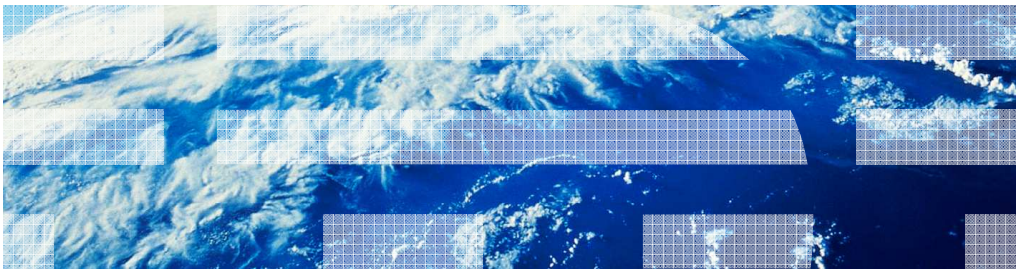
WebSphere Business Process Management

WebSphere Integration Developer
WebSphere Enterprise Service Bus
WebSphere Process Server

Routing primitives utilizing registry lookup information



WebSphere software

This presentation introduces the routing primitives that perform a registry lookup to obtain information used by the flow, either influencing routing within the flow or identifying endpoints.

Goals and agenda

- Goal
  - Provide an introduction to mediation primitives with these characteristics:
    - Classified in WebSphere® Integration developer as routing primitives
    - Perform a registry lookup to obtain information used to influence routing
- Agenda
  - Contrast basic characteristics of these primitives
  - Introduce the basic functionality the endpoint lookup primitives
    - Endpoint lookup
    - UDDI endpoint lookup
    - Gateway endpoint lookup
  - Discuss dynamic invocation following an endpoint lookup
  - Introduce the basic functionality of:
    - Policy resolution
    - SLA check

The goal of this presentation is to introduce a set of primitives that share some common characteristics. These primitives are classified within WebSphere Integration Developer as routing primitives. Within that grouping, these are the primitives that have the capability to perform a registry query, obtaining information that ultimately affects some aspect of routing or other mediation behavior. How routing is affected varies, such as selection of endpoint addresses or controlling flow path within the mediation.

The presentation starts out by highlighting and contrasting the basic characteristics of all these primitives. The basic functionality of the three endpoint lookup primitives is then presented, specifically the endpoint lookup primitive, the UDDI endpoint lookup primitive and the gateway endpoint lookup primitive. This is followed by a section looking at dynamic invocation considerations in scenarios that include one of these primitives.

Finally, the basic functionality of the other registry based routing primitives is examined, specifically the policy resolution primitive and the SLA check primitive.

Contrasting the characteristics

Characteristic	Endpoint Lookup	UDDI Endpoint Lookup	Gateway Endpoint Lookup***	Policy Resolution	SLA Check
Registry used	WSRR*	UDDI**	Built-in	WSRR*	WSRR*
Registry instance identified by	Admin name	Admin name	Singleton	Admin name	Admin name
Query based on	Port type and qualifiers	UDDI information model	Virtual endpoint name	Mediation or endpoint	Endpoint, consumer and context IDs
Supported endpoint types	Almost all	Web service	Web service	Almost all	Almost all
Sets endpoint lookup context	Yes	Yes	Yes		
Sets target address and alternate target addresses	Match policy	Match policy	Yes		
Sets dynamic promoted property alias values				Yes	
Controls flow path by output terminal fired					Yes

* WSRR – WebSphere Service Registry and Repository

** UDDI - Universal Description, Discovery and Integration

*** Gateway endpoint lookup summary ignores characteristics of the action lookup method

3 Routing primitives utilizing registry lookup information

This chart is a summary of the similar and contrasting characteristics of the primitives that make registry lookups. Those shown in blue are used to discover endpoint addresses that can be used for dynamic invocation of services. Those in green use information from the registry to control mediation flow behavior.

The endpoint lookup primitives each uses a different registry to perform the lookup, while the primitives that control behavior each make use of WebSphere Service Registry and Repository.

Except for the built-in registry, the endpoint for the registry to use is identified by an administrative name of a registry configuration object maintained in the server configuration. The built-in registry is a singleton configured in the server itself.

Each of the primitives has a unique set of query parameters. These are explained in the subsequent sections for the individual primitives.

The next characteristic is the supported endpoint types. Those primitives utilizing WebSphere Service Registry and Repository support all endpoint types except for EJB and EIS bindings. The UDDI and gateway endpoint lookup primitives only support Web service endpoints. Also, for the endpoint lookup primitives, the endpoints are the result of the query, whereas for the other primitives, the endpoint is part of the data used to make the query.

All of the endpoint lookup primitives set the endpoint context in the SMO. The gateway endpoint lookup also sets the target and alternate target addresses in the SMO, whereas the other two only conditionally set these fields, based on the setting of a match policy property.

The policy resolution primitive sets values into the dynamic properties context, providing values for promoted property aliases that will influence subsequent primitives in the flow that use those aliases.

The SLA check primitive controls the flow path taken based on the response received from the registry.

Section

Endpoint lookup primitive

The endpoint lookup primitive is described in this section.

Endpoint lookup primitive - Overview of function

- Performs a lookup of endpoints from WebSphere Service Registry and Repository
  - Lookup is based on selection criteria
  - Initializes the SMO with results for downstream use by the mediation flow
- Numerous criteria can be used for selection
  - Which registry to use for the lookup
    - Available registries are administratively defined within a WebSphere cell
  - Specifics of the requested service port type
    - Name, Namespace, Version
  - Latest compatible version of a versioned SCA module
  - Binding type specifies the protocol supported by the binding
  - Associated classification, based on Web Ontology Language (OWL)
  - Associated properties and property values
- Support almost all the binding types
  - Requires WebSphere Service Registry and Repository V7
  - Exceptions are the EJB bindings and EIS bindings

The endpoint lookup primitive uses the WebSphere Service Registry and Repository to find service provider endpoints, based on a set of selection criteria. The results of the lookup are reflected in the service message object, allowing them to be used downstream in the mediation flow.

There are many different criteria that can be used for selection. First, multiple registries can be configured for use by the servers within a WebSphere cell, and the endpoint lookup can specify which of these registries should be used. The service port type of the requested service can be qualified based on name, namespace and version. Also, if you are using versioned SCA modules, you can request the latest compatible version to a specified version.

