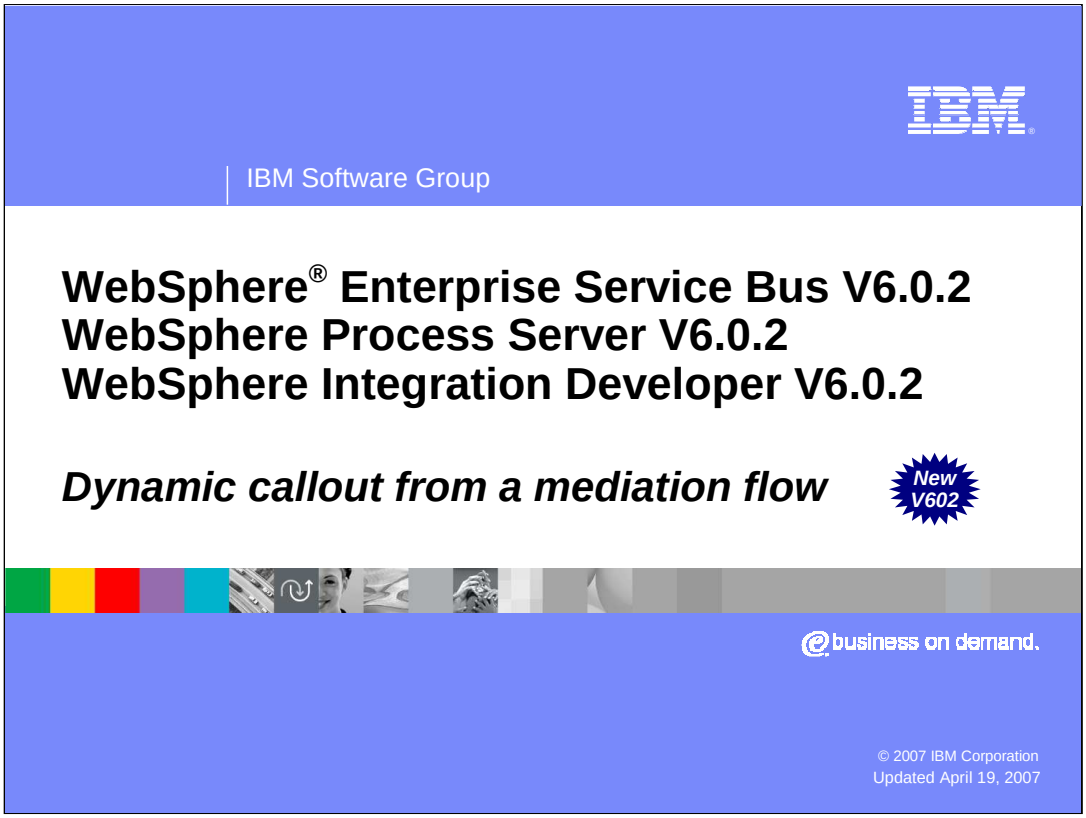
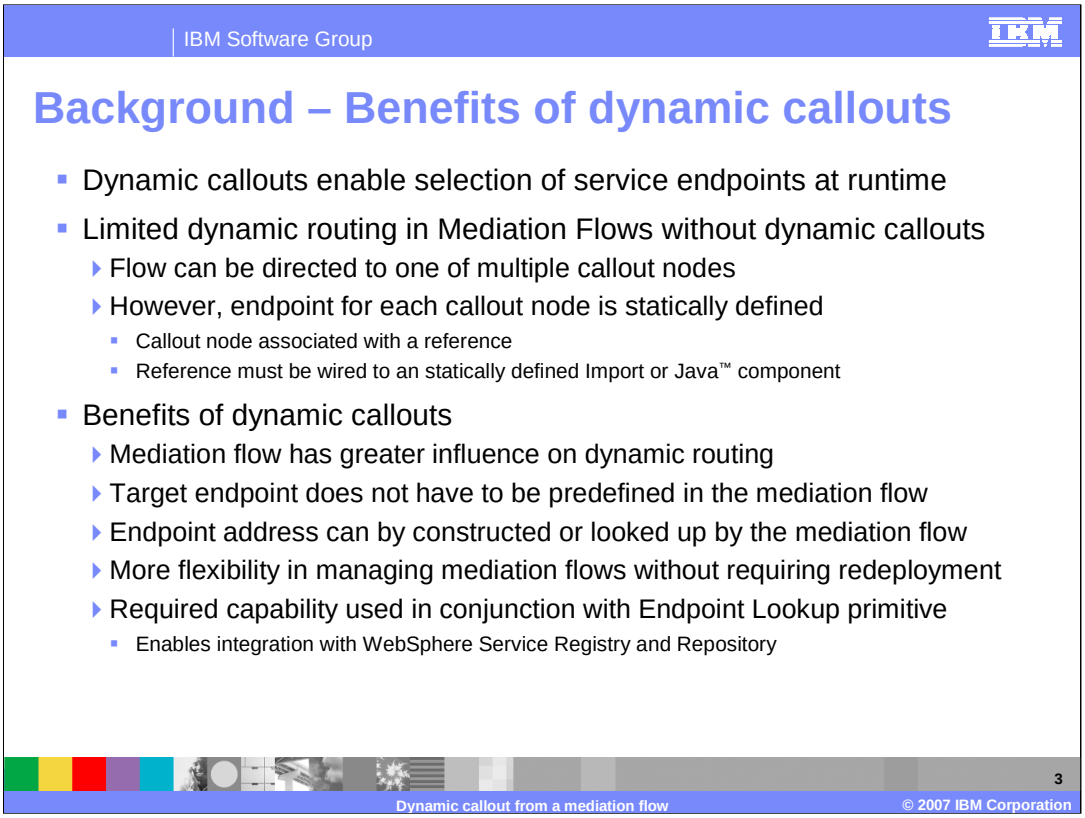


