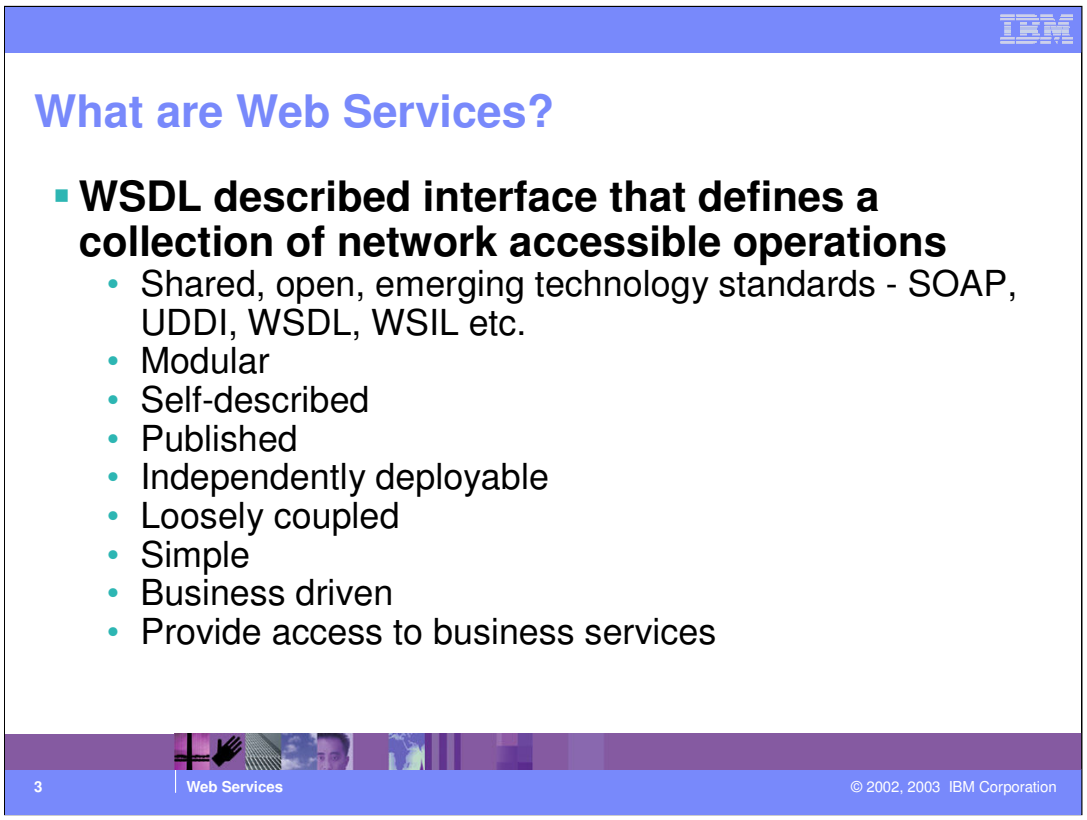

IBM
Web Services
WebSphere® Application Server Version 5
WebSphere. software
e business software
Updated 5/16/2003
© 2002, 2003 IBM Corporation

Objectives

- **Web Services Overview**
  - What are Web Services?
  - Roles and Functions of Web Services
- **Web Services Runtime**
  - Base Support
  - IBM WebSphere UDDI Registry (Private UDDI)
  - Web Services Gateway (WSGW)
- **Problem Determination**
- **Web Services Security**
- **Technical Previews**
- **Summary**



So, what is a Web Service? Definitions will vary, but here is ours. Simply, an "Interface that describes a collection of networked accessible operations". For developers, one way to think of Web Services is as a component model. A fairly coarse grained component model that gives us a building block from which to build solutions from. It's not a subroutine library or replacing Java collection classes or C++ libraries or anything else you use. It's function. Could be a stock quote service, could be something more complicated like a credit authorization, or perhaps an entire business process like loan authorization, a service composed of other nested services.

The other way to think about Web Services is how most people talk about them, that is the mechanism itself..., XML messages sent over the HTTP transport protocol.

Modular by design because inherently they are interface oriented.

Described using a service description language. WSDL (Web Services Description Language).

Published by making its description available to potential users.

Found by sending queries to that registry and receiving the binding details of the service(s) that fit the parameters of the query.

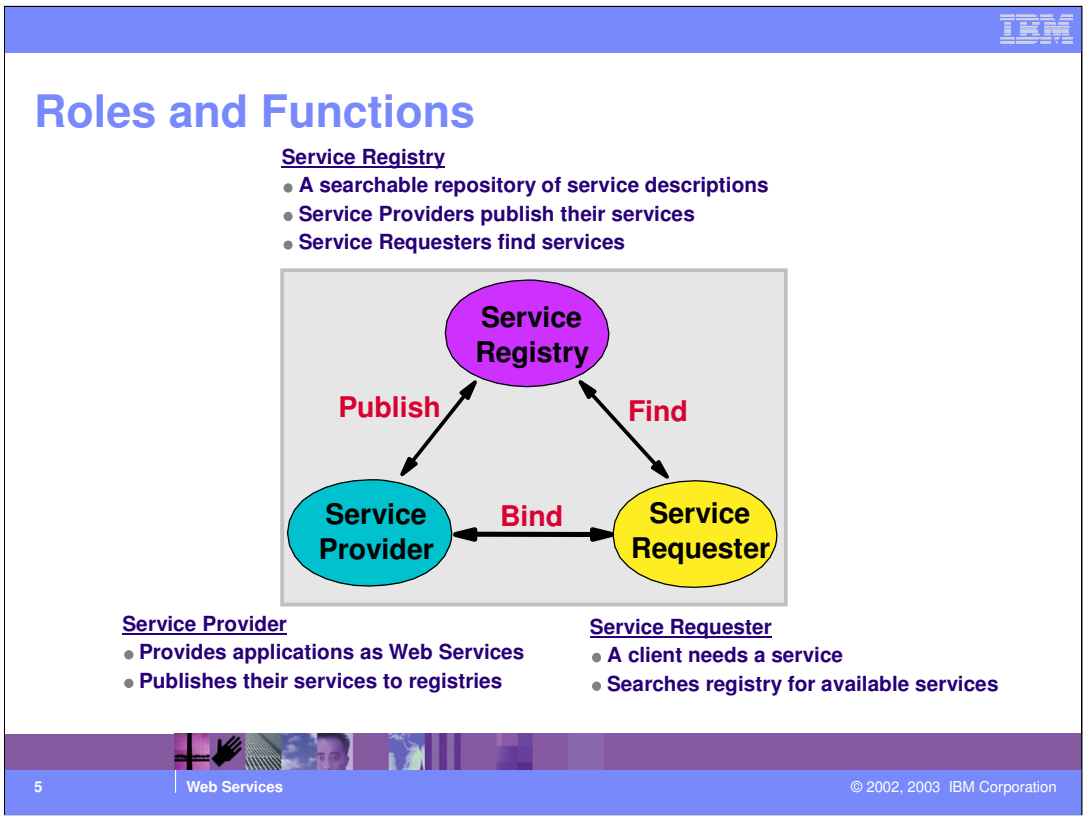
Bound by using the information contained in the service description to customize the connection to be created.

Invoked over a network by using the information contained in the binding details of the service description and possibly composed with other services into new services.

Business and Technical Benefits

- **Revenue benefits**
  - Improved relationships with customers/partners
  - Work force productivity
  - Innovation and reduced cycle times
  - Share processes without sharing technology
- **Cost reduction**
  - Deliver new business solutions faster
  - Better information flow and knowledge
  - Consistent infrastructure
  - Based on industry standards avoiding costly proprietary implementation
  - Standards hide underlying implementation allowing for more efficient debugging/testing
- **Unification of applications**
  - New ways of accessing old applications
  - Applications can be integrated without regard to implementation details
  - Applications can dynamically navigate, discover, and interact over the Internet (loosely coupled)
- **Organized information**
  - Targeted, more relevant

There are many benefits of Web Services.

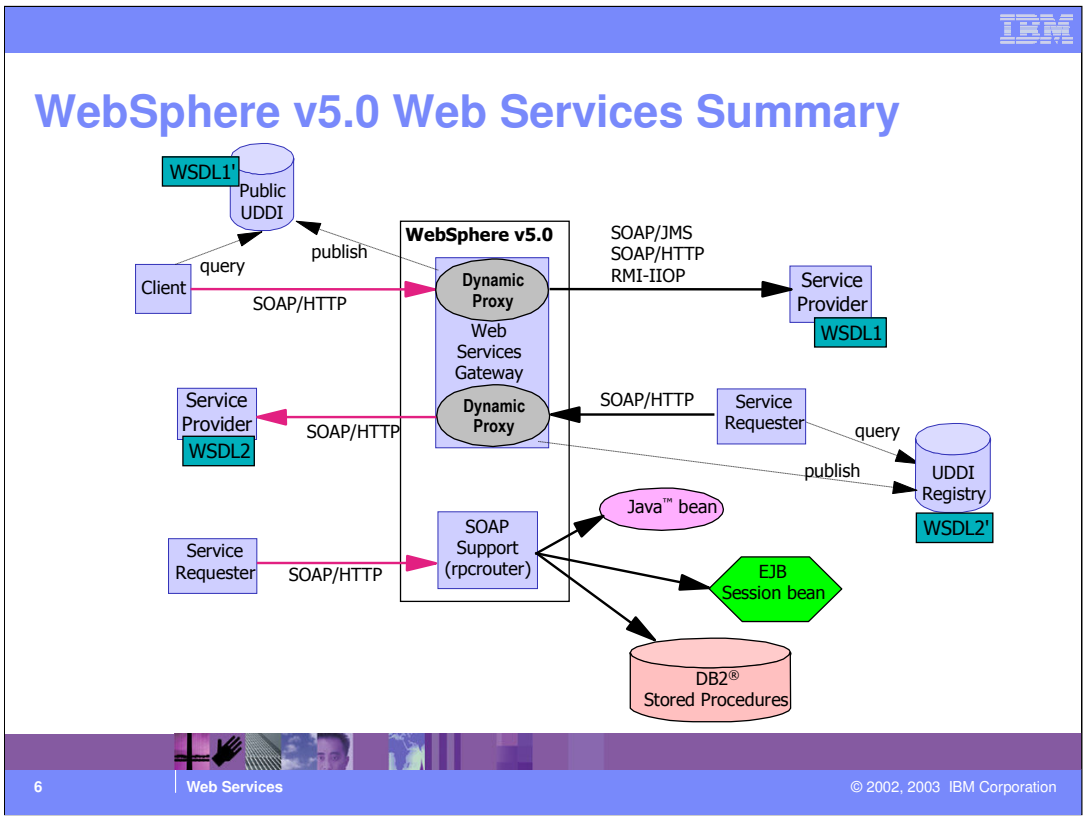


The functional enhancements and the commitment to open standards that characterize the WebSphere V5.0 release go well beyond mere support of J2EE 1.3. In fact, IBM believes that the market of information technologies is mature enough to move on to the next step in terms of distributed, Internet-enabled applications. That's where Web Services play a key role.

