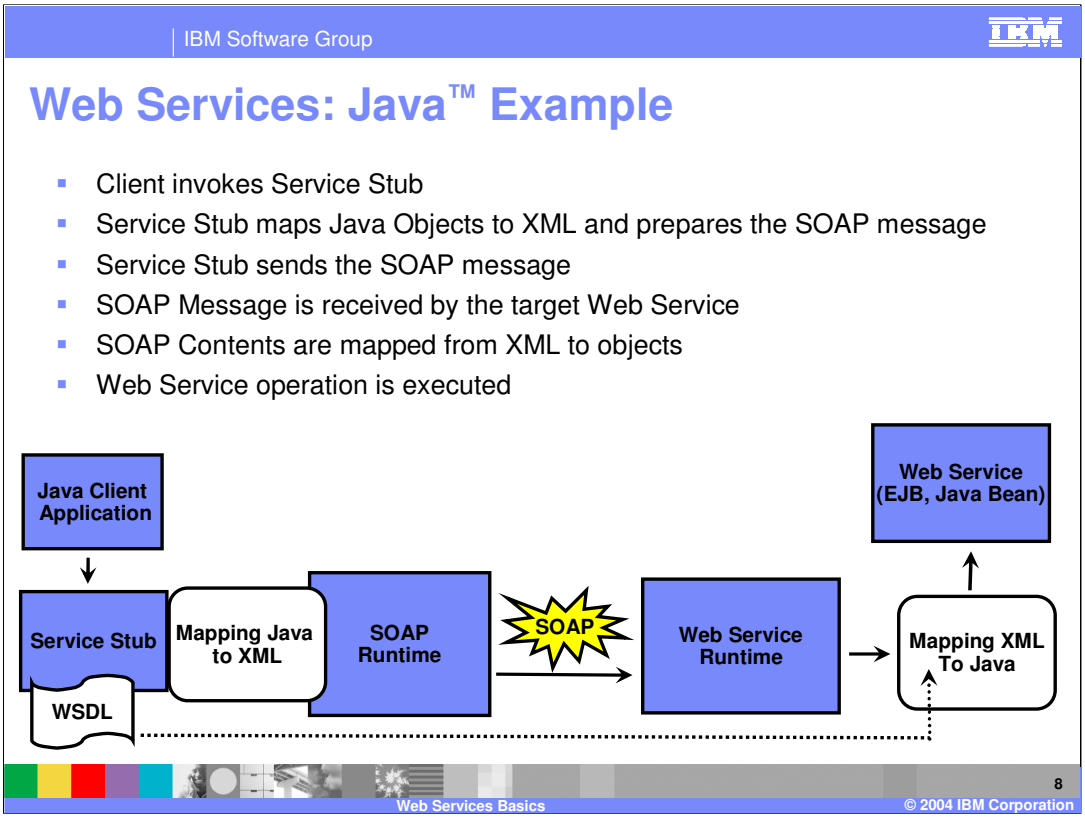


This is the big picture explanation of Web Services. There are three roles in the triangle: the Service Provider, the Service Registry, and the Service Requestor. Web Services are about creating interfaces. They achieve this by creating a standard description of the services being provided by the Service Provider in a WSDL document. In particular, the WSDL document tells the client exactly where the service is being hosted, what are the operations that can be called, what are the input parameters, and what to expect back as a response from the service provider.

The service provider can exist on the internet or an intranet and it may choose to publish its services on a registry, usually either a UDDI registry or an ebXML registry. This can be a public registry that anybody can access, or a private registry, accessible from within the Intranet.

Ultimately – whether through a UDDI registry or via other means, the service requester is interested in getting hold of the WSDL document that describes the service to be accessed. Using the WSDL document, the requester can generate a client that consumes the service.

The communications among these components use XML which ensures interoperability and support.



