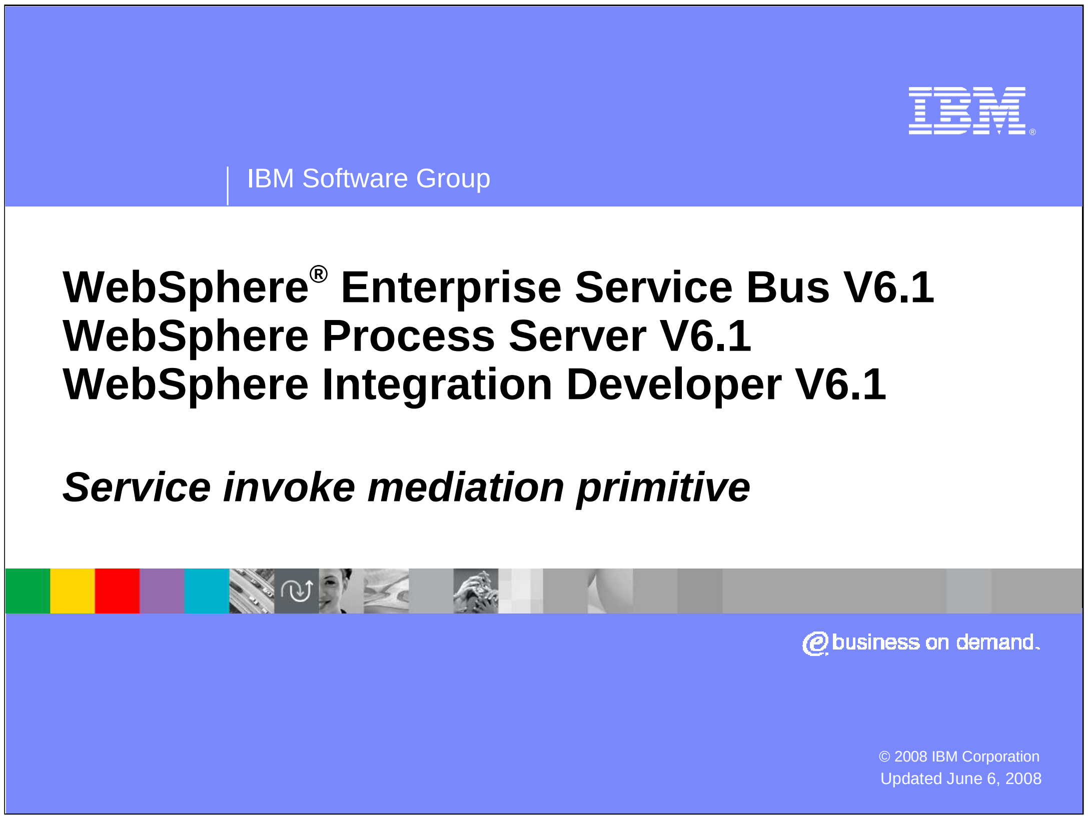


IBM Software Group

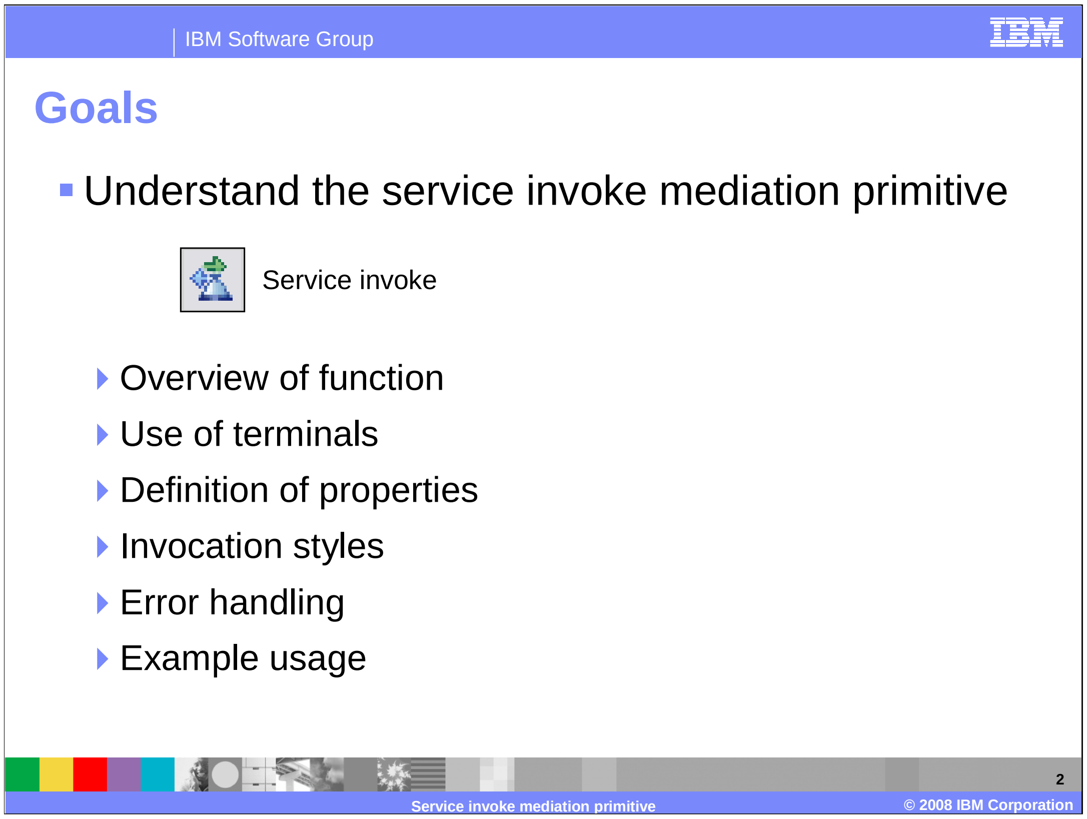
WebSphere® Enterprise Service Bus V6.1 WebSphere Process Server V6.1 WebSphere Integration Developer V6.1

Service invoke mediation primitive

business on demand.

Updated June 6, 2008

This presentation provides a detailed look at the service invoke mediation primitive.