The Service Registry role owns a directory of all of the services available. The Service Registry represents a new, potentially very lucrative business model. The Registry owner could charge a fee for the use of their directory. Service Providers lists (or advertise) the Service offering in the registry. Service Requesters on the other hand query the Service Registry about the services available. Once Service Registry provides the binding information to the requester, it is no longer involved in the communications between the provider and requester.

The Service Provider has developed services that they make available as Web Service. These services will be hosted on their Application Server. A service is invoked by a requester through an XML message. These XML messages are generally carried across the Internet through a network-neutral standard protocol called Simple Object Access Protocol (SOAP). The Service Provider describes the service they are making available with a standard encoding called Web Services Description Language (WSDL).

The Service Requester is the business that requires a certain business function to be fulfilled. From an architectural perspective, this is the application that is looking for (queries the Service Registry) and then binds to and invokes the service. The requester has to find the service before invoking it - this process of discovery involves accessing a directory where the services are published. The access occurs through a set of standard APIs defined in the UDDI standard. UDDI stands for Universal Description, Discovery and Integration specification.



Web Services in WebSphere v5.0 have been enhanced with the addition of WebSphere UDDI Registry and the Web Services Gateway. The WebSphere Application Server can act as both a Service Provider as well as a Service Requester for Web Services. Some of those scenarios are shown.

The other new addition the WebSphere Web Service tooling is the UDDI Registry. The UDDI Registry provides a means of storing and categorizing information on Web Services available privately or publicly.

Web Services Gateway, which is installed into WebSphere v5.0, allows for Web Services to be exposed and available as services not bound to SOAP. Through the Gateway a service can be accessed from the client through SOAP over HTTP and actually called by SOAP over JMS. There are other protocol conversions that also can take place with the Web Service Gateway. The Gateway can also be used to expose Web Services to internal clients over different bindings from SOAP.

WSDL defines an XML format for describing network services as a set of endpoints that operate on messages. WSDL documents allow developers to expose their applications as network-accessible services. By using UDDI and WSIL, other applications can discover WSDL documents and bind with them to execute the business processed defined.

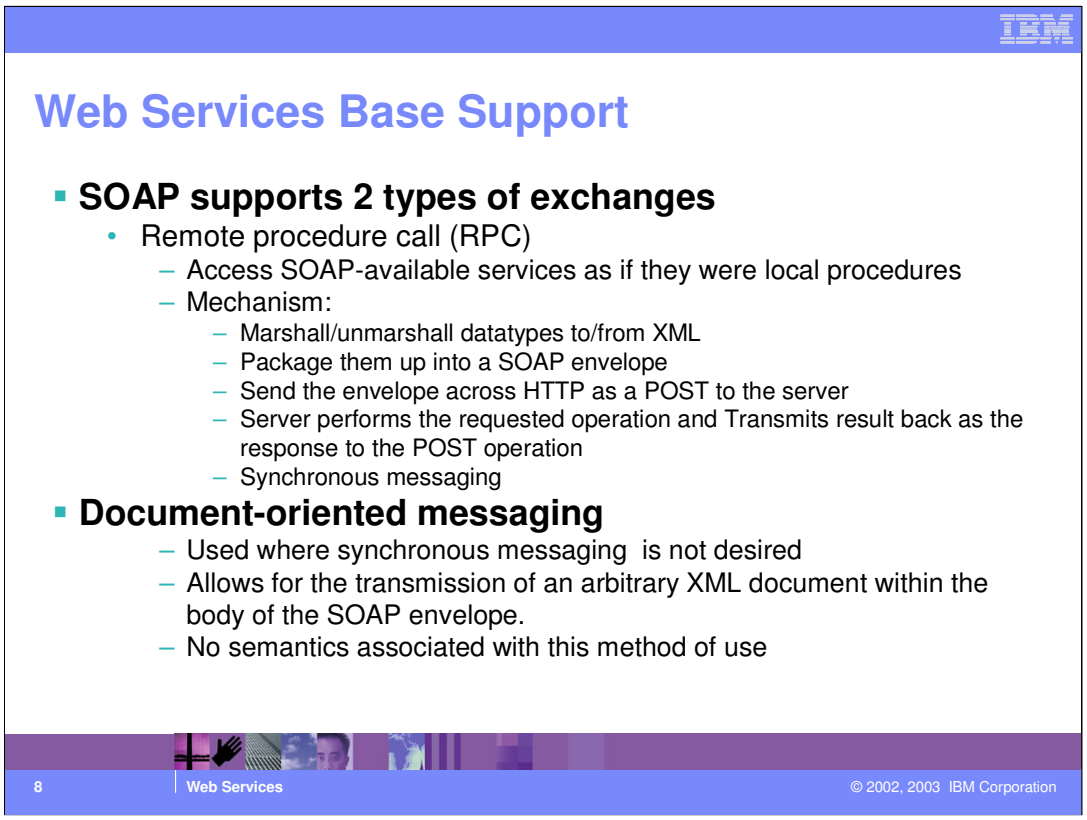
SOAP is the Simple Object Access Protocol. IBM's implementation of SOAP builds upon Apache SOAP.

WSDL allows a service provider to specify the following for a Web Service:

The name of the Web Service and addressing information

The protocol and encoding style to be used when access the Web Service

The type information like the methods, parameters, and data types describing the interface of the Web Service.

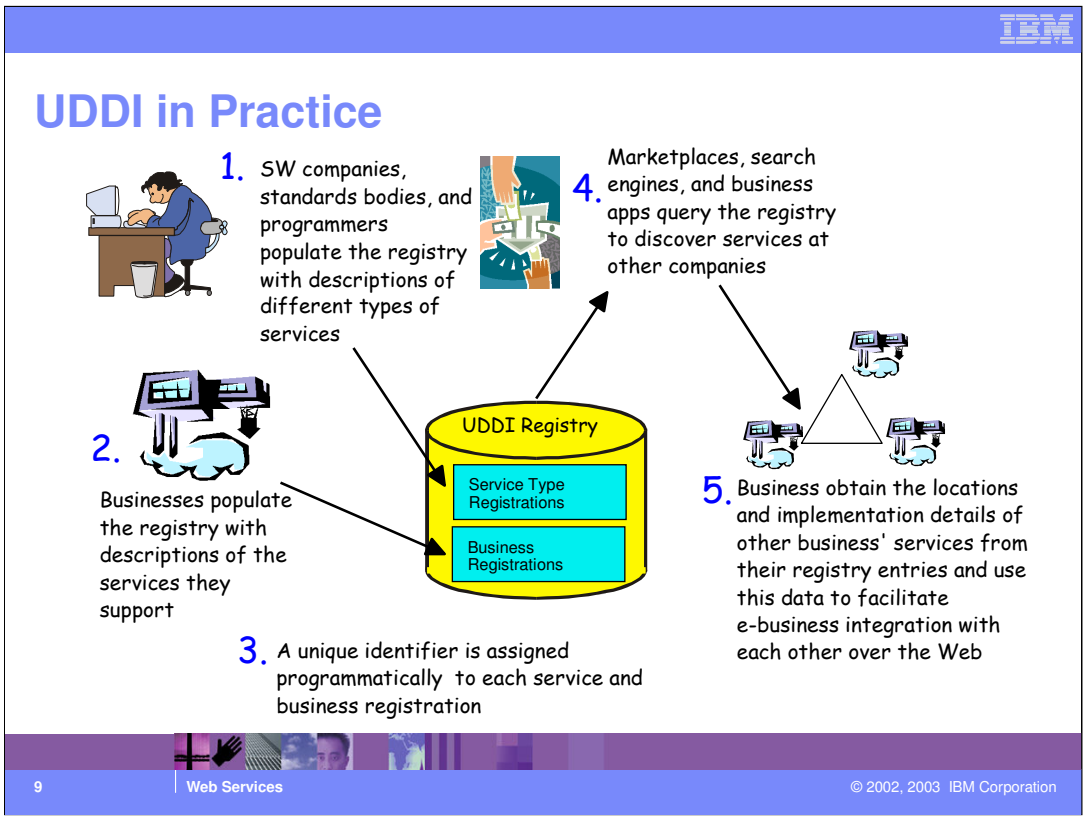


Web Services need a neutral and structured way to represent business information. XML provides an ideal framework for describing what a service can do and how to invoke it - and even for actually invoking a service, passing parameters, and getting responses back.

Service providers and service requesters interact by exchanging XML messages. XML has become the standard mechanism for exchanging messages in the Web Services arena.

There are two types of message exchanging with SOAP. The first is a remote procedure call where a method is executed on the service provider's system. A response is expected for this type of message

Style (RPC vs. Document) is orthogonal to invocation mode (e.g. request/response vs. one-way messaging vs. solicit/response vs. notify).