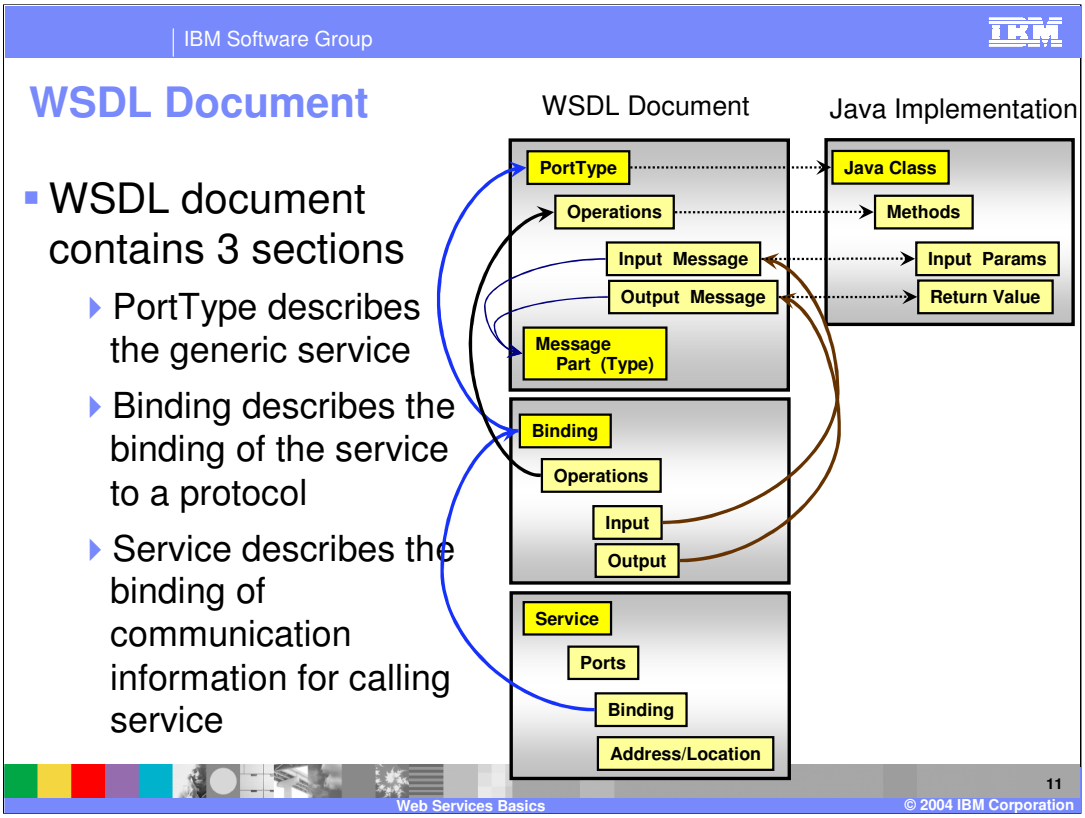
Here is a more technical example of how a Web Service is called. A Java client application invokes a Web Service proxy described in a WSDL document. The WSDL document is used to generate a Service Stub, which is a J2EE artifact that is used to call the target Web Service. The service stub will prepare the SOAP message to be sent and map the Java types from the client application to XML types that can be sent in the SOAP message. The service stub will then send the SOAP message to the target service using HTTP as a transport protocol. The SOAP message will be received by the target Web Service, where the Web Service runtime will consume the message. This will map the XML in the SOAP message back to Java types, and call the method being described by the Web Service.

Now for an explanation of the Web Services Description Language or WSDL.

What is WSDL?

- Provides an industry standard way to describe Web Services
- To describe services for calling, must be readable and compatible with different technologies
- WSDL is XML based and extensible

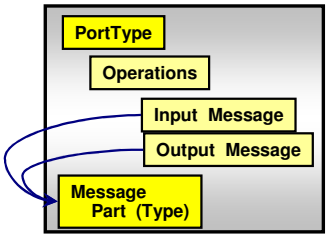
The Web Services Description Language or WSDL is a crucial part of a Service-Oriented Architecture, as it provides a standard way to describe services. Using WSDL, you can describe the interface for a Web Service once and then provide different bindings and services for different clients based upon their platform or language. This provides support for communication in mixed environments. This implementation also has a consistent client programming model. This leads to another very important concept of Service-Oriented Architectures. The adoption of an SOA has greater client requirements than server requirements. It is up to the client to be able to map their objects into the generic XML format which is described in the Service interface and to invoke the service based upon the binding and service information provided.