This presentation provides a detailed look at the dynamic callout capabilities of a mediation flow, enabling the dynamic specification of service provider endpoints. The capability was first introduced in version 6.0.2.

Goals

- Introduce dynamic callouts
  - Understand the benefits provided
  - Look at how it works at runtime
  - Look at the development time configuration
  - Examine a usage scenario
  - Provide information to help with problem determination

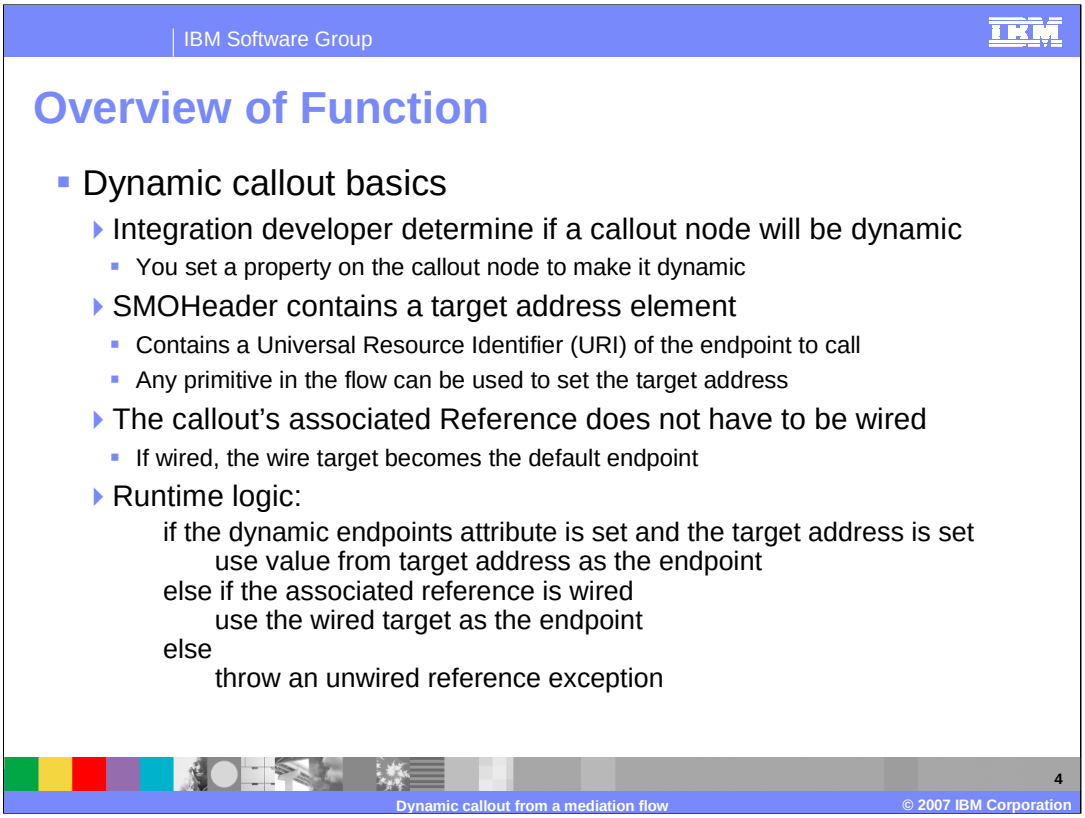
The goal of this presentation is to provide you with a full understanding of dynamic callouts. The dynamic callout capability is used to enable the dynamic runtime selection of service provider endpoints. This presentation starts by looking at the benefits that dynamic endpoints provide. Then the function is looked at in terms of how it behaves at runtime and how it is configured using WebSphere Integration Developer. To help you understand how this function might be used, an example usage scenario is explained. Finally, this capability has potential for runtime errors to occur, so problem determination information is provided.