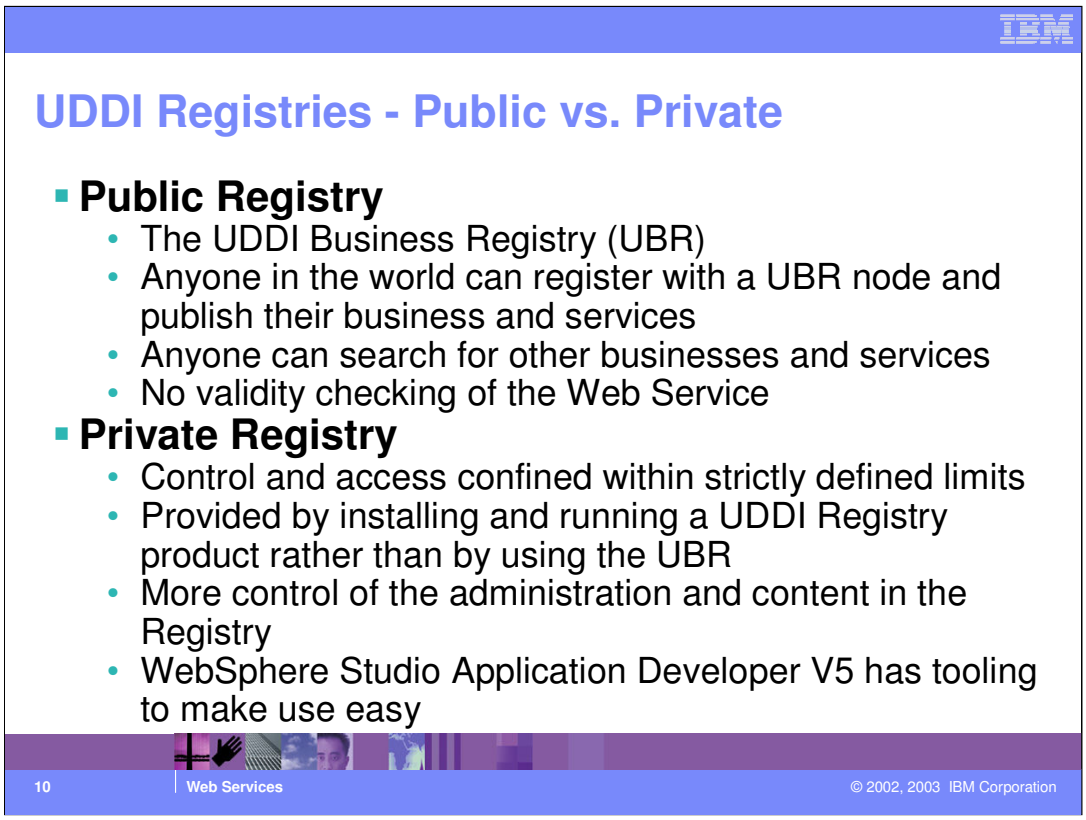
The Unit Test UDDI Registry is a value-add function to the Web Services picture for Version 5. With the test (or private) registry, you can publish and test your internal applications in a secure, private environment.

IBM's private registry is called the IBM WebSphere UDDI Registry.

The IBM WebSphere UDDI Registry is an implementation of the SOAP-based APIs, as defined by Version 2 of the UDDI specification that allow publish and query functions programmatically.

Additional information on the IBM WebSphere UDDI Registry can be found in the InfoCenter for WebSphere Version 5 and other UDDI information can be found at http://www.uddi.org.

Businesses populate the registry with descriptions of the services that they support. The registry assigns a unique identifier to each service description and business registration, storing the identifiers in the registry. Service requesters can query the registry to discover services.



Public Registry

is an implementation of the UDDI specification

UBR nodes (the "cloud") are run by UBR operators (currently (12/02) IBM, Microsoft®, NTT.com, and SAP)

data is automatically replicated across the nodes

Positive side is universality and global nature, but it is not so good for ensuring privacy and validity of information

Private Registry

May be confined to:

a department

a company

a consortia of companies

an industry group

etc.

UDDI Example Scenarios

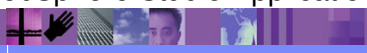
A mid-sized manufacturer needs to create 400 online relationships with customers, each with their own set of standards and protocols

A flower shop in Australia wants to be “plugged in” to every marketplace in the world, but doesn’t know how

A B2B marketplace can get catalog data for relevant suppliers in its industry, along with connections to shippers, insurers, etc.

Unit Test UDDI Registry Details

- **Installing and Configuring the Registry - Runtime and Tooling**
  - An optional install component with WebSphere Application Server: Network Deployment
  - Automatic file deployment, database creation and uddi.ear file installation with scripts
  - Pre-configured for Cloudscape-based UDDI Registry
  - Configurable for DB2
  - Configuration performed through a properties file called uddi.properties
  - Unit Test UDDI Registry available with the WebSphere Studio Application Developer
- **Accessing the Registry**
  - SOAP Interface
  - UDDI4J
  - EJB Interface
  - Browser-based User Console
    - http://localhost:9080/uddigui
  - WebSphere Studio Application Developer's Web Services Explorer



Installation creates a UDDIReg directory under the WebSphere Network Deployment Install Directory

Cloudscape is intended for development or testing use only. DB2 is used for production environments.

Cloudscape is the default database backend.

Configuring for DB2 requires running database creation scripts and modifying the properties file

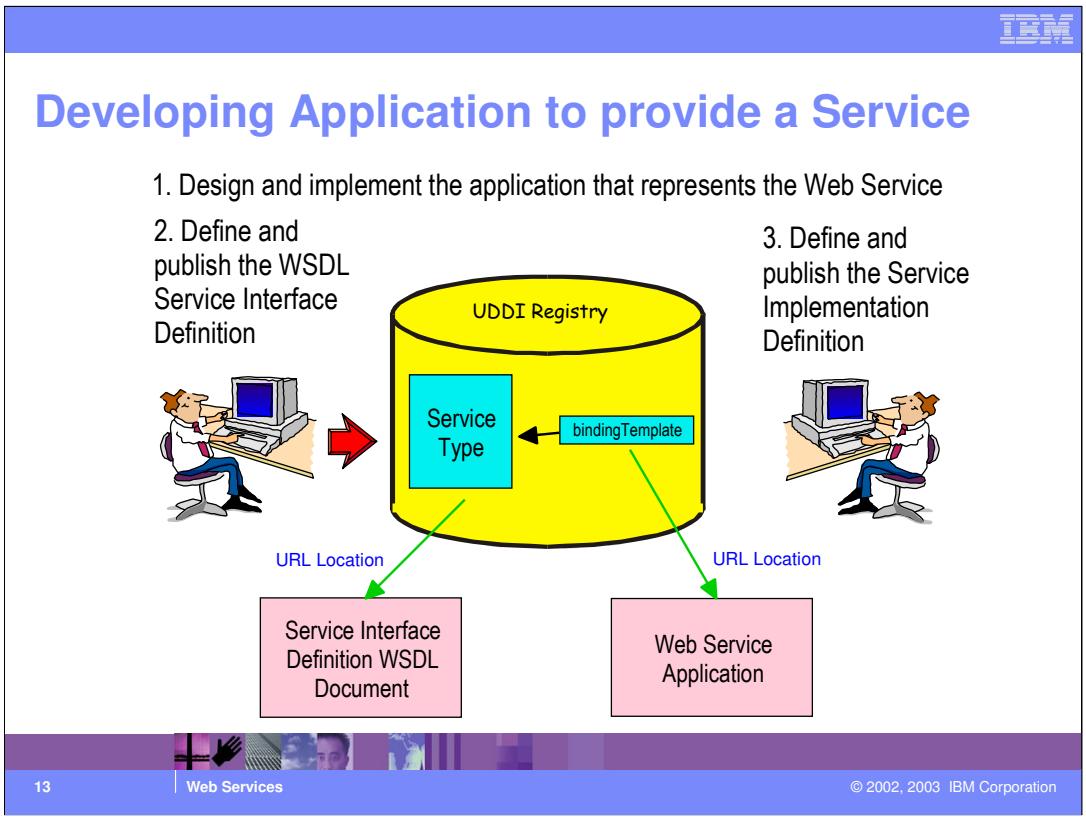
The properties file resides in WebSphere's properties directory

The Unit Test Registry is also available to be installed within the Application Developer - more on this later in this presentation.

UDDI4J is a Java API which simplifies the process by hiding the SOAP XML messages and HTTP connection implementation

The user console is accessed via http://hostname/uddigui or https://hostname/uddigui

Using the WebSphere Studio Application Developer V5 Web Services Explorer, you can access the private (or any v2 compliant) UDDI Registry.