A WSDL file is made up of 3 sections; a port type, bindings, and services. A WSDL port type can be associated with multiple bindings. And each binding can in turn be associated with multiple services. So a single port type can be associated with any number of services. These sections contain various XML sections. Types provide a container for data type definitions using some type system, such as an XML schema. Messages are an abstract type definition of the data being communicated. A message can have one or more typed parts. The port type is an abstract set of one or more operations supported by one or more ports. Operation is an abstract definition of an action supported by the service that defines the input and output message, as well as option fault messages. The binding is a concrete protocol and data format specifications for a particular port type. The binding information contains the protocol name, the invocation style, a service ID, and the encoding for each operation. A service is represented as a collection of related ports. Each port is a single endpoint, which is defined as an aggregation of a binding and a network address.

WSDL: Interface Information

```
<wsdl:message name="getQuoteRequest">
    <wsdl:part element="intf:getQuote"
          name="parameters"/>
</wsdl:message>
<wsdl:message name="getQuoteResponse">
    <wsdl:part element="intf:getQuoteResponse"
          name="parameters"/>
  </wsdl:message>
<wsdl:portType name="StockQuote">
    <wsdl:operation name="getQuote">
      <wsdl:input message="intf:getQuoteRequest"
          name="getQuoteRequest"/>
      <wsdl:output message="intf:getQuoteResponse"
          name="getQuoteResponse"/>
    </wsdl:operation>
  </wsdl:portType>
```



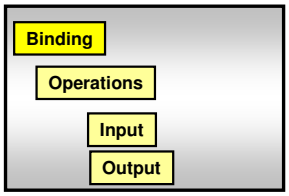
Here is an example of the interface information for a Web Service. The example is a stock quote service which accepts a message (getQuoteRequest) and returns a response (getQuoteResponse). The getQuoteRequest message contains a single part, called ticker, of type string. The getQuoteResponse message contains a single part, called result, of type float. The types of the parts are based upon a generic XML format. The messages and operations are also generic with no specifics about the implementation in the interface information.

The client application or the Service Requestor would call the generic operation getQuote on the portType StockQuote. As long as the client application can format the information for the parts into the appropriate XML format, the messages can be completed in preparation for calling the service.

While this example is very simple, it showcases all the major points of a service interface information.

WSDL: Binding Information

```
<wsdl:binding name="StockQuoteSoapBinding" type="intf:StockQuote">
    <wsdlsoap:binding style="document"
  transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="getQuote">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input name="getQuoteRequest">
        <wsdlsoap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="getQuoteResponse">
        <wsdlsoap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
```

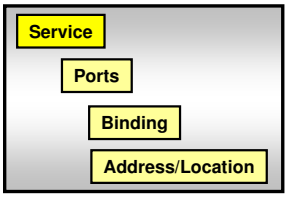


At execution time, the generic calls made by the service requestor are converted to a format which is understandable to the service implementation. The binding information in the WSDL has the specifics on what method, function, or transaction a generic operation will map to. In this example the getQuote operation maps to a SOAP operation of the same name. If the implementation operation had a different name, it would be listed in the SOAP Operation tag. For the input and output messages, the mapping of the parts to a specific type uses standard SOAP encoding for converting from the generic type to the SOAP implementation.

Notice that in this example the conversion is mapping XML objects to SOAP objects. While this is expected for Web Services, when you talk about a larger Services concept, the conversion can be performed from XML objects to any type of objects which can be supported by the runtime environment.

WSDL: Service Information

```
<wsdl:service name="StockQuoteService">
    <wsdl:port binding="intf:StockQuoteSoapBinding" name="StockQuote">
      <wsdlsoap:address
  location="http://localhost:9080/TestWeb/services/StockQuote"/>
    </wsdl:port>
  </wsdl:service>
```



Besides the binding information, the service information is also used at execution time. The service information is used when establishing a connection to the service. The service information in the WSDL has the location for the service and contains the information on what form of communication should be used when calling the service. In this example, for a request to the StockQuoteService location and specifically the StockQuotePort service, a SOAP connection will be established with the URL location listed.

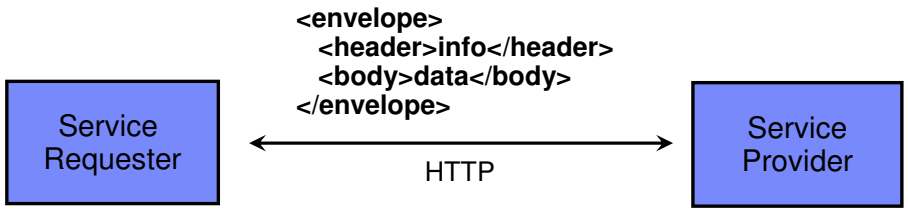
Section

SOAP
(Simple Object Access Protocol)

Now for a description of the Simple Object Access Protocol or SOAP.