Dynamic callouts enable a mediation flow to select or define a service endpoint dynamically at runtime. Mediation flows without dynamic endpoints can still make runtime routing decisions, but that capability is limited. They can choose between multiple callout nodes defined in the flow. However, each of those callout nodes is statically defined through its association with a reference on the mediation flow component. That reference must be wired to a statically defined Import or to a Java component.

Dynamic callouts provide several benefits. They enable the mediation flow to have a much greater influence over routing decisions. Rather than the endpoint address being predefined as part of the mediation flow, it can be constructed or looked up at runtime. One of the great benefits is that an endpoint used by the flow can be changed without requiring the mediation application to be redeployed.

The use of repositories to hold service provider information is an increasingly important aspect of service oriented architecture. Integration of the WebSphere Enterprise Service Bus and WebSphere Process Server with the WebSphere Service Registry and Repository is a key part of this. The Endpoint Lookup mediation primitive, which provides integration with the WebSphere Service Registry and Repository, would not be useful without this dynamic callout capability.



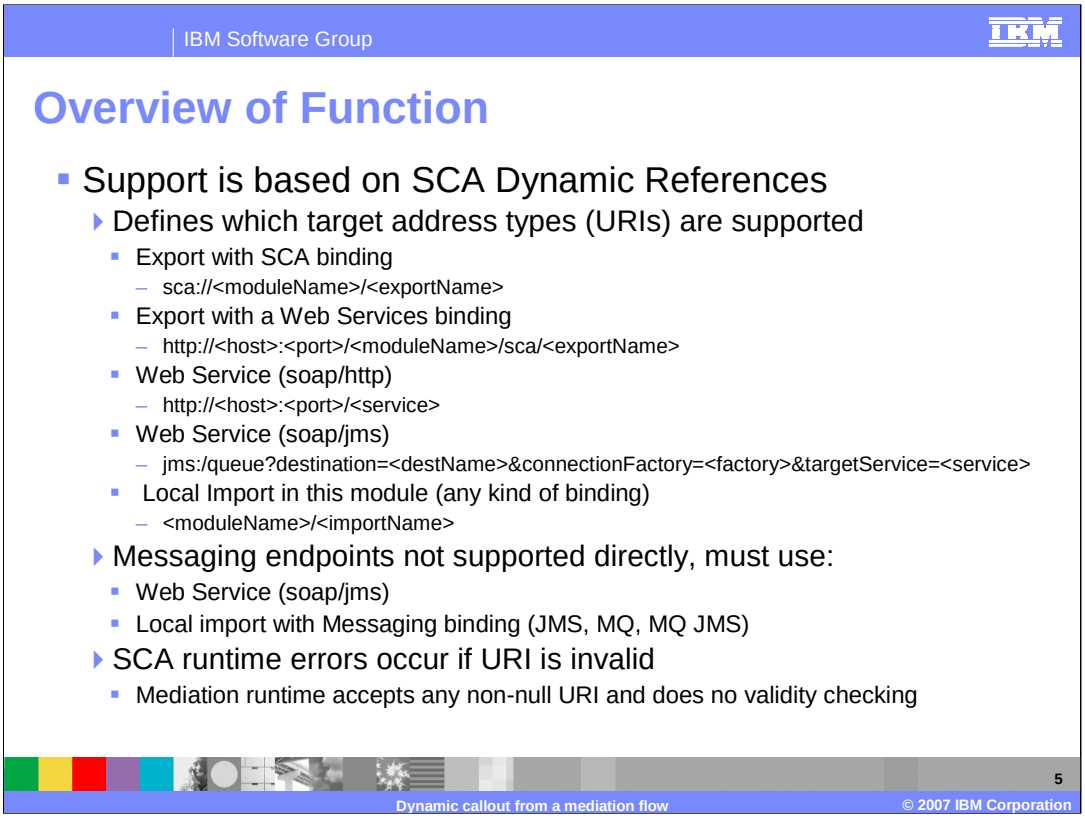
The use of dynamic callouts are really fairly straight forward.

It is your decision whether or not to make a callout dynamic. There is an attribute in the Properties view of the Callout node that you select to enable it.

There is a target address element in the SMOHeader that needs to be set. When developing the mediation flow, you need to have the logic defined that sets the target address element with the URI identifying the endpoint.