Step 1

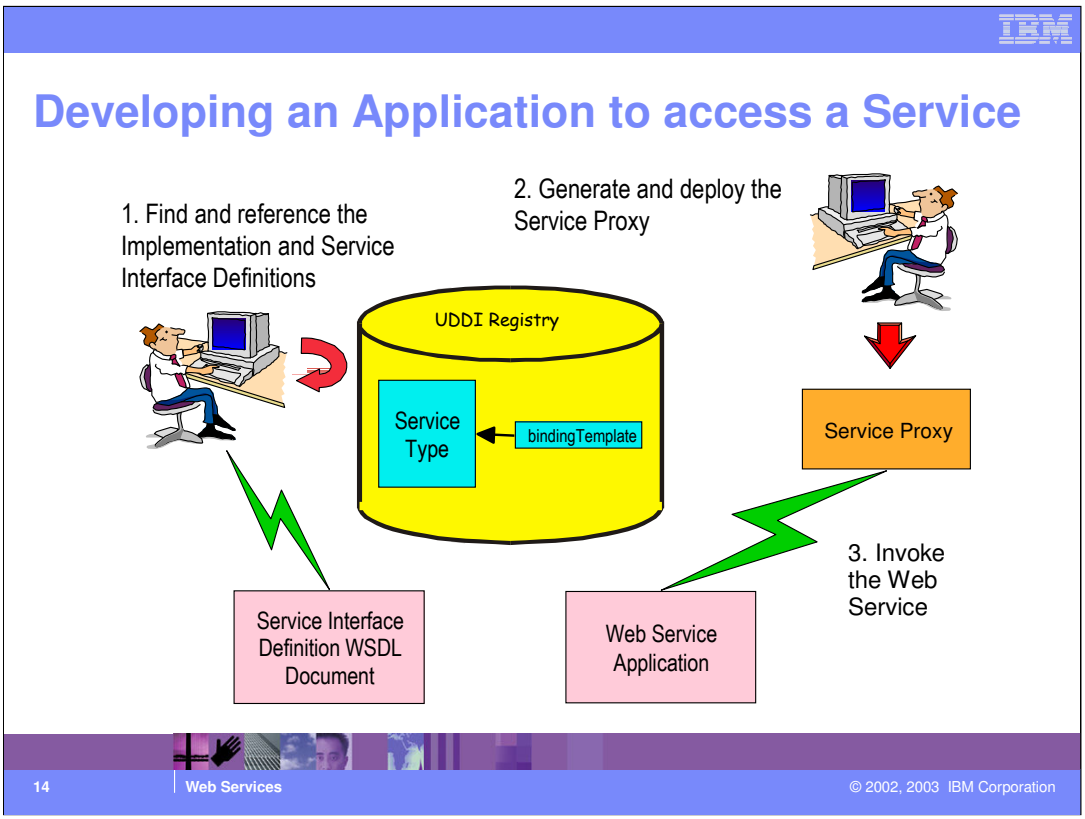
This involves the design and coding required to implement the Web Service

Step 2

The WSDL Service Interface Definition document defines the interface and mechanics of the service interaction (e.g. message structures, data types)

This step involves publishing a Service Type definition to the UDDI Registry (stored in a tModel structure). The tModel contains an element called overviewDoc which references a URL at which the WSDL Service Interface Definition document is located (the WSDL file is not stored in the Registry directly). Note that this step might not necessarily be performed by the service provider. For example, a standards body may publish a Service Type representing a standard web service interface (a hotel booking interface for example).

Step 3 involves publishing a Service to the UDDI Registry (stored in a businessService structure within a businessEntity structure). The Service registry entry contains a bindingTemplate structure whose tModelInstanceInfo element references the Service Type created in Step 2.



Step 1

The developer interrogates the UDDI Registry to find:

the Service entry

the associated WSDL Service Interface definition (via the referenced Service Type)

the network location of the service (this is contained in the accessPoint element within the bindingTemplate)

Step 2

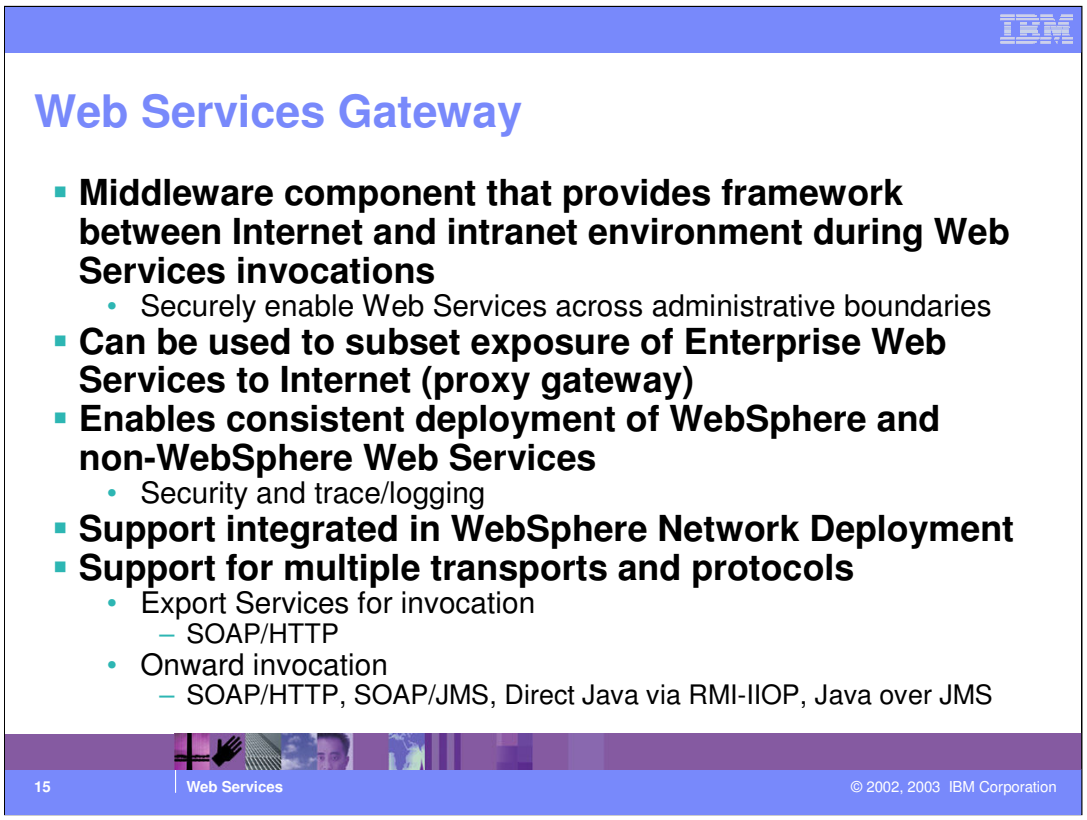
The Service Proxy contains all of the code that is required to access and invoke a Web Service. Typically, the development tooling will generate this automatically

The Service Proxy is deployed with a client application

Step 3

The client application is run and uses the Service Proxy to invoke the Web Service

Note: It is possible that the service interface definition is located at run time (called Runtime Dynamic Binding) rather than at build time as above (Static Binding)



The Web Services Gateway itself if a J2EE application.

The Web Services Gateway allows interaction with public and private UDDI registries. External services can be exposed as internal services. Likewise, internal services can be exposed externally.

The gateway provides a bridge between the Internet and Intranet and therefore between services located in those two environments

The gateway handles conversion between protocols. For example, external services can be invoked with SOAP/HTTP and SOAP/JMS or RMI/IIOP can be used for internal services.

Code-free configuration of services through the gateway. There is no development involved.

A SOAP/HTTP channel is supported (two channels are shipped - one for external customers the other for internal).

Single deployment to multiple channels

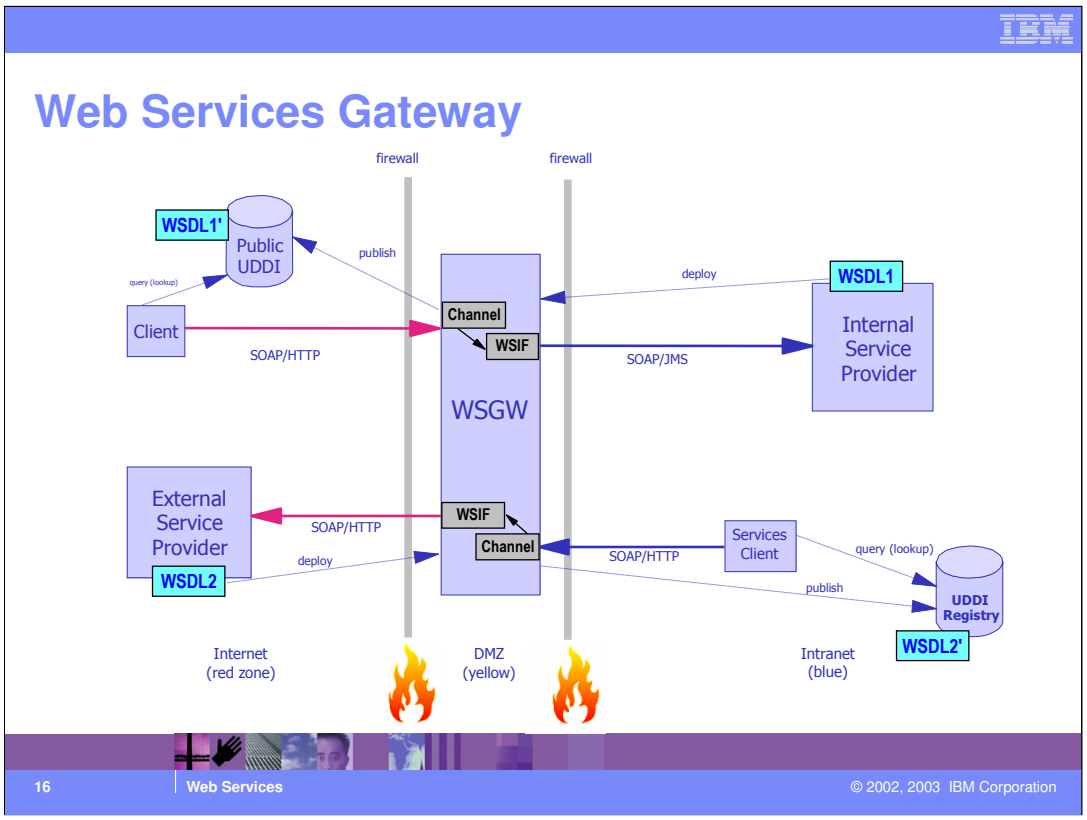
Support for multiple target service types - e.g. SOAP/HTTP, SOAP/JMS, EJB, and Java

Configurable intermediation through filters

Support for discovery through WSDL and UDDI

Support for publication through WSDL, WSIL and UDDI

Authentication and authorization support (configurable for each service and/or operation)



This chart shows the final result of deploying your binding information to the Web Services Gateway.

