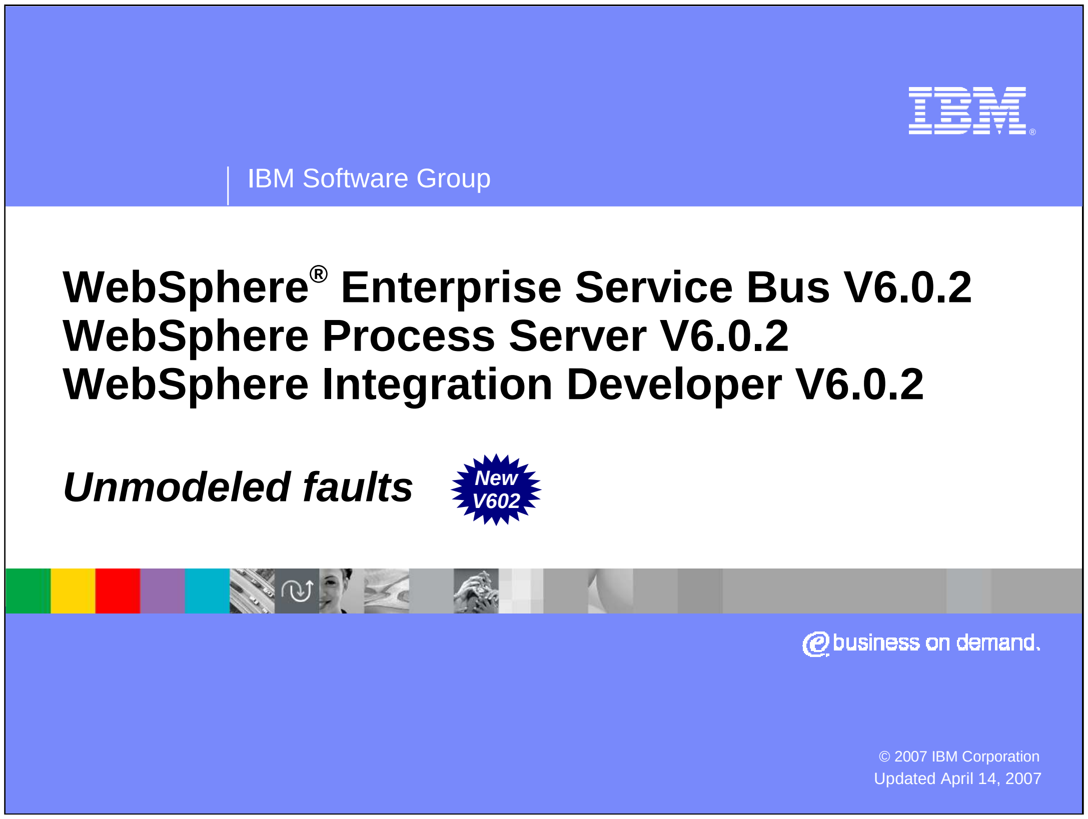


IBM Software Group

WebSphere® Enterprise Service Bus V6.0.2 WebSphere Process Server V6.0.2 WebSphere Integration Developer V6.0.2

