This presentation will explain the new nodes for WebSphere Service Registry and Repository in WebSphere Message Broker Version 6.1.

Agenda

- Brief overview
- Nodes for WebSphere Service Registry and Repository access
  - EndpointLookup
  - RegistryLookup
- Proxy and data cache
- Usage patterns

This presentation will first provide a brief overview of what a service registry and repository is. It will then cover the new EndpointLookup and RegistryLookup nodes.

It will then cover how these nodes can be used in a production scenario, and the impact of performance, and then finally presents some examples scenarios.

What is a registry? …. A repository

Registry?

- Hold meta-data
  - Describes the service artifacts
- Allows you to search for artifacts in the repository

Repository?

- Stores artifacts, entities
  - WSDL
  - XSD
  - Other XML documents
  - SCDL

WebSphere Service Registry and Repository consists of two logical components.

The first component is the repository. The repository stores artifacts, and in the case of WebSphere Service Registry and Repository these are artifacts associated with services, such as WSDL, XSD, other XML documents such as SCDL and BPEL.

The second component is the registry. This holds meta-data associated with the artifacts stored in the Repository. The registry holds data that describes the service artifacts. This includes state data associated with a particular artifact. It also includes associated information such as Service Name, Namespace and Version. These allow you to search for the artifacts in the repository, such as the WSDL.

WebSphere Service Registry and Repository offers advanced functions for several well-known SOA metadata types. The key metadata document types are WSDL, XSD, WS-Policy and SCDL. Special services are provided for these document types, including "shredding" of the documents upon receipt into Logical Derivations.

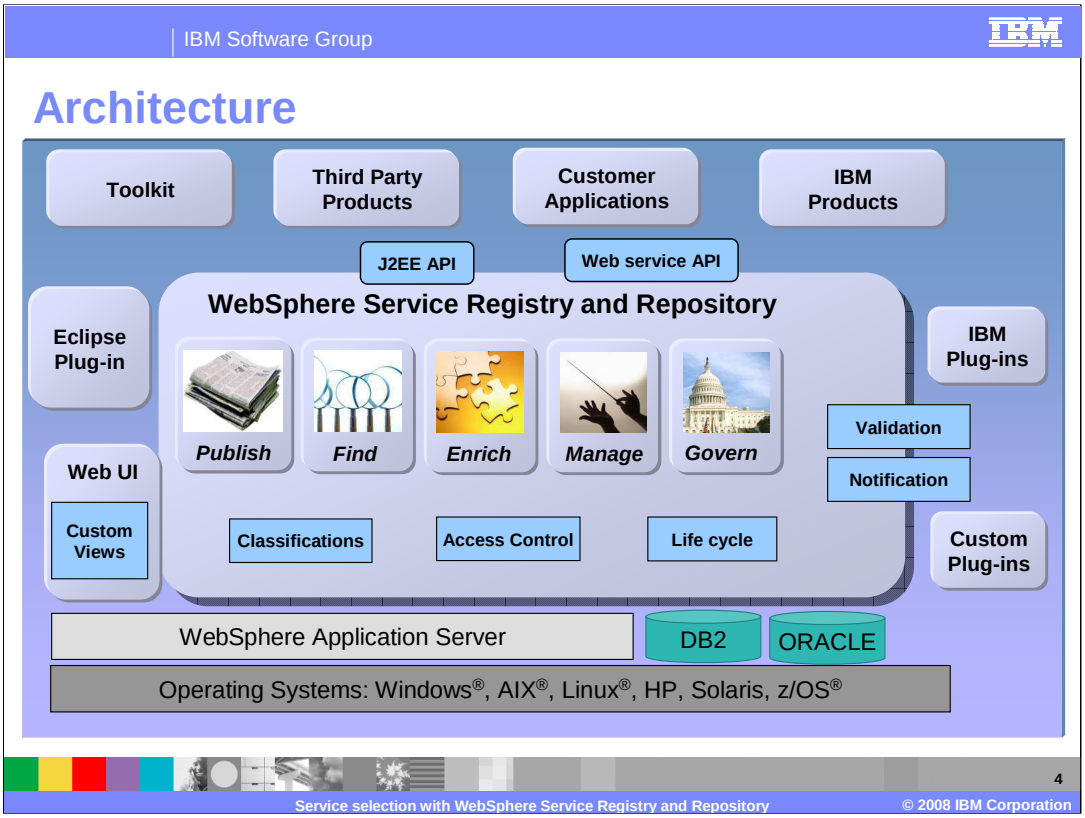
Other types of service metadata can be stored using a generic content type of “XML Document”. Documents of type “XML Document” are not decomposed into the logical derivations.

However, WSDL also contains some meta-data associated with a service. Thus, it is possible to hold certain parts of a WSDL document in a repository and other parts in a registry.

WebSphere Service Registry and Repository contains both a registry and a repository for two key reasons.

First, an integrated set of capabilities provides a complete set of capabilities in a single component.

And second, it is necessary to keep the artifacts in sync with their metadata. If the registry implementation is separate from the repository then it is possible for the two to get out of sync.



This slide shows the overall architecture of WebSphere Service Registry and Repository. This is a J2EE application that installs on top of WebSphere Application Server. It can contain information about many artifacts including customer applications, IBM products, and products from other vendors.

WebSphere Service Registry and Repository provides two APIs – a Java API and a Web service API.

When using the Web service API, you send and receive SOAP messages with an XML payload.

When using the Java API, you send and receive Service Data Object structures.

A servlet-based Web User Interface is the primary way for users to interface with WebSphere Service Registry and Repository.

WebSphere Service Registry and Repository supports lookup and publish scenarios, metadata management and analysis scenarios, and functions that support SOA governance.

An Eclipse plug-in is also provided to support lookup, retrieval and publishing of service metadata from the development toolkit.

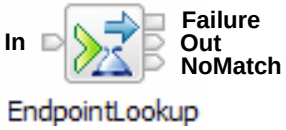
WebSphere Service Registry and Repository can be considered as two separate modules:

The Registry and Repository core component provides basic metadata storage and retrieval. It also provides a framework for extensible validators and an extensible event based notification system that can publish events for any operations performed on the registry. The validators are applied to create, update or delete operations.

Message Broker client

- Two nodes that query an instance of WebSphere Service Registry and Repository (default version 6.0.2)
- Result placed into LocalEnvironment
  - Payload is not modified
- Results of query propagated to Out terminal

- Generic Node for retrieval of any
- WebSphere Service Registry and Repository entity
- Able to search entity names
- Retrieve entire contents of entity into the broker
- Use entity information at runtime



- Focused on the retrieval of endpoints defined within WSDL
- Retrieves EPR and Metadata, not whole artifact
- Able to search port type names within WSDL imported into repository
- Use endpoint for dynamic routing at runtime

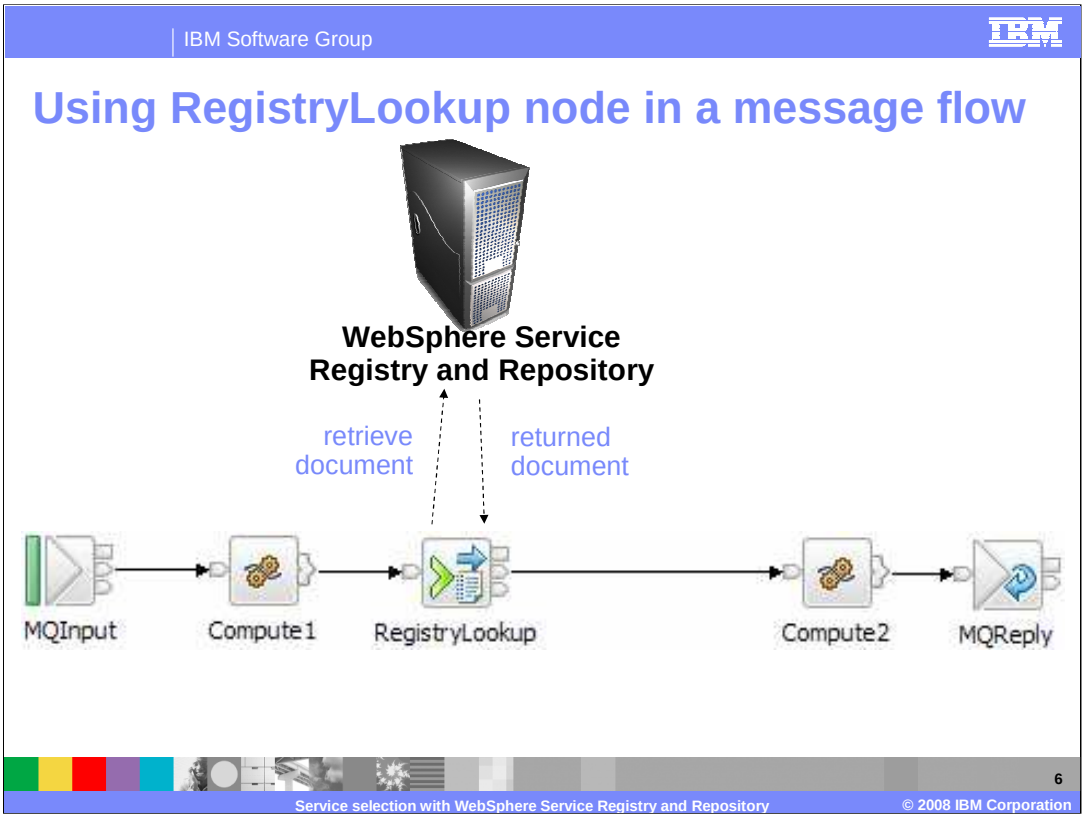
Message Broker version 6.1 provides two nodes to support the interface to WebSphere Service Registry and Repository.

The EndpointLookup node is used to retrieve the service endpoint information related to a WebSphere Service Registry and Repository defined service, for example, WSDL. It does not perform additional filtering or selection, other than that specified by the property matching. If a single service is requested, it will set the endpoints in the context so that this service information is correctly placed in the domain context. The existing Message Broker nodes will correctly invoke the service.

This node is independent from any other domain context.

Support is currently limited to query for endpoints for Web services. This can be used by the SOAP nodes, or the HTTP nodes.

