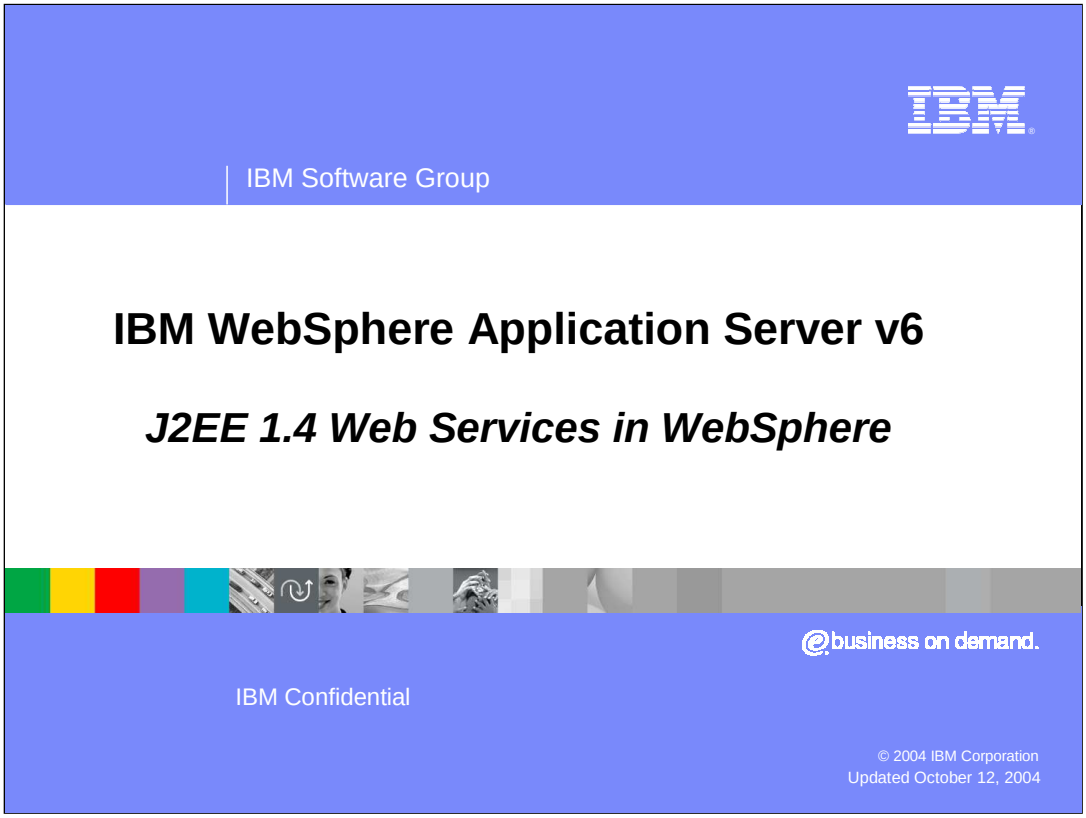

IBM Software Group
IBM WebSphere Application Server v6
J2EE 1.4 Web Services in WebSphere
e business on demand.
IBM Confidential
© 2004 IBM Corporation
Updated October 12, 2004

Agenda

- Web Services Support in WebSphere v6
- Web Services Interoperability (WS-I) Support
- New Web Services Features in v6
- WS-Security enhancements in v6
- Gateway Changes
- UDDI/JAXR
- Summary and References

Section

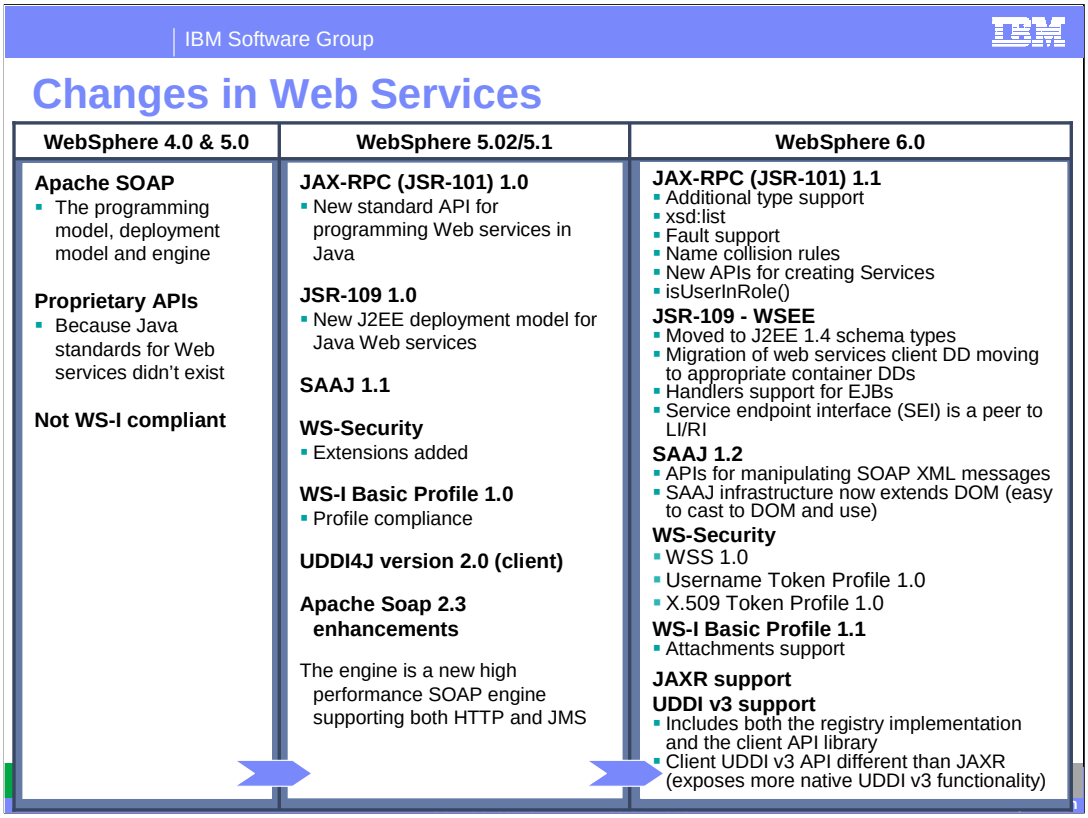
Web Services in WebSphere V6

Web Services Support in WebSphere

- IBM WebSphere Web Service engine implements J2EE 1.4 Web Services Standards
- Focus is on standards compliance and interfaces
  - Insure interoperability
- Enhanced performance – SAX based
- Integration into WebSphere administration and tooling
  - Ease Web Services development and deployment

Web Services and XML Support

- **Standards / Portability** - XML Schema definitions for all deployment descriptors
- **JAX-P 1.2** - New properties for XML parsers
- **JAX-R** - XML registry API
- **JAX-RPC** - APIs for representing WSDL-based services as RPCs in Java (and vice-versa)
- **JSR 109** - Web services programming and deployment model
- **SAAJ 1.2** - SOAP Attachments API for Java
- **WS-I** – Ability to create WS-I compliant services



What's new?

•

What used to happen?

•

Why does the customer care?

•

Top 3 points of this slide?

1. Web Services has evolved from a propriety solution to more standard environment with each release
2. Web Services is becoming more robust and viable
3. We are moving to a SOA environment…

- **Distributed and Z/0S <<<===**
- **Distributed only**
- **Z/OS only**

Additional Notes…

-

Section

WS-I

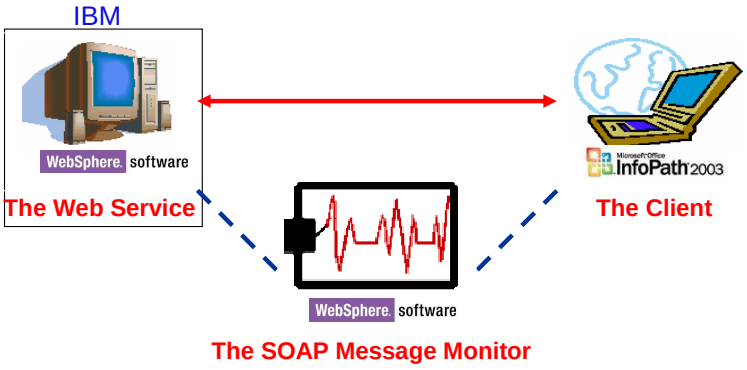
Web Services: Interoperability (WS-I)

- Organization focused on promoting interoperability between Web Services
- Provide guidance in the standardization of Web Services between different platforms, applications, and programming languages
- Defines profiles, which are a set of different specifications
  - WS-I Basic 1.0 Profile currently available

Additional information on WS-I is available at http://www.ws-i.org.

WS-I Overview

- Allows for Interoperability
  - Use 2 out-of-the-box products from 2 different vendors and develop a Web service that is compliant to WS-I BP 1.0



Development: Interoperable Web Service

- Our First Task …

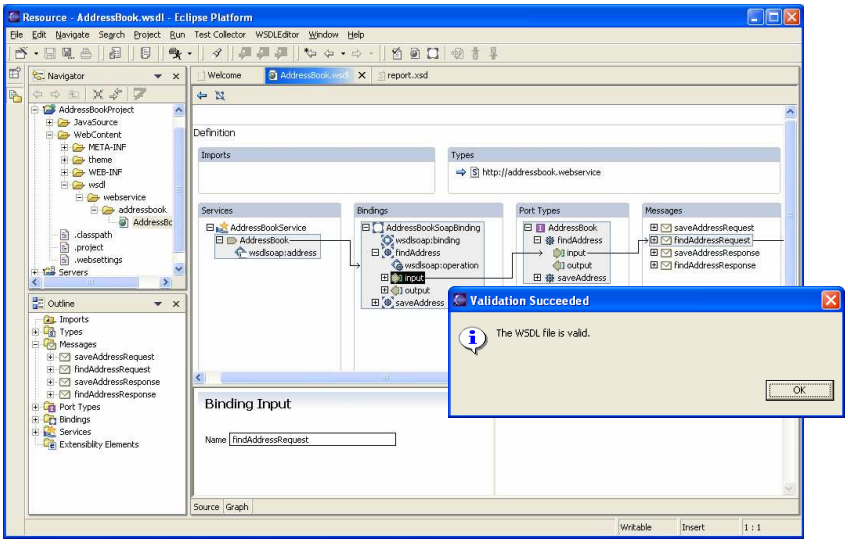
  Use WebSphere Studio Application Developer

  to design and develop a Web service

  Select the level of WS-I compliance

Development: The top-down Approach

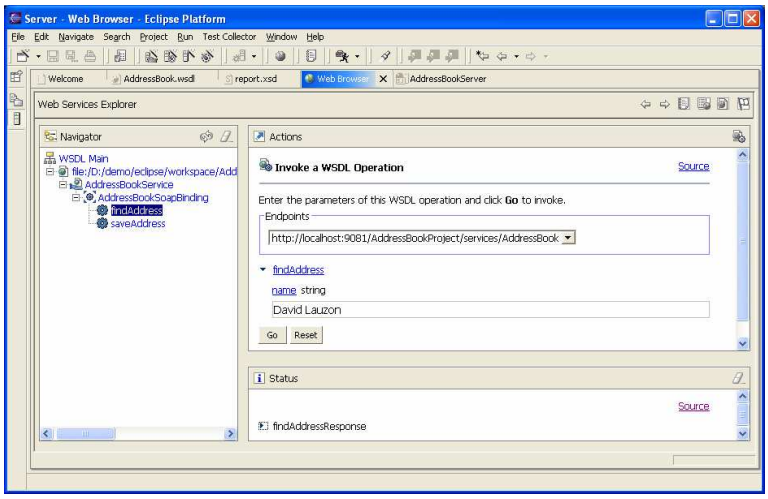
- Creating the WSDL file using the WSDL Editor