Web Services Protocol: SOAP

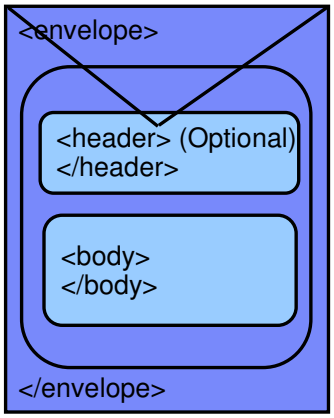
- Network-neutral service access protocol to transfer XML messages between Service Roles
- SOAP provides a simple method to communicate between heterogeneous systems



SOAP is the XML format that Web Services use to communicate. Soap is a network neutral protocol that transfers XML messages between Service Requestors and Service Providers. Because SOAP uses a platform independent means to send information (XML), it allows a simple means to communicate in a heterogeneous environment. While SOAP messages are most often sent using HTTP, SOAP is not limited to any protocol, and can be sent using JMS and RMI-IIOP among other options. HTTP simply provides the simplest and most expedient protocol for ending SOAP messages.

SOAP Message Format

- A SOAP message is an envelope containing 0 or more headers and exactly 1 body
  - The envelope provides a container for control information, the addressee of the message, and the message itself
  - Headers contain optional control information and provide extensibility for the SOAP message structure
  - The body contains message identification and its parameters



A SOAP message is formatted within the concept of a SOAP envelope. The envelope provides a container for control information, the addressee of the message, and the message itself.

The SOAP envelope contains zero or more headers containing optional control information about the message. The header references who has to do what with the message.

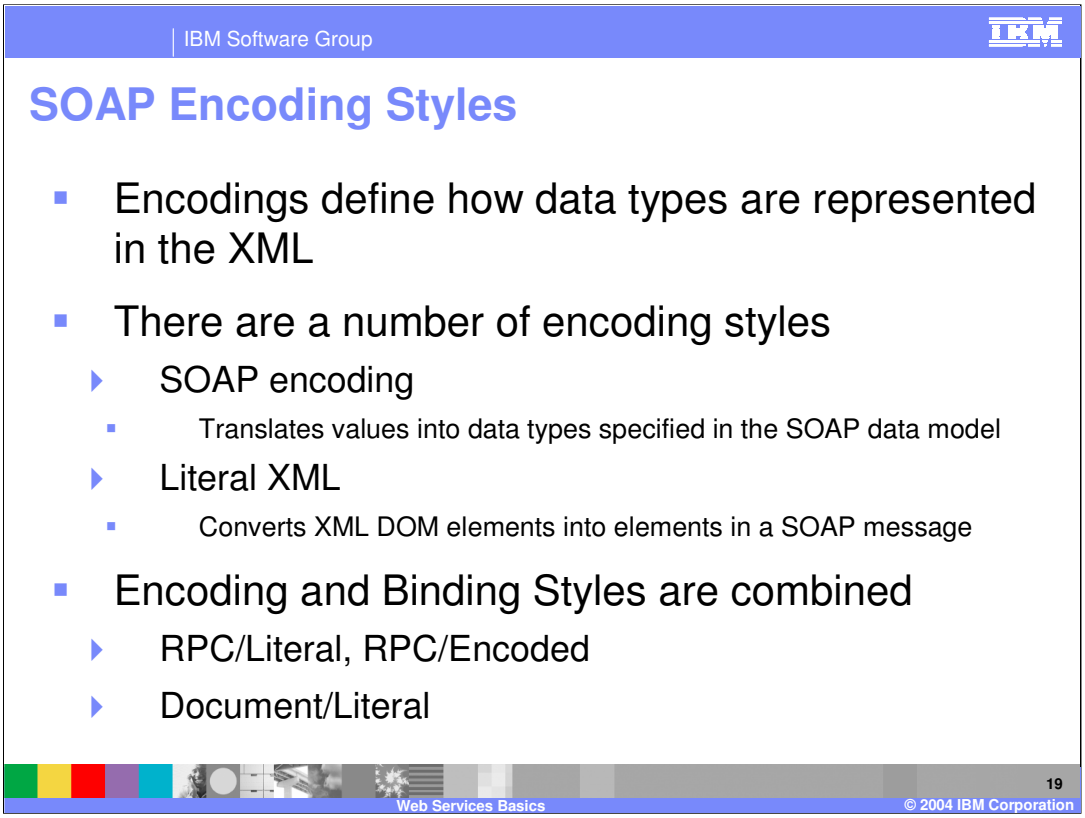
The SOAP envelope must also contain one body section. The body references what actually has to be done when invoking the service. The Body is encoded as an immediate child element of the SOAP envelope element.