There is a binding type property that is used to indicate which protocol the selected bindings support.

The Web Ontology Language (OWL), provides a classification system which can be used as part of the selection criteria. Registered services can be associated with name value pairs which can also be used as part of the selection criteria. These selection criteria reflect the underlying capabilities of the WebSphere Service Registry and Repository.

The endpoint lookup primitive supports all the binding types supported by WebSphere Service Registry and Repository V7. The only SCA binding types that are not supported are the EJB bindings and the EIS bindings used by WebSphere adapters.

Endpoint lookup primitive - Overview of function (continue)

- EndpointLookupContext in SMO is set, defining for each selected endpoint
  - Reference including endpoint address, port type and other properties
  - Annotations associated with the endpoint in the registry
- SMO header optionally set for dynamic endpoint usage
  - Setting of the target and alternate target addresses
  - Controlled by the match policy property
- Match policy settings
  - Return first matching endpoint and set routing target
  - Return endpoint matching latest compatible SCA service version
  - Return all matching endpoints and set alternate routing targets
  - Return all matching endpoints

The EndpointLookupContext in the SMO contains the selected endpoints returned from the registry. For each endpoint there is the endpoint reference which defines the URL, the port type and other properties. Each endpoint also has registry annotation information defining properties, classifications and relationships defined for the endpoint in the registry.

The SMO header fields, target address and alternate target addresses, are optionally set with the results of the lookup. These are the fields used by service invoke primitives and callout nodes for dynamic endpoint invocation. The setting of these fields is controlled by specification of the match policy property. Enabling the setting of these fields allows the primitive to initialize the SMO, preparing the SMO for dynamic endpoint usage by service invoke primitives and callout nodes.

The match policy setting determines how the target and alternate target addresses are affected by the endpoint lookup primitive.

The first match policy setting is called return first matching endpoint and set routing target. This policy results in the target address being set and causes the alternate target addresses to be cleared of any addresses it might already contain. At this point, the SMO is ready to be used by a service invoke primitive or a callout node configured for dynamic invocation. However, it is not ready to be used for service call retry with alternate target addresses.
The next policy, called return endpoint matching latest compatible service version, has the identical affect on the SMO as the previous policy, and therefore results in the same downstream behavior. The difference between them is that this policy incorporates the SCA module version into the criteria when making the selection from the registry.

The next match policy setting is called return all matching endpoints and set alternate routing targets. This results in the target address being set with the first endpoint returned and the alternate target addresses being set with the other endpoints returned. At this point, the SMO is ready to be used by a service invoke primitive or a callout node configured for dynamic invocation and for service call retry with alternate target addresses.

Finally, there is the match policy setting called return all matching endpoints. In this case, the target address and alternate target addresses are not set. Therefore, when using this match policy, downstream processing in the mediation must perform some logic that selects an endpoint and places it into the target address. The logic might also set other endpoints in the alternate target addresses. Specifically what logic is performed depends upon your application requirements.

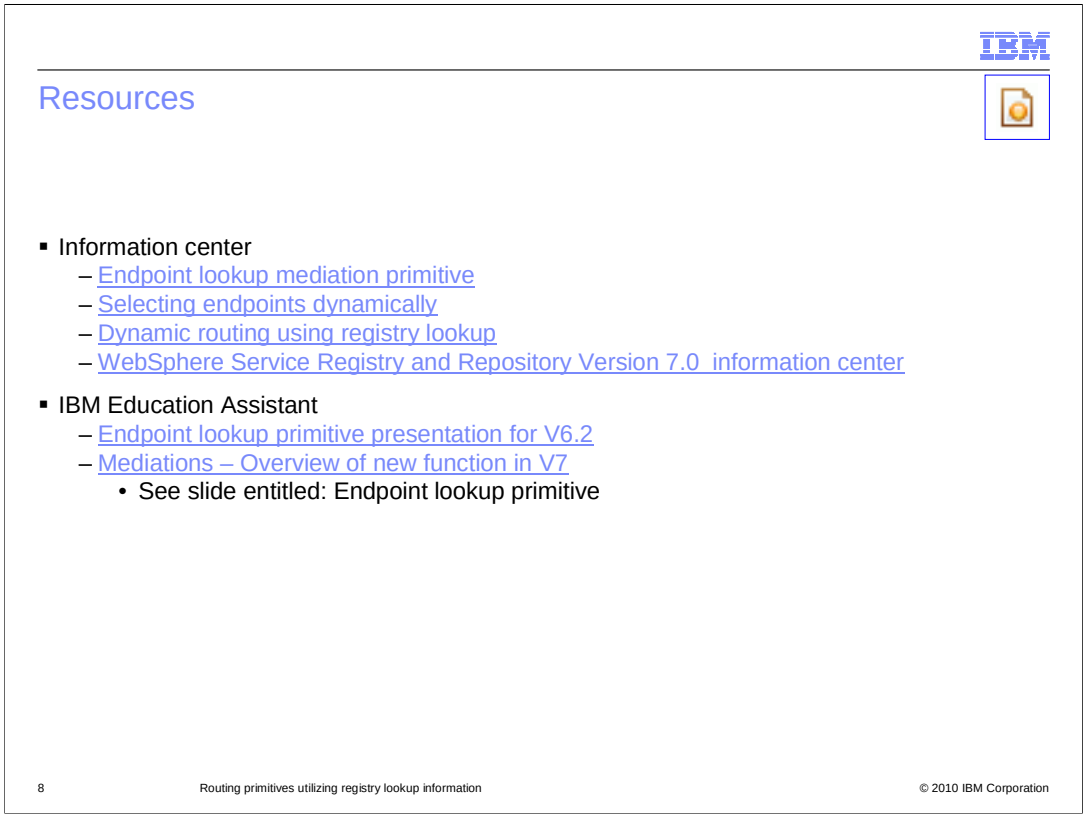
Service registry cache

- Server runtime provides a registry cache
  - Intended to boost performance for registry lookups
    - Caching is an effective strategy because registries are not normally subject to frequent updates
  - The cache is not exposed by the Mediation Flow editor
    - It is a hidden implementation detail in WebSphere Integration Developer
  - The cache is exposed to the administrator
    - Registries are administratively configured in the server
    - Registry definitions contain cache management information
    - Registry definitions are maintained at the cell level
    - Each server in the cell has one cache per configured registry

Performing frequent lookups to the WebSphere Service Registry and Repository can be expensive in terms of performance. Registries do not normally have frequent updates, so caching is a viable way to address this. Therefore, WebSphere Enterprise Service Bus and WebSphere Process Server provide a registry cache which is intended to boost the overall performance of registry lookups by reducing the number of calls made to the registry.