In the top half of the chart, you have created a Web Service internally and have deployed the binding information to the Web Services Gateway. The gateway in turn creates a service proxy and a description file that is made available in a public UDDI. When the client (external in this case) requests that specific service, it retrieves the binding information from the public UDDI registry and invokes the service based on the description. The client does not have to be concerned with the implementation details as the Web Services Gateway hides the implementation.

The gateway handles the Security aspects of this call. There are three primary authentication mechanisms: HTTP(S) based authentication, servlet-based authentication, and SOAP authentication done by document signing and verification.

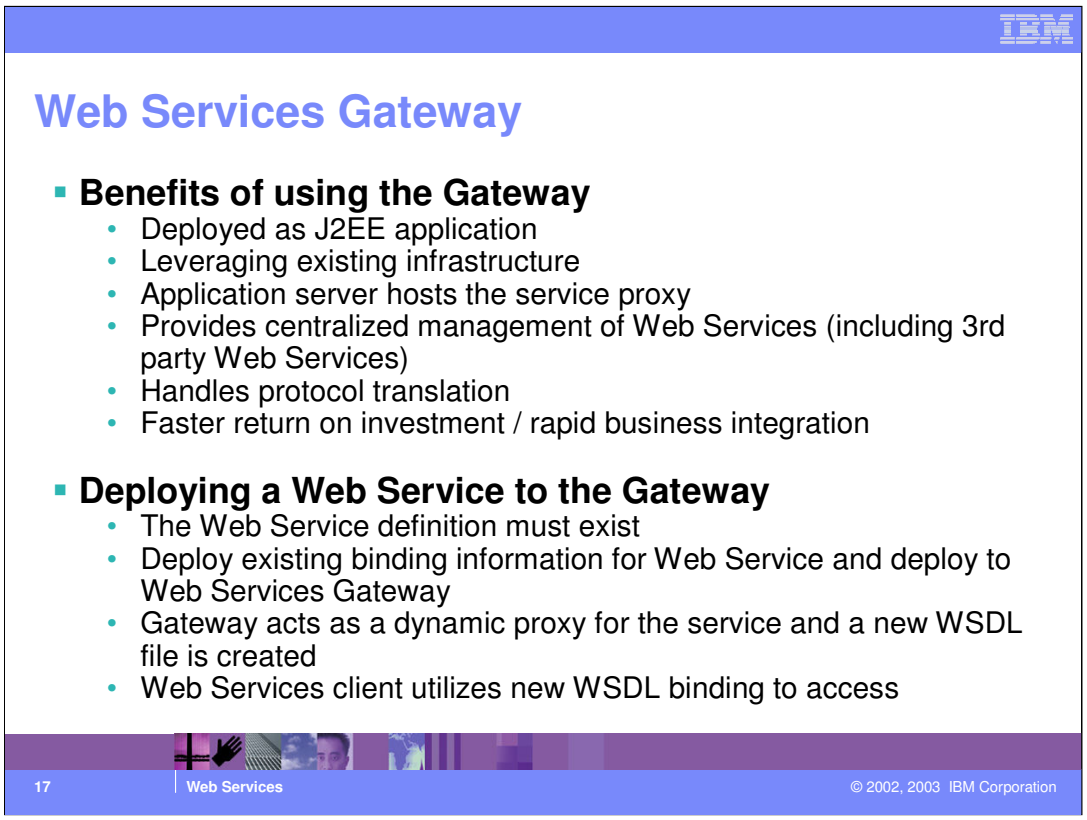
The bottom half of the charts illustrates how an external service could be exposed as an internal service.

Here is a brief description of the protocols illustrated:

1) SOAP/JMS: The JMS message body is a text message, whose text contains a SOAP message which encodes the input and output messages for the service invocation.

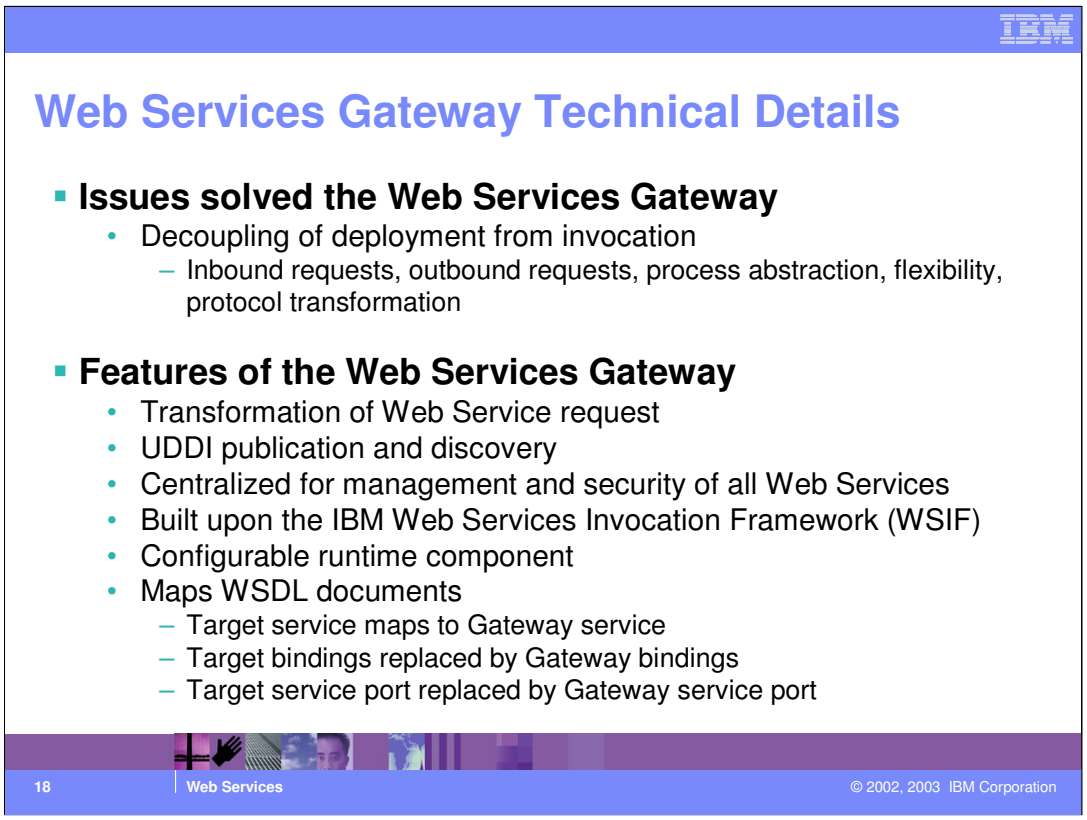
2) Java/JMS: The JMS message body is an object message, and the input and output messages for the service invocation are written into the message body as serialized Java objects.

The Web Services Gateway is configured with a number of transport channels. A transport channel is the underlying transport mechanism for Web Service invocation. These channels provide the provider a choice of transports on which to make the Web Service available. When you map (deploy) a service through the Gateway, you can configure



The Web Service Gateway includes a web-based console for the management of services (deployment, undeployment, viewing deployed services etc.) The gateway currently handles only incoming SOAP/HTTP requests, but support for more channels will be added in the future. Requests passing through the gateway may be sent to a Java class, an EJB, or a SOAP server (including another gateway.)

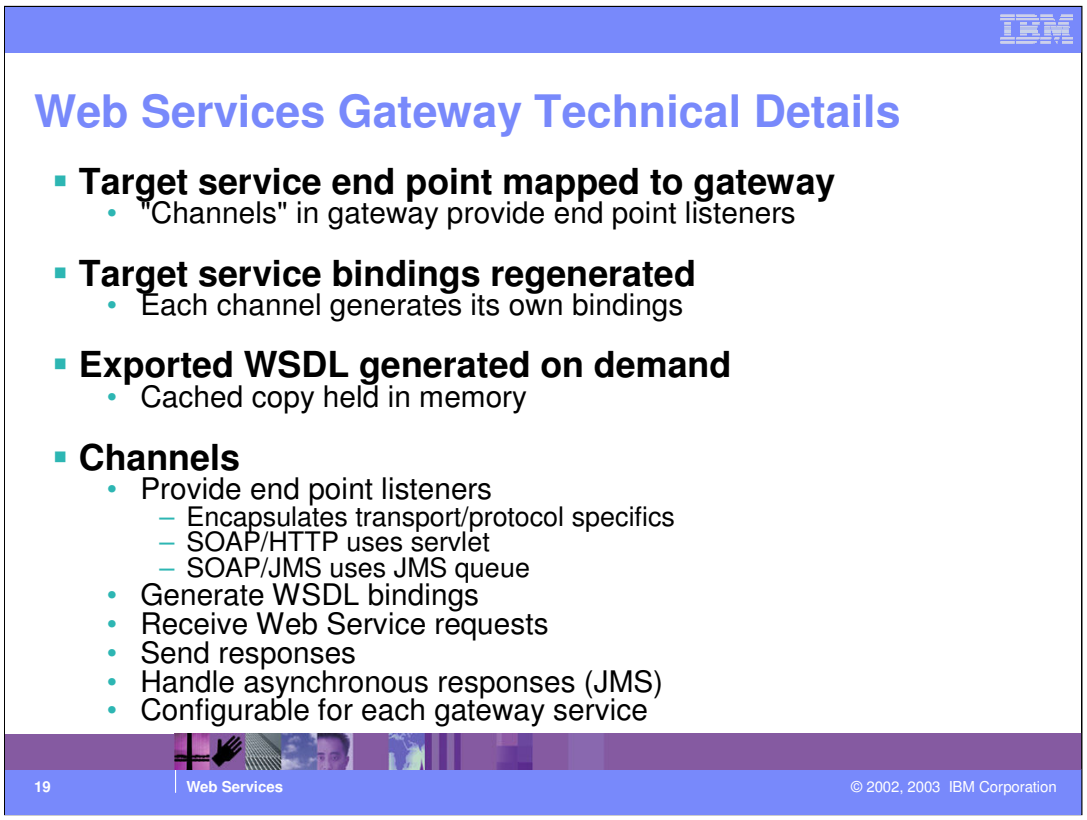
Additional information on the Web Services Gateway can be found on DeveloperWorks. Here's an introduction article to get started: http://www-106.ibm.com/developerworks/webservices/library/ws-gateway. The InfoCenter is also a good resource for information.