The RegistryLookup Node is a generic node that retrieves the meta-data related with a WebSphere Service Registry and Repository entity. It does not perform additional filtering or selection, other than that specified by the property matching. It also does not resolve endpoints, do content based routing, or set domain context for execution of the target service. The message flow developer is responsible for all these functions.



This slide shows how the RegistryLookup node can be used within a message flow.

First, the RegistryLookup node receives a message.

The node then retrieves the entity metadata information from WebSphere Service Registry and Repository using the specified query string. The RegistryLookup node can be used to define a query dynamically within the message. Both the RegistryLookup and the EndpointLookup nodes can accept a query specified within the Local Environment. Accepting a query specified within the Local Environment overrides any property values set on the node. However, it is still mandatory to set values on the properties of the nodes because the nodes cannot deploy without doing so.

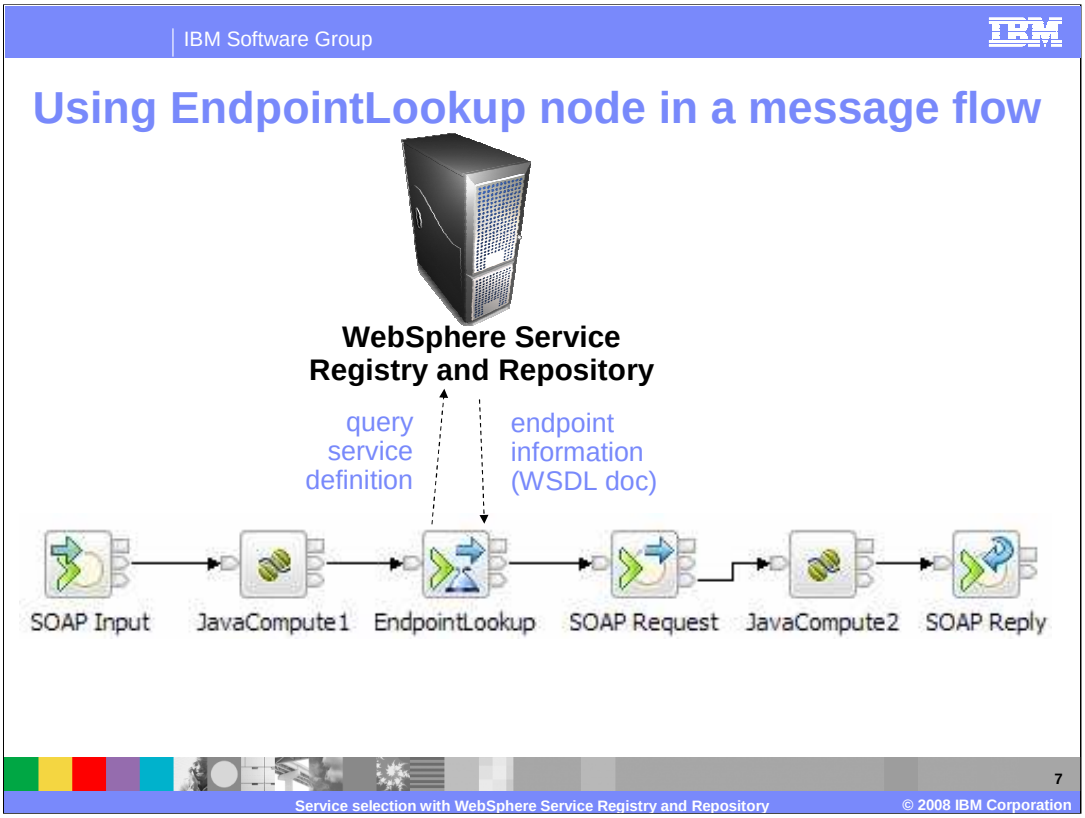
If one or more matches are found, then two outcomes are possible.

First, if Match Policy is set to One, the RegistryLookup node adds the single matching metadata information to the message instance.

Or, if Match Policy is set to All, the RegistryLookup node adds all matching metadata to the message instance, leaving all other message content unchanged.

Metadata is propagated to the Out terminal, where it is available for further processing either by ESQL or by a Java Compute node.

If no matches are found, the RegistryLookup node propagates the input message to the No-Match terminal.



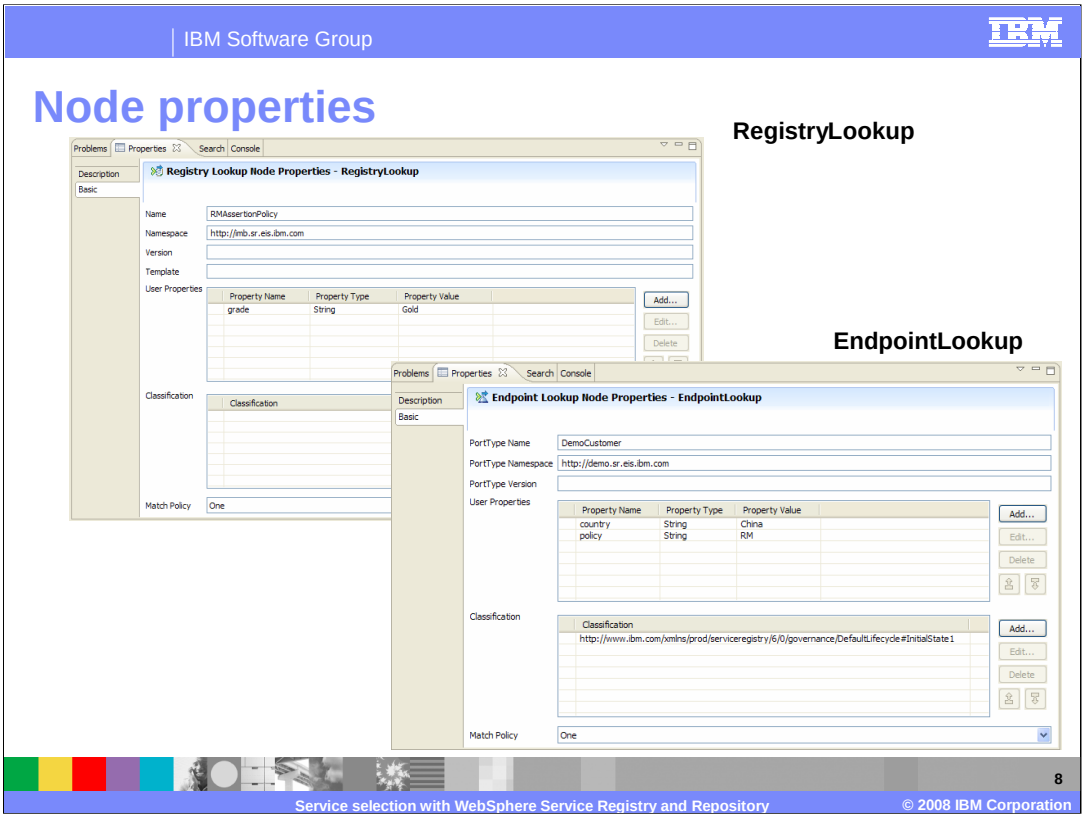
This slide shows how the EndpointLookup node can be used within a message flow.

First, the EndpointLookup node receives a message.

The node then retrieves the IT service endpoint information from WebSphere Service Registry and Repository using the specified query string. As with the RegistryLookup node, the EndpointLookup node can be used to define a query dynamically within the message, and this is done by setting the appropriate properties in the Local Environment. As with the RegistryLookup, it is still mandatory to set values on the properties of the nodes because the nodes cannot deploy without doing so.

If one or more matches are found, then if Match Policy is set to One, and a single endpoint is matched, the EndpointLookup node sets valid endpoints in the domain. Using these, the SOAP Request, SOAP AsyncRequest, and HTTP Request nodes can be used to invoke the service. The EndpointLookup node returns the first result it receives, and sets the endpoint address. However, note that the first result might not be the most current version.

If no matches are found, the EndpointLookup node propagates the input message to the No-Match terminal.



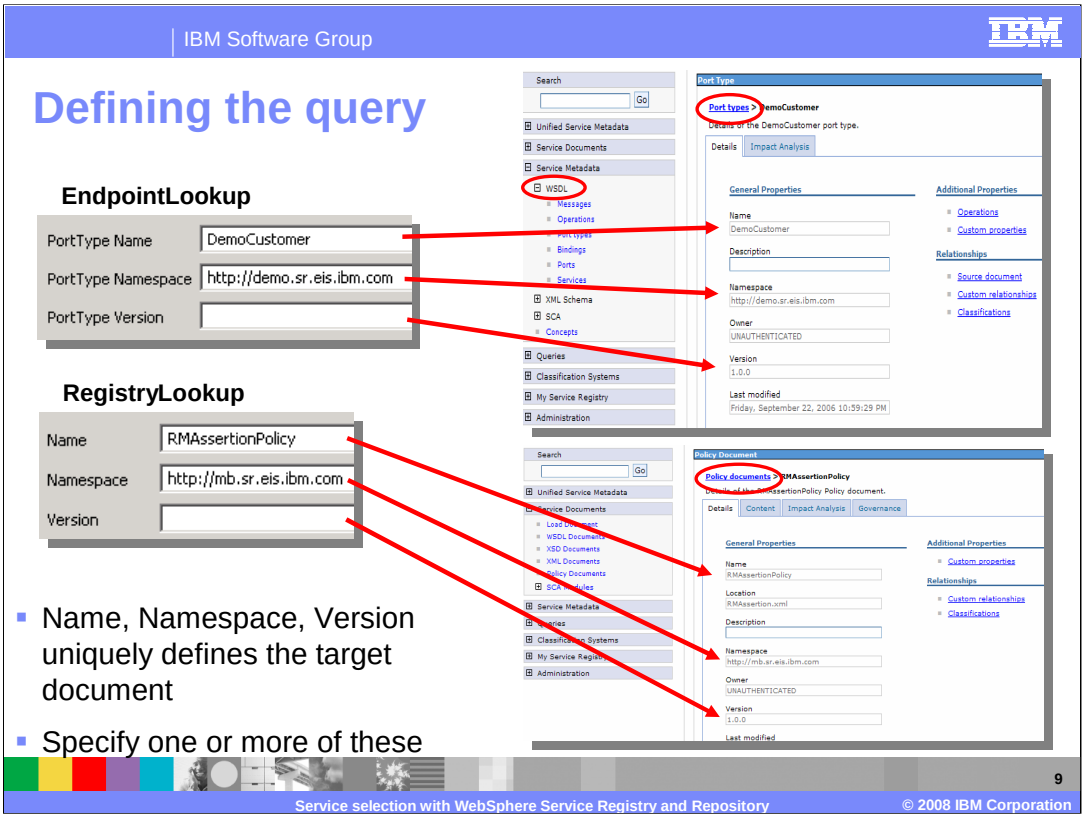
The properties of both lookup nodes are shown on this slide.