Deployment: Interoperable Web Service

- Our Second Task … WebSphere Application Server

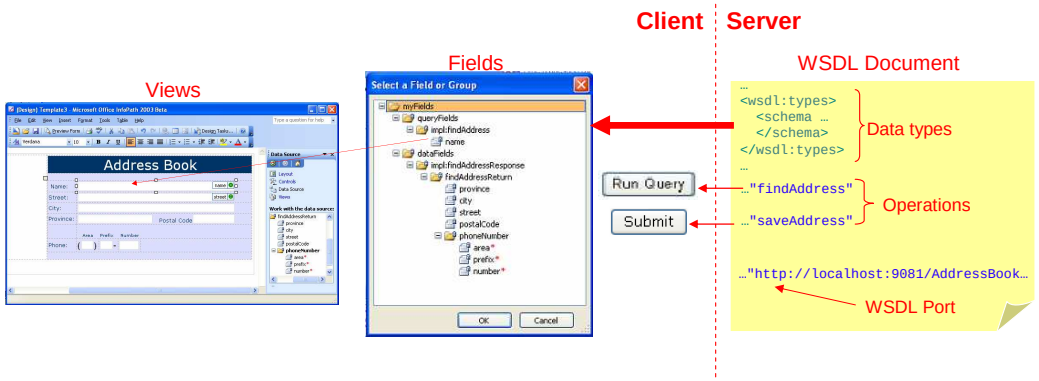
  Deploy and test the web service on



Development - Client by Microsoft

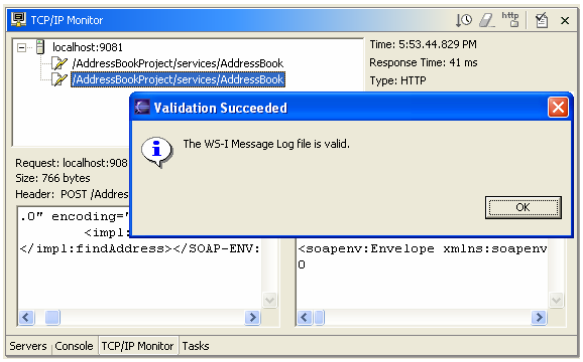
- Our Third Task …

  Use Microsoft Office InfoPath 2003 to create a client



Validating - Monitor by IBM

- Our Last Task …
  Use WebSphere Studio Application Developer
  to capture and validate traffic between the
  Microsoft client and IBM service



Section

New Features in V6

New Web Service Features

- New Features are very focused and unlikely to be needed in average use cases
- Based on specific customer requests for new functionality
  - Custom Bindings
  - Support for generic SOAP elements
  - Multi-Protocol support

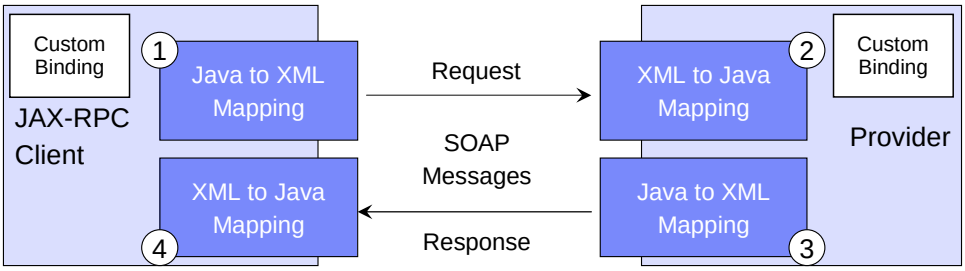
Support for Custom Bindings

- JAX-RPC does not support all schema types
  - Unsupported types can be mapped to literal XML elements represented as SOAP elements
  - This works well for data-centric applications, but not very well for type-centric applications
  - Users may want to map schemas to custom or legacy Java types
- A new CustomBinder interface is provided to be implemented by the binding provider
  - A CustomBinder deals with a particular pair of XML schema type and Java type
  - 2 primary methods; Serialize and Deserialize

JAX-RPC specification defines the standard Java mapping for various XML schema types, unfortunately, due to the complexity of XML schema, there are still some XML schema types which are either essentially un-mapable or very difficult to define mappings, for example, xsd:choice, xsd:anyType, xsd:anyAttribute. To deal with such types, JAX-RPC specification maps them to javax.xml.soap.SOAPElement if the WSDL dictates a literal representation for these types. As a result, SOAPElement becomes part of the Java method signature.

Support for Custom Bindings: Example

- JAX-RPC binds XML data to specific Java types
  - Certain designs may want to override this automatic choice
- Custom Bindings allow the developer to choose the mapping rules
- The SOAP message must use the Literal style
  - Document/Literal or RPC/Literal



JAX-RPC specification defines the standard Java mapping for various XML schema types, unfortunately, due to the complexity of XML schema, there are still some XML schema types which are either essentially un-mappable or very difficult to define mappings, for example, xsd:choice, xsd:anyType, xsd:anyAttribute. To deal with such types, JAX-RPC specification maps them to javax.xml.soap.SOAPElement if the WSDL dictates a literal representation for these types. As a result, SOAPElement becomes part of the Java method signature. In the diagram we see the flow of a message. The message begins as Java data types in the client. These are mapped to XML and sent in a SOAP message to the service provider, where the XML is mapped back Java types. If the provider returns a response, the same process occurs again. A Custom Binding provided by the developer would be used by the client or provider to map to different Java types then those specified in JAX-RPC.

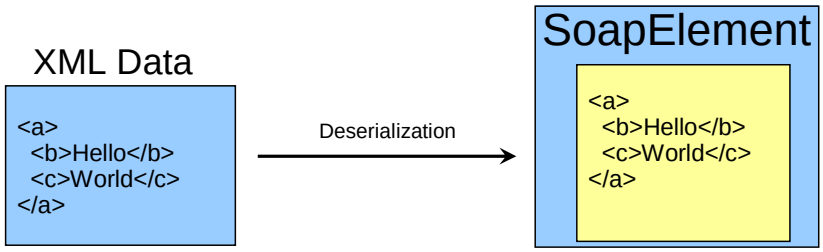
Support for Generic SOAP Elements

- Support For Custom Bindings allows for the specified JAX-RPC bindings to be extended or changed
- Certain designs may prefer to eliminate the binding completely
  - Support for generic SOAP elements gives the developer the ability to disable the normal deserialization process
  - Improves performance of Services that do not require their XML bound to Java types
- Normal binding:
  - public Bean echo(Bean bean);
- Generic binding:
  - public SOAPElement echo(SOAPElement bean);

The –noDataBinding option disabled the “data binding” of xml constructs to java objects. Instead, all parameters and returned values of the SEI methods will be bound to the SAAJ SOAPElement objects.

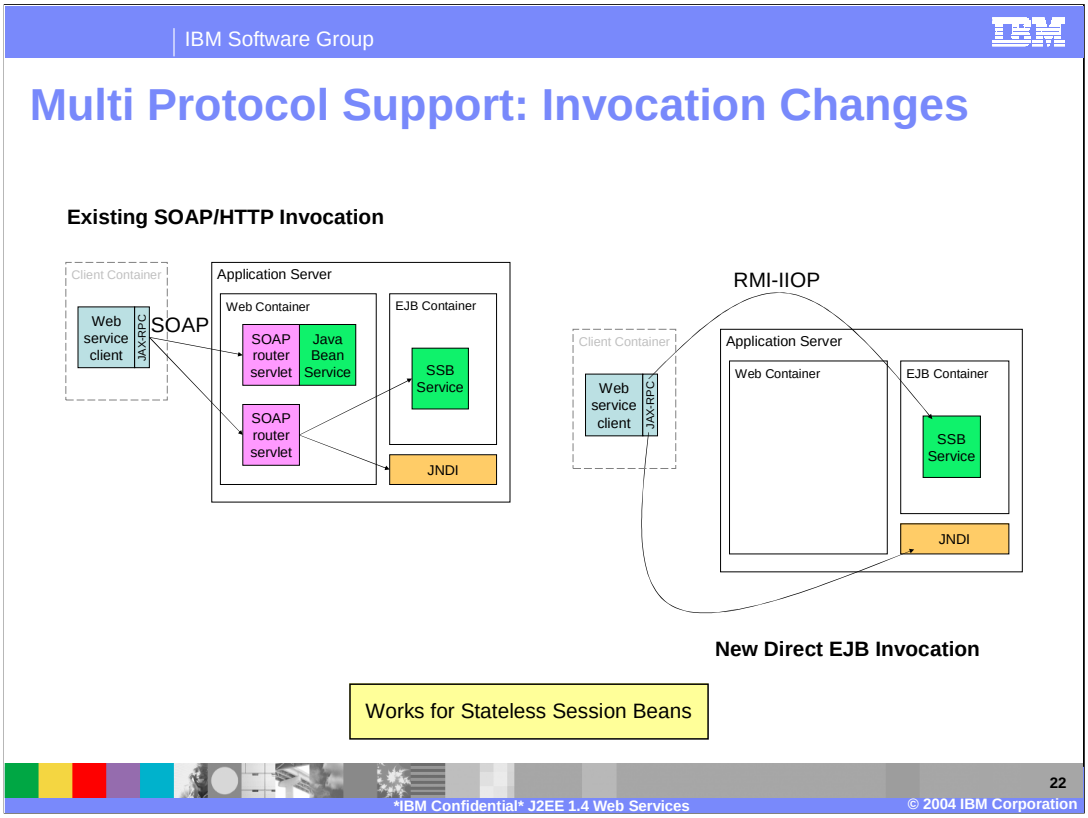
Generic Elements: Example

- WSDL2Java contains a new option –noDataBinding
  - Disables normal deserialization, instead all objects will be returned bound to the SAAJ SOAPElement API
- The SAAJ API will provide a new method for accessing the xml string that represents the SOAPElement tree
  - Public final String toXMLString(…)



Multi Protocol Support

- Extends JAX-RPC support for invoking remote stateless session EJBs with RMI-IIOP
  - More performant method for calling EJB services
- This allows managed clients (defined by JSR 109) to access Web Services through a number of protocols
  - No changes to JAX-RPC client are needed
  - May be extended to include other protocols in the future



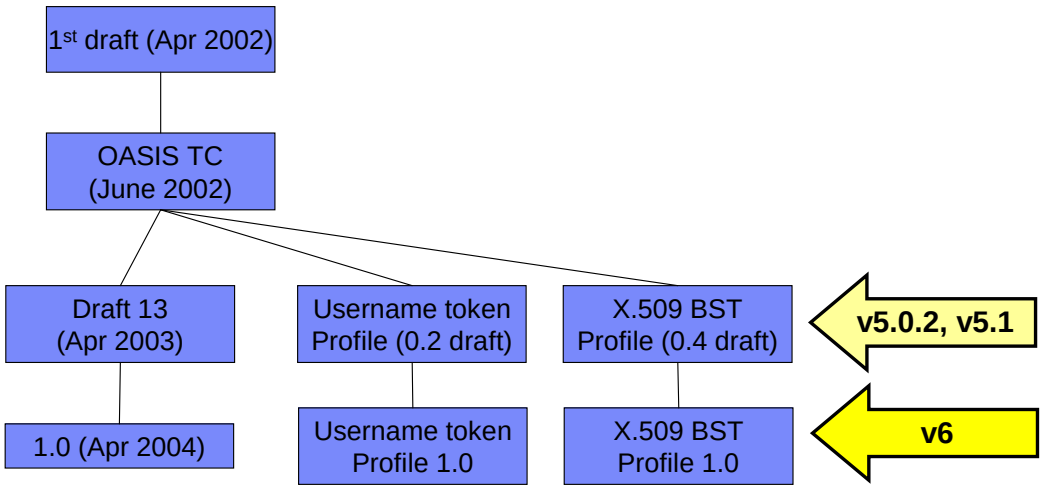
A web service client either running in a managed container (application client container, web container or EJB container) or running unmanaged from a J2SE client or non-Java client, invokes a web service implemented either as a stateless session bean or a java bean behind a router servlet. The HTTP transport protocol is implied by the diagram, however, the existing web services support also includes the option for a messaging (JMS compliant) transport such as MQSeries.