Goals

- Understand the service invoke mediation primitive

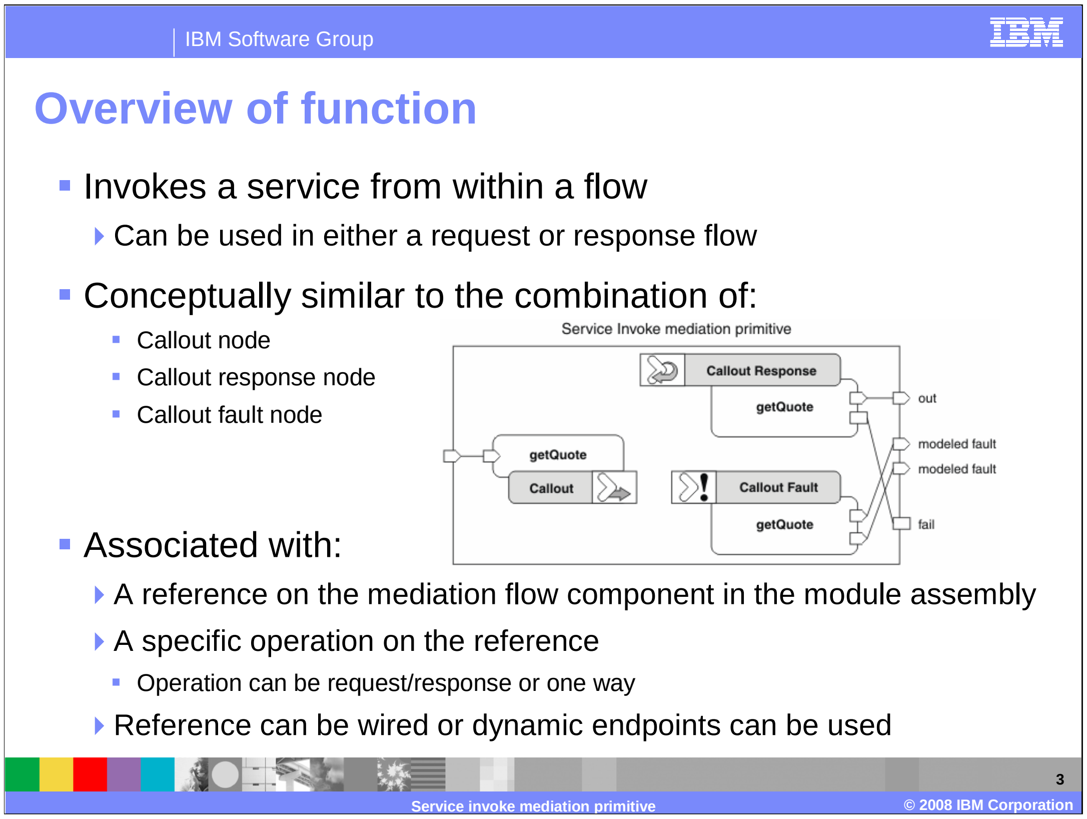
  Service invoke

  - Overview of function
  - Use of terminals
  - Definition of properties
  - Invocation styles
  - Error handling
  - Example usage

The goal of this presentation is to provide you with a full understanding of the service invoke mediation primitive.

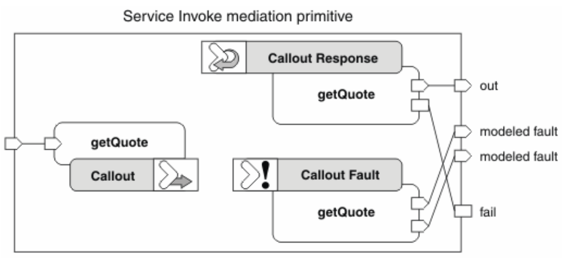
The presentation assumes that you are already familiar with the material presented in the **Mediation primitive common details** presentation and the **Common details – Promoted properties** presentation. These two presentations serve as a base for understanding mediation primitives in general.

This presentation starts with an overview of the service invoke primitive and information about the primitive’s use of terminals and its properties. A discussion of invocation styles is then presented, as it is important for helping you understand the behavior of a mediation flow containing a service invoke. Some error handling information is discussed and an example use case of the service invoke primitive is provided.



Overview of function

- Invokes a service from within a flow
  - Can be used in either a request or response flow
- Conceptually similar to the combination of:
  - Callout node
  - Callout response node
  - Callout fault node

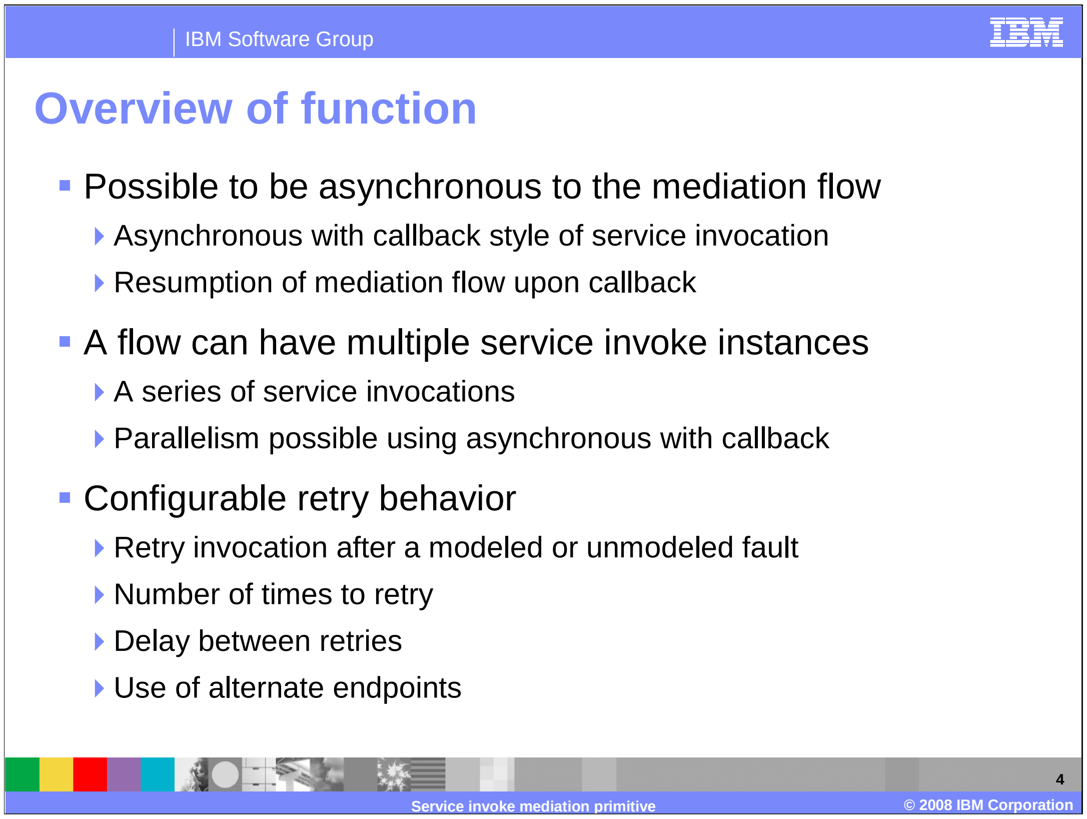


- Associated with:
  - A reference on the mediation flow component in the module assembly
  - A specific operation on the reference
    - Operation can be request/response or one way
  - Reference can be wired or dynamic endpoints can be used

The purpose of the service invoke primitive is to enable you to call an external service from within a mediation flow, either a request flow or a response flow.

In many ways, it is similar to a combination of a callout node and its associated callout response node and callout fault node. This is illustrated in the graphic, where the service invoke is represented as the enclosing rectangle and the relationship between the terminals of a service invoke and the terminals of the three nodes are shown. This is described in more detail later in the presentation.

A service invoke primitive is associated with a reference on the mediation flow component in the assembly diagram, and to a specific operation on the interface of that reference. The operation can be a request response operation or can be a one way operation. The service invoke primitive can be configured to make use of a dynamic endpoint address taken from the SMO rather than using the import or component the reference is wired to in the assembly.