The endpoint lookup primitive and the mediation flow editor do not expose the registry cache. Therefore, from an integration developers perspective, the cache is hidden.

Registries are defined administratively and therefore are exposed to the administrator. The configuration of a registry includes configuration information for the management of the cache. The registry information is maintained at the cell level, and therefore the registry cache configuration information applies to all servers in the cell. There is one cache per registered registry in each server in the cell.



The links on this slide direct you to more detailed information about the endpoint lookup primitive and its use. The information center contains a section on the endpoint lookup primitive that describes its usage, behavior and defines all of its properties. The selecting endpoints dynamically link brings you to a page that links to several topics on dynamic endpoints and how they are used. This is broader coverage of the dynamic endpoint topic, and will help you put the endpoint lookup primitive into perspective in a larger picture. The dynamic routing using registry lookup includes a scenario description that introduces you to how WebSphere Service Registry and Repository is used. The last information center link is to the WebSphere Service Registry and Repository information center, where the real details of the registry are found.

There are also a couple of links to IBM Education Assistant. The first is to the endpoint lookup presentation from V6.2. This contains many details, including more detailed descriptions of the properties, SCA module version handling, management of the administrative registry definition and cache, and problem determination. The second link is to a presentation with a single slide on enhancements made to the primitive between V6.2 and V7. Essentially, support for additional binding types and the binding type property were added in V7.

Section

UDDI endpoint lookup primitive

The UDDI endpoint lookup primitive is reviewed in this section.

UDDI endpoint lookup primitive – Overview of function

- Performs a lookup of endpoints from a UDDI registry
  - Lookup is based on selection criteria
  - Initializes the SMO with results for downstream use by the mediation flow
- Selection criteria is based on UDDI information model, consisting of:
  - Businesses
  - Services
  - Technical models
  - Find qualifiers
- Usage requires understanding of OASIS UDDI V3 specification
- Works with any standard UDDI registry
  - WebSphere Application Server provides a UDDI registry implementation
- Only Web service endpoints supported
  - Supports both HTTP and JMS Web services

As the name suggests, the UDDI endpoint lookup primitive is used to perform a lookup from a UDDI registry. The primitive is configured with selection criteria used for the query to the registry and the SMO is initialized with the results that are returned. The selection criteria is based on the UDDI information model which defines businesses, services, technical models and find qualifiers. In order to effectively use the UDDI endpoint lookup, you need to be familiar with this information model, which is defined in the OASIS UDDI V3 specification. The primitive should work with any compliant UDDI registry. There is a UDDI registry implementation that is provided as part of the WebSphere Application Server product that can be installed into any WebSphere Application Server, WebSphere Process Server or WebSphere Enterprise Service Bus. UDDI only supports Web service endpoints and can be used for either HTTP or JMS Web services.

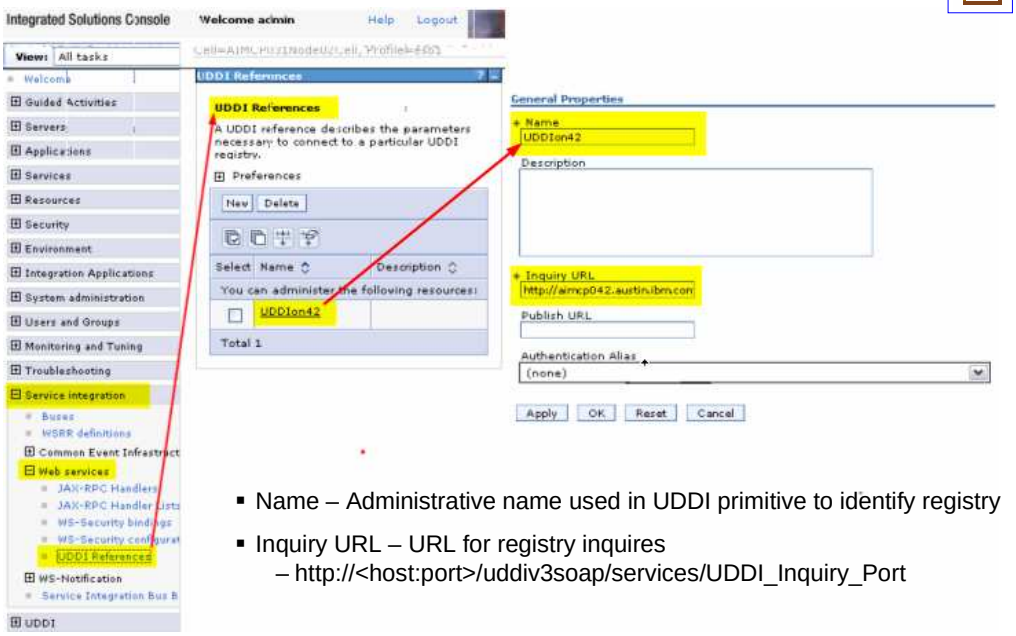
UDDI endpoint lookup primitive – Similar to endpoint lookup

- Similar is operation to endpoint lookup primitive
  - Registry definition
    - Maintained in server configuration
    - Identify in primitive using administrative name
    - Does not maintain a cache in the server
  - Primitive populates the EndpointLookupContext with results of query
  - Primitive optionally populates addresses in the SMO header target and alternate target fields
    - Based on setting of match policy property
    - Same match policies as endpoint lookup except for versioned SCA modules

The UDDI endpoint lookup is similar in behavior to the endpoint lookup primitive. There is an administrative definition maintained in the server that identifies the endpoint of the registry and the primitive identifies the registry by specification of this administrative name. However, there is no UDDI registry cache maintained in the server as there is for the WebSphere Service Registry and Repository.

The UDDI endpoint lookup updates the EndpointLookupContext in the SMO with the results of the query. A match policy property setting determines whether the primitive also sets the target address and alternate target addresses in the SMO. Other than the match policy for versioned SCA modules, the match policy settings are the same as they are for the endpoint lookup primitive.