The new scenario above removes the servlet step by essentially implementing its functionality underneath the JAX-RPC API on the client and using RMI/IIOP to communicate with the service instead of SOAP. Using this approach a service can be invoked in a way it finds more natural. SOAP enabled stateless session bean services are fronted by a 'router module'; by providing a direct route to the EJB, the overhead of the router module is removed.

Section

WS-Security Enhancements in v6

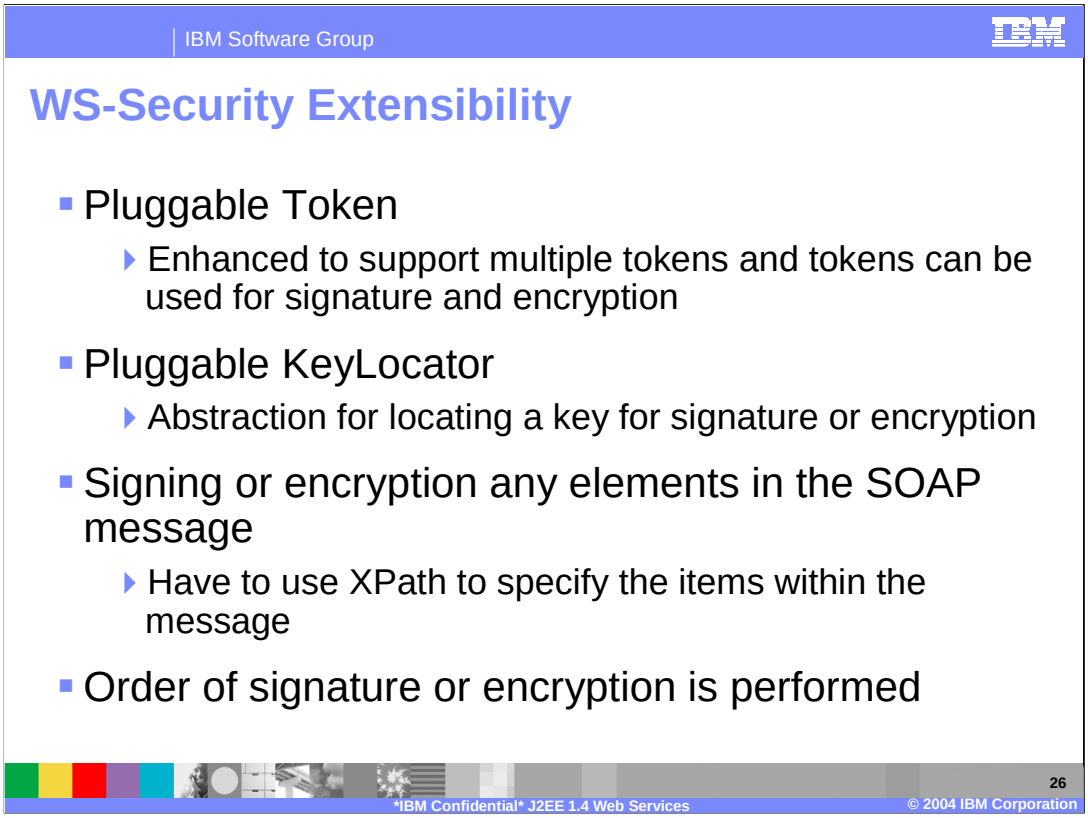
WS-Security Specifications in WebSphere



One of the major design goals is to make WS-Security implementation extensible. This is an important requirement, as the WS-Security specification is very flexible and we can not cover all the possible combinations allowed in the specification. Also, the pluggable architecture is capable to allow others to implement other Web services security specifications like WS-Trust client or WS-SecureConversation, or WS-SecurityKerberos profile.

There will be a technical preview of base Kerberos support as part of WAS V6.

Since the JSRs are still in progress or review status, there are no APIs exposed for WS-Security in 6.0. We use the deployment model to express the security constraints.



In addition to provide completion of the specification, a highly flexible architecture is required so that developers can implement other specifications such as WS-Trust, WS-SecureConversation and WS-SecurityKerberos on the current architecture.

Although we provide a base framework, default implementation classes are provided to support basic functions of WS-Security and profiles so that developers can use the functions without any development. The default implementation classes follow:

X509TokenGenerator: Generates a X509Token object, wrapping X.509 certificate which is returned by X509CallbackHandler

X509CallbackHandler: Retrieves X.509 certificate from key store files. Note that this is not specialized for WS-Security, so it would be defined as WAS-generic SPI.

UsernameTokenGenerator: Generates a UsernameToken object wrapping username and password return by a CallbackHandler such as GUIPromptCallbackHandler.

KeyStoreKeyLocator: Retrieves keys from key store files for signature and encryption

X509TokenConsumer: Processes BinarySecurityToken which includes X.509 certificate, invoking X509LoginModule. An object of X509Token class is created to store in JAAS Subject.

X509LoginModule: Authenticates X.509 certificate. Note that this is not specialized for WS-Security, so it would be defined as WAS-generic SPI.

UsernameTokenConsumer: Processes UsernameToken, invoking UsernameLoginModule. An object of UsernameToken class is created to store in JAAS Subject.

X509TokenKeyLocator: Retrieves a public key from a X.509 certificate which is stored in JAAS Subject as a form of X509Token by X509TokenConsumer

UsernameToken: Represents UsernameToken element, thus has properties for username and password.

X509Token: Represents BinarySecurityToken for X.509 certificate

Backward Level Support for Services

- Web Services with WS-Security in WebSphere v6 have different deployment descriptors than services in v5.X
- WebSphere v6 will include support for J2EE 1.3 services using earlier versions of WS-Security
  - OASIS WS-Security draft 13
- The Admin Console will provide different screens to configure back-level security for back-level services

Section

Web Services Gateway Architecture

WSGW Overview

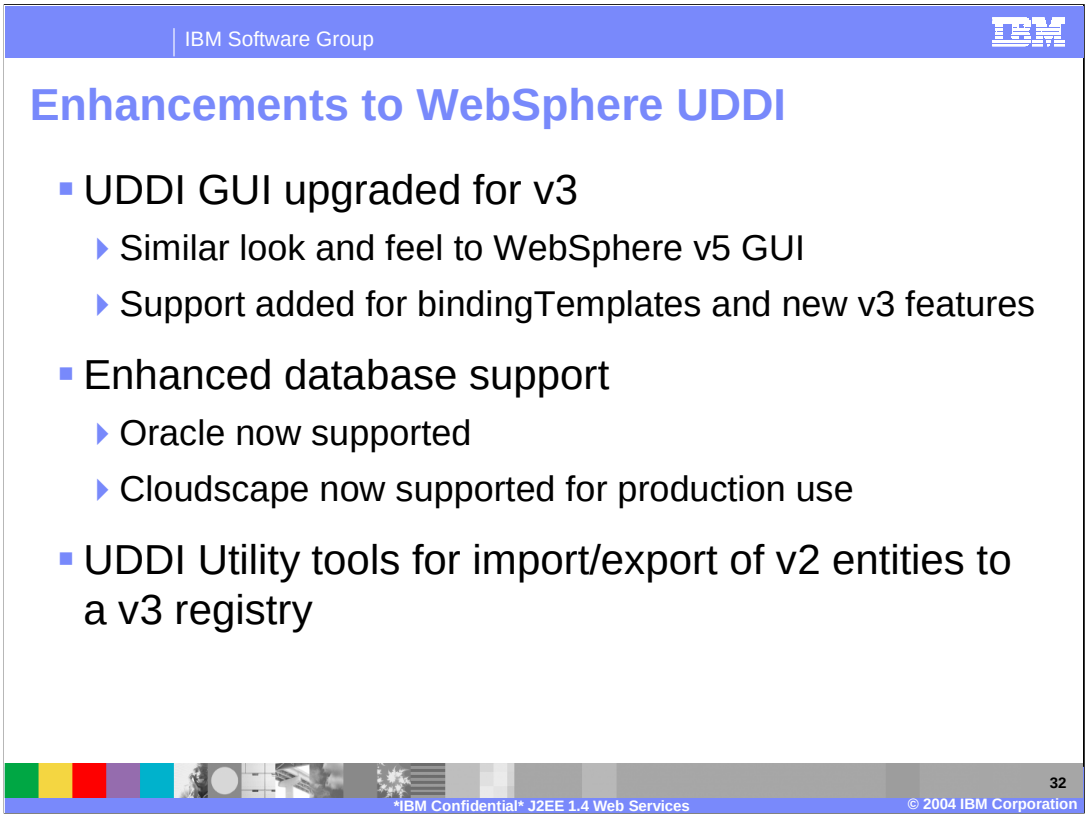
- Known as the Service Integration Bus Web Services Enablement (SIBWS) in v6
- Provides integrated support for Web Services communications using the Service Integration Bus
- Separate installation from the Application Server install
  - Available under <install_root>/installableApps
  - Detailed install instructions available in infocenter

SIBWS Benefits

- Take an internal service that is available at a service destination, and make it available as a Web service
- Take an external Web service, and make it available at a service destination
- Map an existing service – internal or external - to a new Web service that appears to be provided by the gateway
- Change the transport protocol used by the web service at the gateway
- Enable WS-Security on services at the gateway

Section

UDDI and JAXR



- Level of support provided by GUI is similar to that provided in v2. As before, the UDDI GUI is intended for familiarisation with UDDI structures, and for finding data; complex publications are best achieved programmatically.
- The set of supported databases will be further extended in a later v6 release
- IBM UDDI v3 Client for Java is a JAX-RPC style client based on Axis, and is the v3 equivalent of UDDI4J. UDDI4J is still supported, but deprecated. In the Beta, the UDDI v3 client is based on Axis but will probably be Maelstrom-based at GA time
- Utility tooling was introduced in v5.1 to allow data to be exported from one registry and imported into another. This is for import/export of v2 entities from a v2 or v3 registry to a v3 registry. A tool for import/export of v3 entities is not needed as the v3 registry supports adding entities with a supplied key via the normal v3 API
- Custom taxonomy support, allowing users to create their own categorization schemes or value sets, was introduced in v5.0.2. “Value Set” is the terminology used in the v3 specification

Note that the WebSphere v3 registry will still accept v2 requests (and v1)

What is JAXR?

- A uniform and standard API for accessing registries within Java platform
- The goal of JAXR is to enable interoperability between diverse clients and registries
- Provides a union of best features of dominant registry specs (particularly UDDI and ebXML)
  - Current version (1.0), supports **UDDI v2 only**
  - Does not map precisely to UDDI
    - for a precise API mapping, IBM provides UDDI4J and IBM Java Client for UDDI v3

The states goals of the specification are to:

1. Define a general purpose Java API for accessing business registries that allows any JAXR client to access and interoperate with any business registry that is accessible via a JAXR provider.
2. Define a pluggable provider architecture that enables support for diverse registry specifications and standards.
3. Support a union of the best features of dominant registry specifications rather than a common intersection of features. *JAXR is not a least common denominator API*.
4. Ensure support for dominant registry specifications such as ebXML and UDDI, while maintaining sufficient generality to support other types of registries, current or future. 547
5. Ensure synergy with other Java specifications related to XML.

Important Note: the current JAXR specification is written against UDDI **Version 2**.