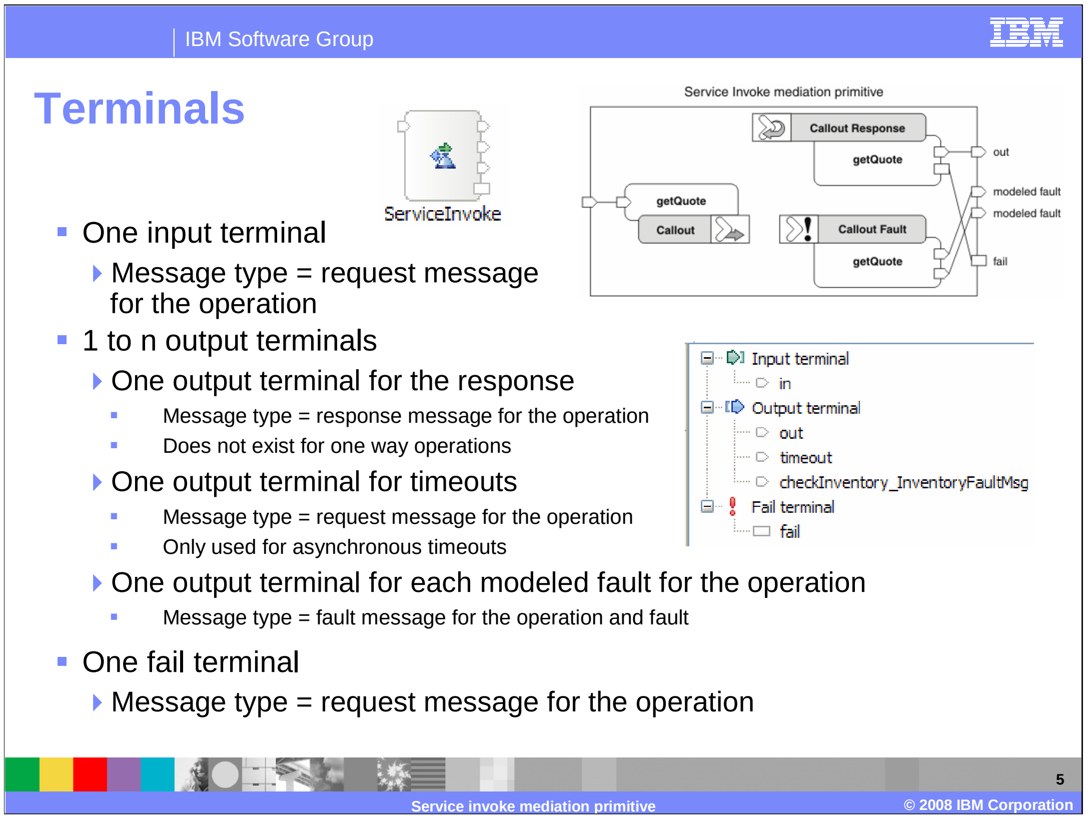
Overview of function

- Possible to be asynchronous to the mediation flow
  - Asynchronous with callback style of service invocation
  - Resumption of mediation flow upon callback
- A flow can have multiple service invoke instances
  - A series of service invocations
  - Parallelism possible using asynchronous with callback
- Configurable retry behavior
  - Retry invocation after a modeled or unmodeled fault
  - Number of times to retry
  - Delay between retries
  - Use of alternate endpoints

When a service invoke primitive makes a call to an external service, it is possible for that call to be either synchronous or asynchronous to the mediation flow. In the case where it is asynchronous, the mediation flow is resumed upon callback. A complete discussion of invocation styles and the effect of synchronous versus asynchronous calls are discussed later in this presentation.

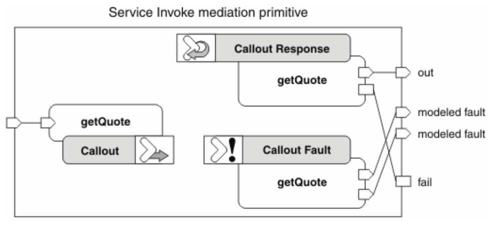
There can be multiple service invoke primitive instances in the same mediation flow. This might be a series of service invocations in a sequential flow. Additionally, when there are multiple paths through a flow with each containing a service invoke, the service invocations have the possibility of being processed in parallel if the asynchronous with callback invocation style is used.

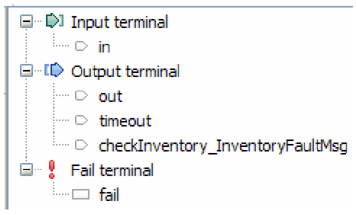
The service invoke primitive can be configured to retry failing service calls. The configuration allows you to specify if it should be done for modeled faults, unmodeled faults or both. Additional configuration properties specify a retry count, a retry delay and if alternate endpoints should be used when retrying the call.



IBM Software Group

Terminals





- Input terminal
  - in
- Output terminal
  - out
  - timeout
  - checkInventory_InventoryFaultMsg
- Fail terminal
  - fail

- One input terminal
  - Message type = request message for the operation
- 1 to n output terminals
  - One output terminal for the response
    - Message type = response message for the operation
    - Does not exist for one way operations
  - One output terminal for timeouts
    - Message type = request message for the operation
    - Only used for asynchronous timeouts
  - One output terminal for each modeled fault for the operation
    - Message type = fault message for the operation and fault
- One fail terminal
  - Message type = request message for the operation

5

Service invoke mediation primitive © 2008 IBM Corporation

This slide looks at the terminals that are used with a service invoke primitive. In the top center of the slide is a screen capture of a service invoke primitive as it appears in a flow. On the center right is a screen capture of the terminals panel from the properties view. As was mentioned earlier, the service invoke can be thought of as a combination of the callout node, callout response node and callout fault node, which is illustrated in the upper right corner.

A service invoke primitive has one input terminal, called in, whose message type is defined by the operation being called on the external service. This is similar to the in terminal of a callout node.

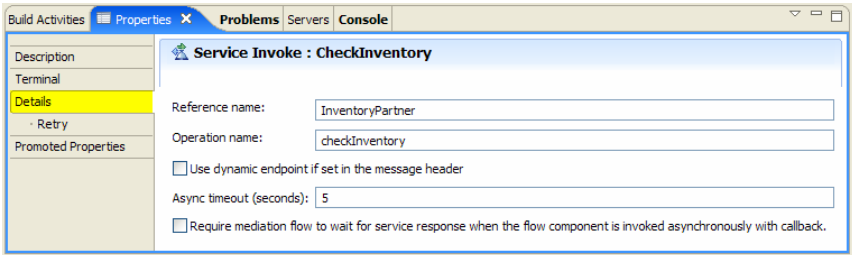
The service invoke can have from one to n output terminals. The first output terminal is called out. It has a message type defined by the response for the operation being called. This can be compared to the out terminal of a callout response node. For a one way operation, this terminal does not exist.

The next output terminal is called timeout and is used when an asynchronous timeout occurs waiting for the response. The message type is the same as the message type of the in terminal. For synchronous operations, this terminal is present but will never be fired. Callouts have no equivalent terminal to the timeout terminal. They use the fail terminal of the callout response node to return asynchronous timeouts.

The remaining output terminals represent the faults defined for the operation being called, with one terminal for each fault. The message type of each terminal is defined by the fault. These are the same as the terminals of the callout fault node.

Finally, there is the fail terminal whose message type is the same as the message type of the in terminal. This terminal can be compared to the fail terminal of a callout response node.

Properties


Build Activities | Properties | Problems | Servers | Console

Description
Terminal
Details
· Retry
Promoted Properties

Service Invoke : CheckInventory

Reference name: InventoryPartner

Operation name: checkInventory

☐ Use dynamic endpoint if set in the message header

Async timeout (seconds): 5

☐ Require mediation flow to wait for service response when the flow component is invoked asynchronously with callback.

- Properties controlling service invocation
  - Reference name
  - Operation name
  - Use dynamic endpoint
  - Async timeout
  - Asynchronous with callback not allowed

The Details panel of the Properties view is shown here.

The first two properties are the **Reference name** and the **Operation name**. They define the reference, and therefore the interface, that this service invoke is associated with and the specific operation on that interface to call. On this panel, these properties are read only. They are specified when the service invoke primitive is created and cannot be changed.

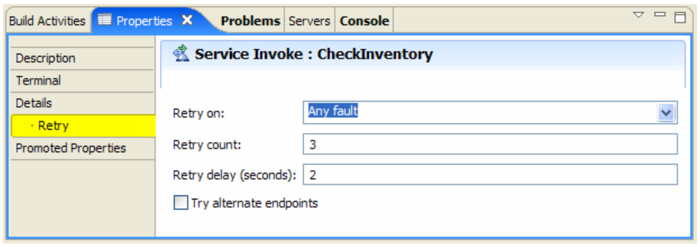
The **Use dynamic endpoint,** if set in the message header property, causes the service invoke primitive to check the SMO for a target URL address in the field headers/SMOHeader/Target/address. If there is one set, that endpoint URL is used rather than the reference's wire target. If a target URL address is not set in the SMO, the wire target of the reference is used.

The **Async timeout** property specifies how many seconds to wait before for a response before the timeout terminal is fired. If the call is synchronous, this property is ignored.