As with all callouts, it must be associated with a Reference on the Mediation Flow Component. However, the Reference does not need to be wired on the assembly diagram. If it is wired, the wire target becomes the default endpoint for cases where the target address is not set.

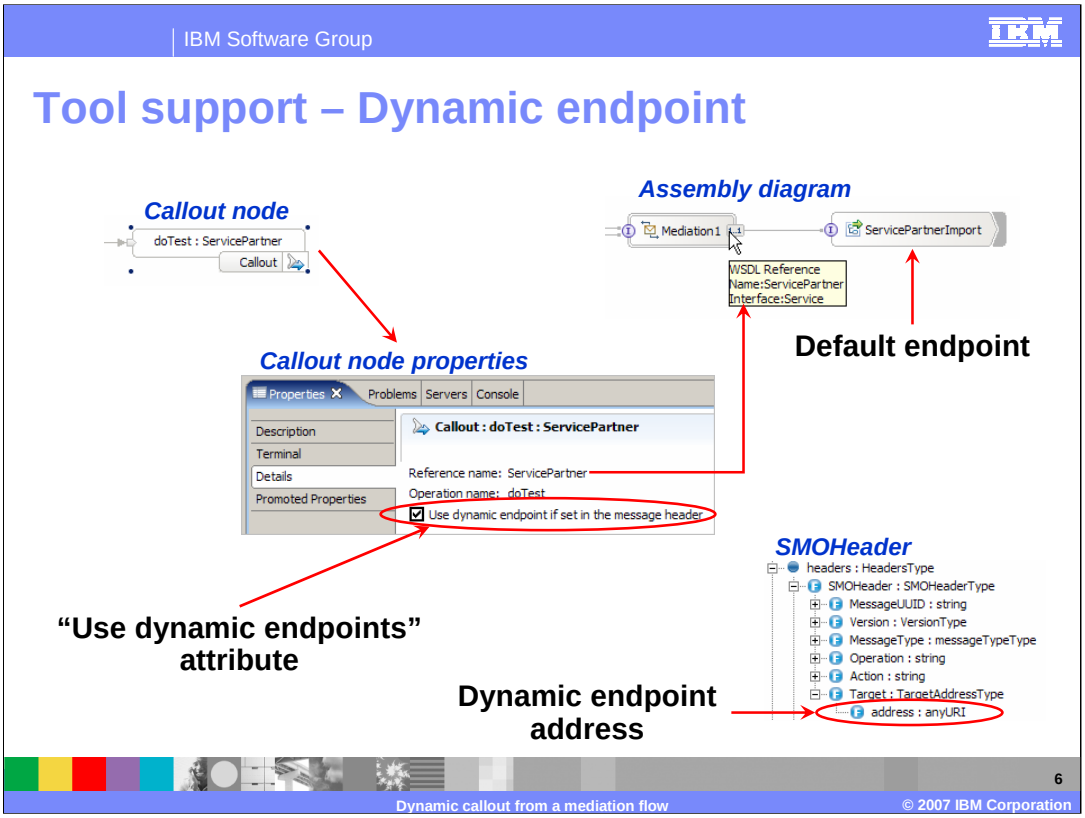
There is a pseudo code description of the runtime logic on the slide. Basically, when the use dynamic endpoints attribute has been set on the callout and the SMOHeader target address element contains a URI, that URI will be used to call the endpoint. If neither of those is true, but the Reference on the assembly diagram is wired, the wired target will be the called endpoint. If the Reference wasn't wired, an unwired reference exception will be thrown and the mediation flow will be terminated.