The Web Services Gateway separates the actual implementation of a service from how it is accessed. This provides for the higher level of abstraction and flexibility results. The deployment infrastructure can be modified without notifying all of the service requesters.

Web Services Invocation Framework (WSIF) invoke Web Services defined using WSDL directly, hiding the complexities of the underlying protocols.

The Web Services Gateway forwards incoming requests to the target service and then sends the responses back to the originator.



A request to the Web services Gateway arrives through a channel, is translated into an internal form, then passed through any filters that are registered for the requested service, and finally sent on to the service implementation. Responses follow the same path in reverse.

A deployed Gateway service, channel, filter, etc. information is stored in a file called wsgw.cfg (a serialized Java object). The Gateway version of the WSDL service definition (implementation and interface) files are generated as needed from the target service WSDL and cached. If the target service WSDL becomes unavailable, then so does the Gateway version (at least temporarily).

WSIF enables the Web service developer, to move away from the usual Web services programming model of working directly with the SOAP APIs, towards a model where you interact with representations of the services. So you can work with the same programming model regardless of how the service is implemented and accessed.

WSIF is a runtime architecture for Service-Oriented Applications

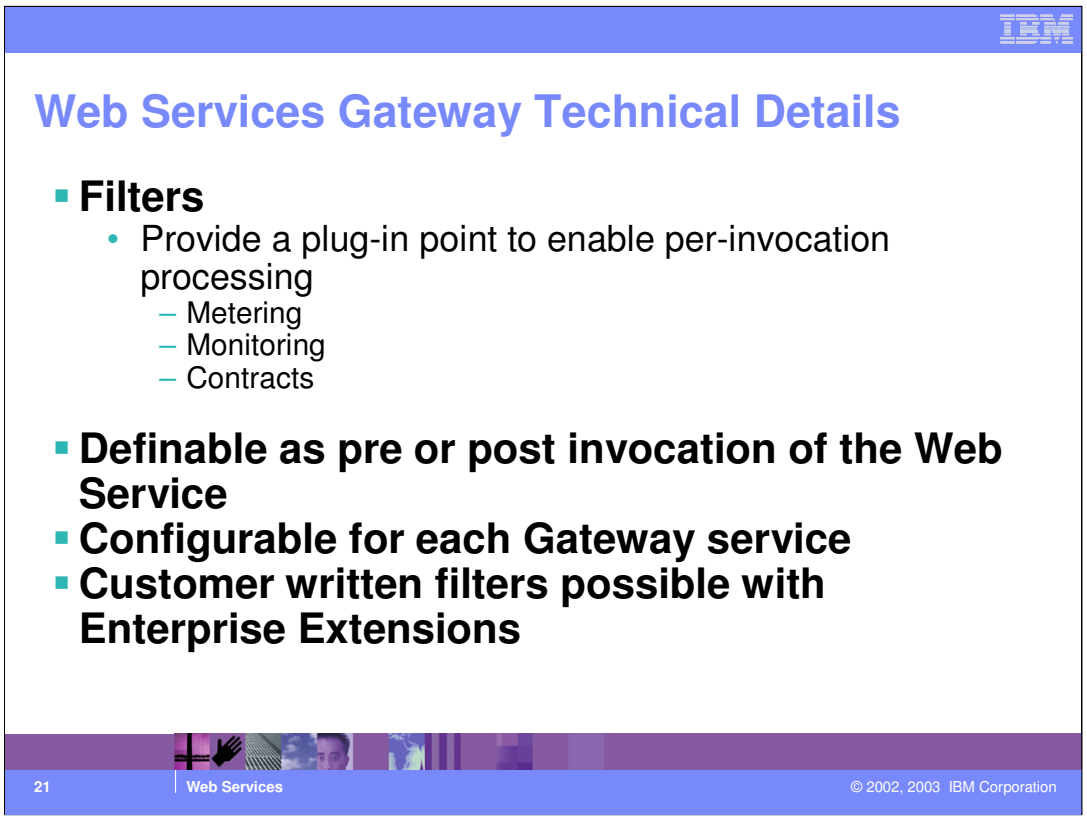
The Gateway deals internally with WSIFMessage object that the Channel converts to/from the wire format.

With the interface and binding information moved out of the application code, it is possible to use other protocols using same client model.

WSIF allows a binding-independent mechanism for Web service invocation.

WSIF free client code from the complexities of any particular protocol used to access a Web service.

WSIF enables dynamic selection between multiple bindings to a Web service and helps the development of Web service intermediaries.

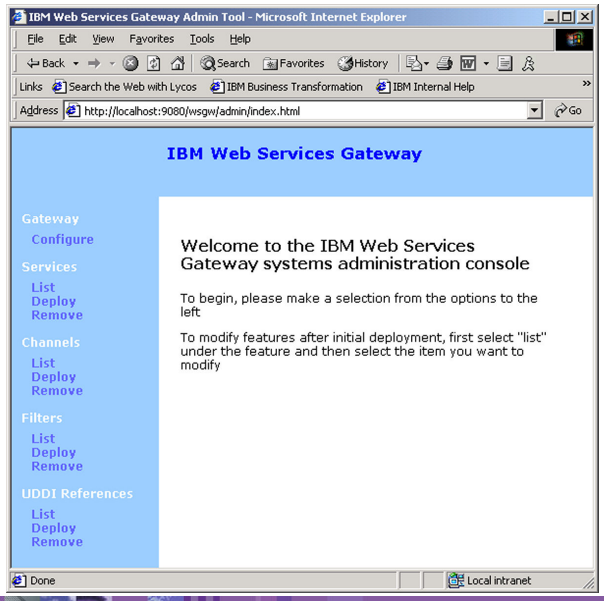


Filters are used to intercept service invocations which come into the Web services Gateway, and the responses coming out of the Gateway. Filters can perform a wide range of tasks, from logging messages, to transforming their content, to terminating an incoming request. Filters are deployed to the Web services Gateway, however, they are not used until they are registered for used with a deployed Web Service.

To install a filter, Click Filters on the main administration page. Then click Deploy. A form is displayed for you to specify the filter deployment details. Fill in the Filter name – this is the name by which the filter is known within the Web Services Gateway. This name must be unique within the Web Services Gateway.

Filter samples will be available with the WebSphere Programming Model Extensions.

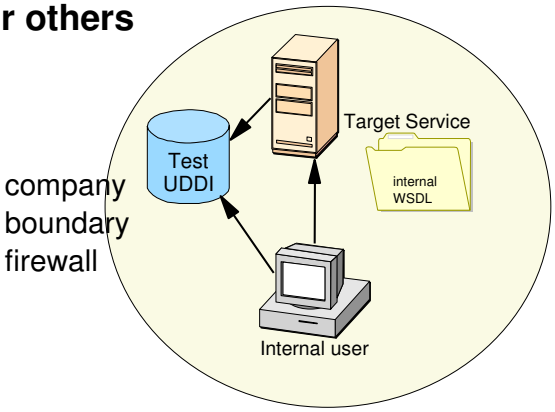
Web Services Gateway



The Web Services Gateway, a J2EE Application, has a browser-based interfaced. It is located at the URL shown in the Address bar of the browser window. You can specify information about the Gateway instance, configure channels, filters, UDDI references, and work with services (list, deploy, undeploy, etc.). D

Web Services Gateway - Outline View

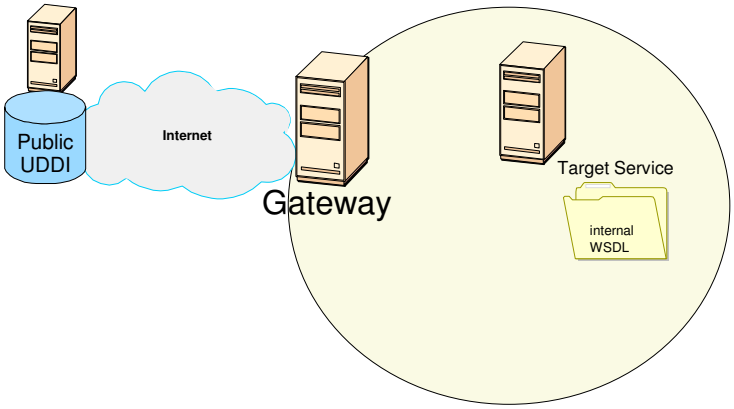
- **Internal service (Target Service) has been developed - published to WebSphere UDDI Registry**
- **After internal use and testing, want to make the service available to partners or others**



This chart illustrates the current environment largely as it exists today. Now with the WebSphere UDDI Registry, internal services can be tested and when desired, externalized to other parties. This externalization is done through configuring the Web Service using the Web Services Gateway.