The final property on this panel can be referred to as **Asynchronous with callback not allowed** or is sometimes referred to as force synch. The actual wording on the panel says: Require mediation flow to wait for service response when the flow component is invoked asynchronously with callback. This property setting is used to prevent the asynchronous with callback invocation style, thus not allowing the service invocation to run on a separate thread from the mediation flow. Note that this property is implicitly set for any service invoke primitives that exist in a flow between a fan out and fan in primitive.

Properties for retry

- Retry on
  - Never
  - Any fault
  - Modeled fault
  - Unmodeled fault
- Retry count
- Retry delay
- Try alternate endpoints



Service Invoke : CheckInventory

Retry on: Any fault

Retry count: 3

Retry delay (seconds): 2

☐ Try alternate endpoints

* See separate Service call retry presentation for details

The properties for retry are specified on their own panel.

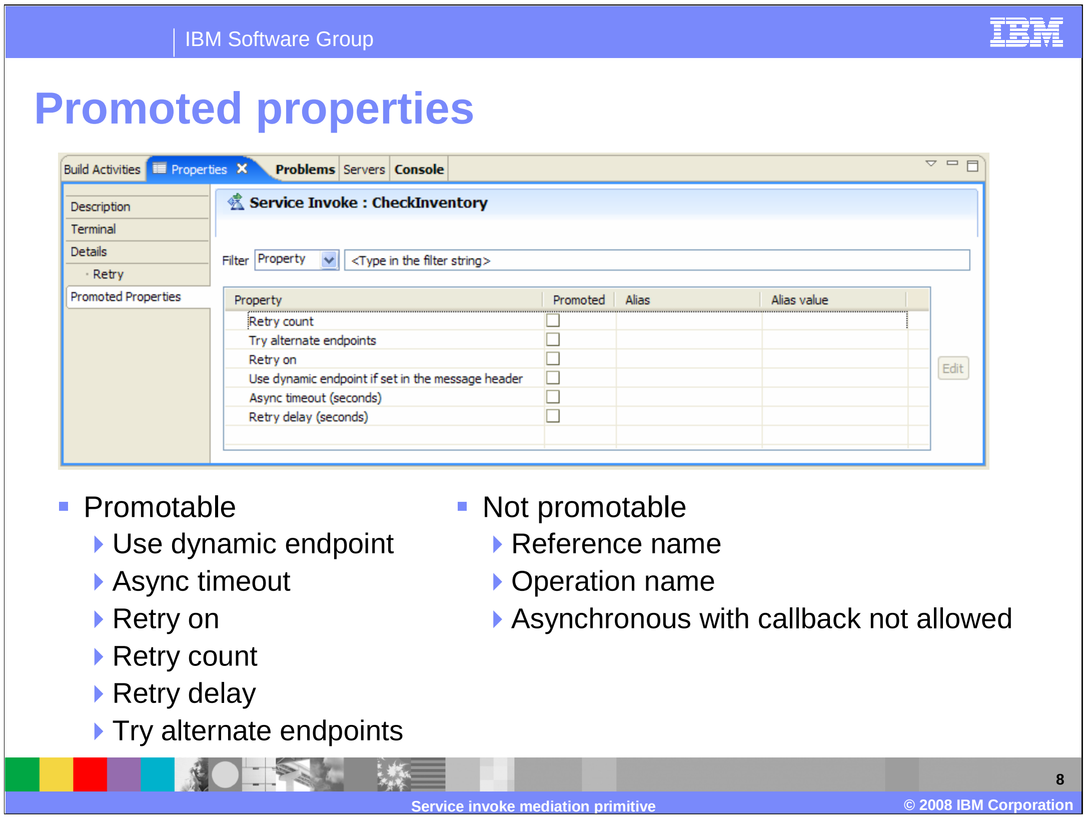
The **Retry on** property can be set to never, indicating not to perform retry processing. It can also be set to Modeled fault, Unmodeled fault or Any fault, indicating which type of faults should result in retry processing.

The **Retry count** property specifies how many times to attempt a retry when faults continue to occur during retry processing. Once the retry count is reached, the fault that occurred on the final attempt is returned to the flow.

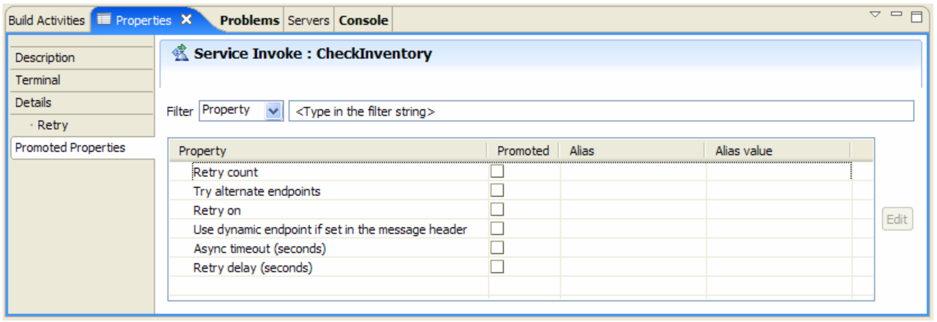
The **Retry delay** specifies how many seconds to wait between a fault being returned and the next retry attempt.

Finally the **Try alternate endpoints** specifies that endpoint URLs from the SMO should be used to perform the retries. These are located at headers/SMOHeader/AlternateTarget/address.

Details of retry processing, as it applies to callouts and service invoke primitives, is explained in a separate presentation entitled **Service call retry**.



Promoted properties



Service Invoke : CheckInventory

Filter Property <Type in the filter string>

Property	Promoted	Alias	Alias value
Retry count	☐		
Try alternate endpoints	☐		
Retry on	☐		
Use dynamic endpoint if set in the message header	☐		
Async timeout (seconds)	☐		
Retry delay (seconds)	☐		

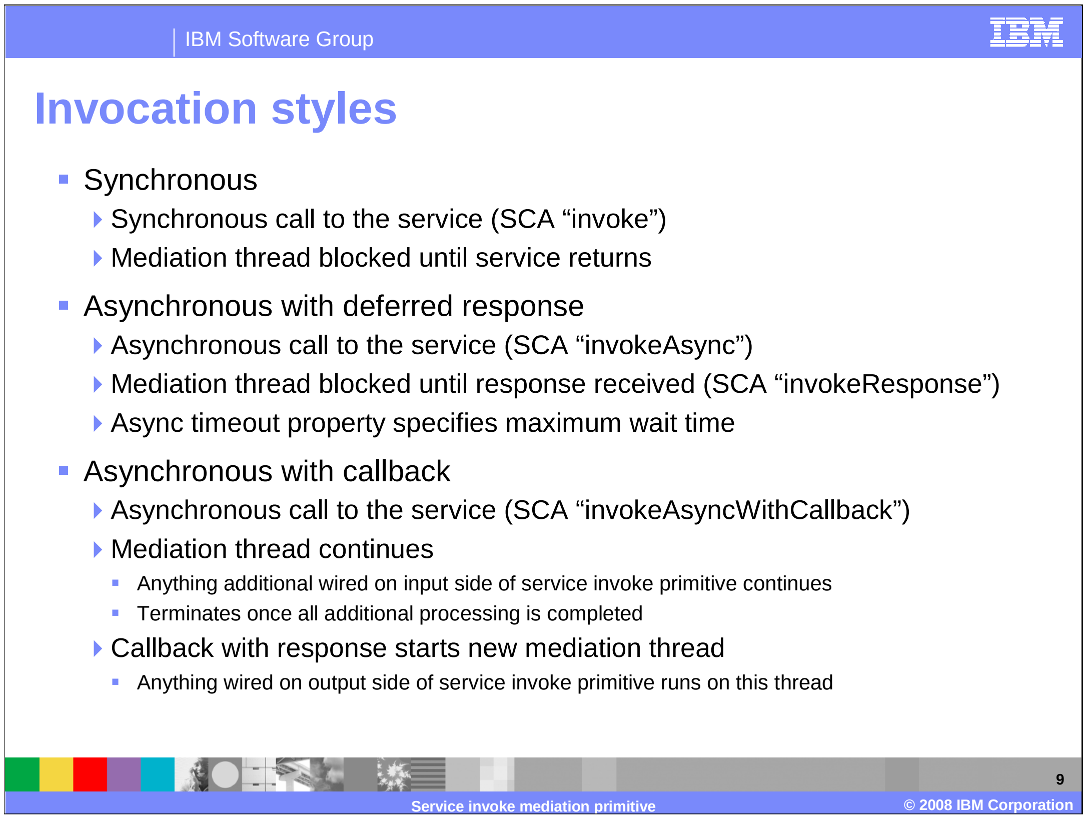
- Promotable
  - Use dynamic endpoint
  - Async timeout
  - Retry on
  - Retry count
  - Retry delay
  - Try alternate endpoints
- Not promotable
  - Reference name
  - Operation name
  - Asynchronous with callback not allowed

This slide shows the Promoted Properties panel for a service invoke primitive.