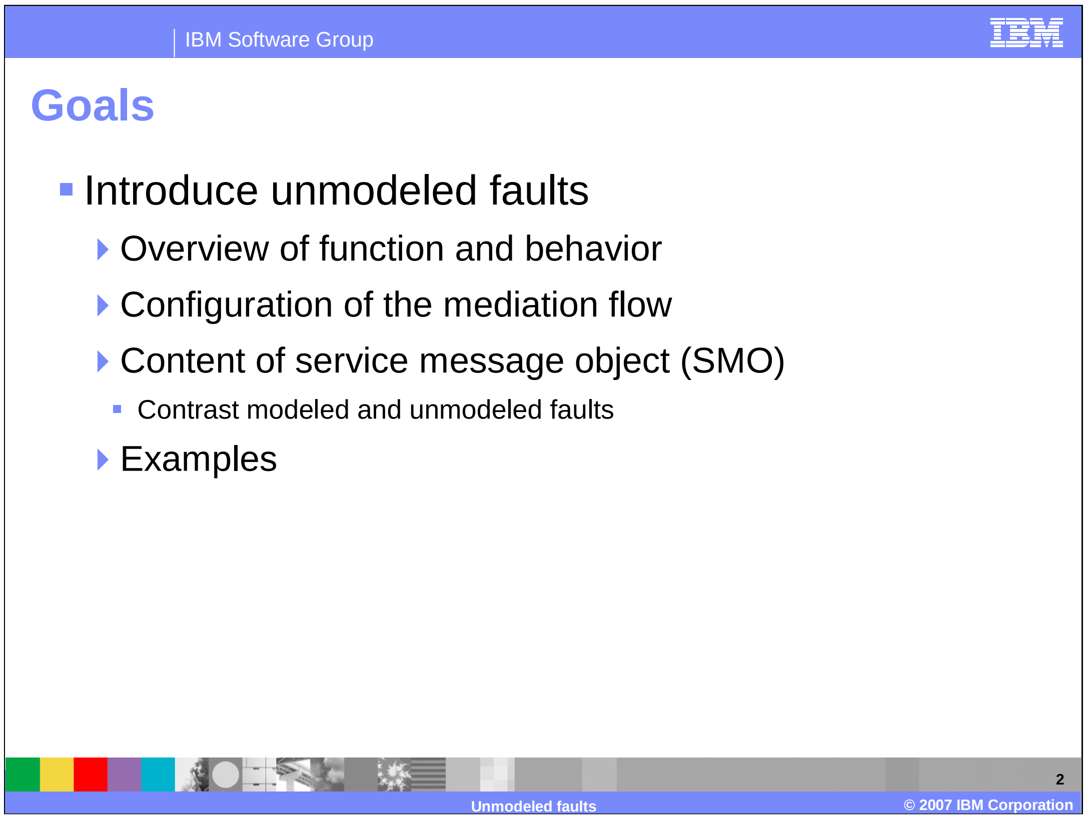
Unmodeled faults

New V602

business on demand.

Updated April 14, 2007

This presentation provides a detailed look at unmodeled faults and how they are handled in mediation flows. This capability is new in version 6.0.2.