UDDI registry administrative reference in server configuration



- Name – Administrative name used in UDDI primitive to identify registry
- Inquiry URL – URL for registry inquires
  - http://<host:port>/uddiv3soap/services/UDDI_Inquiry_Port

The screen captures on this slide show the administrative definition of a UDDI registry in the administrative console. From the menu panel on the left, navigate to the UDDI References panel by selecting Service integration, Web services, UDDI references. From here you can create new UDDI references or open existing ones. The name specified is the name you need to use in the UDDI endpoint lookup primitive. An example of the correct URL syntax for a UDDI registry is shown.

Resources

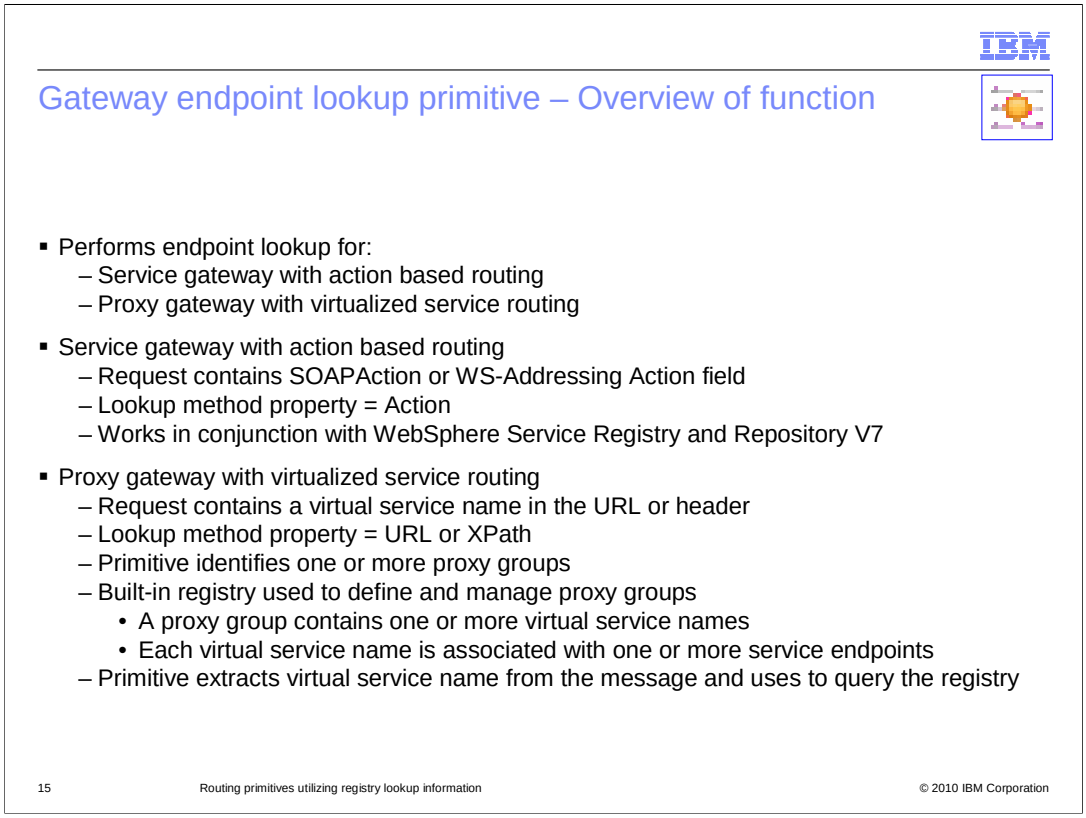
- Information center
  - UDDI endpoint lookup mediation primitive
- WebSphere Application Server information center
  - Using the UDDI registry in WebSphere Application Server
- Specifications
  - OASIS UDDI V3

The links on this slide direct you to more detailed information about UDDI and the UDDI endpoint lookup primitive. The information center contains a section on the UDDI endpoint lookup primitive that describes its usage, behavior and defines all of its properties. The next link is for the WebSphere Application Server information center, to the main page containing links to various UDDI topics. These include sections on learning about UDDI, using the UDDI registry, installing and configuring the UDDI registry in a WebSphere Application Server, and troubleshooting issues with the UDDI registry. Finally, there is a link to the UDDI specification which is published by the OASIS consortium. The link provided should take you to the latest published version.

Section

Gateway endpoint lookup primitive

The gateway endpoint lookup primitive is looked at in this section.



The gateway endpoint lookup primitive is designed to perform endpoint lookups for a couple of specialized cases. One is the service gateway with action based routing and the other is the proxy gateway with virtualized service routing.

In the case of service gateway with action based routing, you configure the lookup method to the value action. The inbound messages need to contain a SOAPAction or WS_Addressing Action field. The primitive queries WebSphere Service Registry and Repository V7 which can be configured to have actions associated with the endpoints. Endpoints with a matching action are returned.

For the proxy gateway, a virtual service name is passed in the inbound message, either as part of the URL or contained within the header. The lookup method property identifies if the virtual service name is found in the URL or by using an XPath expression to its location in the header. The primitive also identifies one or more proxy groups. Within the server there is a built-in registry that you configure to contain the proxy group information. Each proxy group contains one or more virtual service names, and each virtual service name is associated with one or more endpoints. The primitive extracts the virtual service name from the message, queries the built-in registry, which returns the appropriate endpoints for the specified virtual service name.

Gateway endpoint lookup primitive – Overview of function (continue)