The properties **Use dynamic endpoint** and **Async timeout** are both promotable.

Also, all of the properties associated with retry processing are promotable, which are **Retry on**, **Retry count**, **Retry delay** and **Try alternate endpoints**.

The **Reference name** and **Operation name** are not promotable. Changing either of these requires corresponding development time changes to the flow. Likewise, the **Asynchronous with callback not allowed** has the potential to require changes to the logic of the flow and is therefore not promotable.


IBM Software Group

Invocation styles

- Synchronous
  - Synchronous call to the service (SCA “invoke”)
  - Mediation thread blocked until service returns
- Asynchronous with deferred response
  - Asynchronous call to the service (SCA “invokeAsync”)
  - Mediation thread blocked until response received (SCA “invokeResponse”)
  - Async timeout property specifies maximum wait time
- Asynchronous with callback
  - Asynchronous call to the service (SCA “invokeAsyncWithCallback”)
  - Mediation thread continues
    - Anything additional wired on input side of service invoke primitive continues
    - Terminates once all additional processing is completed
  - Callback with response starts new mediation thread
    - Anything wired on output side of service invoke primitive runs on this thread

9

Service invoke mediation primitive

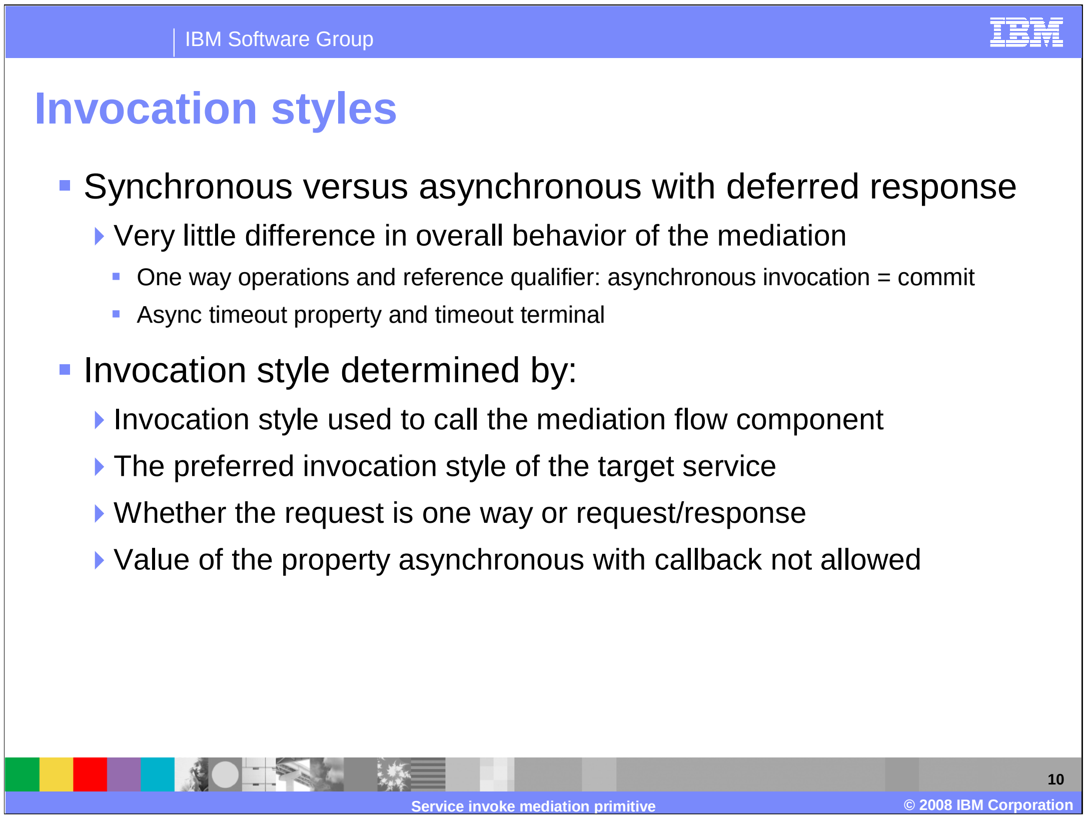
© 2008 IBM Corporation

These next few slides discuss the topic of invocation styles used by service invoke primitives. The invocation styles are the service component architecture defined styles of invocation which provide the underlying implementation. This slide describes these SCA defined styles as they apply to mediation flows.

The **Synchronous** style of invocation uses the SCA API invoke operation to make the call. This causes the tread on which the mediation is running to block until the response is received.

The **Asynchronous with deferred response** style of invocation uses the SCA API invokeAsync operation. From an SCA perspective, this allows the thread making the call to continue in parallel, and then it can use the SCA invokeResponse operation to receive the response. However, the service invoke primitive performs the invokeResponse immediately after the invokeAsync, so effectively no parallel processing occurs. It does allow the asynchronous timeout property to cause a timeout if the response is not received within that timeframe.

The **Asynchronous with callback** style of invocation uses the SCA API invokeAsyncWithCallback. From an SCA perspective, this allows the thread making the call to continue in parallel, and the response to be received on a new thread kicked off by a callback. As this applies to the mediation flow, the thread on which the mediation is running will continue if there is more work to be done. Basically, this means that in a mediation with multiple flow paths, anything wired on the input side of the service invoke will continue until all is completed. The thread then terminates. The response is received on a new thread which is kicked off starting on the output side of the service invoke primitive.