The RegistryLookup node uses the name of the registry artifact as the key property of the node. This is used to query WebSphere Service Registry and Repository to retrieve the artifact itself from the repository.

The three properties that define the query are Name, Namespace and Version. At least one of these properties are needed, although all three can be specified. If you leave all blank, you will not be able to save the properties in the toolkit.

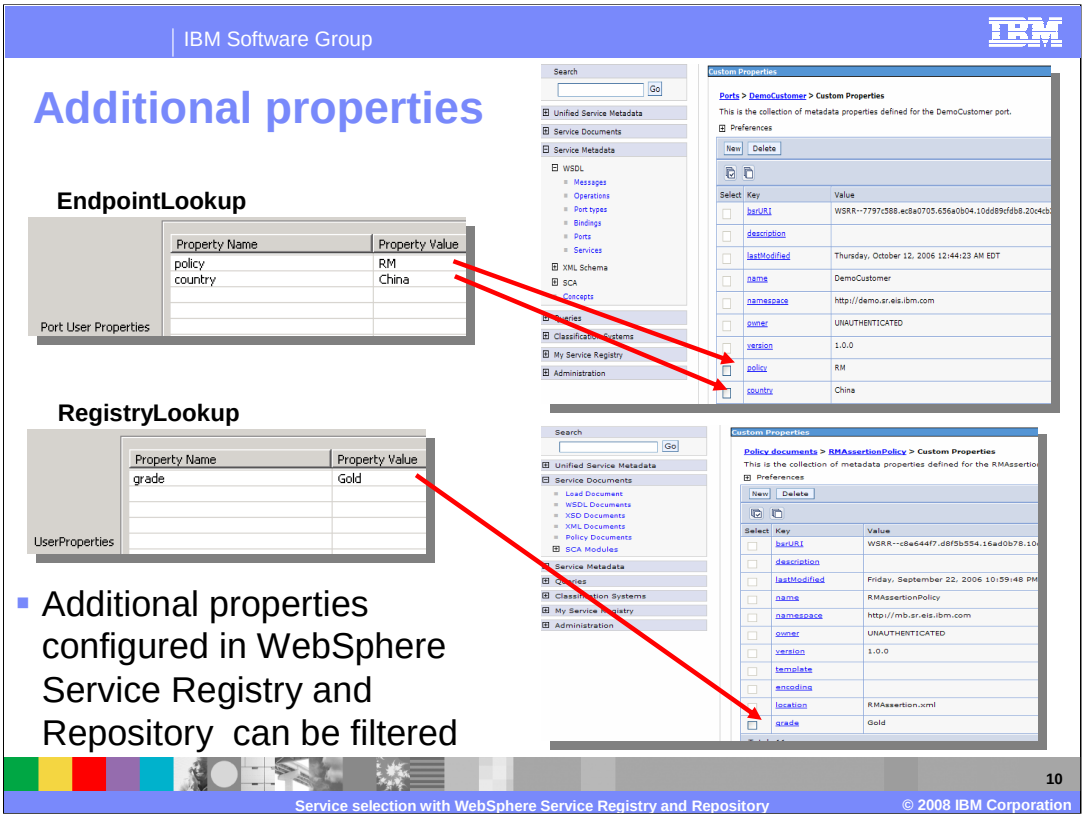
Similarly, the EndpointLookup node has three possible properties that can be used to perform the WebSphere Service Registry and Repository query.

These are port-Type Name, port-Type Namespace, and port-Type Version. As with the RegistryLookup node, at least one of these properties must be specified, although you can specify all three.



This slide shows how the properties specified for the Message Broker nodes correspond to the values of the parameters in WebSphere Service Registry and Repository.

The message broker node properties are shown on the left, and the WebSphere Service Registry and Repository parameters are shown on the right. For both of the nodes, at least one property must be specified. If the query matches more than one item within the repository, all results will be returned to the message flow. This is discussed later in the presentation.

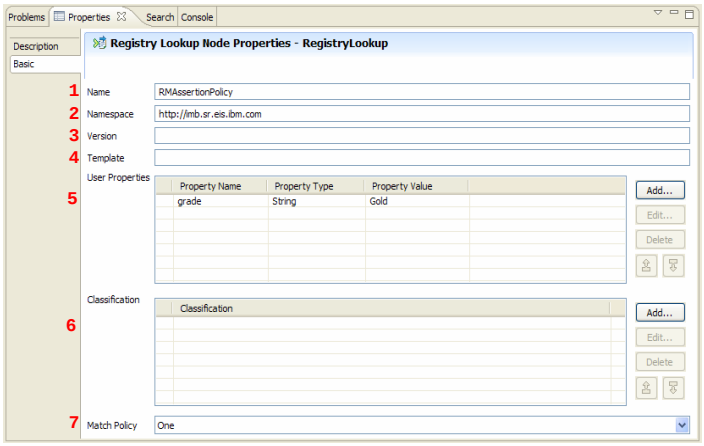


Both WebSphere Service Registry and Repository nodes provide the facility to add additional properties. These are known as user properties, and correspond to “Custom Properties” in WebSphere Service Registry and Repository.

This slide shows how several user properties have been added to the message flow node properties. This is achieved by clicking the “Add” button on the Basic properties tab, and using the wizard to specify required values.

Overriding the query properties

- The broker's LocalEnvironment can be used to override values set on the node



1. `LocalEnvironment.ServiceRegistryLookupProperties.Name`
2. `LocalEnvironment.ServiceRegistryLookupProperties.Namespace`
3. `LocalEnvironment.ServiceRegistryLookupProperties.Version`
4. `LocalEnvironment.ServiceRegistryLookupProperties.Template`
5. `LocalEnvironment.ServiceRegistryLookupProperties.UserProperties.<Property Name>`
6. `LocalEnvironment.ServiceRegistryLookupProperties.Classification.<Classification name>`
7. `LocalEnvironment.ServiceRegistryLookupProperties.MatchPolicy`

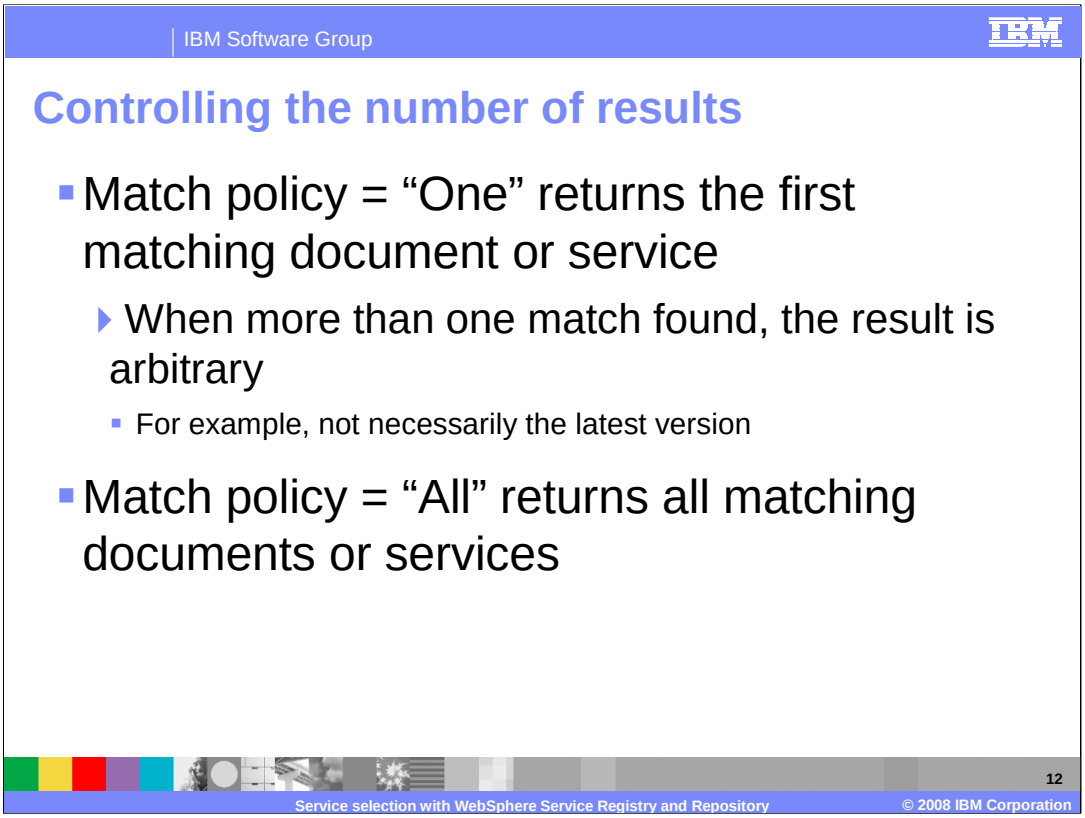
As mentioned earlier in this presentation, you can use the Local Environment to override the properties that have been defined in the Endpoint or RegistryLookup nodes in the message flow. The Endpoint and Registry nodes do not manipulate the body of the message.

The actual message is not changed by the EndpointLookup node, however, the Local Environment is updated to reflect the search results.

The Local Environment properties that can be used are shown here, and the corresponding property on the node.

As with the node properties, with the EndpointLookup node, at least one of the Name, Namespace or Version properties must be set. If this is not done, the flow will not deploy.

Unlike the User Properties defined on the EndpointLookup and RegistryLookup nodes, X-Path and ESQL expressions are not supported when overriding the properties from the Local Environment.