Currently there are a variety of specifications for XML registries. These include:

The ebXML Registry and Repository standard, which is sponsored by the Organization for the Advancement of Structured Information Standards (OASIS) and the United Nations Centre for the Facilitation of Procedures and Practices in Administration, Commerce and Transport (U.N./CEFACT)

The Universal Description, Discovery, and Integration (UDDI) project, which is being developed by a vendor consortium

eCo Framework, developed as part of the eCo Framework Project. This was chartered by CommerceNet in August, 1998, with Commerce One as the primary corporate sponsor, and focuses on business to business solutions

JAXR compared with UDDI4J and IBM Java Client for UDDI v3 is discussed in further detail in a later section of this presentation

JSR093 Group contained individuals from Extol Inc., IONA, Oracle, Bowstreet, IBM Corporation ,Hewlett-Packard Company ,Tibco Extensibility Inc., Cyclone Commerce, Sun Microsystems, CISCO, Encoda Systems, Inc., webGain, SilverStream, webMethods Corporation, BEA Systems

JAXR Capability Profiles

- JAXR API methods are categorized by a Capability Profile
  - Level 0 – UDDI v2 support
  - Level 1 – ebXML support and Level 0 support
- JAXR Provider must declare the capability level for its implementation
- IBM's JAXR implementation is Level 0 compliant

The JAXR API categorizes its API methods by a small number of capability profiles. Currently only two capability profiles are defined (level 0 and level1). The capability level is defined in the API documentation for each method in a class or interface in the JAXR API.

There is no assignment of capability level to interfaces and classes in the JAXR API. Capability assignment is done at the method level only.

A JAXR provider must declare the capability level for its implementation of the JAXR API. A JAXR client may discover a JAXR provider's capability level by invoking methods on an interface named CapabilityProfile as defined by the JAXR API. If a JAXR provider declares support for a specific capability level then it implicitly declares support for lower capability levels. For example, a JAXR provider that declares support for the level 1 profile implicitly declares support for level 0 profile.

A JAXR provider must implement the functionality described by the JAXR API for each method that is assigned a capability level that is less than or equal to the capability level declared by the JAXR provider.

A JAXR provider must implement all methods that are assigned a capability level that is greater than the capability level declared by the JAXR provider, to throw an UnsupportedCapabilityException. A JAXR provider must never implement any other behavior for methods assigned a greater than the capability level declared by the JAXR provider. The reason for this restriction is that it is necessary to ensure portable behavior for JAXR clients for any JAXR provider within a specific capability level.

Support for the level 0 profile is required to be supported by all JAXR providers. The methods assigned to this profile provide the most basic registry capabilities. JAXR providers for UDDI must be level 0 compliant.

Support for the level 1 profile is optional for JAXR providers. The methods assigned to this profile provide more advanced registry capabilities that are needed by more demanding JAXR clients. JAXR providers for ebXML must be level 1 compliant.

Examples of Level 1 capability:

The JAXR RegistryPackage (used to group logically related RegistryObjects together) is Level 1 only

The DeclarativeQueryManager interface, which provides a more flexible search API allowing SQL queries, is Level 1 only

Section

Summary and Reference

Summary

- Discussed
  - Web Services support in WebSphere v6
  - Creating a WS-I compliant service
  - New Web Services functions
  - WS-Security enhancements
  - New features for the Gateway
  - Support for UDDI V3 and JAXR

Resources: JSR 101, 109

- JSR 101 (JAX-RPC)
  - http://java.sun.com/xml/jaxrpc/index.html
- JSR 109
  - http://jcp.org/jsr/detail/109.jsp
  - http://www-106.ibm.com/developerworks/webservices/library/ws-jsr109/index.pdf
- Introduction to Web Services
  - http://java.sun.com/webservices/docs/ea2/tutorial/doc/IntroWS.html
- WS-I Basic Profile
  - http://www-106.ibm.com/developerworks/webservices/library/ws-basicprof.html?dwzone=webservices

Resources: Standards and Organizations

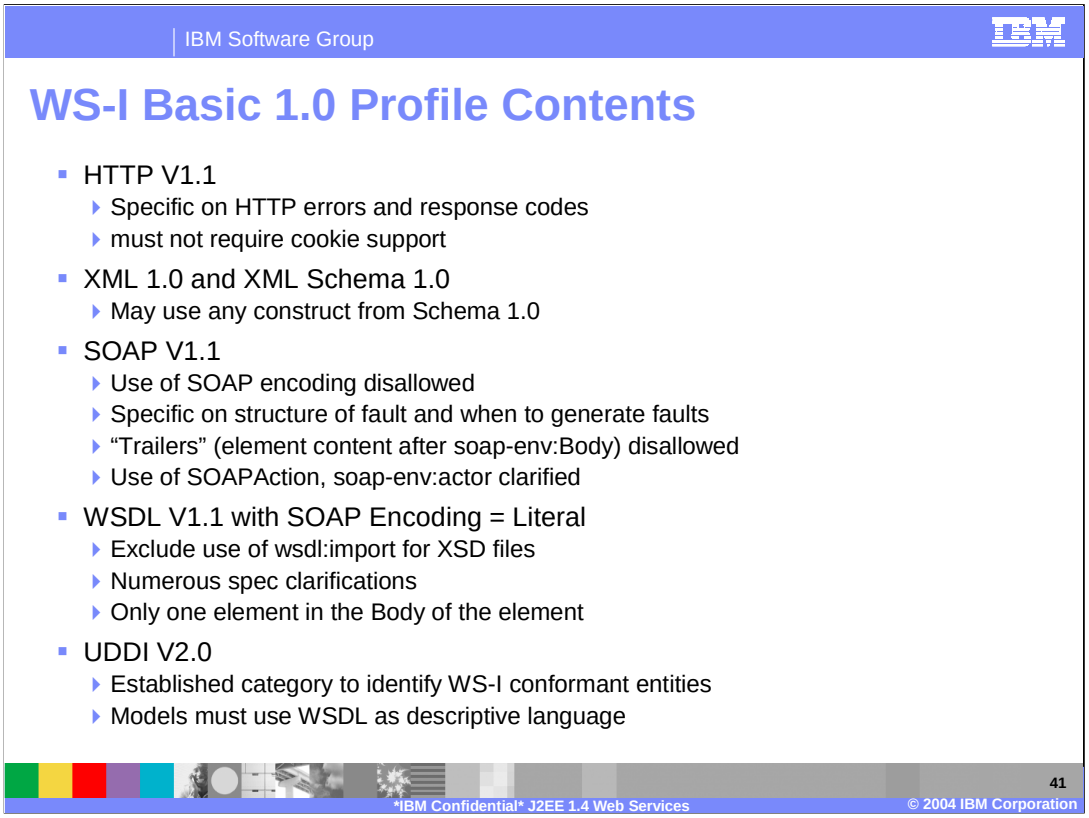
- http://www.w3.org/TR/SOAP/
- http://www.w3.org/TR/wsdl
- http://www.uddi.org
- http://www.WS-I.org
- http://xml.apache.org/soap
- http://www.xmethods.com
- http://www.oasis-open.org/

Resources: IBM, WSAD, Eclipse

- 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

Section

Appendix



These are the different specifications which make up the WS-I Basic 1.0 Profile. Vendors will strive to become WS-I Basic 1.0 compliant rather than claim compliance with the individual specifications. With this action, interoperability is more likely to be obtained between different Vendors.

Section

Detailed Information on New Features

Section

Support for Custom Bindings

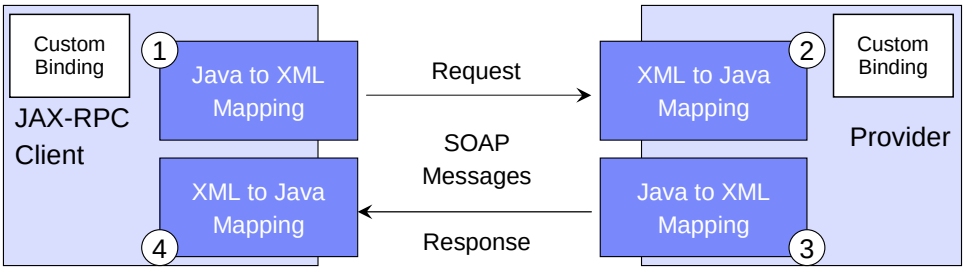
Support for Custom Bindings

- JAX-RPC does not support all schema types
  - Unsupported types can be mapped to literal XML elements represented as SOAP elements
  - This works well for data-centric applications, but not very well for type-centric applications
  - Users may want to map schemas to custom or legacy Java types
- A new CustomBinder interface is provided to be implemented by the binding provider
  - A CustomBinder deals with a particular pair of XML schema type and Java type
  - 2 primary methods; Serialize and Deserialize

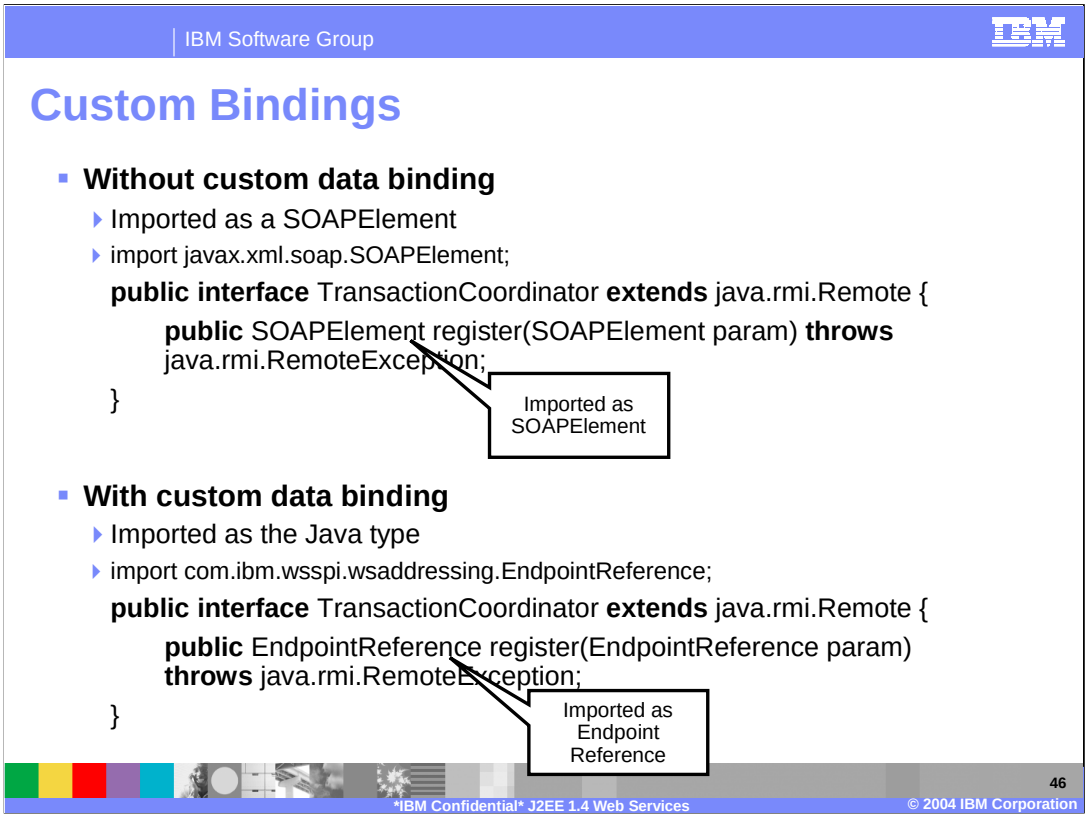
JAX-RPC specification defines the standard Java mapping for various XML schema types, unfortunately, due to the complexity of XML schema, there are still some XML schema types which are either essentially un-mapable or very difficult to define mappings, for example, xsd:choice, xsd:anyType, xsd:anyAttribute. To deal with such types, JAX-RPC specification maps them to javax.xml.soap.SOAPElement if the WSDL dictates a literal representation for these types. As a result, SOAPElement becomes part of the Java method signature.