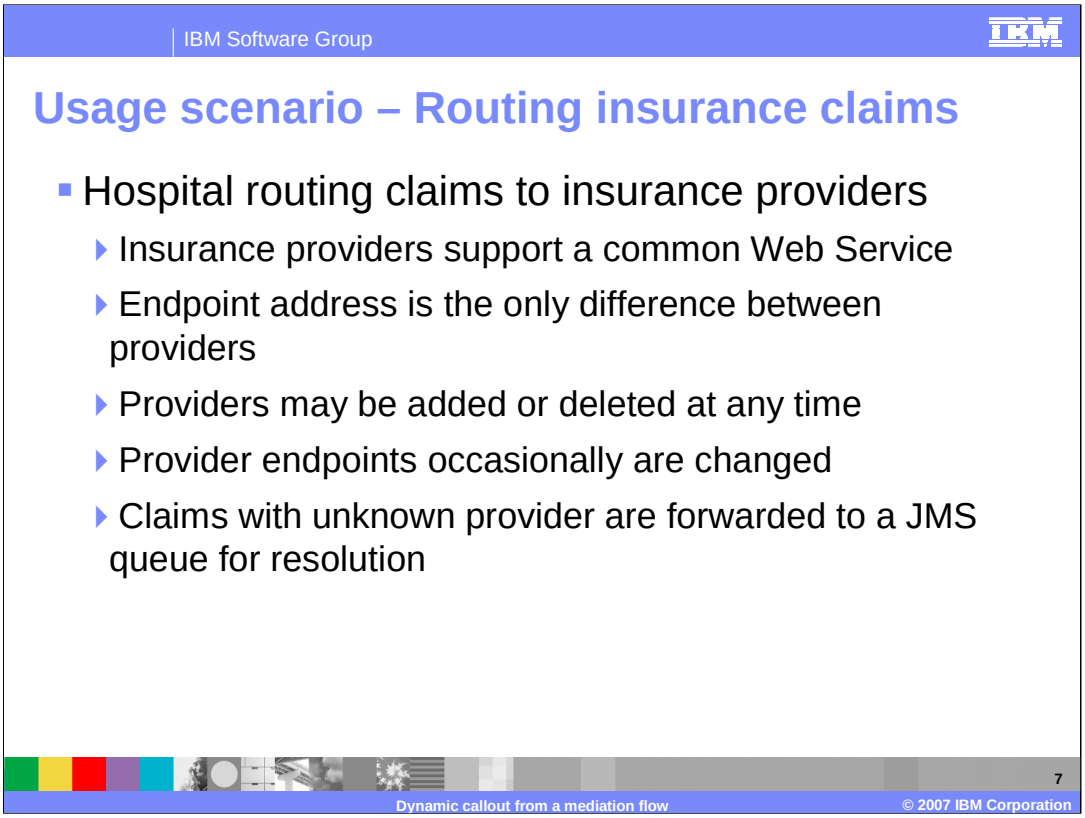
The underlying support that enables dynamic callouts is the service component architecture dynamic references support. Because this is the mechanism used, only those URI types that are supported by SCA dynamic references are valid to use with dynamic callouts.

There are basically five types of URIs that are supported. These are listed in the slide, along with their syntax descriptions. The first two are for Exports in an SCA Module, one supporting an Export with an SCA binding and the other an Export with a Web Service binding. The next two are for Web Service, for soap over http and for soap over jms. The last is for an SCA import that is defined in the same mediation module as the mediation flow with the dynamic callout. The interesting thing about this last one is that it opens up support for endpoints using different protocols. For example, if you wanted to use a JMS or MQ endpoint, you can configure an Import that can then be dynamically called by the callout.

The mediation runtime support makes no attempt to validate the URI. When anything is in the target address element of the SMOHeader, it will be used. If it is not a valid URI, SCA runtime errors will occur.



This slide illustrates all the pieces that play a part in the dynamic callout support. In the upper left is the callout node as it appears in the mediation flow editor, and the Details panel of the Properties view for the callout is shown in the center. Making the callout dynamic is done by selecting the Use dynamic endpoint if set in the message header check box. Also in the properties is the Reference name which identifies the Reference associated with this callout. In the upper right is the assembly diagram showing this Reference. It is wired to an Import which will become the default endpoint to be used when the target address is not set in the SMO. In the lower right the SMOHeader is shown, containing an element called Target which contains an element called address. This is where the URI needs to be set.