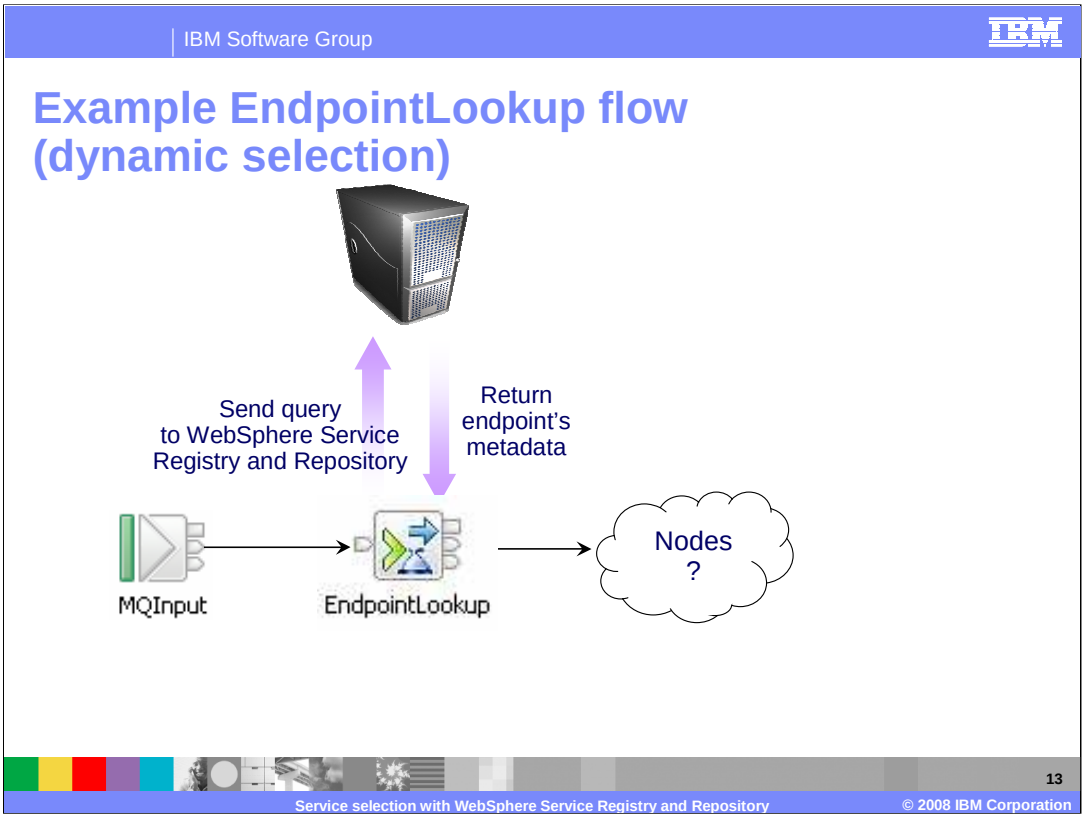


If one or more matches are found, then this information is passed to the message flow.

In this case, if *“Match Policy”* is set to One and a single endpoint is matched, the EndpointLookup node sets valid endpoints in the domain. The SOAP Request, SOAP Asynchronous Request, and HTTP Request nodes, can be used to invoke the service. The EndpointLookup node returns the first result it receives, and sets the endpoint address. However, the first result might not be the most current version.

If *“Match Policy”* is set to All, the EndpointLookup node adds all matching endpoints information to the message instance, leaving all other message content unchanged. These results are then sent to a Java Compute node which selects the current version and sets the endpoint address. You must propagate the Local Environment tree with the message. You can use this option to select the most current version of a particular Web service.

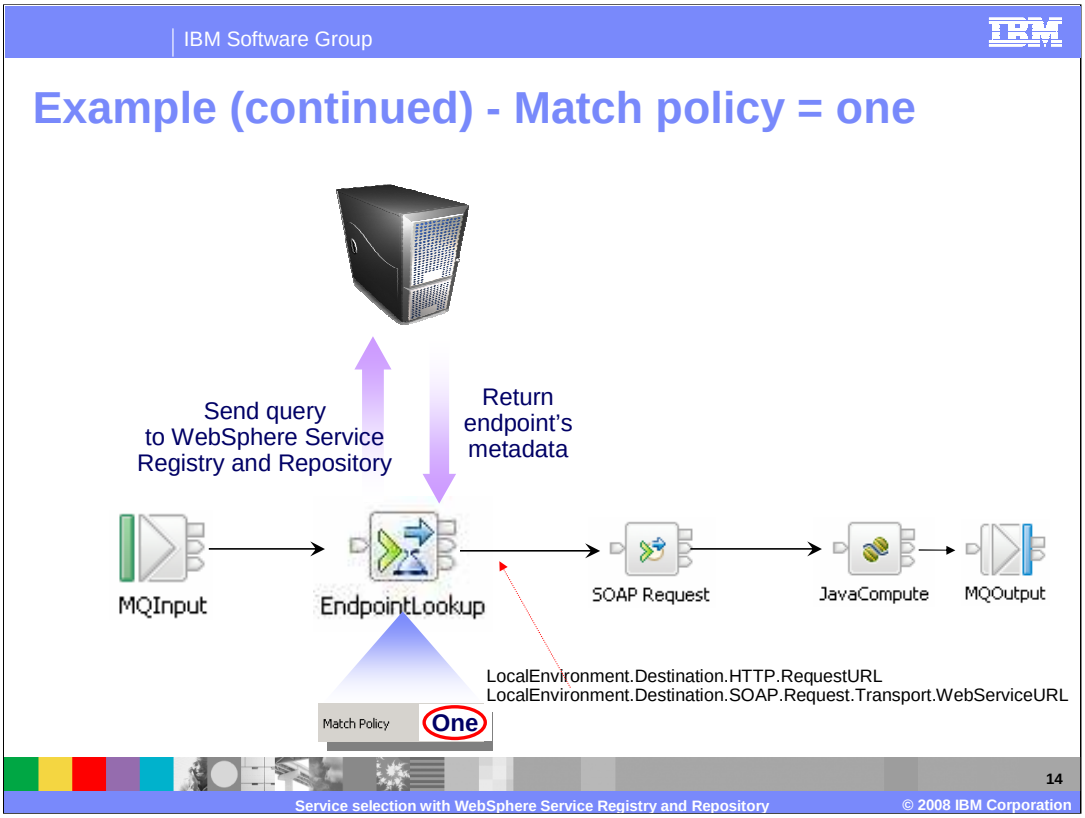
Metadata information is propagated to the Out terminal, where it is available for further processing either by ESQL or by a Java Compute node.



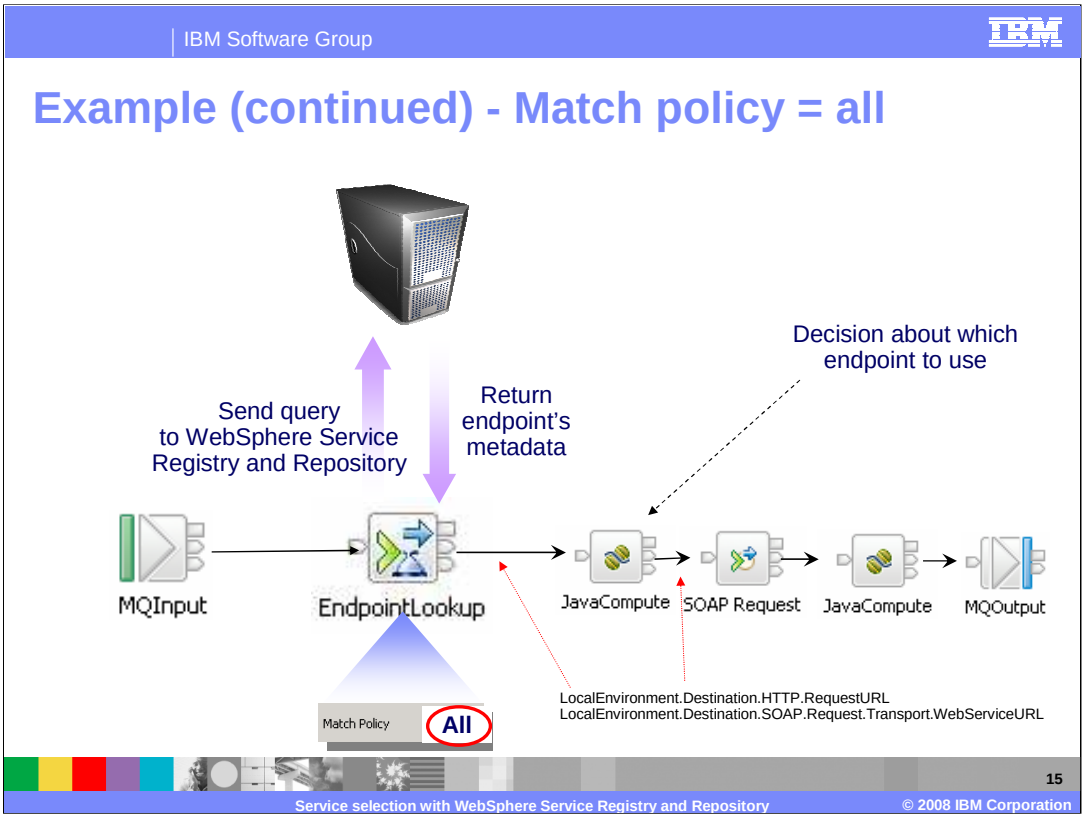
The next three slides show an example of how a service can be dynamically selected, based on information return from the WebSphere Service Registry and Repository system.

First, the MQ Input node receives a message and propagates this to the EndpointLookup node.

This node uses information defined within its properties to query WebSphere Service Registry and Repository.