SOAP Binding Styles

- Controls how the elements under the SOAP body are constructed
- SOAP supports 2 different communication styles
  - Document or Message Oriented
    - Places an XML element into the SOAP body
    - A lower level of abstraction requiring more programmatic work
  - Remote Procedure Call (RPC)
    - Adds extra elements to the XML to simulate a method call
    - A synchronous invocation of an operation returning a result

SOAP messages have 2 binding styles, that specify how XML elements within the SOAP body will be created. The two choices are Document or Remote Procedure Call (RPC). Document creates the simplest XML, and thus offers the best performance. RPC adds extra elements to simulate a method call.

The essence of the distinction lies in the above use of a *type* attribute versus an *element* attribute. With document style, you're placing into the SOAP body an XML element that is fully specified under the <wsdl:types> element. With RPC style, you're just specifying the type so the SOAP engine, using RPC rules, can make it a parameter, or a return value, and then place that parameter and any siblings inside an operation.



SOAP encodings are used to tell the SOAP runtime how to translate from data structures constructed in a specific programming language into SOAP XML, and back. This translation process is referred to as serialization and deserialization, it can also be called marshalling and unmarshalling. The most popular form of binding and encoding styles has become Document Literal. This combination has proven to provide the best performance, while creating the least complex SOAP message. This has led to this style being more generally accepted across vendors and developers.

Section

UDDI

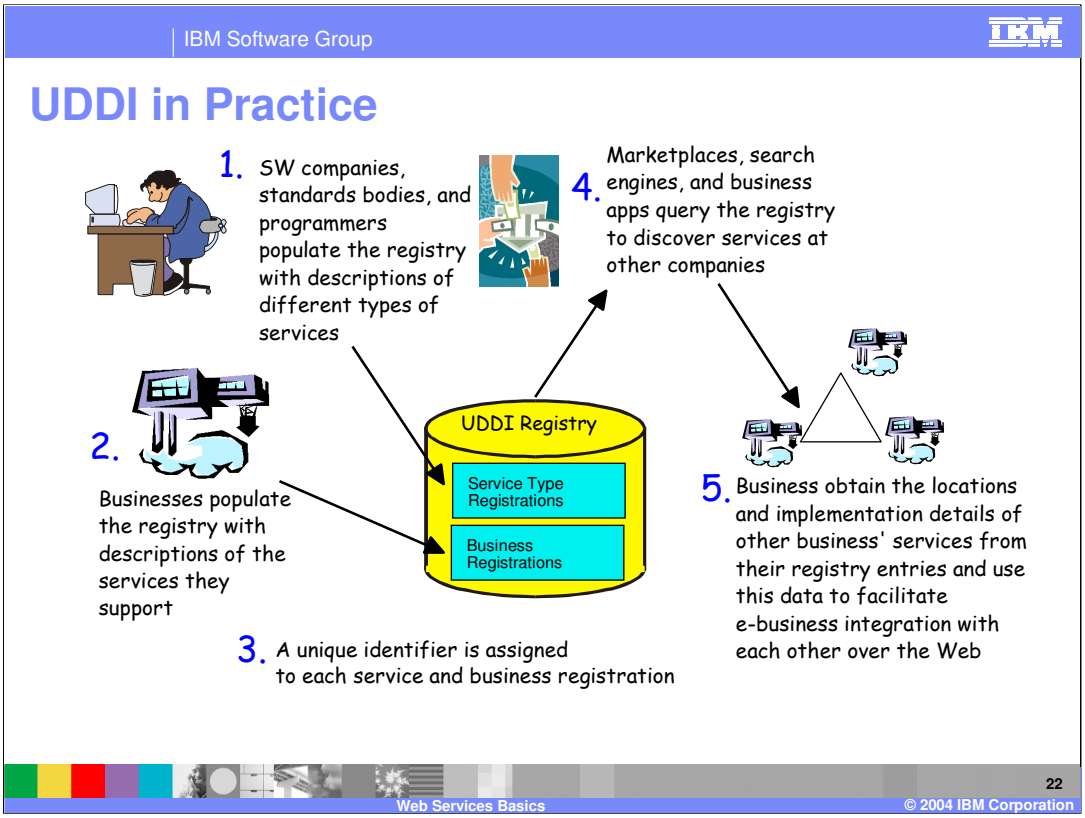
Universal Description, Discovery, and Integration