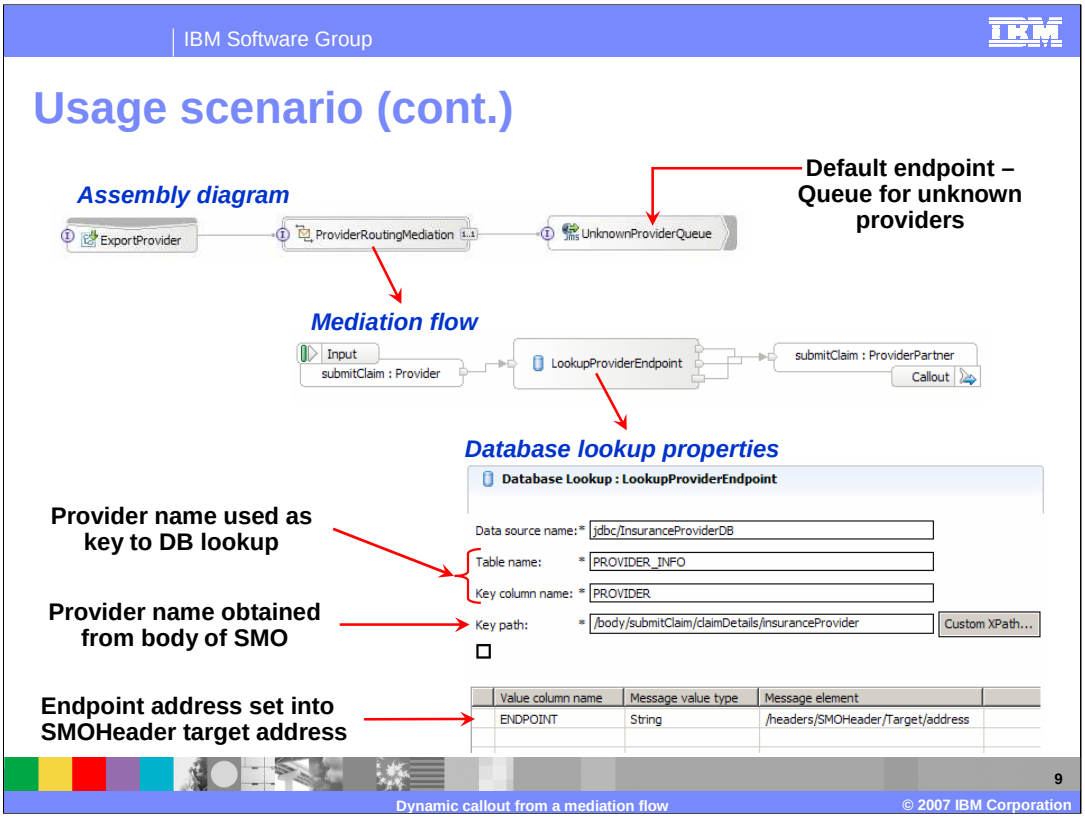
These next couple of slides provide an example scenario for the use of dynamic callouts.

The scenario is a hospital that sends insurance claims to many different insurance providers. There is a common Web Service definition that all the insurance providers support, but of course each insurance provider has their own endpoint address. The hospital has to deal with the fact that it may need to add or remove insurance providers from time to time, and on occasion an existing provider may change their endpoint address. Finally, when there is a claim for a provider that is not recognized, there needs to be some way to get human intervention to look at the claim and decide what to do. There is a JMS queue provided for this.

Usage scenario (cont.)

- Mediation flow
  - Input claim data contains provider name
  - Database lookup primitive uses provider name to lookup endpoint
    - Database table with provider information contains current endpoint to use
    - Places the endpoint into the target address of the SMO header
  - Callout configured to use the dynamic endpoint address
  - Callout wired to an import with JMS binding
    - Default endpoint
    - Only goes here when target address is not set by the database lookup

The mediation flow logic has the following elements. First, each claim contains the name of the insurance provider who is to handle the claim. There is a database lookup primitive which goes against a database table with provider names and endpoint addresses. The provider name is used as the key, and the endpoint address is placed into the SMOHeader target address field by the database lookup. The callout is configured to use dynamic endpoint addresses so that the claim is sent to the appropriate insurance provider. In the case where the named insurance provider is not in the database, the SMOHeader target address will remain null. In this case, the default endpoint is used. It is an Import with a JMS binding that sends the claim to a queue being monitored by the hospital's business office.