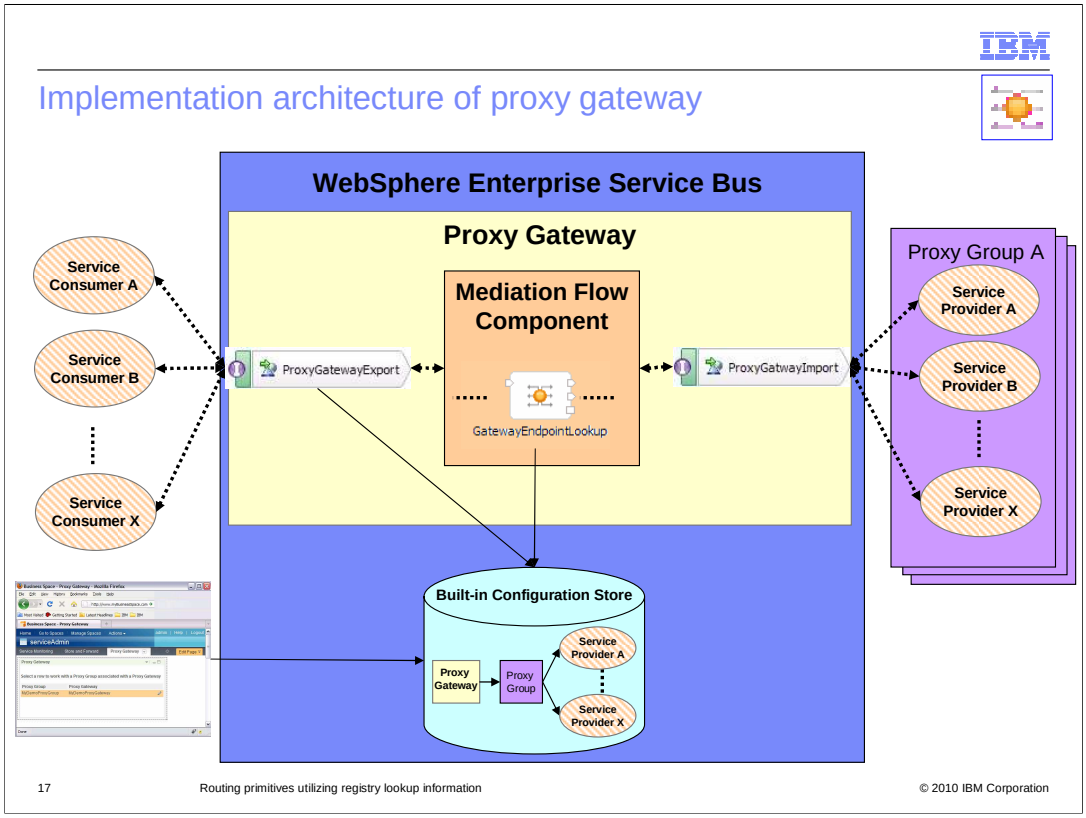
- The primitive sets the endpoint information in the SMO:
  - EndpointLookupContext
  - Target and alternate target addresses fields
- Only Web service bindings are supported
- Proxy gateway
  - Patterns explorer in WebSphere Integration Developer used to create a proxy gateway
  - Business space widget provided to define and manage the proxy groups
  - Supports generation of the WSDL needed to invoke the gateway for a virtual service
    - Invoke gateway with → http://<PGExport>/<VirtualServiceName>?wsdl
    - Returns a WSDL definition of the service provider with the endpoint and WS-Policy information associated with the proxy gateway

For both the action based and virtualized service routing, the primitive takes the result returned by the registry and sets the EndpointLookupContext and the target address and alternate target addresses in the SMO. Also, in both cases, only Web service bindings are supported

WebSphere Integration Developer provides a patterns explorer that can be used to generate the basic implementation of a proxy gateway. The gateway is generated based on some configuration parameters you provide, and then you complete the implementation to perform the specific function you require. This simplifies the work needed to develop a proxy gateway.

When you install a proxy gateway application into the server, the proxy groups specified in the gateway lookup primitive are registered with the built-in registry. You then use a business space widget that allows you to administer the virtual service names and endpoints associated with the proxy group.

When a client invokes a service endpoint through a proxy gateway, they need to have a WSDL definition that has both the service providers definition combined with the gateway's endpoint and Web service policy information. This WSDL can be generated by invoking the gateway, passing a URL containing the proxy gateway export, the virtual service name, with ?wsdl appended to the end.

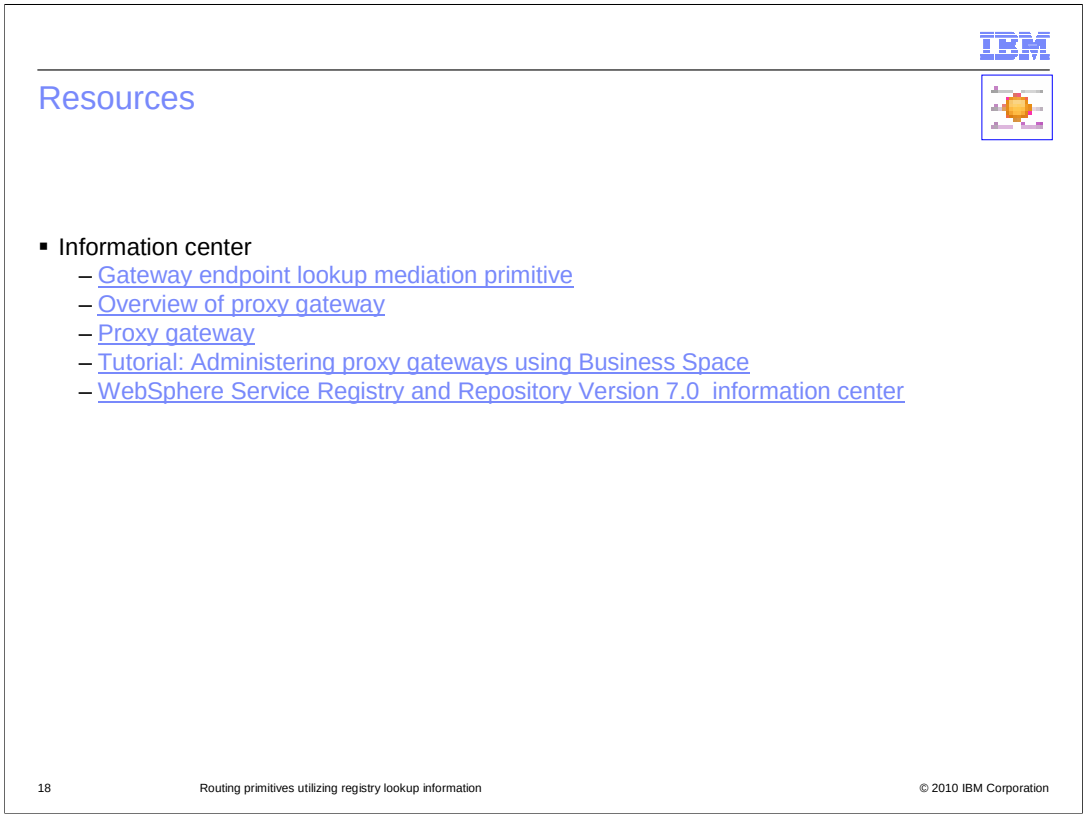


The architecture of a proxy gateway is shown in this diagram, providing a better view of how the gateway endpoint lookup participates in the proxy gateway. At the middle of the picture you can see the gateway endpoint lookup within the mediation flow component, showing that there is likely flow logic of some kind before and after it. The mediation flow component is contained within an SCA mediation module with an export and import that support the service gateway interface. This is the proxy gateway application.