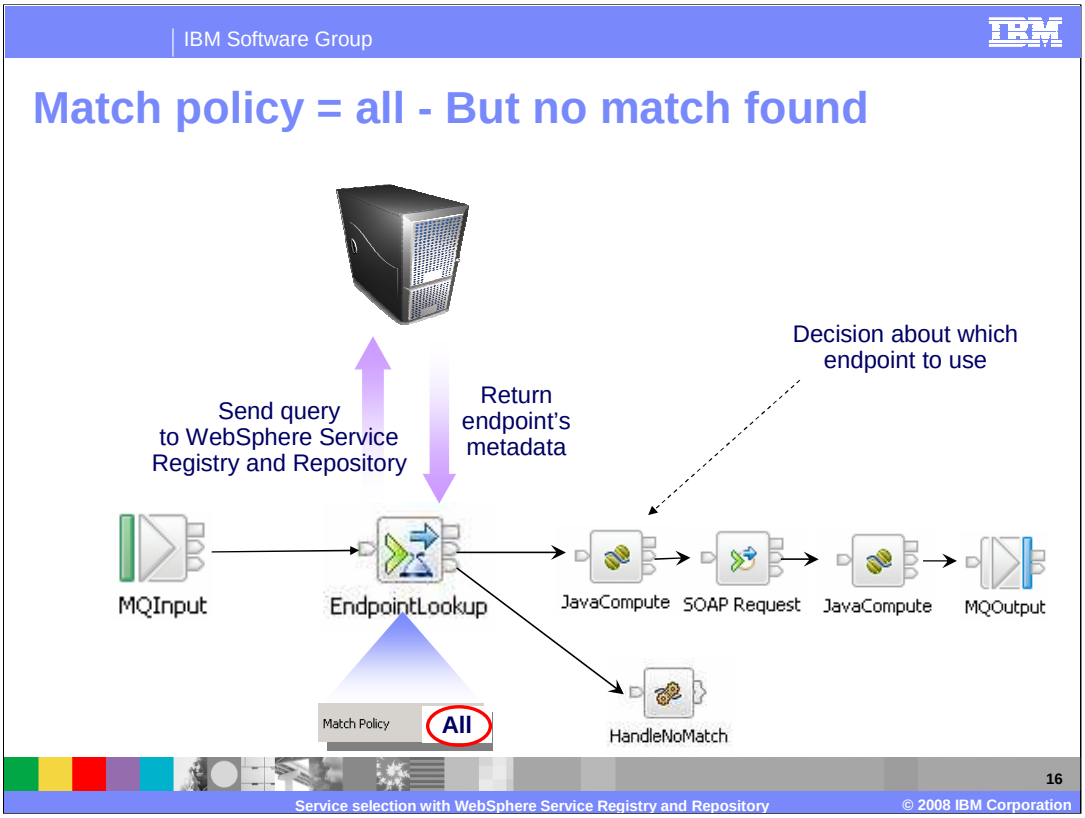


If the Match Policy is set to "One", the EndpointLookup node will be connected directly to a SOAP Request node. The information needed by the SOAP node will be placed into the Local Environment and propagated to the 'out' terminal of the Lookup node.

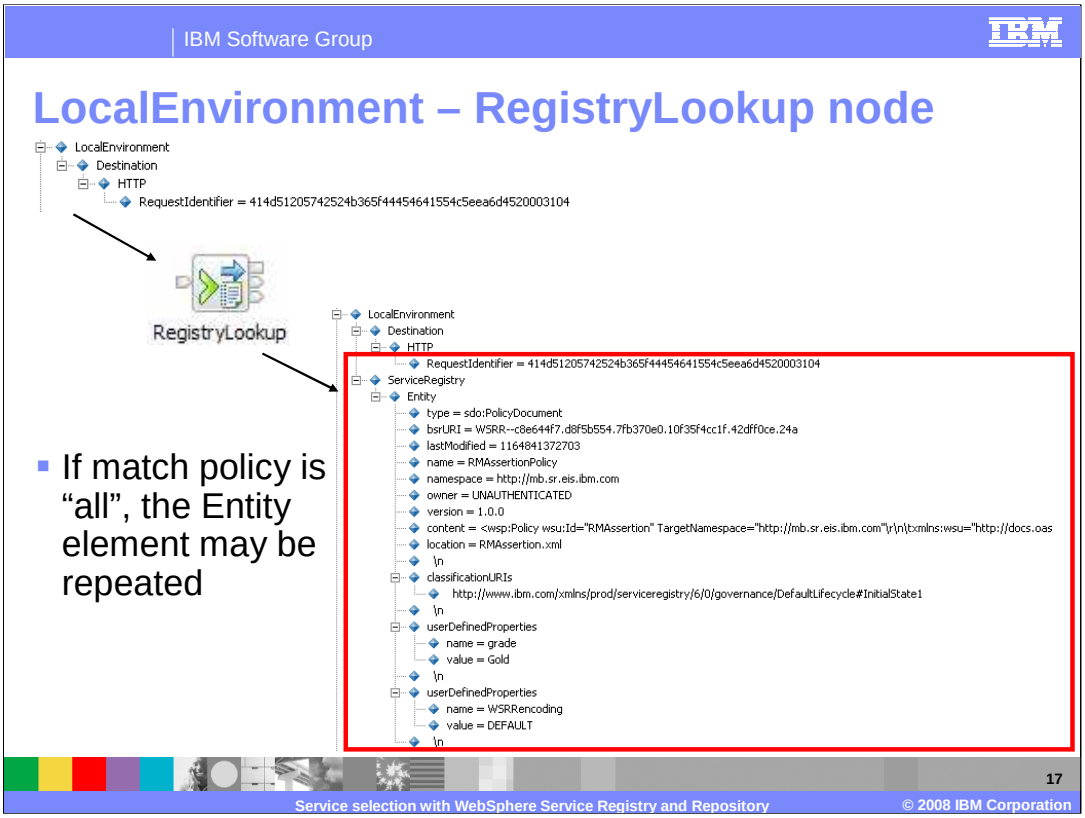


If the Match Policy is set to "All", one or more results are returned.

Information is placed into Local Environment and propagated to 'out' terminal of the EndpointLookup node. In this case, this information will be passed to a Java Compute Node, which will make a decision on which Web service to invoke, based on the information from WebSphere Service Registry and Repository.



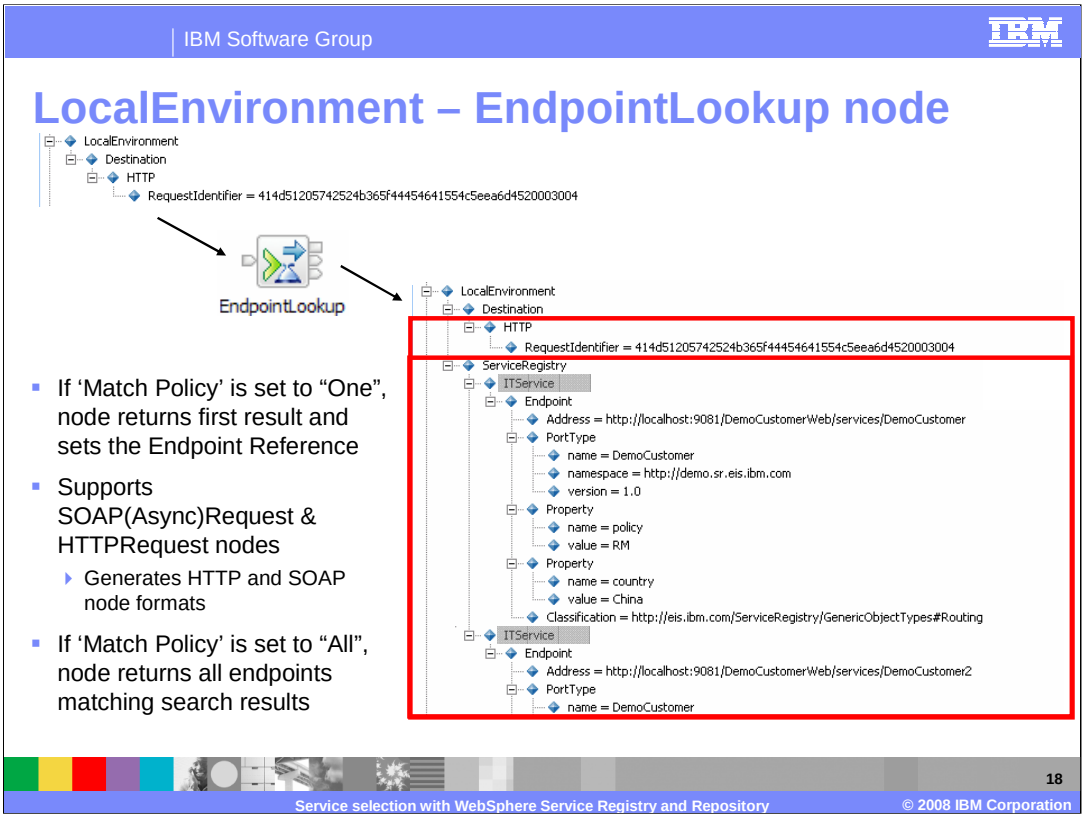
If no results are returned from WebSphere Service Registry and Repository, the message is propagated to the 'No Match' terminal. You should handle this scenario in the normal way, by connecting a node to this terminal.



When using the RegistryLookup node, the retrieved WebSphere Service Registry and Repository metadata is attached to the Local Environment tree.

This screen capture of the Local Environment shows an example where just a single element has been retrieved.

If the Match Policy is set to "All", then the Local Environment will contain multiple instances of the Entity element, depending on the number of elements returned by WebSphere Service Registry and Repository.



When using the EndpointLookup node, the retrieved WebSphere Service Registry and Repository service data is attached to the Local Environment tree. If the Match Policy is set to "One" then the node also sets LocalEnvironment.Destination.HTTP.RequestURL. This setting overrides the HTTP Request Node URL property, permitting a dynamic URL setting for HTTP nodes.

The "Service Registry" part of the Local Environment is populated with information about the service, which is then used by a downstream SOAP Request node.

Other entity information (for example customer properties and classifications) may be added enabling further processing choices.