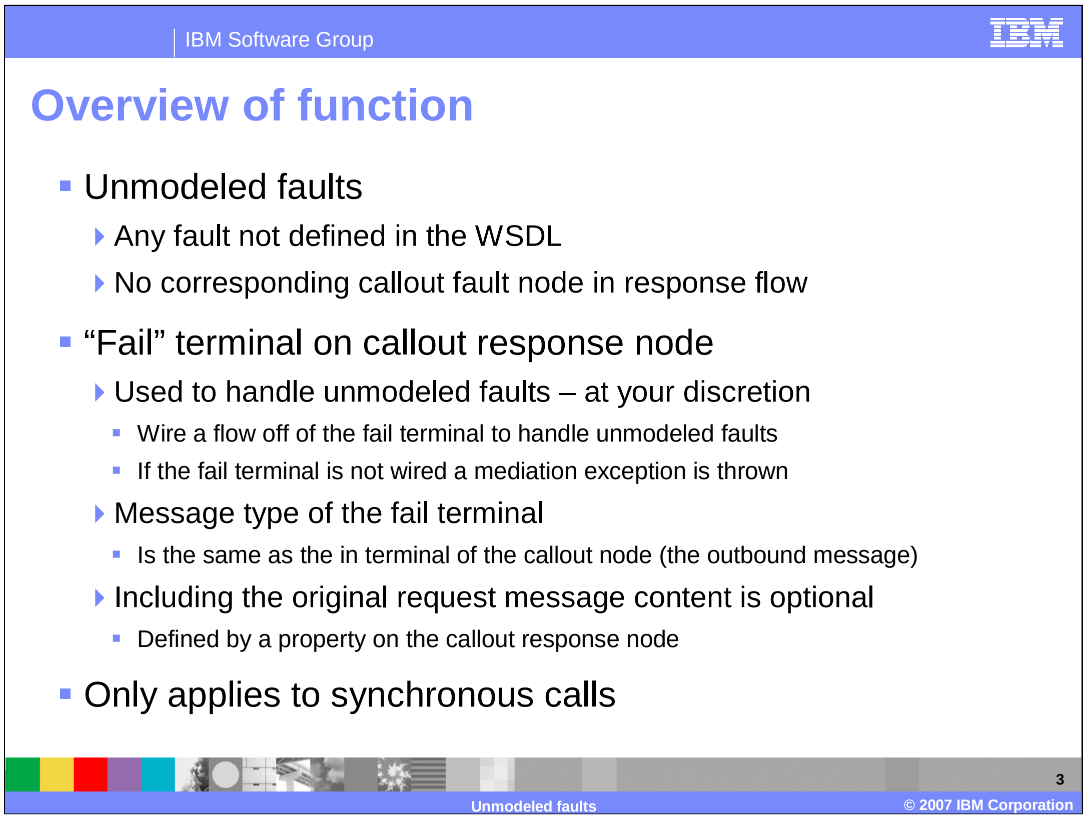
Goals

- Introduce unmodeled faults
  - Overview of function and behavior
  - Configuration of the mediation flow
  - Content of service message object (SMO)
    - Contrast modeled and unmodeled faults
  - Examples

The goal of this presentation is to provide you with a full understanding of unmodeled faults. The presentation begins with an overview of the function and behavior of unmodeled faults and then looks at how to configure your mediation flow in WebSphere Integration Developer.

You will see how both the unmodeled fault and modeled fault information are represented in the service message object, or SMO.

Finally, some examples are provided. These show you how you can make use of the unmodeled fault capabilities within your mediation flows.


IBM Software Group

Overview of function

- Unmodeled faults
  - Any fault not defined in the WSDL
  - No corresponding callout fault node in response flow
- “Fail” terminal on callout response node
  - Used to handle unmodeled faults – at your discretion
    - Wire a flow off of the fail terminal to handle unmodeled faults
    - If the fail terminal is not wired a mediation exception is thrown
  - Message type of the fail terminal
    - Is the same as the in terminal of the callout node (the outbound message)
  - Including the original request message content is optional
    - Defined by a property on the callout response node
- Only applies to synchronous calls