At the bottom center of the slide is shown the built-in registry, containing proxy groups with the virtual services and defined endpoints. You can see that both the export and the gateway endpoint lookup interact with this registry. On the lower left you can see the business space widget that you use to administer the registry.

The service consumers are seen on the left and the service providers are seen on the right. The service provider endpoints are associated with proxy groups defined in the registry.

A basic flow goes like this. A consumer invokes the export in the proxy gateway specifying a virtual service name. The message is passed from the export to the mediation flow component where some processing takes place, such as header manipulation. The gateway endpoint lookup then extracts the virtual service name and queries the registry for endpoints associated with that virtual name defined within a proxy group specified by the primitive. The endpoints returned are set into the SMO, which are used to control the dynamic invocation to the specified service provider through the service gateway import.



The links on this slide direct you to more detailed information about the gateway endpoint lookup primitive and its use. The information center contains a section on the gateway endpoint lookup primitive that describes its usage, behavior and defines all of its properties. The overview of proxy gateway explains the basic concepts of how a proxy gateway works, reinforcing what has been presented here. The next link, proxy gateway, brings you to a page with links to several topics mostly concerned with how you administer a proxy gateway using the business space widget. To complement that, the next link is to a tutorial that also brings you through administering a proxy gateway, providing specific steps and screen captures to guide you.

Finally, the link to the WebSphere Service Registry and Repository Version 7.0 information center is provided for information about configuring the repository to support action base routing.

Section

Dynamic invocation following an endpoint lookup

In this section there is a short discussion about managing dynamic invocation in a flow after one of the endpoint lookup primitive types has initialized the SMO with target service information.

Dynamic invocation and endpoint lookup primitives

- Basic Background
  - Service invoke primitives and callout nodes support dynamic invocation
  - The target and alternate target fields in the SMO specify the endpoints to call
  - Endpoint lookup primitives set the target and alternate target fields
  - Nothing additional needs to be done between the endpoint lookup and the dynamic call
- However, you might need to consider
  - Is the address set in the target actually the preferred endpoint to call?
  - Is the ordering of alternate addresses appropriate when retry with alternate address?
- Providing your own logic
  - EndpointLookupContext contains all of the results of the registry query
    - Properties, classifications, relationships
  - Provide mediation flow logic that initializes the target and alternate target addresses according to your criteria

Service invoke primitives and callout nodes can be configured to support dynamic invocation. When they are configured this way, the endpoint supplied in the target address field of the SMO header is used to invoke the service. They can also be configured to retry failing calls by using the endpoints supplied in the alternate target addresses field. These are the target address fields that the various endpoint lookup primitives set when configured to do so. When this is done, the endpoint lookup fully initializes the target fields so that dynamic invocation can be done without further processing between the endpoint lookup primitive and the service invoke primitive or callout node.

However, when designing your flow, you should be aware that the endpoint set into the target address field might not be the optimum one for you scenario. You might have some criteria you want to apply to selecting which of the returned endpoints is most appropriate to use. This is also true of the order of endpoints in the alternate target addresses field.

In the case where you have some criteria that you require to be applied, you can make use of the results set into the EndpointLookupContext. This includes additional information for the endpoints returned from the registry, such as property settings, classifications and relationships. You can use this information to perform some logic to select the most appropriate endpoint based on your criteria, and then setting the target address and alternate target addresses accordingly.

Resources

- IBM Education Assistant presentations from V6.2
  - Dynamic invocation
  - Service call retry
  - Service invoke mediation primitive

The links here direct you to presentations that provide additional background information relevant to the discussion presented on the previous slide. They cover the topics of dynamic invocation, the automatic service call retry capabilities and the workings of the service invoke primitive.

Section

Policy resolution primitive

The next primitive to be examined is the policy resolution primitive.

Policy resolution primitive – Overview of function

- Performs lookup of mediation policies from WebSphere Service Registry and Repository
- Mediation policies are used to dynamically control the mediation flow
  - Used to dynamically set property values for promoted properties
- Mediation policies can be associated with
  - The SCA module containing the flow
  - An endpoint identified in the target address field of the SMO
- Policy scope property set to specify which polices you want to retrieve
  - Module
  - Target service
  - Intersection
- Gate conditions can be associated with policies
  - Condition added as attachment to the policy in the registry
  - Policy resolution primitive specifies the name of the condition and an XPath to its value