Support for Custom Bindings: Example

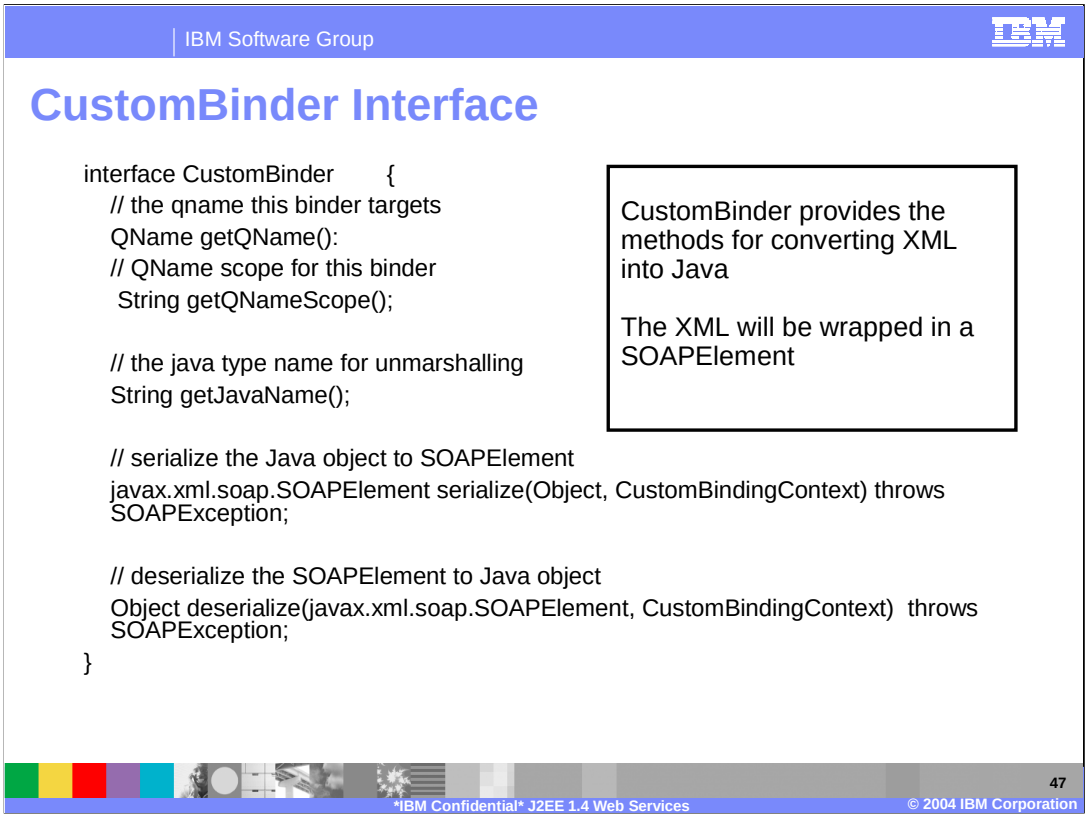
- JAX-RPC binds XML data to specific Java types
  - Certain designs may want to override this automatic choice
- Custom Bindings allow the developer to choose the mapping rules
- The SOAP message must use the Literal style
  - Document/Literal or RPC/Literal



JAX-RPC specification defines the standard Java mapping for various XML schema types, unfortunately, due to the complexity of XML schema, there are still some XML schema types which are either essentially un-mappable or very difficult to define mappings, for example, xsd:choice, xsd:anyType, xsd:anyAttribute. To deal with such types, JAX-RPC specification maps them to javax.xml.soap.SOAPElement if the WSDL dictates a literal representation for these types. As a result, SOAPElement becomes part of the Java method signature. In the diagram we see the flow of a message. The message begins as Java data types in the client. These are mapped to XML and sent in a SOAP message to the service provider, where the XML is mapped back Java types. If the provider returns a response, the same process occurs again. A Custom Binding provided by the developer would be used by the client or provider to map to different Java types then those specified in JAX-RPC.



Mapping to SOAPElement is fine for certain usages, especially for data centric applications, but it's not desirable for type centric applications because SOAPElement deals with the raw SOAP message itself, not the predefined Java type. Even for data centric applications, SOAPElement may not be always the desirable choice either, user may desire for other data objects such as SDO.



The custom binder is an interface to be implemented by the binding provider to deal with a particular pair of XML schema type and Java type. *CustomBinder* interface has two primary methods: *serialize* and *deserialize*; and both methods deal with a Java object and *javax.xml.soap.SOAPElement* (for brevity, use “SOAPElement” for the rest of this document when appropriate). Once the binder is recognized by the engine runtime, it communicates with the runtime via the SOAPElement in the following ways:

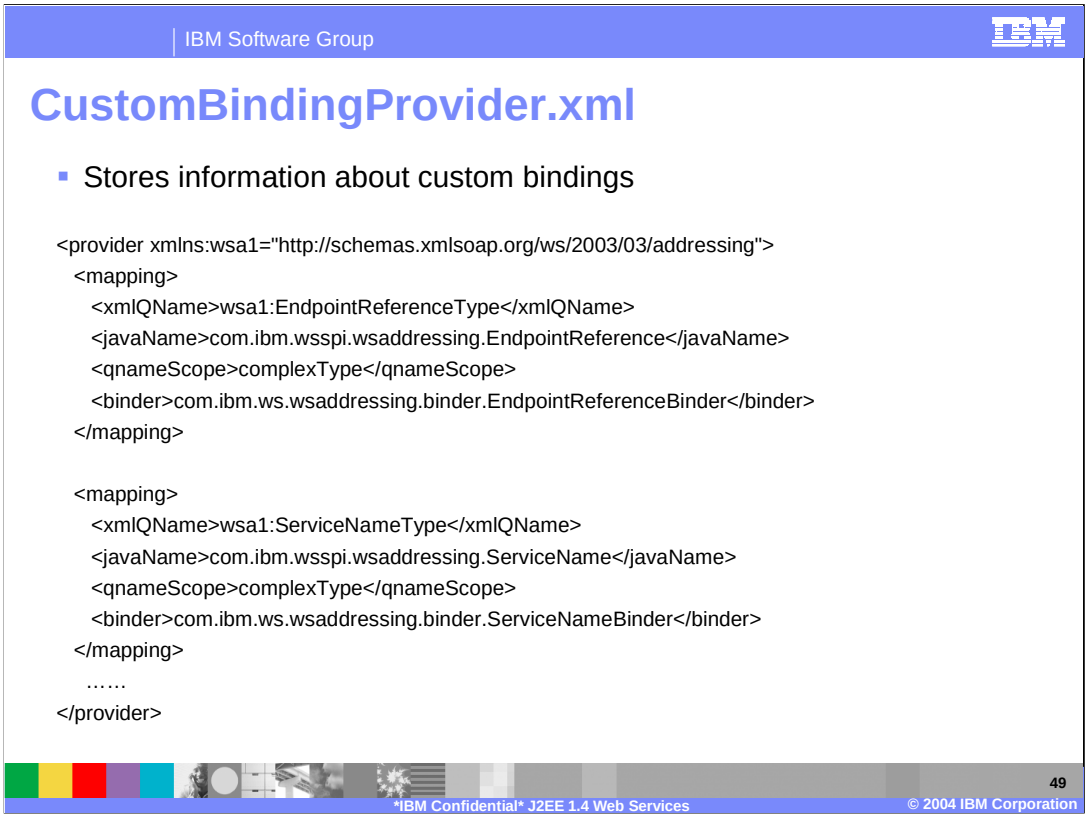
For serialization, the binder is invoked by the runtime and it receives a Java object and a SOAPElement. Java object is the data to be serialized, and SOAPElement is the context element for the serialization. Unlike the conventional serializer which writes out the raw data, the custom binder here is to produce an intermediate form of SOAP message: SOAPElement. The runtime takes care of writing the SOAPElement to the raw data. Obviously, it’s easier to create the SOAPElement rather than the raw text since SAAJ APIs are available to help.

Similarly for deserialization, the runtime builds an appropriate *javax.xml.soap.SOAPElement* instance and passes it to the binder which then deserializes it to a Java object.

QNameScope

- The CustomBinder interface contains an attribute qNameScope
  - Indicates whether the binder deals with the named type or the anonymous type in the XML
  - The value for qnameScope is ‘element’ for the anonymous types
  - Or a value of ‘complexType’ or ‘simpleType’ for named types
- To combine a number of custom bindings to support a custom application a Custom Binding Provider can be used
  - Normally created for a specific custom schema file which requires the custom data binding
  - Declared in the /META-INF/services/CustomBindingProvider.XML file
  - Provided as part of the jar file

The custom binder has an attribute named “qnameScope” to indicate whether it deals with the named type (i.e. complexType, simpleType) or the anonymous type (i.e. element). As such, the value for “qnameScope” is either element, complexType or simpleType. To some extent, the “qnameScope” affects the contract between the runtime and binder.



The provider is declared through a specific XML file (/META-INF/services/CustomBindingProvider.xml) which includes all the necessary information for the tool and runtime to locate the right binder. This XML file is normally packaged along the binder classes in the same jar file.

xmlQName: The XML named type

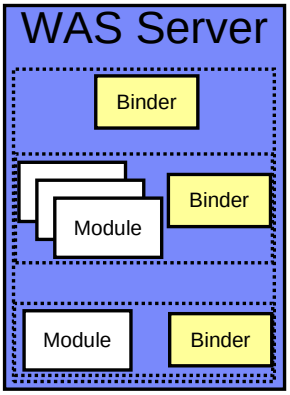
javaName: The Java names type

qnameScope: The scope of the type

Binder: The name of the custom binder to be used when this type is encountered

Binding Levels

- Depending on where the Custom Binding Provider is located it has different levels of visibility
  - Custom Binding Provider is packaged in a jar file
- Packaged with an application module
  - Visible only to the module
- Configured as part of a shared library
  - Visible to any module sharing the library
- Placed in the WAS system directory
  - Visible to the entire WAS runtime



Provider Discovery

- **Provider Discovery**
  - Locate providers at /META-INF/services/CustomBinderProvider.xml within supplied jar files
  - Runtime:
    - System level provider: jar files have to be visible to WAS runtime classloader such as $WAS/lib or $WAS/classes
    - Application level provider: jar files have to be visible to application classloader such as /WEB-INF/lib
  - Command line options
    - -classpath besides the system level discovery
- WSDL2Java
  - Discover all custom binding providers
  - For each xml type encountered, query the providers to obtain the Java type
  - Burn the custom data binding information into the stub

Section

Support for Generic SOAP Elements

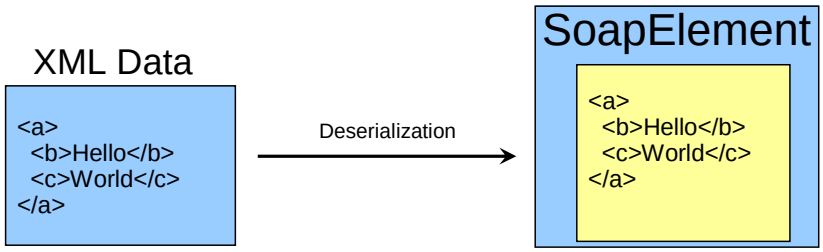
Support for Generic SOAP Elements

- In normal JAX-RPC flows, SOAP elements are deserialized into Java data types
- Support For Custom Bindings allows for the specified JAX-RPC bindings to be extended or changed
- Certain designs may prefer to eliminate the binding completely
  - Support for generic SOAP elements gives the developer the ability to disable the normal deserialization process
  - Improves performance of Services that do not require their XML bound to Java types

The –noDataBinding option disabled the “data binding” of xml constructs to java objects. Instead, all parameters and returned values of the SEI methods will be bound to the SAAJ SOAPElement objects.