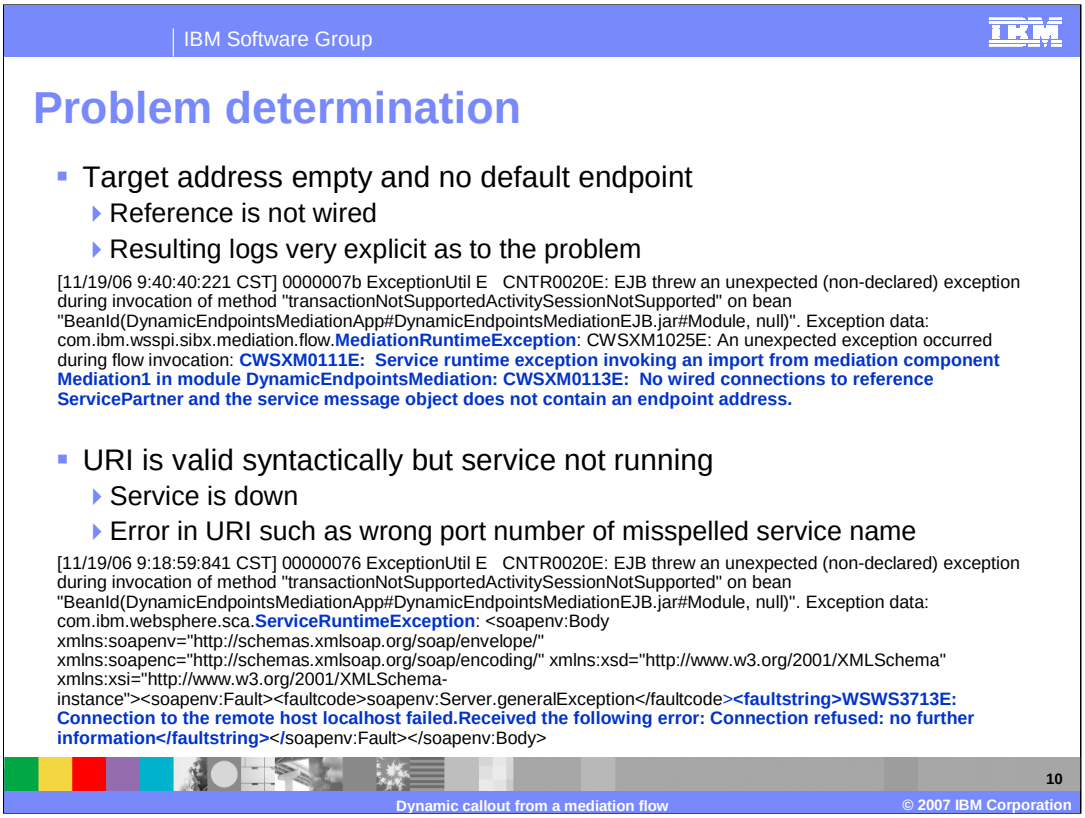


Given the scenario described on the previous slide, here are the interesting screen captures.

On the top is the assembly diagram, showing the Export used to call the mediation, the mediation flow component, and the Import with the JMS binding that will be the default endpoint. It will receive the claims for unknown providers.

The middle of the slide shows the mediation flow. It contains a database lookup primitive and a callout which has been enabled for dynamic endpoints.

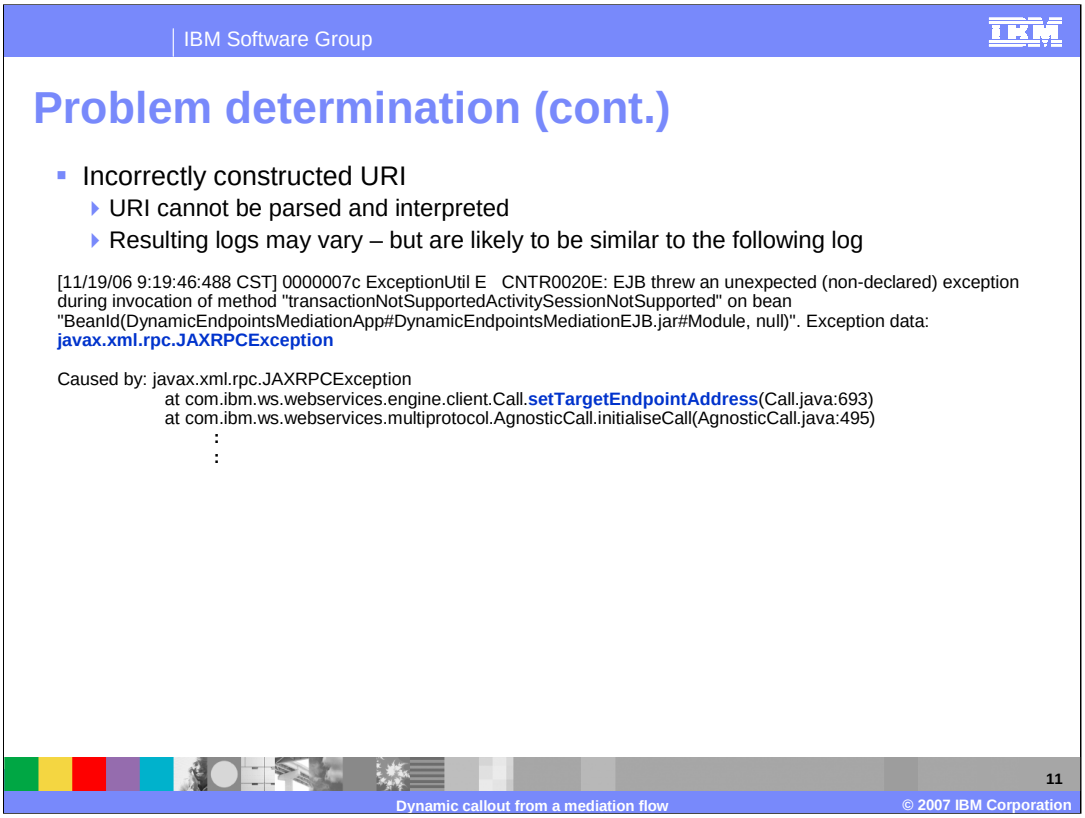
In the lower right are the database lookup properties. You can see that the database lookup is configured to do a lookup from the InsuranceProviderDB using the PROVIDER_INFO table with the PROVIDER column for the key. The key value is picked up from the InsuranceProvider input parameter contained in the body of the message. The ENDPOINT column from the database is placed into the SMOHeader Target address element.