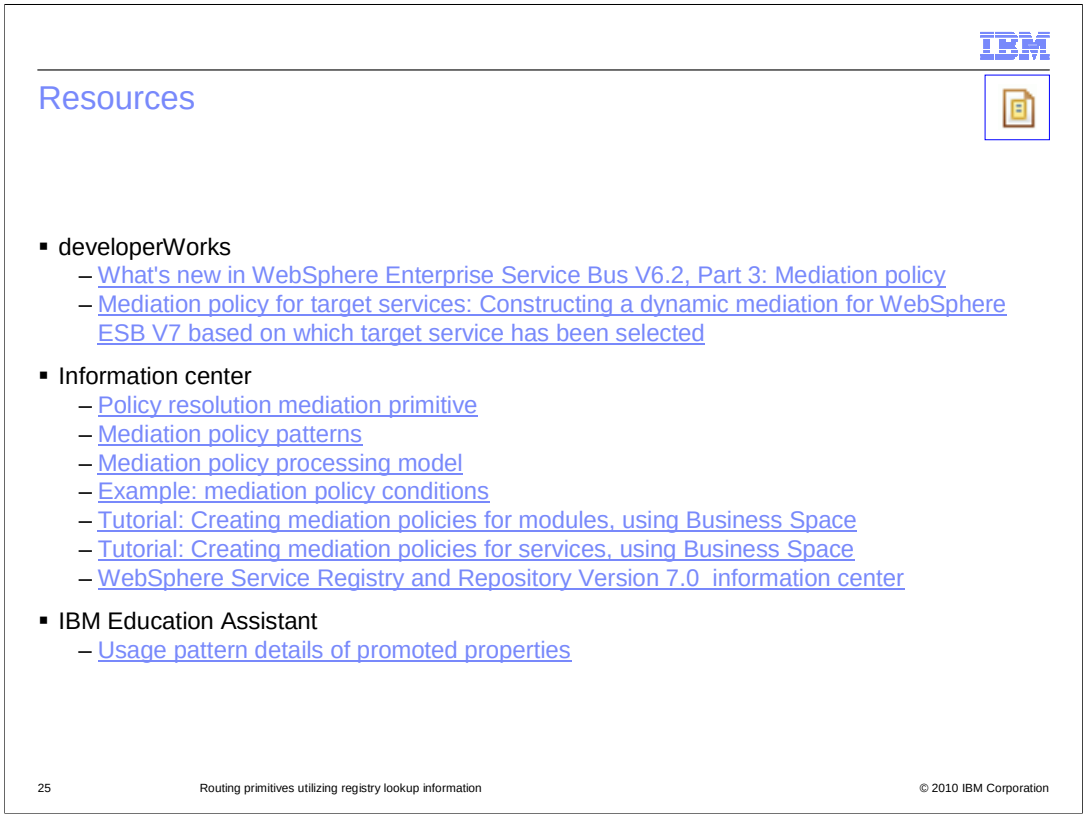
The policy resolution primitive is used to lookup mediation policies from the WebSphere Service Registry and Repository. Mediation policies are used to dynamically control the mediation flow by setting property values for promoted properties that influence the processing of other primitives in the flow. The policies can be associated with an SCA module within which the policy resolution primitive is running or with a service endpoint that is identified by the value in the target address field of the SMO. You must configure the policy resolution primitive using the policy scope property to indicate which of the policy types you want retrieved. One of the options is intersection, which merges the policies for module and target service. You can use gate conditions associated with policies to further qualify which policies are returned. These conditions are associated with the policies in the registry The primitive is configured to include the condition in the query. The value for the condition is obtained from a field in the message identified using an XPath expression.

Business space widgets supporting policy

- Business Space widgets provided to interface to WebSphere Service Registry and Repository for policy administration
  - Easier to administer than when using WebSphere Service Registry and Repository interface directly
  - Interface is designed to reflect the WebSphere ESB policy concepts
- Business space widgets can be used to
  - Create new mediation policies
  - Create policy attachments
    - Module scope
    - Target Service scope
  - Definition of gate conditions

There are business space widgets provided that enable administration of policy artifacts in the WebSphere Service Registry and Repository. These might be administered using the WebSphere Service Registry and Repository administrative interfaces. However, the business space widgets provide an interface that is closely aligned with WebSphere ESB concepts of policy and therefore are more intuitive when configuring policy to be used with WebSphere ESB.

The business space widgets provided enable you to create new mediation polices, create the policy attachments at either the module or target service scope and associate gate conditions with the attachments.



The policy resolution primitive cannot be understood without some understanding of mediation policy as it applies in WebSphere Enterprise Service Bus. This slide provides you links to many topics related to mediation policy that will help you understand the big picture.

To start out, there are a couple of developerWorks articles. The first was published in the V6.2 timeframe and addresses the topic of mediation policy, which was first introduced in that release. The second one addresses the scenario where mediation policies are attached to target services, which is a new capability added in V7.

There are several links to the information center, the first being to the policy resolution primitive itself, describing its usage, behavior and defines all of its properties. Following that are links that describe mediation policies and how they are used, along with an example and a couple of tutorials. A link to the WebSphere Service Registry and Repository for V7.0 is also provided as information about configuring the registry with policy information is provided there.

Finally, in IBM Education Assistant, the presentation on usage patterns for promoted properties contains several slide describing the use of dynamic properties, a fundamental element of mediation policy in WebSphere ESB.

Section

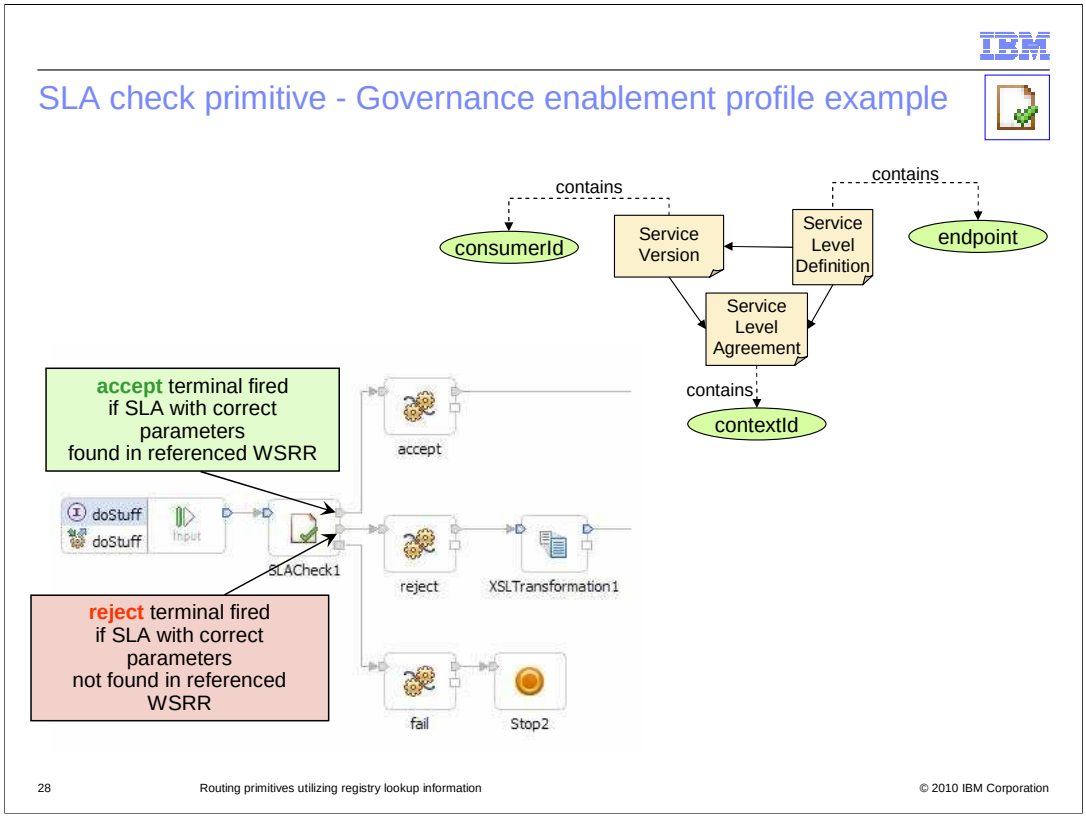
SLA check primitive

This section introduces the SLA check primitive. SLA stands for service level agreement.