3

Unmodeled faults

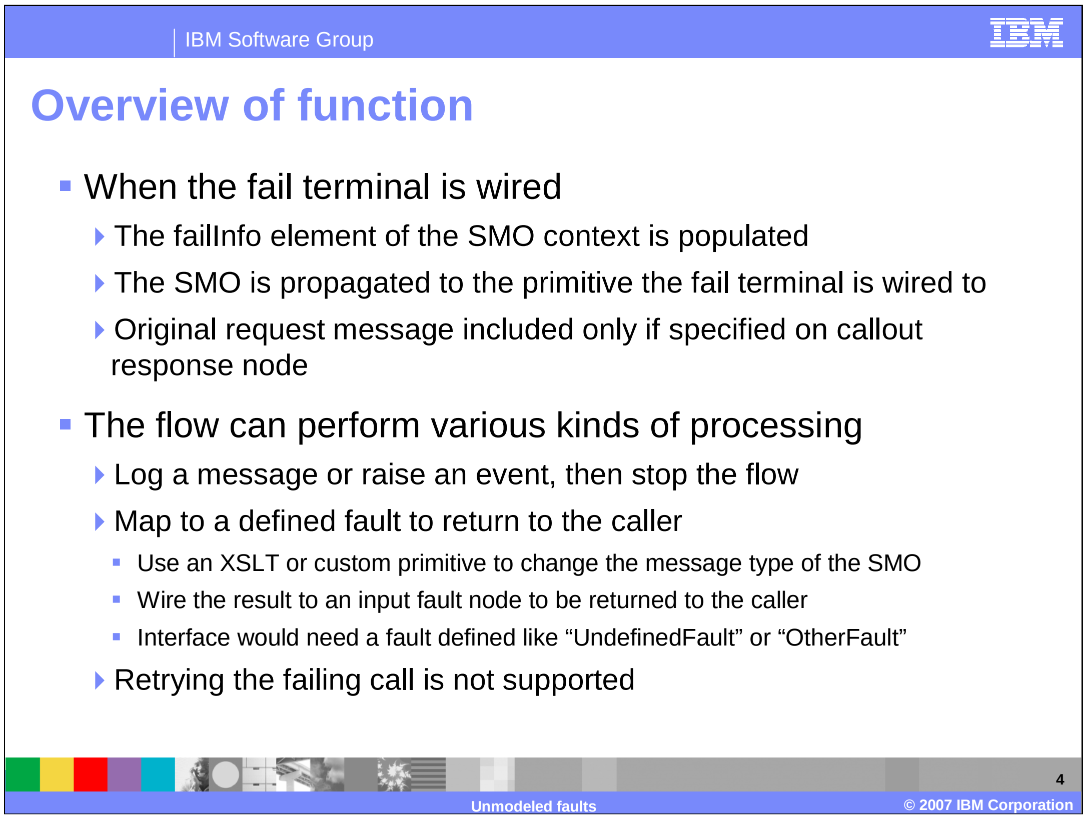
© 2007 IBM Corporation

An unmodeled fault is any fault that is not defined in the WSDL definition of the interface. Since the fault is not defined in WSDL, there is no callout fault node shown in the response flow of the mediation flow editor.

The handling of unmodeled faults makes use of a fail terminal that is on the callout response node. You can choose to provide special handling for unmodeled faults by wiring the fail terminal of the callout response node to some appropriate mediation logic. If you choose not to handle the unmodeled faults, leave the fail terminal unwired. This results in the mediation flow ending with an exception when an unmodeled fault is raised.

The message type of the fail terminal on the callout response node is the outbound message type defined for the in terminal of the callout node. The content of the original request message can be included in the message flowing from the fail terminal. However, that capability is optional and is controlled by a property on the callout response node.

A very important point to understand is that unmodeled fault support only works for synchronous calls. Unmodeled faults occurring on asynchronous calls to service providers are handled by the service component architecture infrastructure and the unmodeled fault is never returned to the mediation flow.



IBM Software Group

Overview of function

- When the fail terminal is wired
  - The failInfo element of the SMO context is populated
  - The SMO is propagated to the primitive the fail terminal is wired to
  - Original request message included only if specified on callout response node
- The flow can perform various kinds of processing
  - Log a message or raise an event, then stop the flow
  - Map to a defined fault to return to the caller
    - Use an XSLT or custom primitive to change the message type of the SMO
    - Wire the result to an input fault node to be returned to the caller
    - Interface would need a fault defined like “UndefinedFault” or “OtherFault”
  - Retrying the failing call is not supported

4

Unmodeled faults

© 2007 IBM Corporation

When the fail terminal of the callout response node is wired, the unmodeled fault information is placed into the failInfo element of the SMO. The SMO is then propagated to the mediation primitive wired to the fail terminal. If the callout response node was configured to include the original request message, all of the content of the outbound SMO is included.

The flow that is wired off of the fail terminal can do any processing that is possible in a mediation flow. A highly probable implementation would be to raise an event with the event emitter primitive or to log a message using the Message Logger primitive, and then end the flow with a stop primitive. It is also possible to provide logic that would return a modeled fault to the original requestor. This can be done by using an XSLT primitive or custom mediation primitive to change the message type of the SMO so that it matches the message type of the input fault node. If you were to do this, you would most likely define a fault on your interface which was a catch all fault, named something like UndefinedFault or OtherFault.

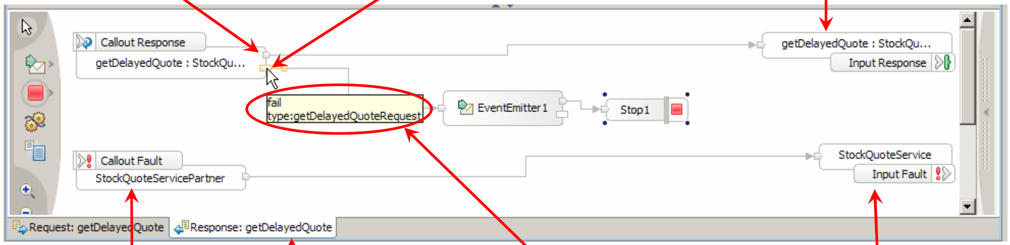
The mediation flow runtime does not provide the capability to retry the failing call.