The next couple of slides deal with some of the possible errors you may see when working with dynamic callouts. The most important parts of the log messages for the errors are highlighted in blue.

The first is the case of no default endpoint. The target address does not get set in the SMOHeader and the Reference is not wired. There is a very explicit error produced for this. It is a MediationRuntimeException that states the reference is not wired.

The next case is when the URI is syntactically correct but the service cannot be reached. This could be because the service is not running at the moment, or possibly the URI has a mistake such as an incorrect port number or the service name is misspelled. This will result in a ServiceRuntimeException that contains a soap fault indication connection refused.

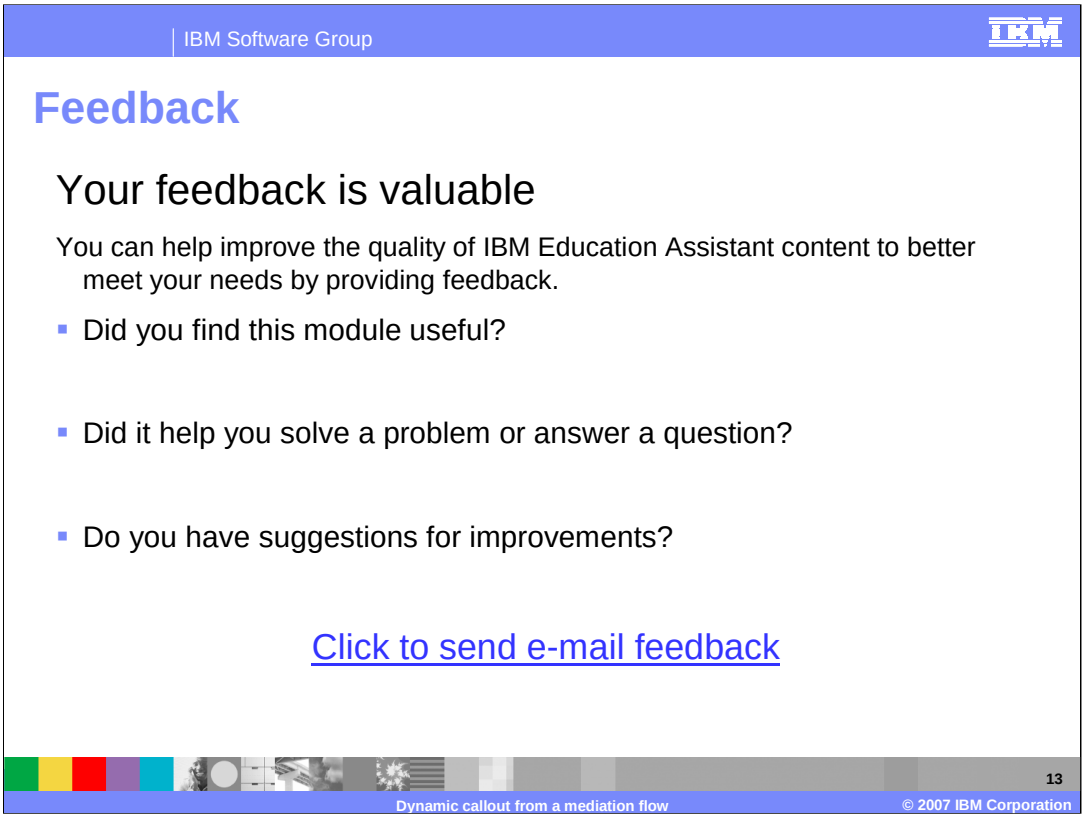


When the URI is incorrectly constructed so that it is syntactically incorrect, the resulting errors may vary depending upon the specific errors in the URI. Here is one example, where a JAXRPCException is thrown in a setTargetEndpointAddress method call.

Summary

- Introduced dynamic callout
  - Examined the benefits
  - Runtime support
  - Development time configuration
  - Examined a usage scenario
  - Provided problem determination information

In this presentation you were introduced to dynamic callouts. The presentation started by looking at the benefits that dynamic endpoints provide. Then the dynamic callout function was looked at in terms of how it behaves at runtime and how it is configured using WebSphere Integration Developer. To help you understand how this function might be used, an example usage scenario was explained. Finally, since this capability has potential for runtime errors to occur, some problem determination information was provided.



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:

IBM WebSphere

EJB, Java, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. 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.