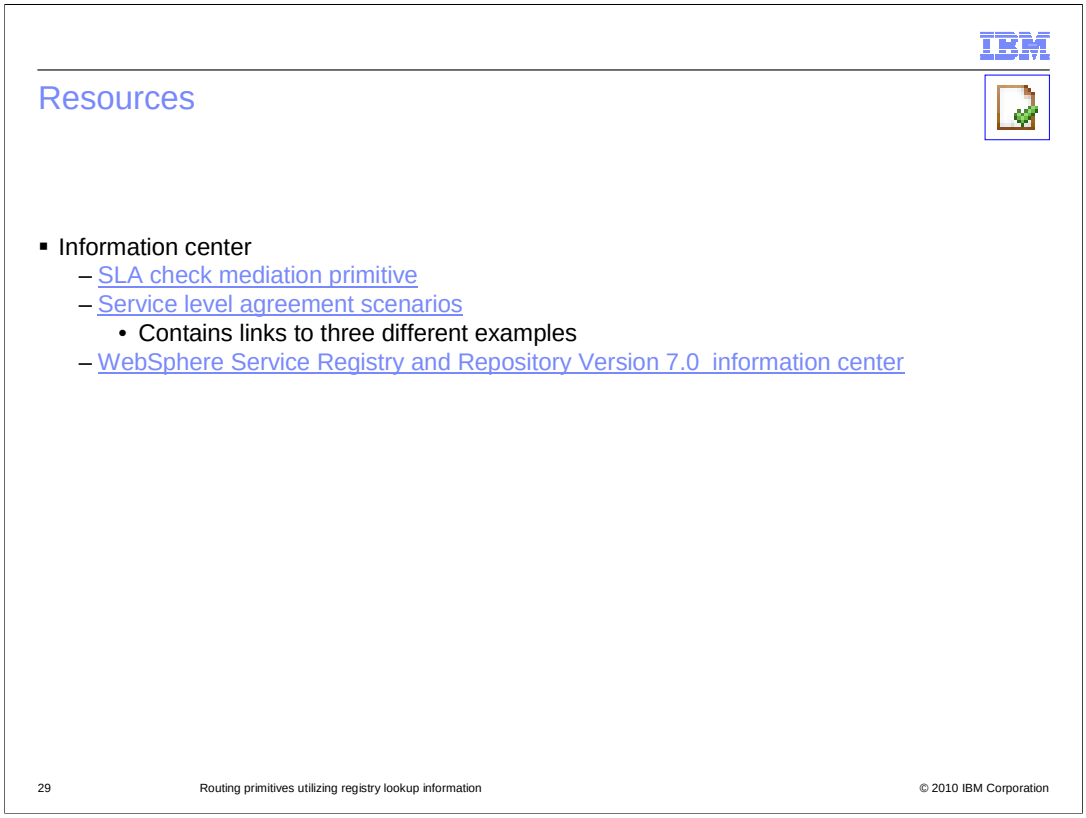
SLA check primitive – Overview of function

- Provides capability to enforce a service level agreement (SLA) within a mediation flow
- SLAs are maintained in WebSphere Service Registry and Repository
  - They are part of governance enablement profile
- Enforces SLAs that can be between
  - Client and mediation
  - Mediation and service
  - Client and service
- Three values can be sent in the query
  - Endpoint – required
  - Consumer ID – optional
  - Context ID – optional
- The registry responds indicating if the there is an acceptable SLA
  - The SLA check primitive fires different output terminals depending upon the response

The purpose of the SLA check primitive is to enable the enforcement of service level agreements within a mediation flow. The service level agreements are maintained in WebSphere Service Registry and Repository as part of the governance enablement profile. The service level agreements can be between the client and the mediation, between the mediation and the service or between the client and the service. It is through queries to the registry that the check is performed, passing in an endpoint, consumer ID and context ID. All three of these can be specified as a literal value or as an XPath to the value. The endpoint is required and the consumer ID and context ID are optional. When the primitive sends the query, the registry responds indicating if it accepts or rejects the request based on the configured SLAs. Different output terminals are fired by the primitive based on the response.



In the upper right corner you can see the relationships that come into play in the registry. There is a service level definition that contains an endpoint and identifies a service version and a service level agreement. The service level agreement contains the context ID. The service version contains the consumer ID. In the lower left is a mediation flow containing an SLA check primitive. When the registry responds that a service level agreement has been found that matches the parameters in the query, the primitive fires the accept terminal, otherwise it fires the reject terminal.



Here are the links to additional resources for understanding the SLA check primitive. The first is to the documentation for the primitive itself, describing its usage, behavior and detailing the properties. The next link describes service level agreement scenarios. It also contains links to three examples, covering the SLAs between client and the mediation, between the mediation and the service and between the client and the service. The last link is to the information center for WebSphere Service Registry and Repository V7.0 where the details about governance enablement profiles and service level agreements is found.

Summary

- Examined the routing primitives that perform registry lookups
  - Contrasted basic characteristics of these primitives
  - Introduced the basic functionality the endpoint lookup primitives
    - Endpoint lookup
    - UDDI endpoint lookup
    - Gateway endpoint lookup
  - Discussed dynamic invocation following an endpoint lookup
  - Introduced the basic functionality of:
    - Policy resolution
    - SLA check

The presentation began by highlighting and contrasting the basic characteristics of the routing primitives that perform registry lookups. The basic functionality of the three endpoint lookup primitives was presented, specifically the endpoint lookup primitive, the UDDI endpoint lookup primitive and the gateway endpoint lookup primitive. This was followed by a section looking at processing scenarios that include one of these primitives. Then the basic functionality of the other registry based primitives was examined, specifically the policy resolution primitive and the SLA check primitive.

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_WBPMv7_RoutingRegistryPrimitives.ppt

This module is also available in PDF format at: ../WBPMv7_RoutingRegistryPrimitives.pdf

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

Trademarks, disclaimer, and copyright information