Generic Elements: Example

- WSDL2Java contains a new option –noDataBinding
  - Disables normal deserialization, instead all objects will be returned bound to the SAAJ SOAPElement API
- The SAAJ API will provide a new method for accessing the xml string that represents the SOAPElement tree
  - Public final String toXMLString(…)



Generic Elements: Binding Example

- Normal binding:
  - public Bean echo(Bean bean);
- Generic binding:
  - public SOAPElement echo(SOAPElement bean);
  - the SEI exposes the SAAJ interface
  - Use the toXMLString() method to retrieve the xml content from the SOAP element

The DOM tree isn't created until the SOAPElement object is queried. This will defer deserialization. Here we see an example of the changed that would occur to an object of type Bean when you use no data binding. In the normal binding the method invocation uses type Bean. With a generic binding they type becomes SOAPElement and you would have to use the toXMLString() method within the service to retrieve the XML content from the SOAPElement.

Changes in Web Services Engine

- **There are a number of changes to the Web Services Engine to support this change**
  - Under normal conditions a full SAAJ tree is constructed when an SAAJ element is deserialized
  - The engine will detect usage of the SAAJ implementation and in these cases the XML string will be attached to the SOAPElement
  - IBM's SOAPElement implementation provides mechanisms for storing data in alternate, optimal, structures such as xml string
- **Reasons for using the generic element**
  - The service may be a conduit to another service, in this case the message is only being forwarded
  - The message may need to be manipulated by a different data model (SDO), using the generic element makes it easier to convert
  - A handler may need to manipulate the message in a more generic manner

The DOM tree isn't created until the SOAPElement object is queried. This will defer deserialization.

Section

Web Services Client Caching

What's New?

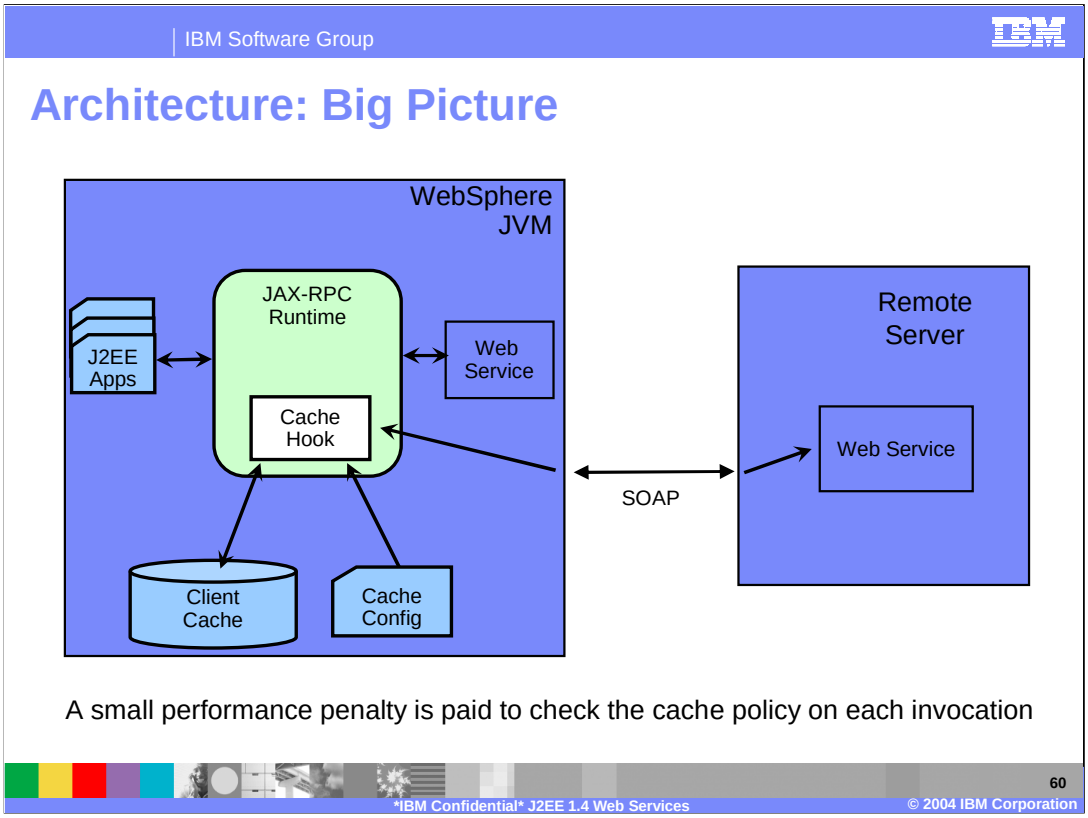
- WebSphere v5.0 can cache
  - Servlets
  - JSPs
  - Java objects
  - Commands
  - Server-side web services
- Can cache portions of pages and responses
- V6.0 adds the capability to cache web service responses within the client's application server

WebSphere 5.0 and 5.1 already contained capability to cache certain resources. 5.1.1 and 6.0 is adding the capability to apply a cache when WebSphere is supporting the client application.

Client Caching

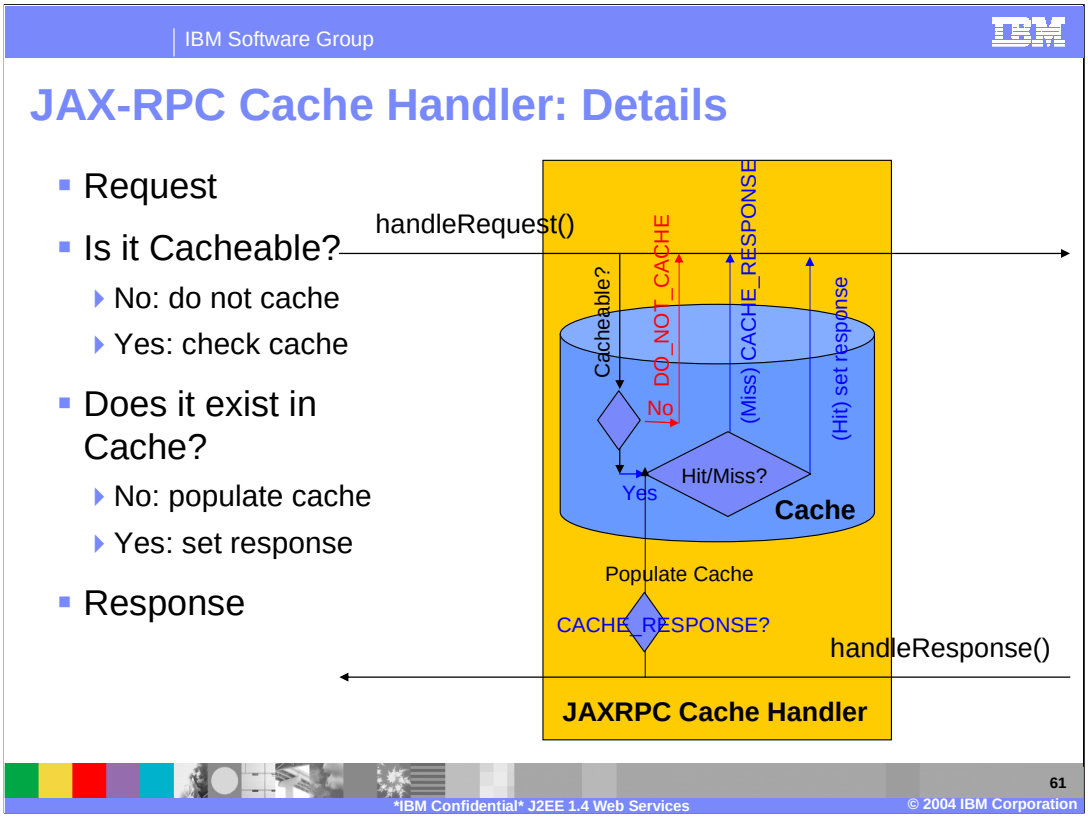
- Increases the performance of Web Services clients by caching responses from remote Web Services
  - Once a response is cached, subsequent calls to the same web service with the same set of request parameters could be responded from cache
- Provided as a JAX-RPC handler
  - Based on the policy specified in the cachespec.xml file
- Choice of methods to invalidate cached values
  - Rule Based, Time Based, APIs

Cache JAX-RPC handler needs to identify a cache policy based on the target web service. Once a policy is found, all the cache id rules specified in the policy will be evaluated one by one until a valid rule is detected. Cache id rules for web services client cache could use service name, port, operation and parameters to identify requests and entries in cache. Similarly, cache ids could also be built using special SOAP header fields if the client generating these SOAP envelopes could add them for better performance.



Within the JVM the JAX-RPC runtime has a hook to the caching service. When a client request comes into the RPC runtime, it is intercepted by the cache handler that checks the cache based on rules found in the cache config XML file. If it doesn't find the information in the Cache, then it will either call the web service within the same WebSphere server, or forward the call on to the target service located elsewhere. The web service can be local or remote to this server. The result would be placed in the cache before being returned to the client.

Cache JAX-RPC handler needs to identify a cache policy based on the target web service. Once a policy is found, all the cache id rules specified in the policy will be evaluated one by one until a valid rule is detected.

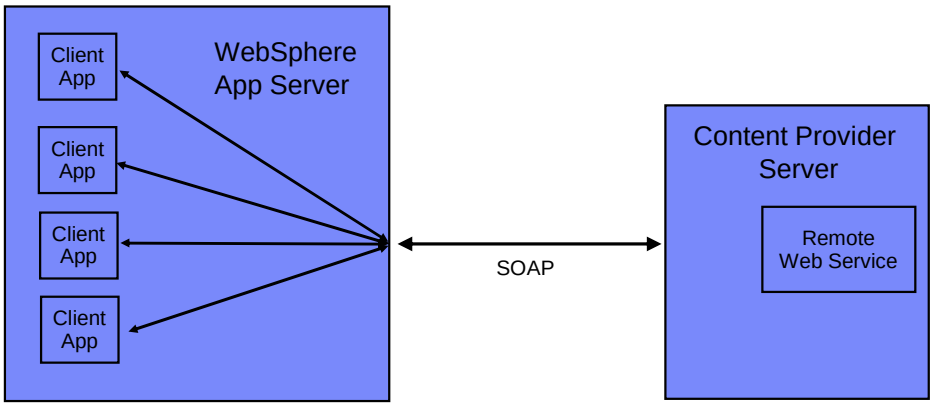


Web services client caching is provided as a JAX-RPC cache handler. In the handleRequest() method, cache config manager is searched for a cache policy based on the target endpoint address specified in the request. Request is not cacheable if a matching cache policy is not found. If a matching policy is found, all the cache id rules in that policy are executed one by one until a valid rule is identified. Result of the first valid cache rule will be the cache key for lookup. If this lookup ends in a cache miss, a property is added to the handler chain's message context to cache the response in handleResponse() method. If this lookup ends in a cache hit, the value from the cache is set as the response and the rest of the request handle chain is blocked. If a SOAP fault is returned, the response is not cached. Else it will be cached in handleResponse() method using the cache key specified in the message context.