Web Services Gateway - Installation

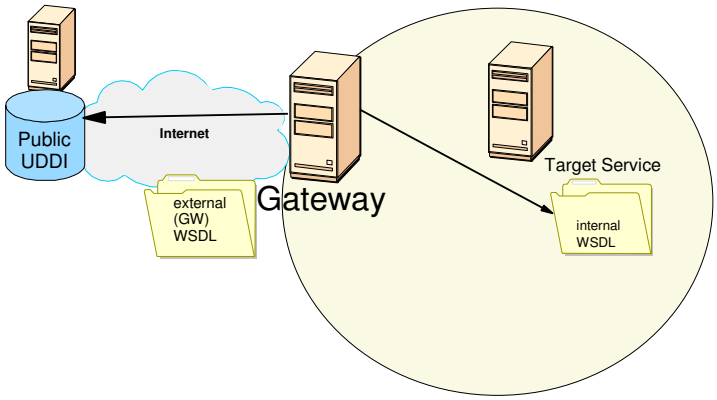
- **Install Gateway**
- **Configure Gateway with channels (e.g. SOAP)**
- **Configure Gateway with public UDDI Registry details**



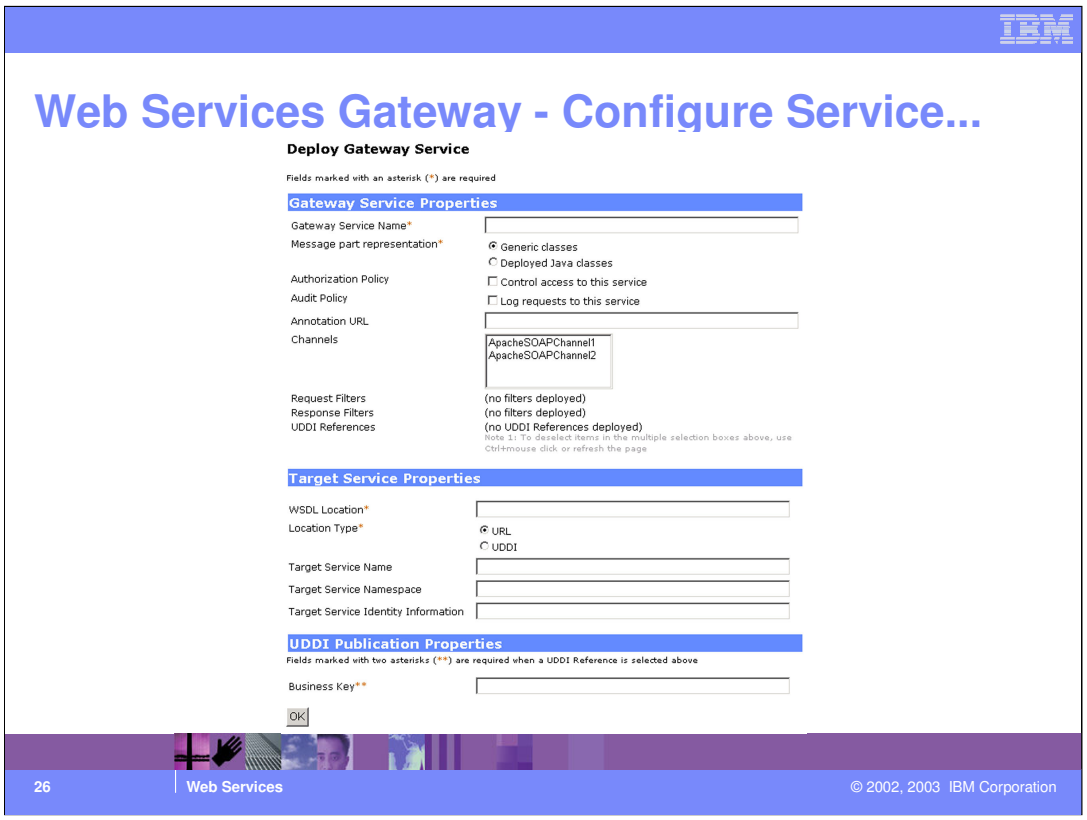
The WebServices Gateway is an installable option on the WebSphere Network Deployment package. The InfoCenter contains excellent documentation on how to complete the installation.

Web Services Gateway - Configure Service

- **Point Gateway at internal WSDL (URL or UDDI)**
- **Choose which channels to make service available over**
- **Gateway publishes "external" WSDL to URL, UDDI and WSIL**



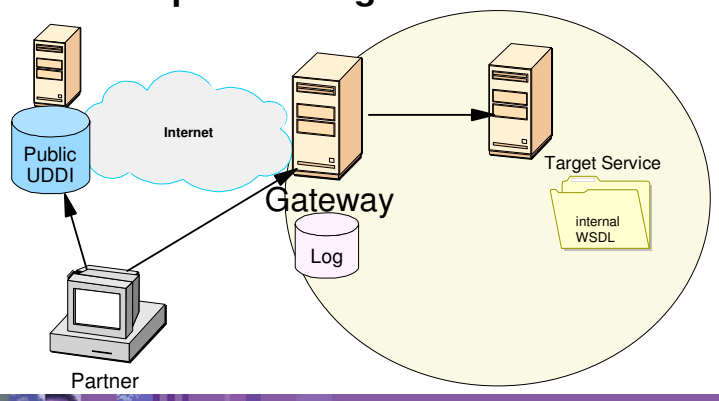
To configure a service on the Gateway, you use a web browser, and bring up the Web Services Gateway GUI console. It is then you choose to deploy a service to the Gateway. There are several choices to make for each Web Service you deploy (e.g. authorization, filters, the channels to make available over, etc.)



This graphic shows how you would configure a service. Each service is configurable - for example, you may use filters on some services and not on others. You can deploy a web service to multiple channels with a single deployment.

Web Services Gateway - Runtime

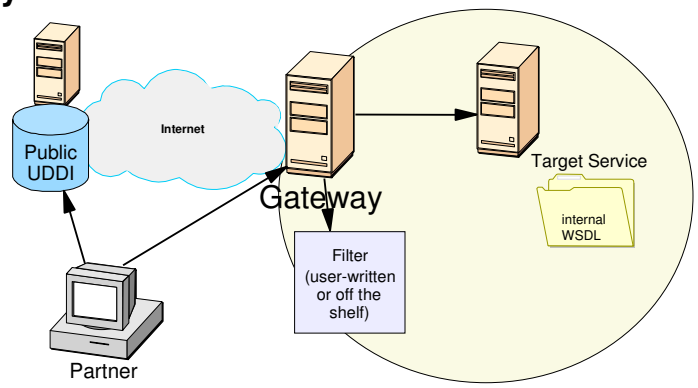
- **Partner queries UDDI (or URL, WSIL)**
- **Partner calls service**
- **Gateway authenticates, authorizes, and audits**
- **Gateway forwards request to target service**

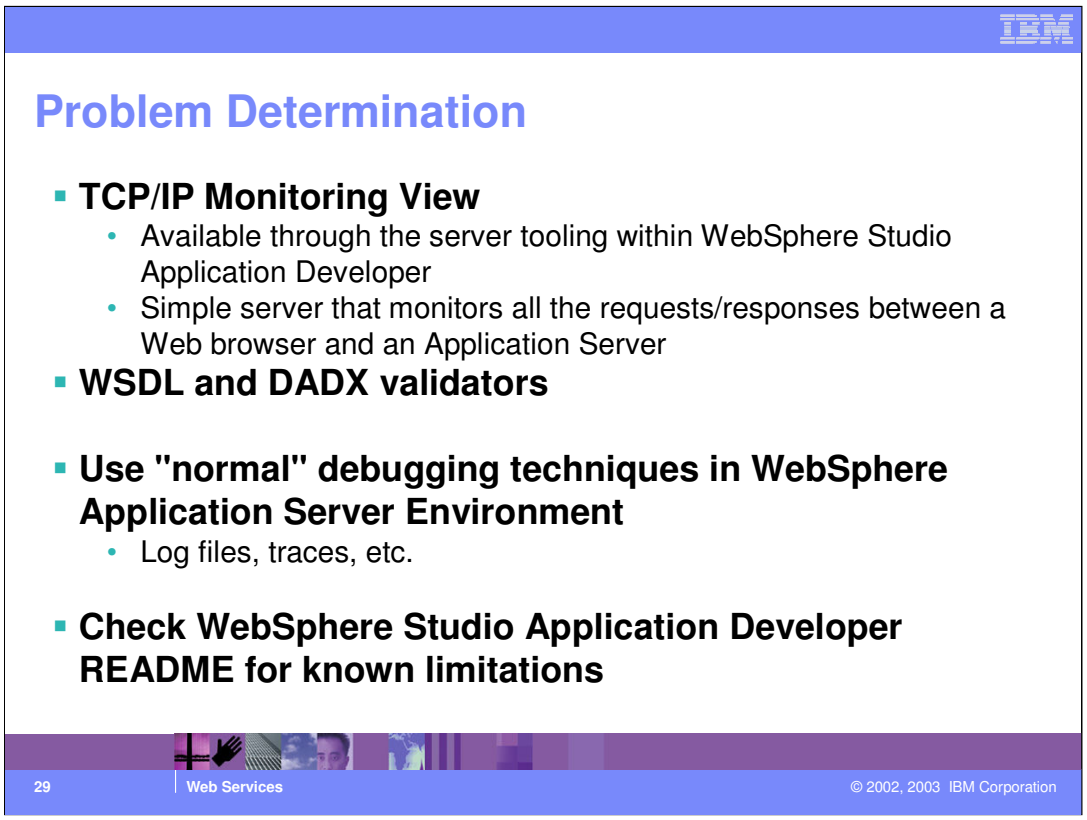


Once the service has been deployed, the third party calls the service and the Web Service Gateway handles passing the request to the target service.

Web Services Gateway - Intermediation

- For intermediation - e.g. metering, custom authorization, load-balancing, routing - the administrator deploys a "filter" (stateless session bean)
- The Gateway calls out to filter before and/or after every request





Problem Determination and Debugging is a large topic that is extremely important especially when your application is not working as designed. As you create and/or consume a Web Service, the Web Services tooling wizard has validators that help prevent problems from occurring.