Next will be a discussion of the Universal Description, Discovery and Integration technology or UDDI. This is the primary Service Registry technology used by WebSphere.

Web Services Registry: UDDI

- UDDI registries are used to advertise and find services and businesses on the web
  - Allows a client to choose from a list of available services based on quality of service issues
- UDDI specification provides standardized formats for programmatic business and service discovery
- It has both a client and server side API to programmatically find, publish and modify registry information by Web Services clients and providers
  - The IBM UDDI version 3 Client for Java is the recommended way to access the UDDI APIs

Service registries are used to find services provided on the internet. They allow service requestors to query a single location for information about a number of services before making a decision and calling a specific service. The UDDI specification provides a standard method to both publish service information to a registry and to query a registry for service information. The UDDI technology does this by providing two sets of APIs, one for server side Web Service development, and the other for creating Web Service clients.



Here is an example of a UDDI registry at work. The first step is for businesses to populate the registry with descriptions of the services that they provide and support. The registry then assigns a unique identifier to each service description and business registration, storing the identifiers in the registry. Through various methods, Web Service clients can then query a registry for information on specific services. The information stored in the registry can then be used to create a client to connect to the target Web Service. In many ways the concept of a Service Registry is similar to the yellow pages in a telephone book. Service providers can opt-in and provide information on their services so that it is easier for consumers to find them.

Summary

- Discussed
  - Web Services Core Technologies
  - WSDL
  - SOAP
  - UDDI

This presentation introduced the basic concepts that make up Web Services. It explained the underlying technologies that Web Services use to communicate, as well as details on the uses for Web Services. Various references and materials are provided on the following slides to help further explain these topics.

Resources

- http://www.ibm.com/software/ad/studioappdev
- http://www.ibm.com/software/webservices
- http://www.ibm.com/developerworks/webservices
- http://www.alphaworks.ibm.com/webservices
- http://www.redbooks.ibm.com
  - SG246891 - WebSphere V5 Web Services Handbook
- http://www.eclipse.org

Template Revision: 11/02/2004 5:50 PM

Trademarks, Copyrights, and Disclaimers

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM	CICS	IMS	MQSeries	Tivoli
IBM(logo)	Cloudscape	Informix	OS/390	WebSphere
e(logo)business	DB2	iSeries	OS/400	xSeries
AIX	DB2 Universal Database	Lotus	pSeries	zSeries

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are registered trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel, ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a registered trademark of Linus Torvalds.

Other company, product and service names may be trademarks or service marks of others.

Product data has been reviewed for accuracy as of the date of initial publication. Product data is subject to change without notice. This document could include technical inaccuracies or typographical errors. IBM may make improvements and/or changes in the product(s) and/or program(s) described herein at any time without notice. Any statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business. Any reference to an IBM Program Product in this document is not intended to state or imply that only that program product may be used. Any functionally equivalent program, that does not infringe IBM's intellectual property rights, may be used instead.

Information is provided "AS IS" without warranty of any kind. THE INFORMATION PROVIDED IN THIS DOCUMENT IS DISTRIBUTED "AS IS" WITHOUT ANY WARRANTY, EITHER EXPRESS OR IMPLIED. IBM EXPRESSLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IBM shall have no responsibility to update this information. IBM products are warranted, if at all, according to the terms and conditions of the agreements (e.g., IBM Customer Agreement, Statement of Limited Warranty, International Program License Agreement, etc.) under which they are provided. Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. IBM makes no representations or warranties, express or implied, regarding non-IBM products and services.

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents or copyrights. Inquiries regarding patent or copyright licenses should be made, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.