Scenarios

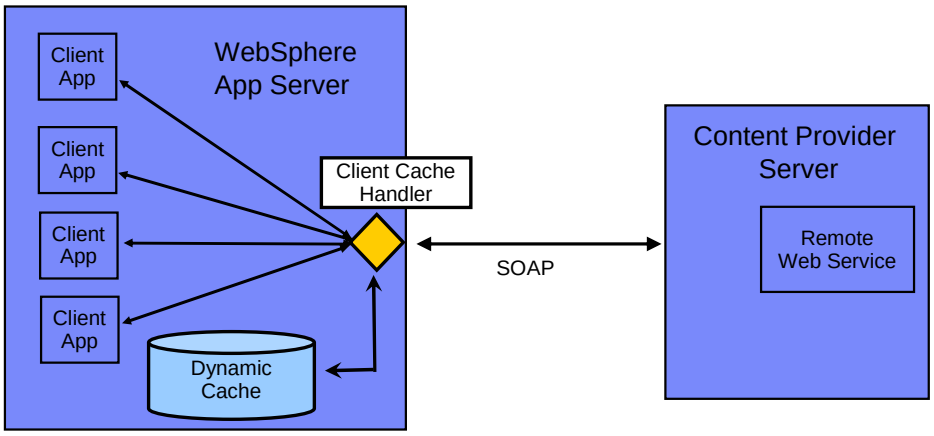
- Enterprise Applications hosted on Content Provider's network exchanging SOAP messages with Enterprise network
- Reverse proxy acting as a gateway by invoking Web Services
  - Proxy can respond without invoking target services
- Split-Tier setup
  - Both client and server running WAS.

External Content Provider Scenario without Cache

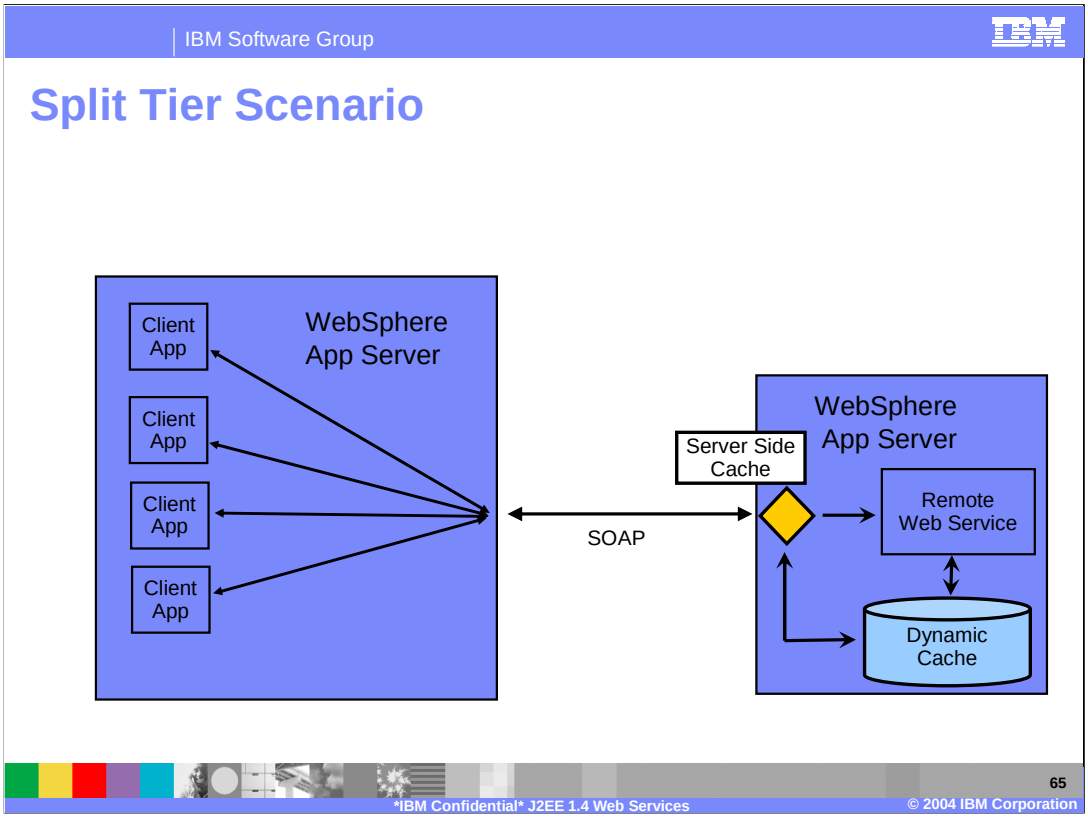


Provider is on the internet. Without caching all calls must travel over the internet. Thus calls to the service can be expensive. By adding a local cache, the cache will be checked before calling the service. This can result in significant performance increase.

External Content Provider Scenario with Cache



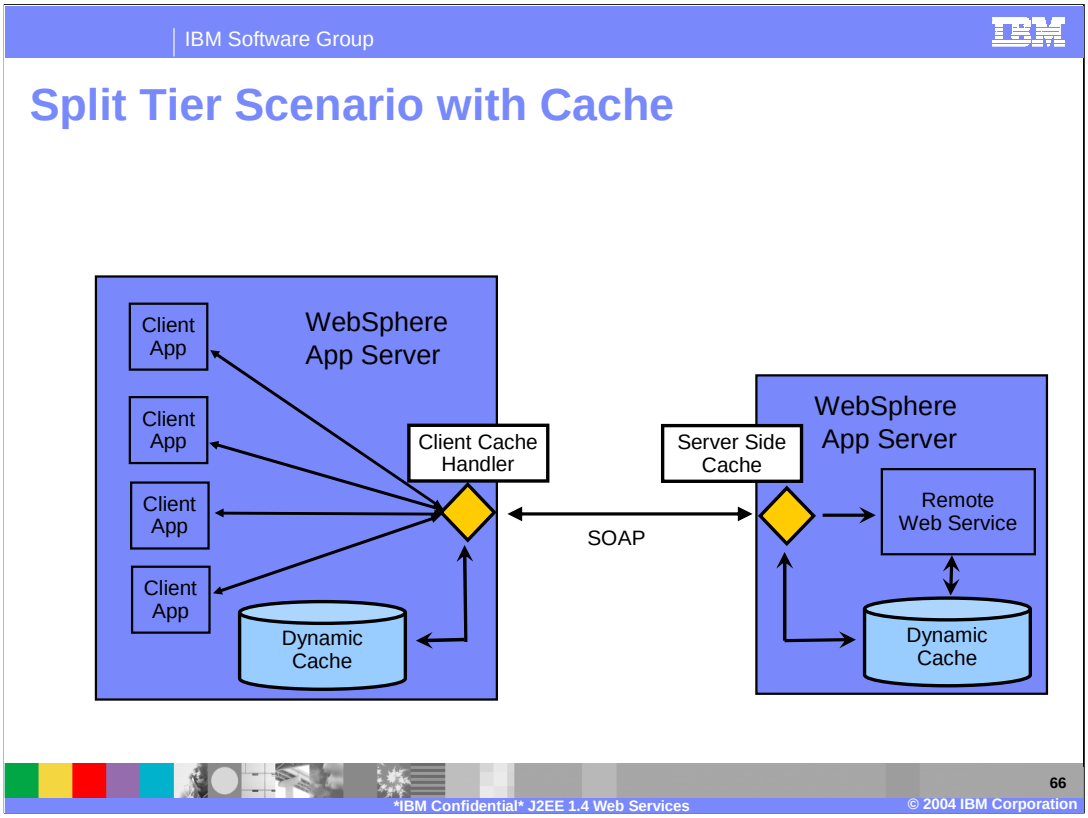
Provider is on the internet. Thus calls to the service can be expensive. By adding a local cache, the cache will be checked before calling the service. The cache will be checked before a request is sent across the internet. This can result in significant performance increase.



This type of caching is very beneficial in a reverse proxy scenario where a Web Services Gateway could respond to requests from cache instead of invoking backend services.

Provider is on the internet. Thus calls to the service can be expensive. By adding a local cache, the cache will be checked before calling the service. This can result in significant performance increase.

This shows the already provided provider side cache capability. Which helps improve performance by checking a cache on the target server. This will result in a performance increase, as objects may not need to be instantiated.



We can increase performance further by providing a second cache on the client side. This cache will be checked before a call is made over the internet. This type of cache can provide a greater performance increase for certain scenarios, and is compatible with a provider side cache as well.

Enabling JAX-RPC Cache

- JAX-RPC caching is enabled if Dynamic cache Service is enabled.
- Configure caching policy in cachespec.xml
  - New type of config entry “JAXRPCClient
  - Supporting new types “part”, “operation”
- The cachespec.xml file is found inside the WEB-INF directory of a Web module.
  - You can place a global cachespec.xml file in the application server properties directory, but the recommended method is to place the cache configuration file with the deployment module.

If dynamic cache is turned on within the administration console, and the presence of cache spec xml file. Cache spec is where the caching is configured. With a new type of entry, JAXRPCClient. This file can be found within the WebSphere application files. It is possible to turn on caching at a global server level, by placing the cache spec into the server properties directory, but this is not recommended. It’s preferable for performance and maintenance to keep it local to an application.

Caching Example

cachespec.xml and cache Ids

- Cache IDs are used to reference entries in the cache
- Cache id from SOAP header entries
  - Best performance
- Cache id from SOAP envelope
  - Hash code
- Cache id from SOAP Body
  - Operation and Part
  - Allows highest level of granularity

There are several ways to specify the way information gets cached. Particularly in the creation of cache ID's which are used by the WebSphere server to store and identify the values stored in cache. WebSphere can create the cache Id a number of ways. From the SOAP header, this is best from a performance perspective, as the body of the SOAP message doesn't need to be parsed in order to determine if the message is in the cache, but this is fairly course grained. The other options allows finer granularity and control over what is cached.

Sample WSDL

```
<definitions targetNamespace=http://TradeSample.com/...>
    <message name="getQuoteRequest">
        <part name="symbol" type="xsd:string"/>
    </message>
    .....
    .....
    <binding name="SoapBinding" type="tns:GetQuote">
        <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
            <operation name="getQuote">
                <soap:operation soapAction=""/>
                <input name="getQuoteRequest">
                    <soap:body namespace="http://TradeSample.com/" use="encoded"
                    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
                </input>
                ......
                </operation>
    </binding>
    <service name="GetQuoteService">
        <port binding="tns:SoapBinding" name="SoapPort">
            <soap:address location="http://TradeSample.com:9080/service/getquote"/>
        </port>
    </service>
</definitions>
```


Here we see portions of an example WSDL for a stockquote service. It contains a method for getQuote, which requires a parameter ‘symbol’ which would be the stock name like IBM. The various bolded areas are information you would need for chahe ID’s

Sample SOAP Request

```
POST /wsgwsoap1/soaprpcrouther HTTP/1.1
….
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope …>
<soapenv:Header>
  <getQuote soapenv:actor="com.ibm.websphere.cache">
    IBM
  </getQuote>
</soapenv:Header>
<soapenv:Body … >
  <getQuote xmlns="urn:ibmwsgw#GetQuoteSample">
    <symbol xsi:type="xsd:string">IBM</symbol>
  </getQuote>
</soapenv:Body>
</soapenv:Envelope>
```