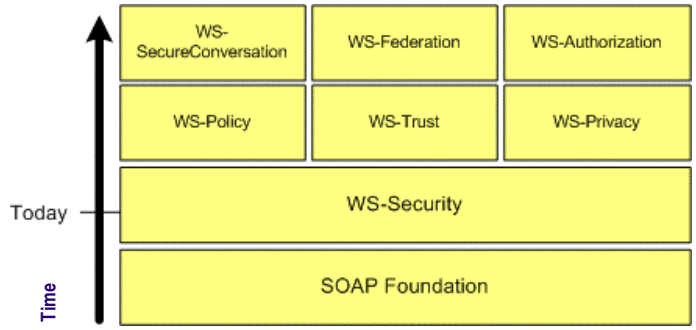
Typically, you would employ the 'normal' WebSphere Application debugging techniques. The log files generally provide good clues as to what may be occurring. Enabling tracing is probably the next step if you cannot determine the problem.

Specifically for Web Services, there is a TCP/IP Monitoring View within WebSphere Studio Application Developer that serves as an point where all requests between a Web browser and Application Server pass through. By using the TCP/IP Monitor, you can view the traffic that is flowing across the network.

Java debug can be used too....as most web services parts are implemented in Java with the exception of DADX and URL.

Web Services Security

- **There is no such thing as absolute security**
- **There are risks and steps to minimize those risks**
- **Aspects of security**
  - Identity, authentication, authorization, integrity, confidentiality, auditing, non-repudiation

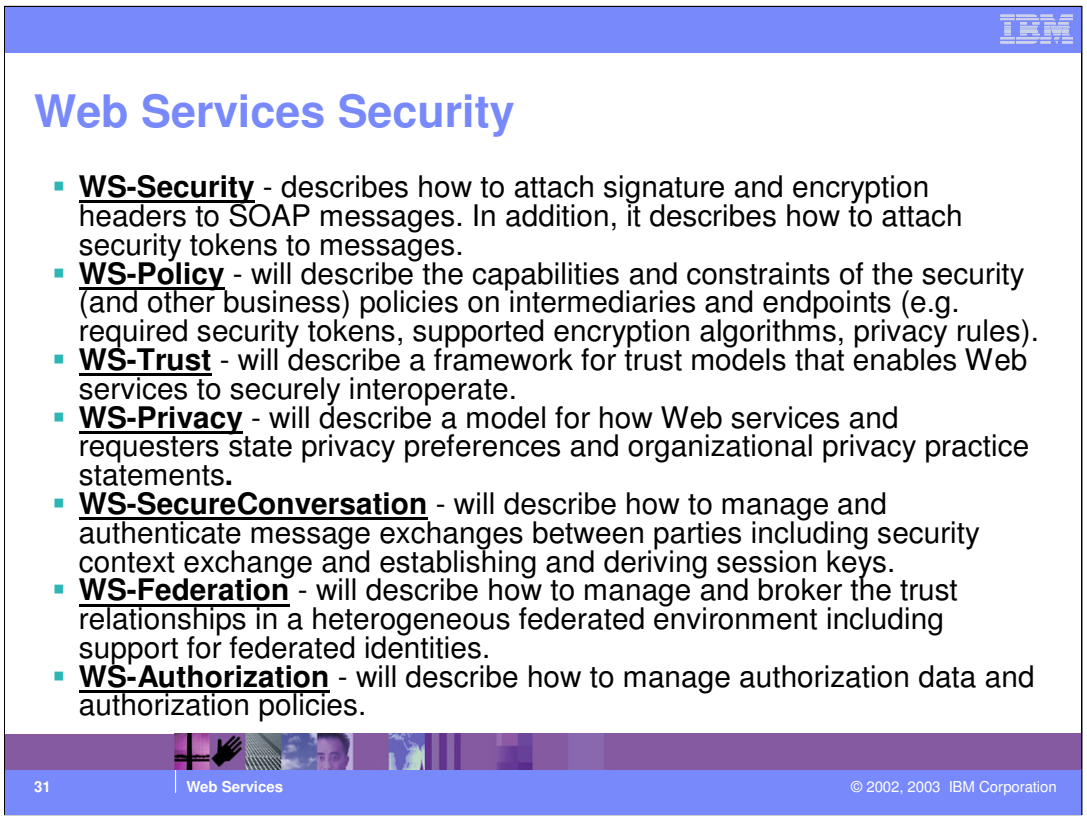


Seven Aspects of Security

1. Identification: who are you?
2. Authentication: how do I know your identity is true?
3. Authorization: are you allowed to perform this transaction?
4. Integrity: is the data you sent the same as the data I received?
5. Confidentiality: are we sure that nobody read the data you sent.
6. Auditing: record of all transactions so we can look for security problems after the fact.
7. Non-repudiation: both sender and receiver can provide legal proof to a third party that the sender did send the transaction, and the receiver received the identical transaction.

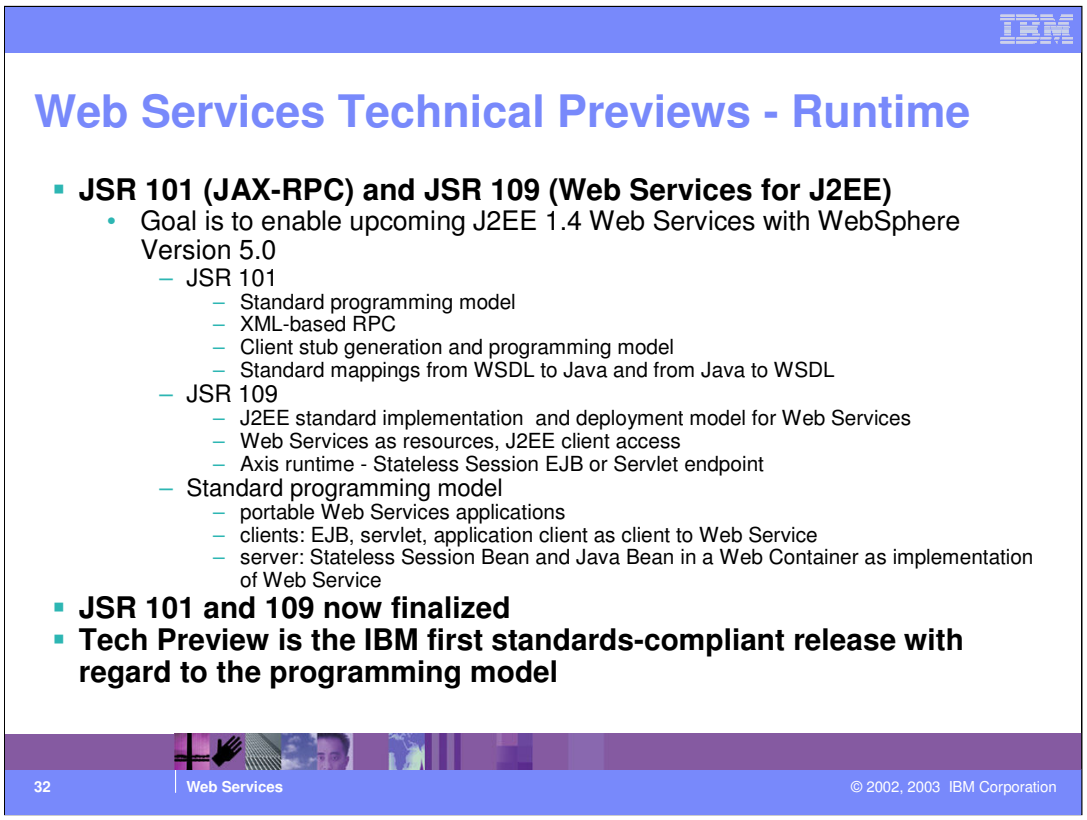
The Web Services security model introduces a set of individual interrelated specifications to form a layering approach to security.

The WS-Security specification (co-developed by IBM, Microsoft, and VeriSign, Inc.) provides the basic elements to add message integrity and confidentiality to Web Services. WS-Security is at the foundation of the Web Services security model that is being proposed.



There is a download available on alphaWorks (http://www.alphaWorks.ibm.com) that provides an implementation of SOAP Security Tokens and Digital Security components of the WS-Security specification. This specification was co-developed by IBM (other participant was Microsoft) to help companies build secure, interoperable Web Service applications. The SOAP Security Token indicates the message sender's properties: name, identity, credentials, and capabilities. It helps identify the message sender to the Web Service provider. WS-Security describes enhancements to SOAP messaging to provide quality of protection through message integrity and message confidentiality.

To learn more about Web Services security, please visit:
http://www.ibm.com/developerWorks/webservices/libray/ws-secure



IBM continues to demonstrate strong leadership in the area of Web Services and the driving evolving open standards

WebSphere Application Server V5 has the following support for Web Services:

- Apache SOAP 2.3 runtime
- DOM-based parsing
- Relies on Web Services Toolkit or WSAD for emitters (no tooling in the runtime to help write stubs)
- Proprietary programming model

Web Services Technical Preview

- Based on Apache Axis
- SAX-based parsing
- Includes command line tooling
- Conforms to Java Web Services standards
- Standards-based programming model
- Applications portable across Application Servers

Security via normal J2EE security mechanisms is accomplished by using J2EE Role-based security, using BasicAuth to collect Authentication data

WS-Security has three possible ways to authenticate

Basic Authorization (user token, userid and password) flows from the client to the server and it is authenticated and the identity is used to invoke the service.

Binary Security Token for X509 Certificate

Identity Assertion (after the signature of the signer certificate is validated, the Identity is used to invoke the service.

Summary

- **Web Services offer many benefits**
- **Industry standards used - XML, SOAP, WSDL, UDDI**
- **Tools for working with all Web Service Roles (Service Provider, Service Broker, and Service Requester)**
- **UDDI provides a mechanism for Service Providers to register their services and for Service Requesters to discover them**
- **The WebSphere UDDI Registry allows a variety of means of access:**
  - SOAP API, UDDI4J, EJB interface, Web-based console
- **Web Services Gateway allows sharing internal Web Services across administrative boundaries**
- **Web Services Security - security models not standardized**
- **Technical Previews demonstrate early J2EE 1.4 support**

Trademarks and Disclaimers