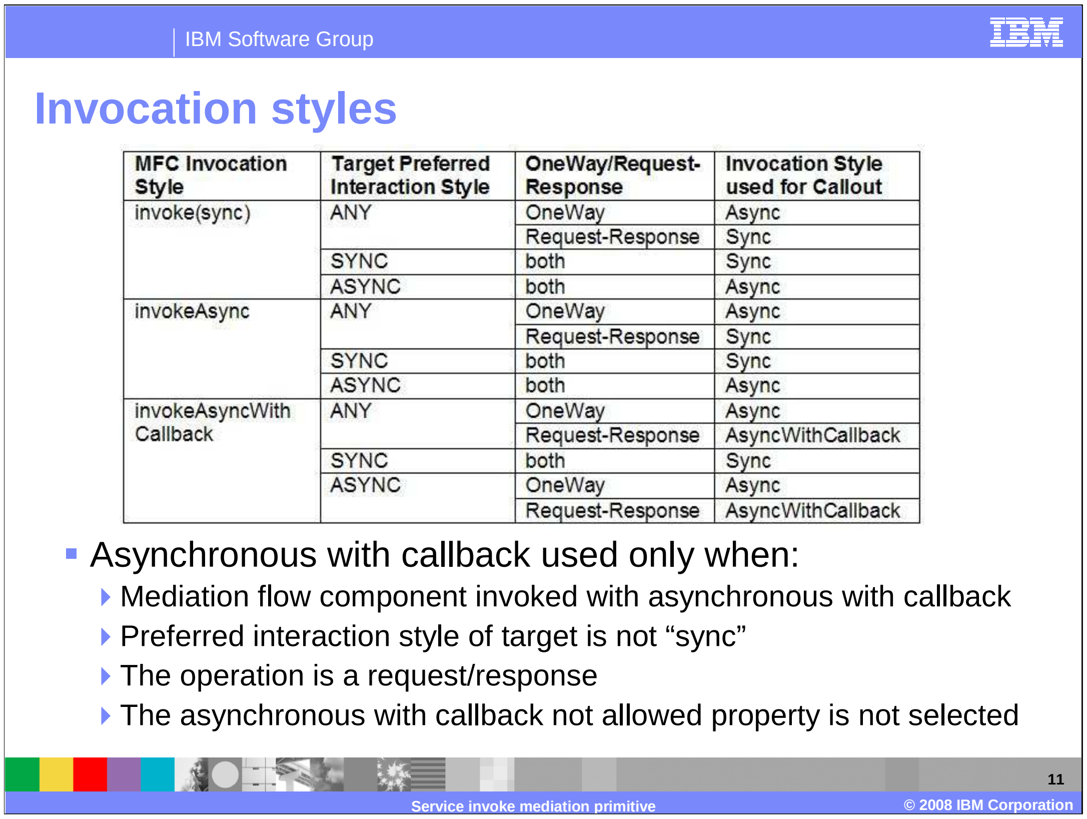


Invocation styles

- Synchronous versus asynchronous with deferred response
  - Very little difference in overall behavior of the mediation
    - One way operations and reference qualifier: asynchronous invocation = commit
    - Async timeout property and timeout terminal
- Invocation style determined by:
  - Invocation style used to call the mediation flow component
  - The preferred invocation style of the target service
  - Whether the request is one way or request/response
  - Value of the property asynchronous with callback not allowed

In the context of the service invoke primitive, there is really very little difference in the behavior you will see between the synchronous and asynchronous with deferred response styles of invocation. This is because the service invoke primitive calls invokeResponse right after invokeAsync, and thus the mediation flow blocks waiting for the response similar to the synchronous style. There are a couple of differences that are seen between these two styles. There is a qualifier used on references called asynchronous invocation which can have a value of call or commit. When a one way operation is invoked using the asynchronous with deferred response style and asynchronous invocation is set to commit, the actual invocation of the service does not happen until the containing transaction commits. The second difference is that a timeout can occur with the asynchronous with deferred response style. Other than these differences, the two styles exhibit the same behavior in a mediation flow.

Which invocation style is used by the service invoke is not based on a simple property setting, but rather on a combination of factors. One of the factors is the invocation style that was used to call the mediation flow component. Then, the preferred invocation style of the target service is considered and whether the operation is one way or request response. Finally, the property asynchronous with callback not allowed prevents an asynchronous with callback style invocation. This is explained in more detail on the next slide.



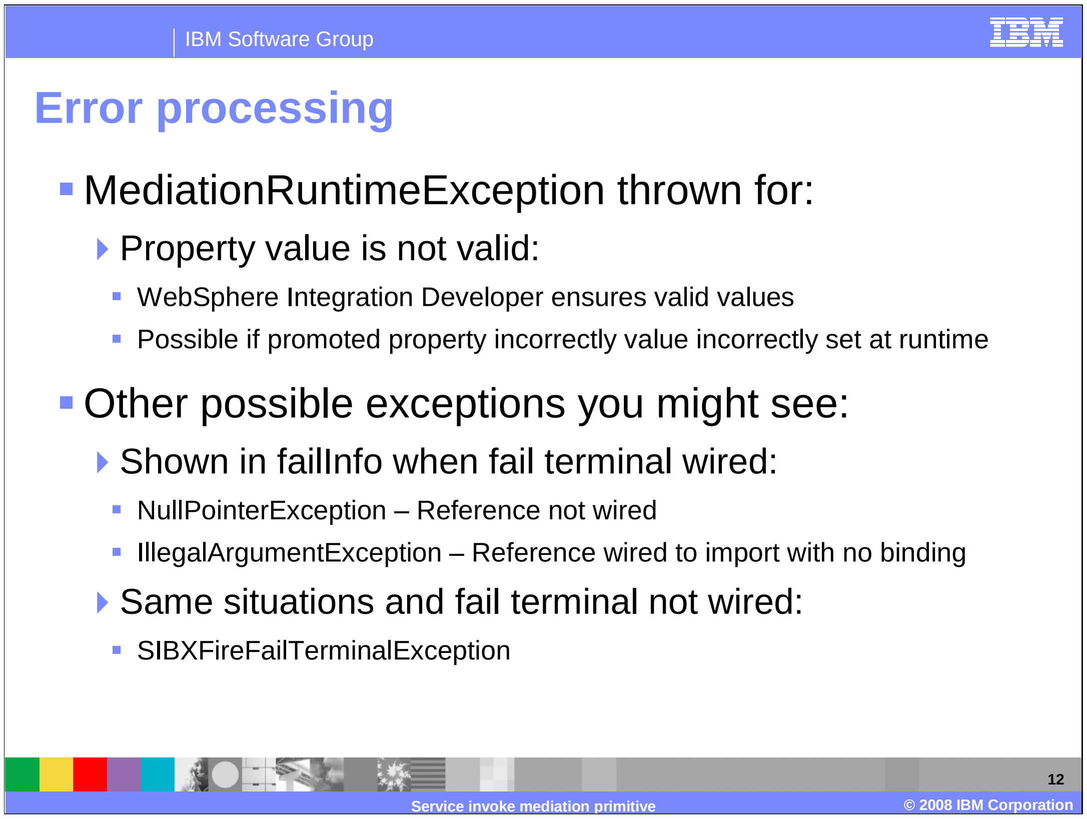
Invocation styles

MFC Invocation Style	Target Preferred Interaction Style	OneWay/Request-Response	Invocation Style used for Callout
invoke(sync)	ANY	OneWay	Async
		Request-Response	Sync
	SYNC	both	Sync
	ASYNC	both	Async
invokeAsync	ANY	OneWay	Async
		Request-Response	Sync
	SYNC	both	Sync
	ASYNC	both	Async
invokeAsyncWith Callback	ANY	OneWay	Async
		Request-Response	AsyncWithCallback
	SYNC	both	Sync
	ASYNC	OneWay	Async
		Request-Response	AsyncWithCallback

- Asynchronous with callback used only when:
  - Mediation flow component invoked with asynchronous with callback
  - Preferred interaction style of target is not “sync”
  - The operation is a request/response
  - The asynchronous with callback not allowed property is not selected

This slide contains a table defining the invocation style based on the various factors described on the previous page. The first column defines the invocation style used to call the mediation flow component. The second column defines the preferred interaction style of the target. The third column defines if the operation is one way or request response and the rightmost column indicates the invocation style used by the service invoke primitive. You might notice that the last column title indicates the invocation style used for callout, but the rules followed are the same for callouts and service invoke primitives. Rather than going through the entire table, it is provided for your reference. However, since asynchronous with callback is the only way to get parallel processing, it is worthwhile pointing out the limited set of circumstances where this interaction style is used. First of all the mediation flow component must have been called using the asynchronous with callback style and the preferred target interaction style cannot have been set to sync. Additionally, the request must be a request response operation and finally the property asynchronous with callback not allowed can not have been set.

At this point it is worth mentioning that splitting and aggregating scenarios using fan out and fan in implicitly set the property asynchronous with callback not allowed. Therefore, one of the most likely places to exploit parallel processing cannot take advantage of it.