Connecting to the WebSphere Service Registry and Repository server

- Configured on the Message Broker runtime as a "Configurable Service"
  - mqsichangeproperties (or configuration manager proxy)
- Example:

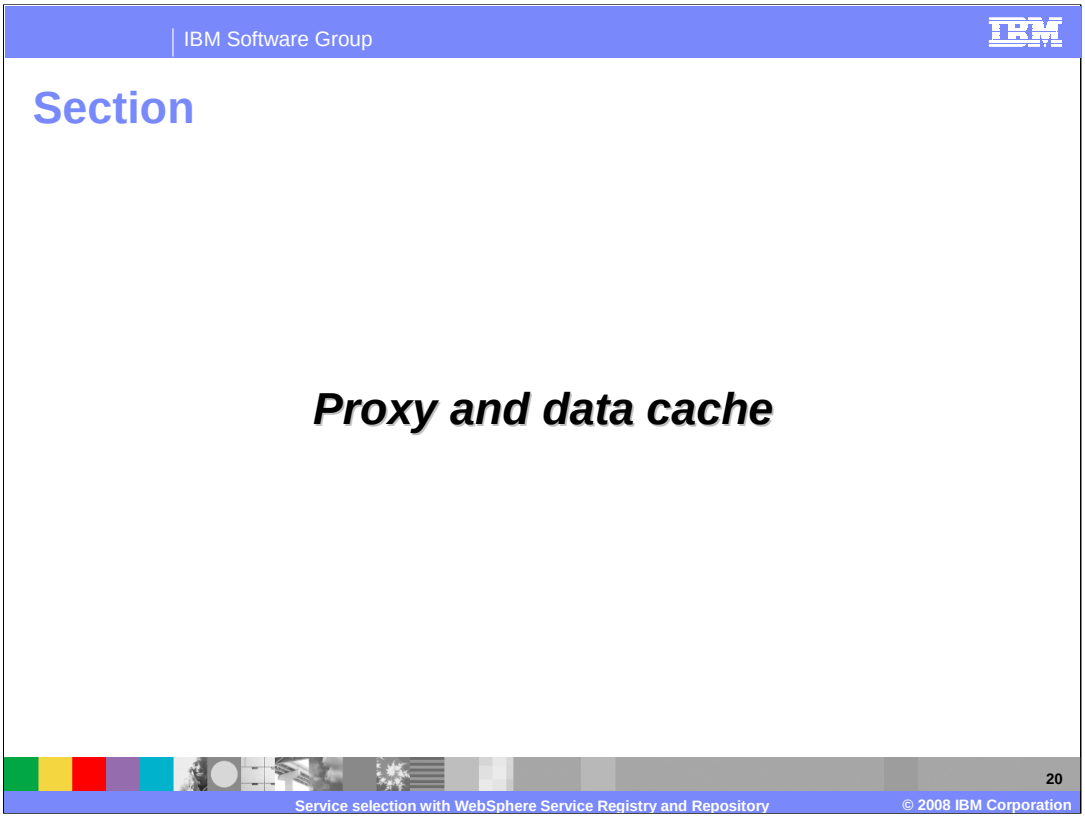
mqsichangeproperties WBRK_BROKER -c ServiceRegistries
-o DefaultWSRR -n endpointAddress
-v http://localhost:9080/WSRRCoreSDO/services/WSRRCoreSDOPort

Where:
-n = name of property to be changed/set
-v = value of property (location of WebSphere Service Registry and Repository system endpoint in this example)

The WebSphere Service Registry and Repository system that the EndpointLookup and RegistryLookup Nodes will use needs to be defined to the Message Broker runtime.

This is done using a Configurable Service, and set using the mqsichangeproperties command. You can also use the configuration manager proxy API, or the proxy exerciser. The name of the configurable service is "Service Registries", specified on the "-c" parameter.

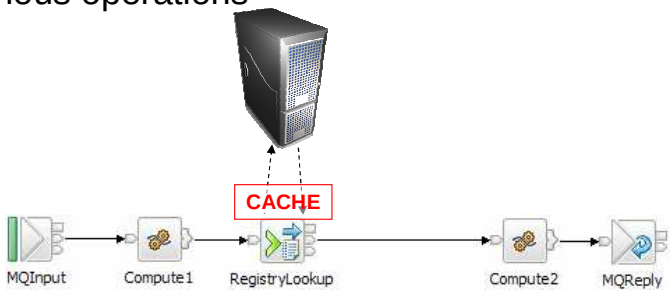
An example of the command is shown on this slide.



This section discusses the caching mechanisms that are provided to ensure optimum performance.

Caching

- The RegistryLookup and EndpointLookup nodes are synchronous operations



- An optional local cache is used to speed up access
- Customizable as a Message Broker “configurable service”
  - Cache preload queries
  - Configurable timeout

An internal cache can be used to store query strings, and their associated documents, that are retrieved from the WebSphere Service Registry and Repository.

The function of the Cache is configured individually for each broker. You can configure the WebSphere Service Registry and Repository Cache strategy by using the pre-defined "ServiceRegistries" configurable service, using the mqsichangeproperties.

The WebSphere Service Registry and Repository Cache is configured at the broker level.

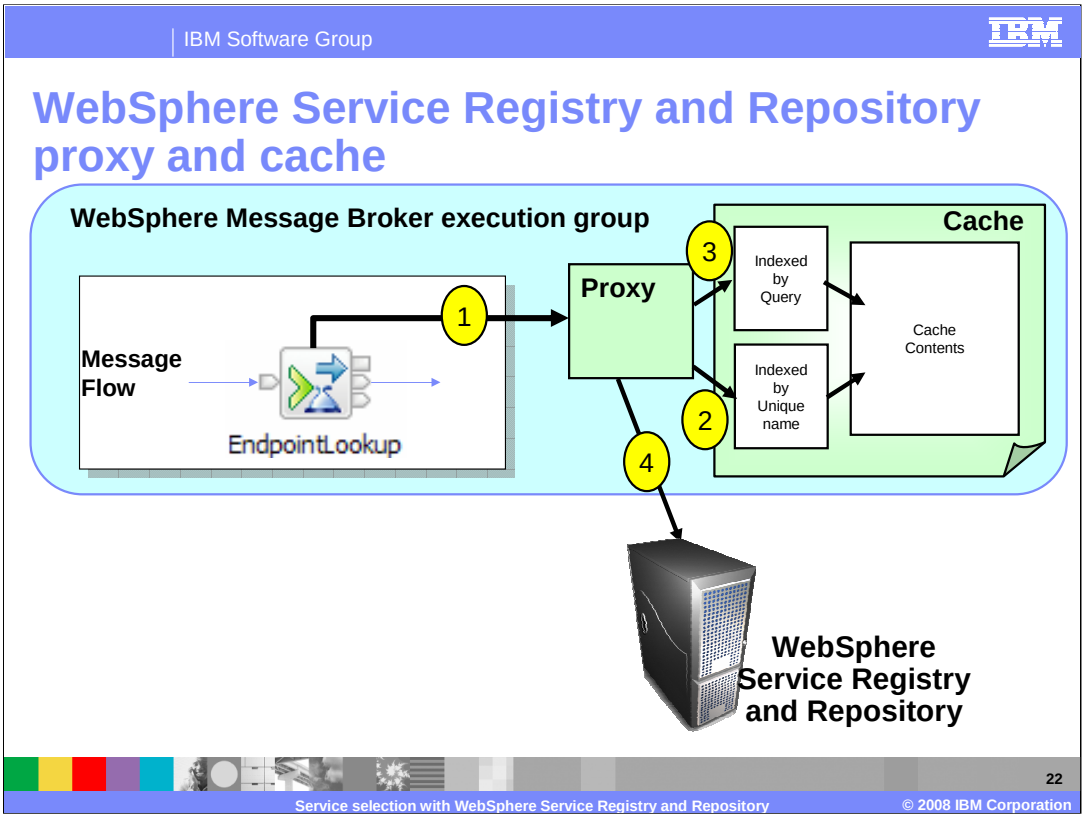
Configuring this cache is optional and does not have to be enabled for the WebSphere Service Registry and Repository nodes to function.

You can cache the data that you retrieve from WebSphere Service Registry and Repository in three ways.

First, with No Cache. In this case, every time a message goes through a WebSphere Service Registry and Repository node, the broker connects to the WebSphere Service Registry and Repository system to retrieve the documents.

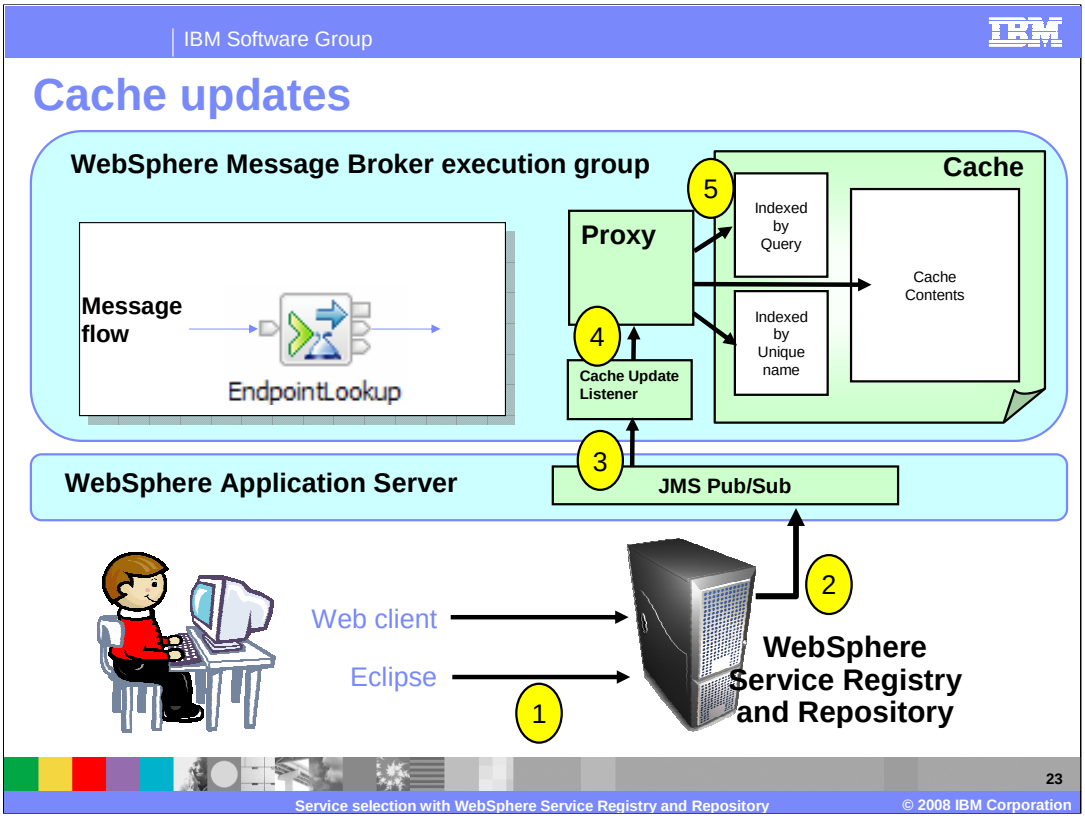
Second, with a Cache Timeout. This is the default setting. You configure this option using the “need Cache” and “Timeout” parameters. If “needCache” is set to True, the broker retrieves the documents from the WebSphere Service Registry and Repository the first time the message flows through a WebSphere Service Registry and Repository node. It stores the documents in an internal cache. The next time the message goes through the flow, the documents are retrieved from the Cache, not from WebSphere Service Registry and Repository. Using this option improves performance. If the Cache Timeout interval is exceeded, the broker marks its internal cache data as not valid and runs the query against the WebSphere Service Registry and Repository the next time a message goes through the node. In this situation the node is not retrieving real-time updates made to WebSphere Service Registry and Repository.