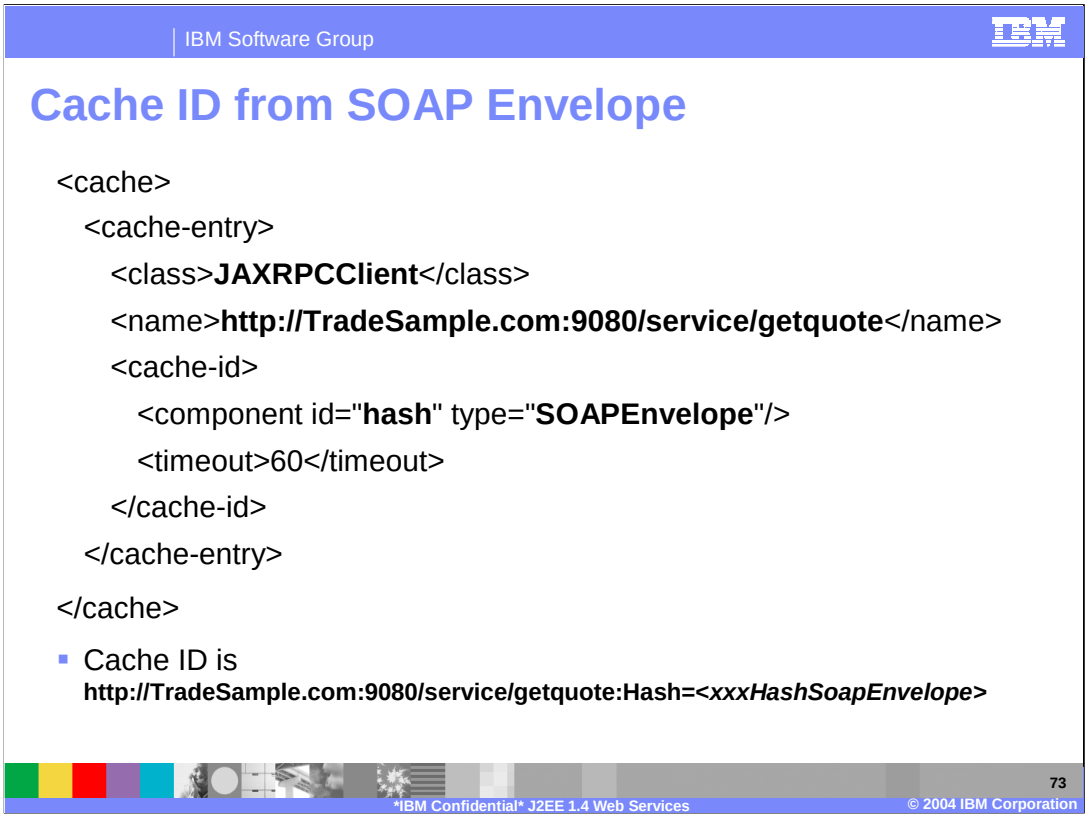

This is a simplified SOAP request. Showing the SOAP envelope and header. With the actor for the value stored in the cache being IBM. And the SOAP body with the getQuoteSample information and the string attribute IBM.

Cache ID from SOAP Header

```
<cache>
  <cache-entry>
    <class>JAXRPCClient</class>
    <name>http://TradeSample.com:9080/service/getquote</name>
    <cache-id>
      <component id="getQuote" type="SOAPHeaderEntry"/>
    </cache-id>
  </cache-entry>
</cache>
```

- Cache ID is **http://TradeSample.com:9080/service/getWQuote:getQuote=IBM**

With this example of a cache entry using the SOAP header to create the cache id, we see the cache entry class is JAXRPCClient. The name is the tradesample service getquote binding. The cache id generated from this is shown on the bottom. So from this cache entry example you would cache a response to the getQuote method for IBM.



This is an example of getting the information from the SOAP envelope. This is slightly less performant then the SOAP header example, as it requires some parsing of the SOAP message to retrieve this information. The name is the SOAP port coming in, and we are saying we want a hash on the SOAP envelope. As we see in our id value that is created containing the hash for the soap envelope.

Cache ID from SOAP Body

```
<cache>
  <cache-entry>
    <class>JAXRPCClient</class>
    <name>http://TradeSample.com:9080/service/getquote</name>
    <cache-id>
      <component id="" type="operation">
        <value>http://TradeSample.com/:getQuote</value>
      </component>
      <component id="symbol" type="part"/>
    </cache-id>
  </cache-entry>
</cache>
```

- Cache ID is
  http://TradeSample.com:9080/service/getquote:operation=http://TradeSample.com/:getQuote/symbol=IBM

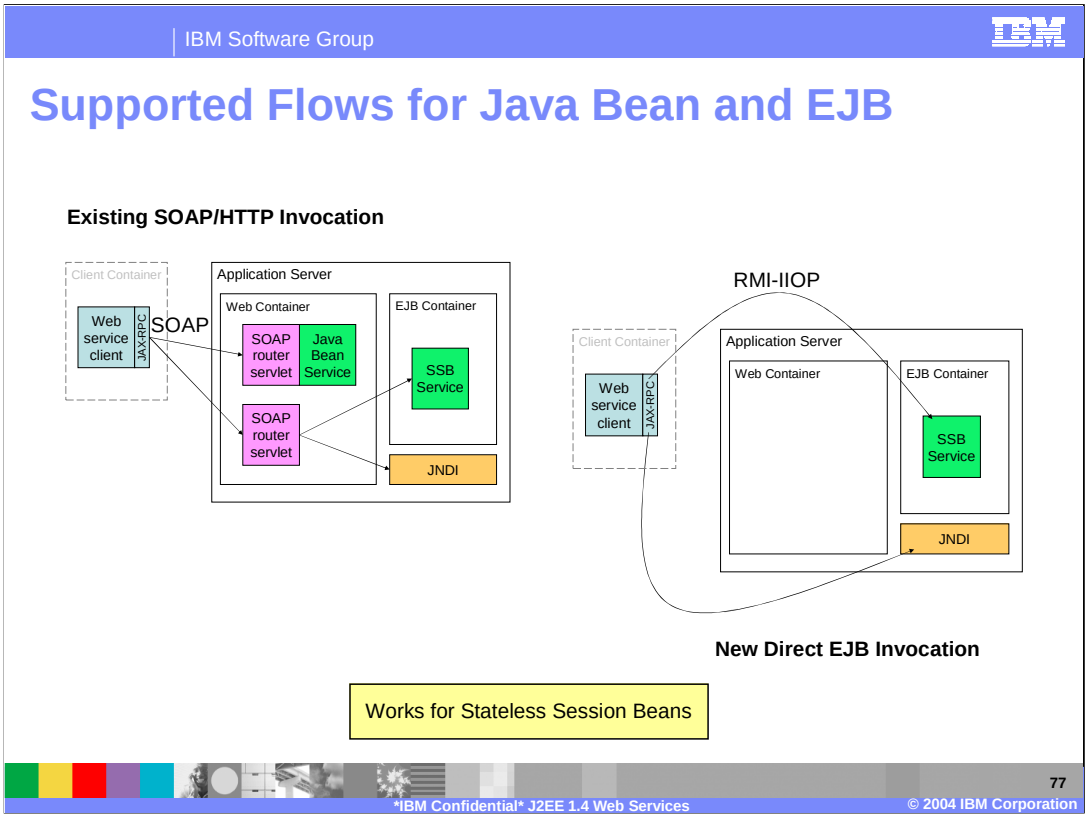
This is an example showing how to create a cache id from the SOAP body. This method allows the greatest control in selecting what is cached, but also requires the largest performance penalty as the entire XML message must be parsed by the cache to retrieve this information. This would allow you to cache certain portions of an XML message that will be common across multiple service requests.

Section

Multi-Protocol Support

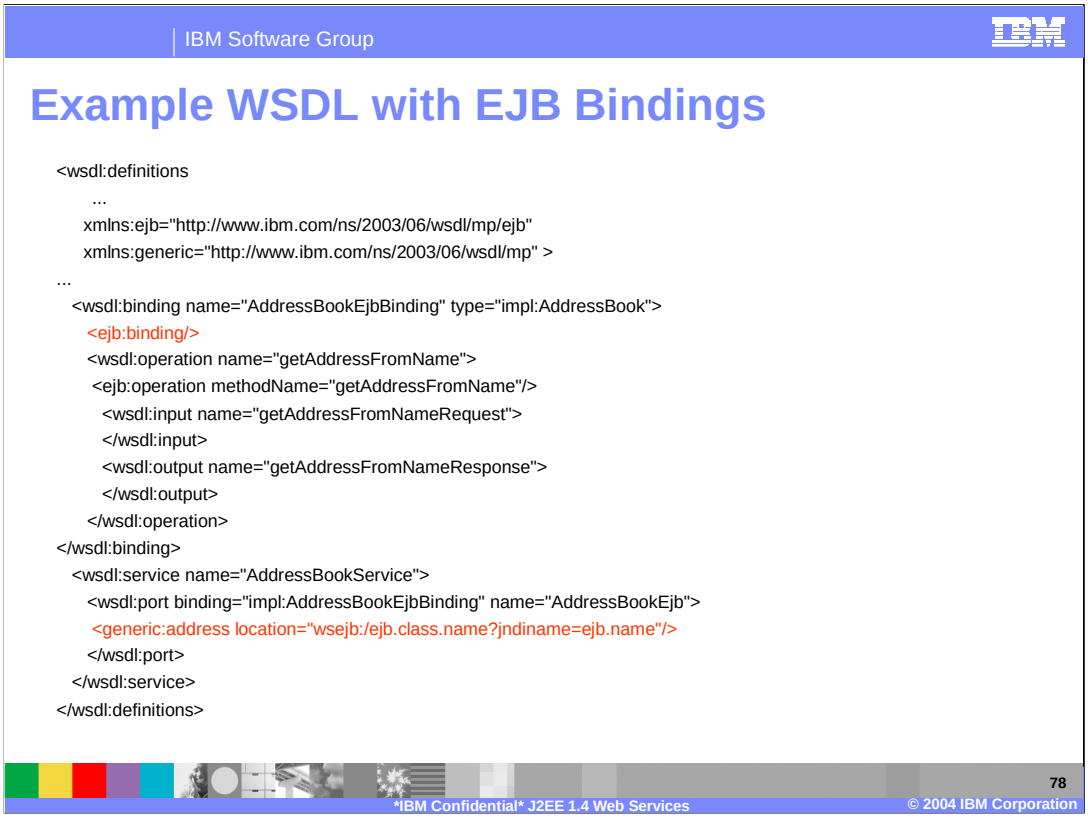
Multi Protocol Support

- Extends JAX-RPC support for invoking remote stateless session EJBs with RMI-IIOP
  - More performant method for calling EJB services
- This allows managed clients (defined by JSR 109) to access Web Services through a number of protocols
  - No changes to JAX-RPC client are needed
  - May be extended to include other protocols in the future



A web service client either running in a managed container (application client container, web container or EJB container) or running unmanaged from a J2SE client or non-Java client, invokes a web service implemented either as a stateless session bean or a java bean behind a router servlet. The HTTP transport protocol is implied by the diagram, however, the existing web services support also includes the option for a messaging (JMS compliant) transport such as MQSeries.

The new scenario above removes the servlet step by essentially implementing its functionality underneath the JAX-RPC API on the client and using RMI/IIOP to communicate with the service instead of SOAP. Using this approach a service can be invoked in a way it finds more natural. SOAP enabled stateless session bean services are fronted by a 'router module'; by providing a direct route to the EJB, the overhead of the router module is removed.



The EJB binding namespace is:

"http://www.ibm.com/ns/2003/06/wsdl/mp/ejb"

The EJB binding extends WSDL with the following elements:

<ejb:binding/>

This element has no attributes.

The methodName attribute is the EJB home interface method name to be invoked for the abstract operation name which encloses it in the WSDL. This is generated to be the same as the abstract operation name.

Changes to Tools

- WSDL2Java
  - No new command-line options
  - Changed to recognize non-SOAP ports and bindings
  - Locator and Stub classes will support non-SOAP ports
  - New Information class contains service information previously contained in the stub
- Java2WSDL
  - Changed to create non-SOAP bindings in the WSDL
  - Command-line now supports EJB bindings
    - java2WSDL –bindingTypes http,ejb –implClass my.pkg.MyEJBClass my.pkg.MySEI

Template Revision: 10/01/2004 4:46 PM

Trademarks and Disclaimers

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

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

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.

Note to U.S. Government Users - Documentation related to restricted rights-Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract and IBM Corp.