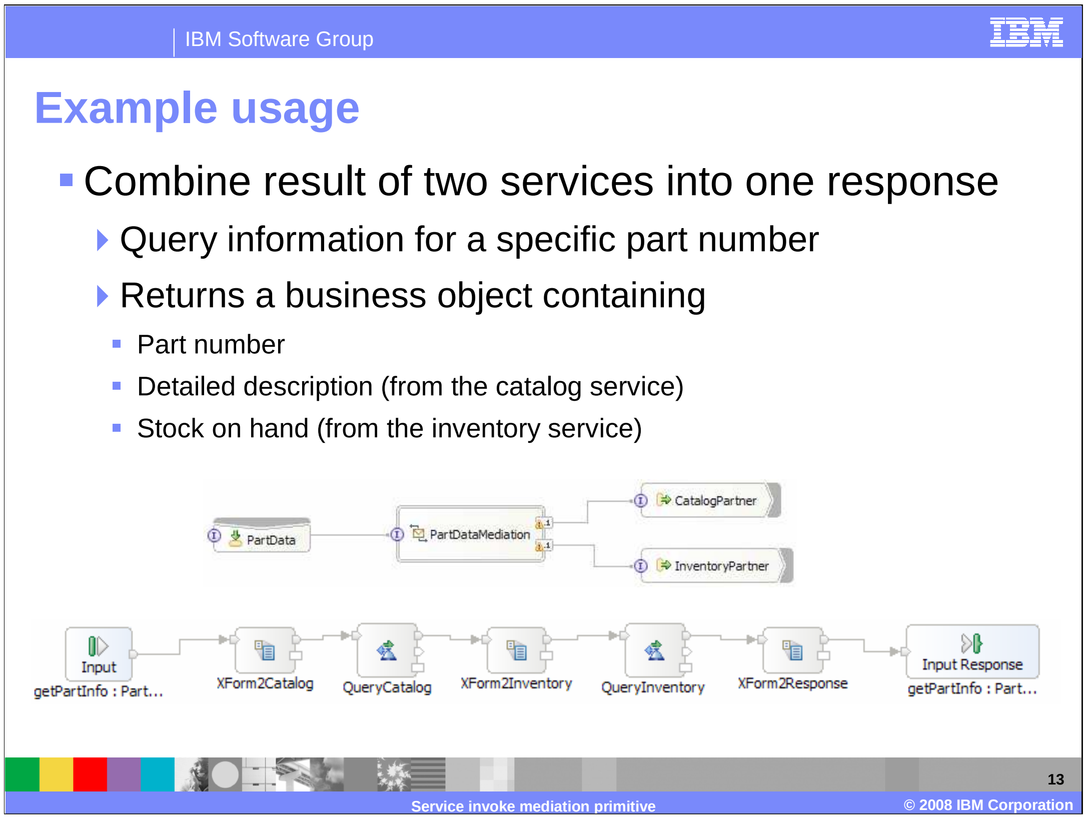
Error processing

- MediationRuntimeException thrown for:
  - Property value is not valid:
    - WebSphere Integration Developer ensures valid values
    - Possible if promoted property incorrectly value incorrectly set at runtime
- Other possible exceptions you might see:
  - Shown in failInfo when fail terminal wired:
    - NullPointerException – Reference not wired
    - IllegalArgumentException – Reference wired to import with no binding
  - Same situations and fail terminal not wired:
    - SIBXFireFailTerminalException

The error processing details and considerations are examined in this slide.

A MediationRuntimeException is thrown when a property value specified for the service invoke is not valid. Because WebSphere Integration Developer validates property values, this problem is only seen when the incorrect property value has been set administratively for a property that has been promoted. The retry on property is an example of one for which this can occur.

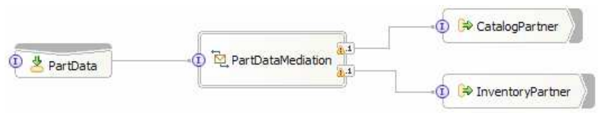
Other possible exceptions you might encounter involve problems with the assembly, and vary depending upon whether the fail terminal is wired. When the fail terminal is wired, the exceptions are seen in the failInfo section of the SMO. An unwired reference results in a NullPointerException and a reference wired to an import with no binding results in an IllegalArgumentException. For the case where the fail terminal is not wired, these same two situations both result in an SIBXFireFailTerminalException.

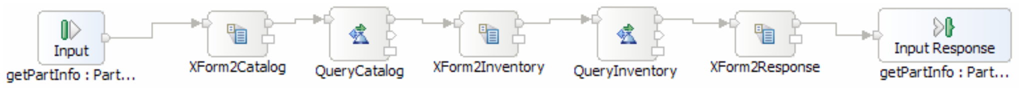


IBM Software Group

Example usage

- Combine result of two services into one response
  - Query information for a specific part number
  - Returns a business object containing
    - Part number
    - Detailed description (from the catalog service)
    - Stock on hand (from the inventory service)





13

Service invoke mediation primitive

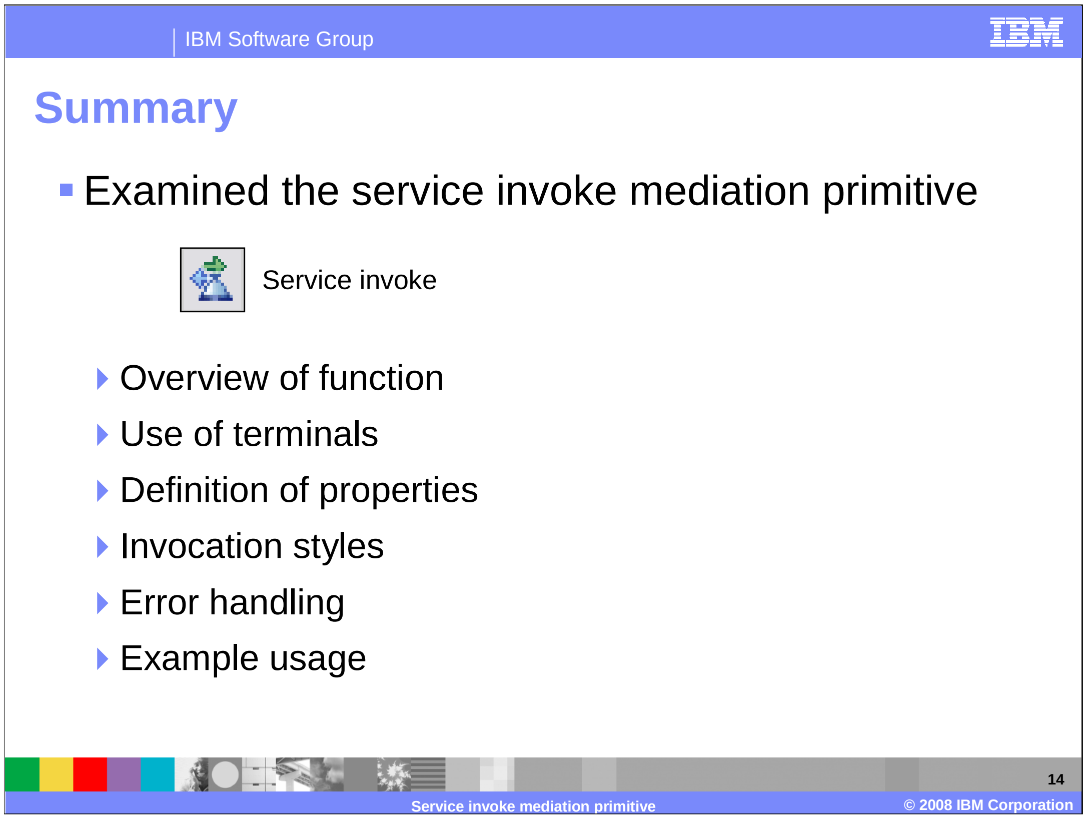
© 2008 IBM Corporation

This slide illustrates one of the possible use cases for the service invoke primitive. In this scenario, the mediation is used to combine the results of calls to two services to build a response to the incoming request. The incoming request provides a part number, and the response is a business object containing the part number, a description of the part and the quantity currently in inventory. The description for the part is obtained from a catalog service and the quantity from an inventory service.

In the slide you can see the assembly diagram showing the export exposing the part data service, the part data mediation containing the flow and the two imports for the catalog and inventory services.