Finally, with Cache Notification. In this scenario, WebSphere Service Registry and Repository publishes a message when a document is added, edited, or deleted. This option is configured using the “enable Cache Notification” parameter. The broker then subscribes to the publish-subscribe topic, and immediately marks its internal cache data as not valid and runs the next query against the WebSphere Service Registry and Repository. The cache is updated with the new values.



This slide illustrates how the caching mechanism works within Message Broker. The cache contents are stored within the execution group of the Message Broker, and can be accessed using a variety of indexes. The cache proxy is the component that performs this local access, and determines whether the cache has been invalidated.

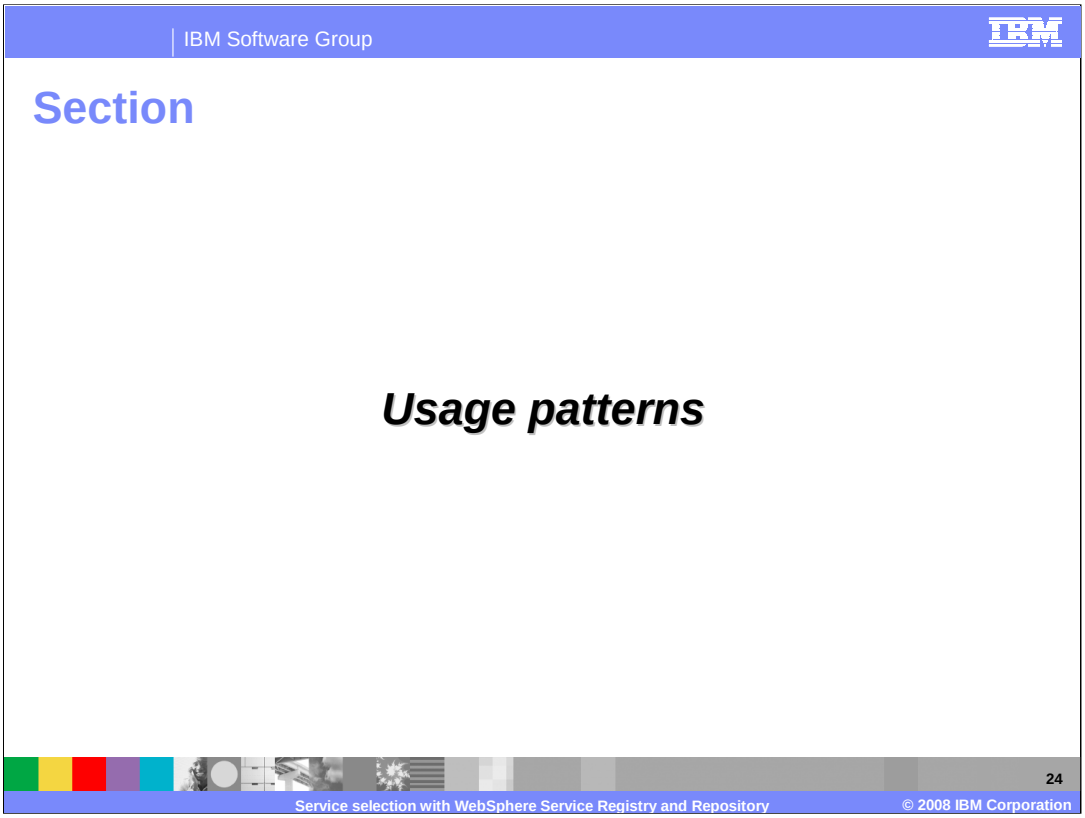
If the proxy determines that the required data is not present in the cache, or has been invalidated through a timeout, then it will connect directly to the WebSphere Service Registry and Repository system. The required data will be retrieved directly from WebSphere Service Registry and Repository.



In the case where WebSphere Service Registry and Repository system is being updated, and you require these updates to be reflected in the Message Broker system, these updates are published using the JMS publish-subscribe facility.

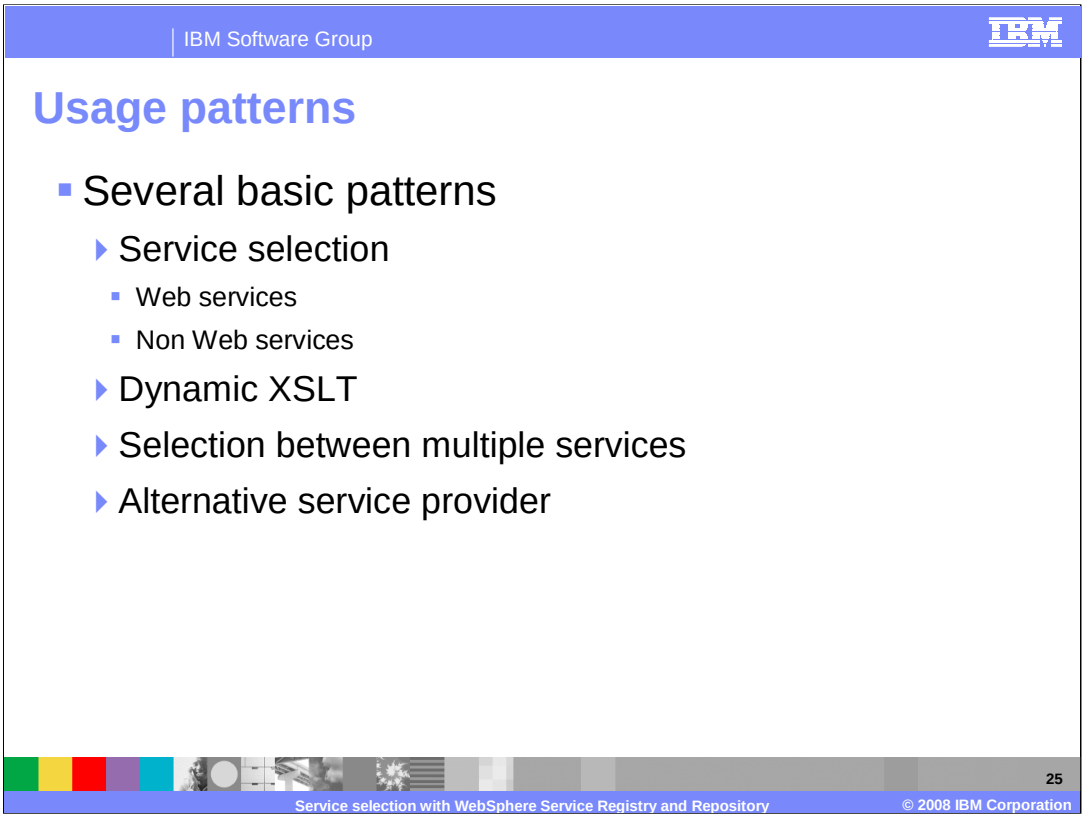
The “cache update listener” has subscribed to the JMS queue, and retrieves any updates published to that queue. When these updates are received, the listener passes these to the cache proxy, which uses them to update the local cache.

Note that although the JMS publish/subscribe system is shown as a separate component on this slide, WebSphere Service Registry and Repository runs as an application on WebSphere Application Server, so the publish/subscribe component uses the same runtime component as WebSphere Service Registry and Repository.



This final section presents some usage scenarios, and shows how message flows can be used in conjunction with WebSphere Service Registry and Repository.

The scenarios are not discussed in detail, but give some suggestions of design patterns that may be employed in a dynamic service architecture.



Four basic patterns are presented.

The first is a basic service selection application. The service itself can be accessed using Web services, or using any other style of connection.

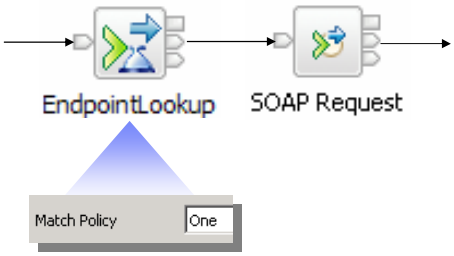
The second provides a facility to store an XSLT transformation in WebSphere Service Registry and Repository, and load and use this dynamically within the Message Flow.

The third presents a technique to allow selection of a service, based on data contained within the incoming message.

And the last presents the case where a service invocation has failed, and the message flow needs to retry with a different service endpoint.

Service selection (basic endpoint address setting)

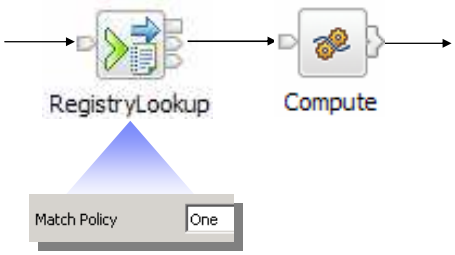
- When Destination is a SOAP/HTTP Web service
  - Use EndpointLookup node to set endpoint address
    - Set match policy = One
  - Use any input node (MQ, HTTP, JMS etc)



The first example is similar to the example presented earlier. The target service is a Web service accessed using the SOAP Request node. The EndpointLookup node is used to retrieve service details, and only one instance of the service data is returned. This is used to complete the SOAP request.

Service selection (Basic endpoint address setting)

- When Destination is NOT a SOAP/HTTP Web service
  - Use RegistryLookup node to retrieve entity metadata (this can include endpoint information)
    - Set match policy = One
  - Use any transformation node to set the endpoint address and add an appropriate transport header



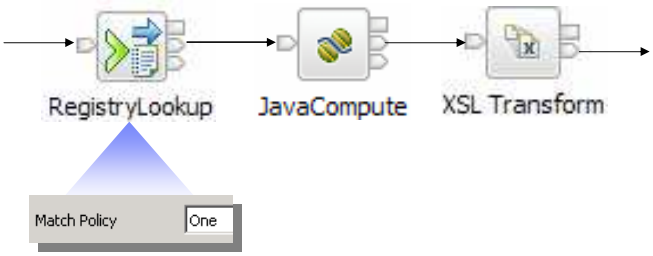
In this example, the target service is not a Web service, but may be provided by a different protocol. For example, this could be a service provided by an MQ interface.

The RegistryLookup node can be used to return the metadata of the service. This can contain any information, and could include information about how to address and invoke the service.

Additional nodes will be required to set the connection details of the service, and create appropriate message header information.

Dynamic XSL transformation

- Transforming a message using a style sheet retrieved from WebSphere Service Registry and Repository
  - Use RegistryLookup node to retrieve XSL style sheet
    - Set match policy = One
  - Use a transformation node to copy the returned style sheet into the input message
  - Use the XSLTransform node to transform the message using the embedded style sheet

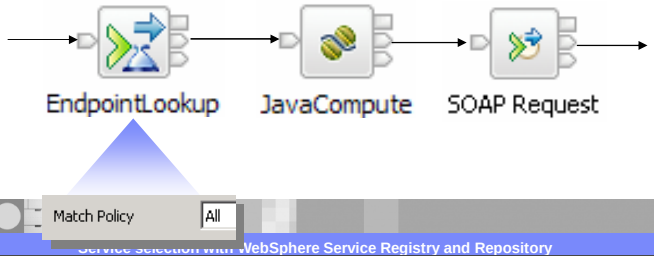


This example shows how an XSL style-sheet can be stored in WebSphere Service Registry and Repository, and retrieved by the message flow for subsequent use within the flow.

The RegistryLookup node retrieves the style-sheet, which is then passed to subsequent nodes in the Local Environment. In this case, the Java Compute node retrieves the style-sheet, and copies it into the message payload. This is then processed by the XSL Transform node, using the embedded style-sheet processing.

Selection between multiple services

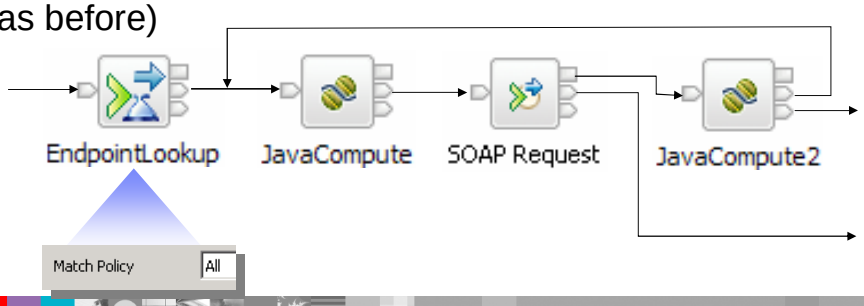
- For example, choosing between premium and standard service
  - Use EndpointLookup node to retrieve a set of endpoints
    - Set match policy = All
  - Use a transformation node to select the required service and copy the endpoint information into the correct place for the SOAP node
- Alternatively, branch earlier in the message flow and use two EndpointLookup nodes



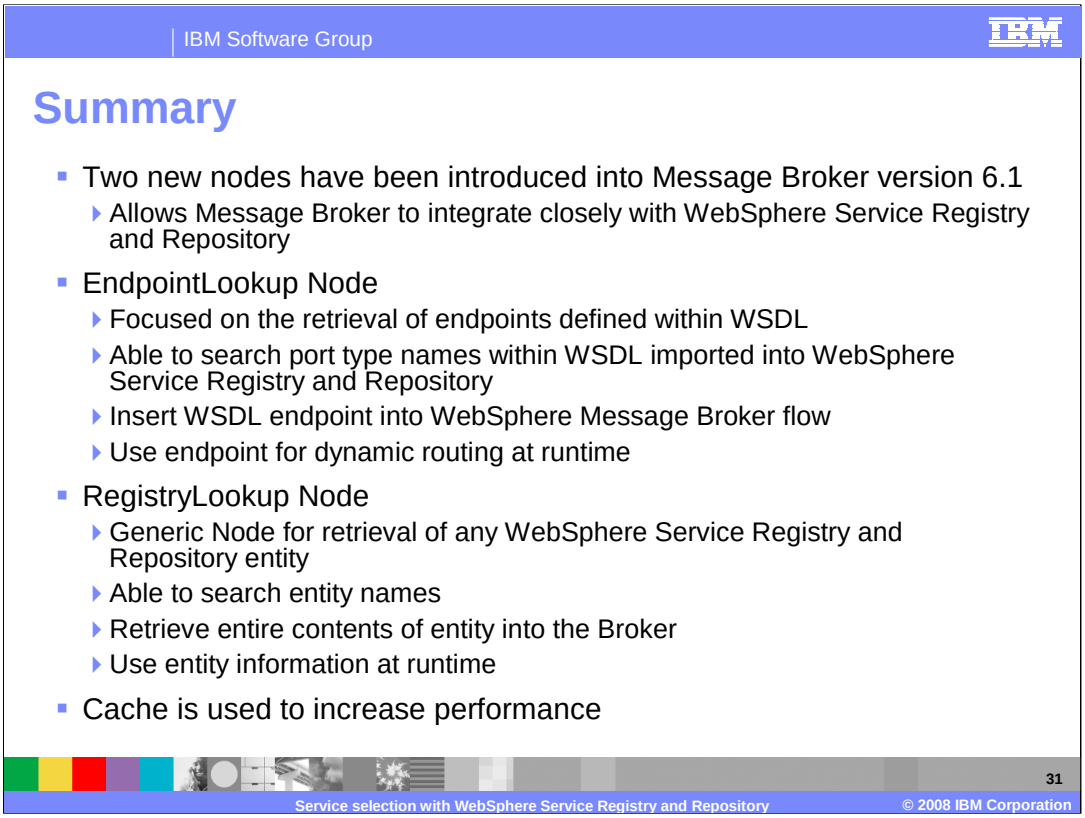
This example represents the scenario where the EndpointLookup node provides information on several services. All information is returned to the Lookup node, and then passed to the Java Compute node to make a decision, before making the request in the SOAP request node.

Alternative service provider

- The Failure terminal is used to denote problems accessing services with the SOAP Request node
  - In this case, mark the endpoint as failed, loop round and select new service
  - Remember to cope with “all services failed” situation
- Use RegistryLookup for non SOAP/HTTP service selection (as before)



This example shows a scenario where the SOAP Request node receives a failure, which is propagated to the second Java Compute node. This node examines the cause of this failure, and sends the output back as input to the first Java Compute node, and the SOAP Request is then re-invoked, but possibly accessing a different Web service.



In summary, Message Broker version 6.1 has introduced two new nodes to support WebSphere Service Registry and Repository.

The EndpointLookup node is used to retrieve information about the service endpoints. Service definitions in the form of WSDL is retrieved from WebSphere Service Registry and Repository, and used by the message flow.

The RegistryLookup node is used to retrieve generic information about any artifact stored in WebSphere Service Registry and Repository. This entity information can be used by the message flow to make runtime decisions. This is not restricted to services which are defined using WSDL.

Finally, a cache mechanism can be used to ensure that optimum performance levels are maintained.

References

- IA9Q: WebSphere Message Broker Version 6 client for WebSphere Service Registry and Repository
  http://www.ibm.com/support/docview.wss?uid=swg24014652
- Publishing a Web service to WebSphere Service Registry and Repository from CICS® Transaction Server version 3.1:
  http://www.ibm.com/developerworks/websphere/library/techarticles/0610_millwood/0610_millwood.html
- Introducing WebSphere Service Registry and Repository - Part 1:
  http://ltsbwass001.sby.ibm.com/cms/developerworks/websphere/library/techarticles/0609_mckee/0609_mckee.html
- Introducing WebSphere Service Registry and Repository - Part 2:
  http://www.ibm.com/developerworks/websphere/library/techarticles/0609_mckee2/0609_mckee2.html
- Introducing WebSphere Service Registry and Repository APIs:
  http://www.ibm.com/developerworks/websphere/library/techarticles/0611_baldwin/0611_baldwin.html
- Build flexible ESB mediations with MessageBroker and WebSphere Service Registry and Repository:
  http://www.ibm.com/developerworks/websphere/library/techarticles/0610_patten/0610_patten.html
- WebSphere Service Registry and Repository information center
  http://publib.boulder.ibm.com/infocenter/sr/v6r0/index.jsp
- WebSphere Service Registry and Repository product page
  http://www.ibm.com/software/integration/wsrr/index.html

For further reference, there are several articles that discuss WebSphere Service Registry and Repository .

Feedback

Your feedback is valuable

You can help improve the quality of IBM Education Assistant content to better meet your needs by providing feedback.

- Did you find this module useful?
- Did it help you solve a problem or answer a question?
- Do you have suggestions for improvements?

Click to send e-mail feedback:

mailto:iea@us.ibm.com?subject=Feedback_about_WMB61_IEA_WSRR.ppt

This module is also available in PDF format at: ../WMB61_IEA_WSRR.pdf

You can help improve the quality of IBM Education Assistant content by providing feedback.

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:

AIX CICS DB2 IBM WebSphere z/OS

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

J2EE, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

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 or changes in the products or programs 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 (for example, 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.