Below the assembly diagram is the mediation flow. Notice that there is no callout for this flow. Calls to the catalog service and inventory service are both required to build the response, so these are both done using the service invoke primitive. The response is then built within the request flow and returned to the caller using the input response node in the request flow.

In order to make use of the service invoke primitives, the message type of the SMO must be transformed to setup each request and to process each response. Therefore, looking at the flow, you see that the first primitive is an XSL transformation, called XForm2Catalog. It takes the part data request message received as input and transforms it into a catalog request message. It also saves the incoming part number in the transient context. The QueryCatalog service invoke primitive calls the catalog service and the resulting SMO is a catalog response message. The next XSL transformation, XForm2Inventory, saves the description from the response in the transient context and sets up the message to be an inventory request message. The QueryInventory service invoke primitive now calls the inventory service and the resulting SMO is an inventory response message. The final step is for the XSL transformation, XForm2Response, to build a part data response message. It gets the part number and description from the transient context and the inventory quantity from the inventory response message body. The input response node then returns this to the caller.



IBM Software Group

Summary

- Examined the service invoke mediation primitive

  Service invoke

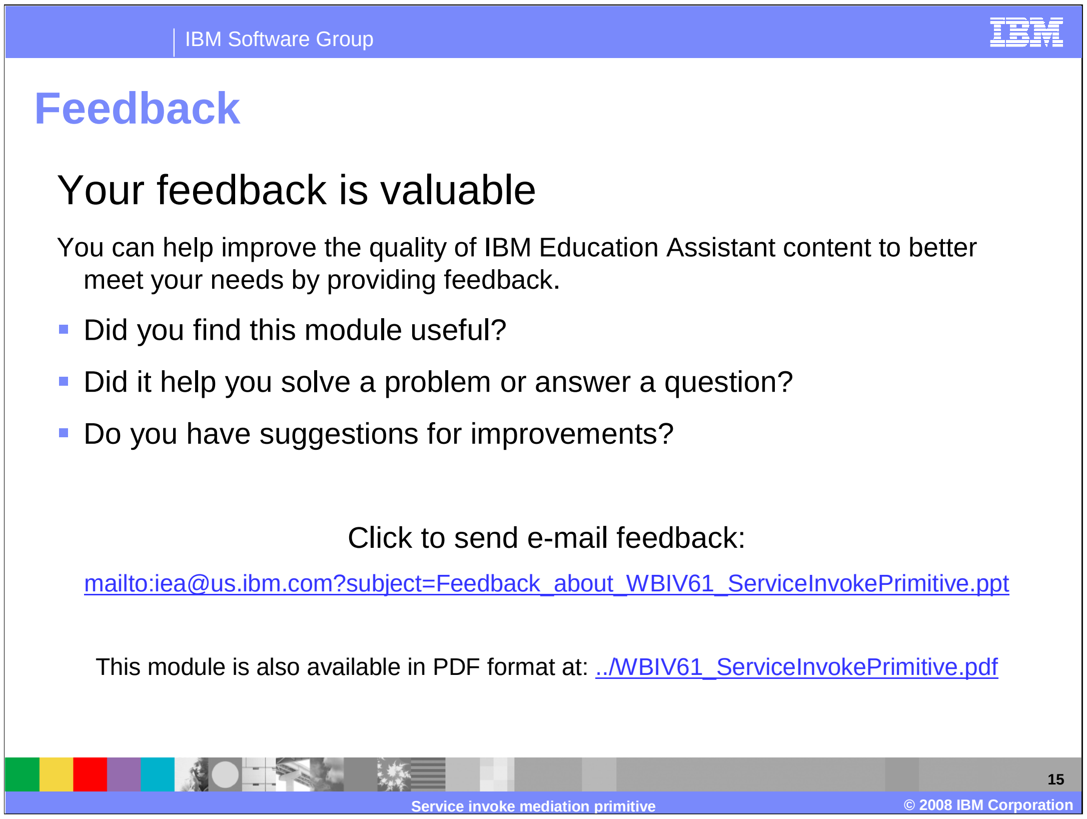
  - Overview of function
  - Use of terminals
  - Definition of properties
  - Invocation styles
  - Error handling
  - Example usage

14

Service invoke mediation primitive

© 2008 IBM Corporation

In summary, this presentation provided details regarding the service invoke primitive, providing an overview of its function and information about the primitive's use of terminals and its properties. A discussion of invocation styles was presented to help you understand the behavior of a mediation flow containing a service invoke. Some error handling information was discussed and an example use case for the service invoke primitive was described.



Feedback

Your feedback is valuable

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

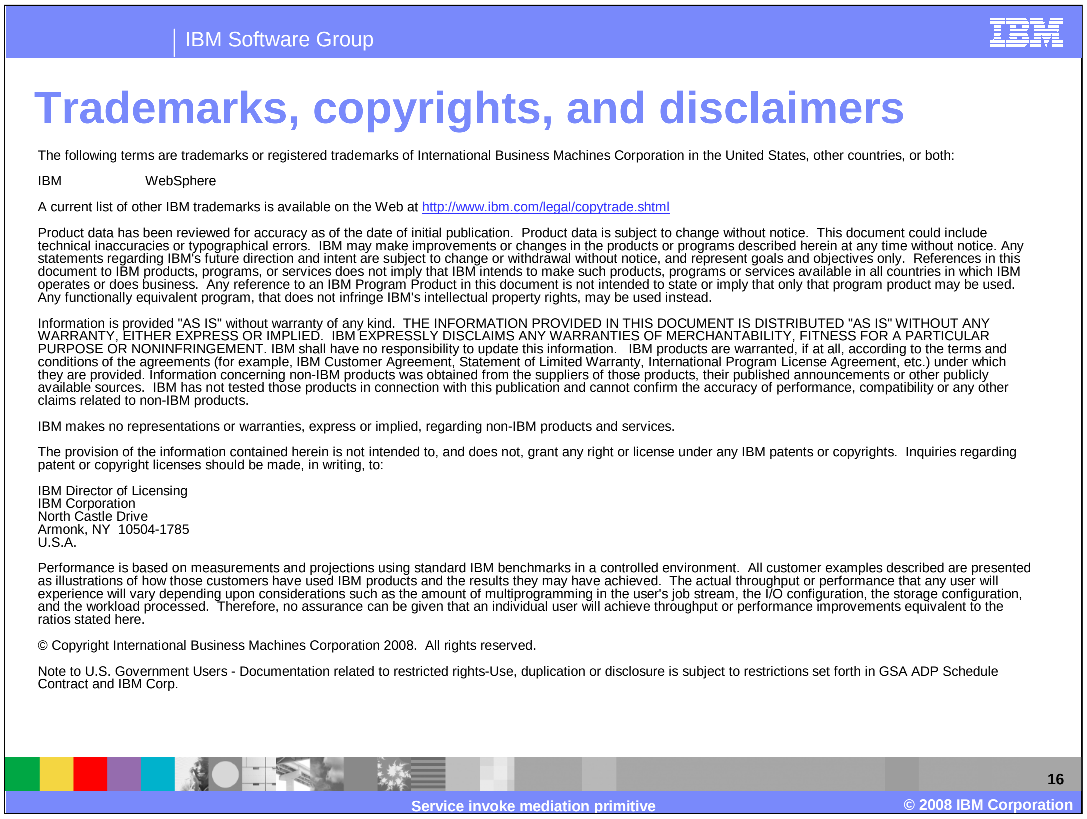
- Did you find this module useful?
- Did it help you solve a problem or answer a question?
- Do you have suggestions for improvements?

Click to send e-mail feedback:

mailto:iea@us.ibm.com?subject=Feedback_about_WBIV61_ServiceInvokePrimitive.ppt

This module is also available in PDF format at: ../WBIV61_ServiceInvokePrimitive.pdf

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



Trademarks, copyrights, and disclaimers