Tooling support – Unmodeled fault

Callout response node – “fail” terminal – unmodeled fault from service

Callout response node – “out” terminal - successful return from service

Input response node – successful return to caller



Callout fault node - receive modeled fault from service

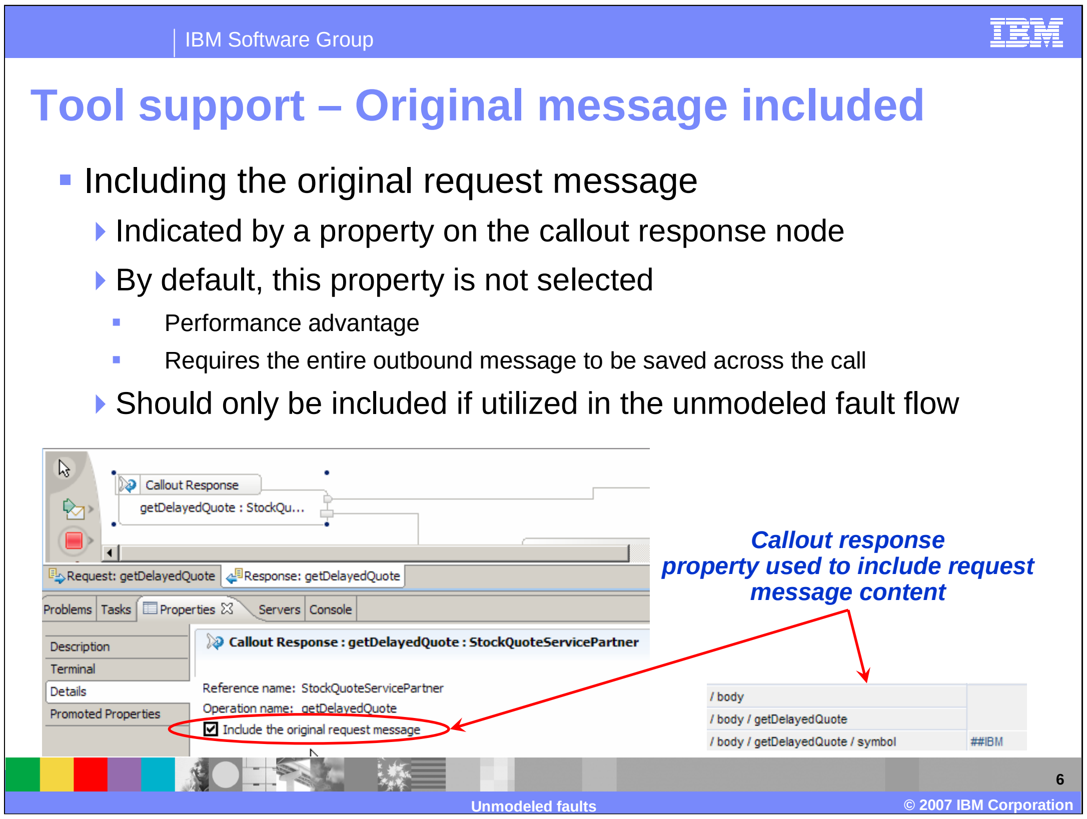
Response flow

“fail” terminal – message type same as for the outbound message

Input fault node - return modeled fault to caller

This slide shows a response flow that is handling an unmodeled fault by raising an event and then stopping the flow without raising an exception.

On the top left you can see the callout response node. The terminal on the top of the node is the out terminal for a normal return, and you can see that it is wired directly to the input response node which returns to the original caller. On the bottom left there is a callout fault node. This node is where a WSDL defined, or modeled fault, enters the response flow. You can see that it is wired directly to the input fault node which returns the fault to the original requestor. Returning back to the top left, the bottom terminal of the callout response node is the fail terminal where unmodeled faults are returned. You can see that the message type for the fail terminal is getDelayedQuoteRequest, the same as the message type of the outbound message. The fail terminal is wired to an event emitter primitive, which produces an event to the Common Event Infrastructure, and then proceeds to the Stop primitive which terminates the flow without raising an exception.


IBM Software Group

Tool support – Original message included

- Including the original request message
  - Indicated by a property on the callout response node
  - By default, this property is not selected
    - Performance advantage
    - Requires the entire outbound message to be saved across the call
  - Should only be included if utilized in the unmodeled fault flow

Reference name: StockQuoteServicePartner
Operation name: getDelayedQuote
[✓] Include the original request message

/ body	
/ body / getDelayedQuote	
/ body / getDelayedQuote / symbol	##IBM

Callout response property used to include request message content

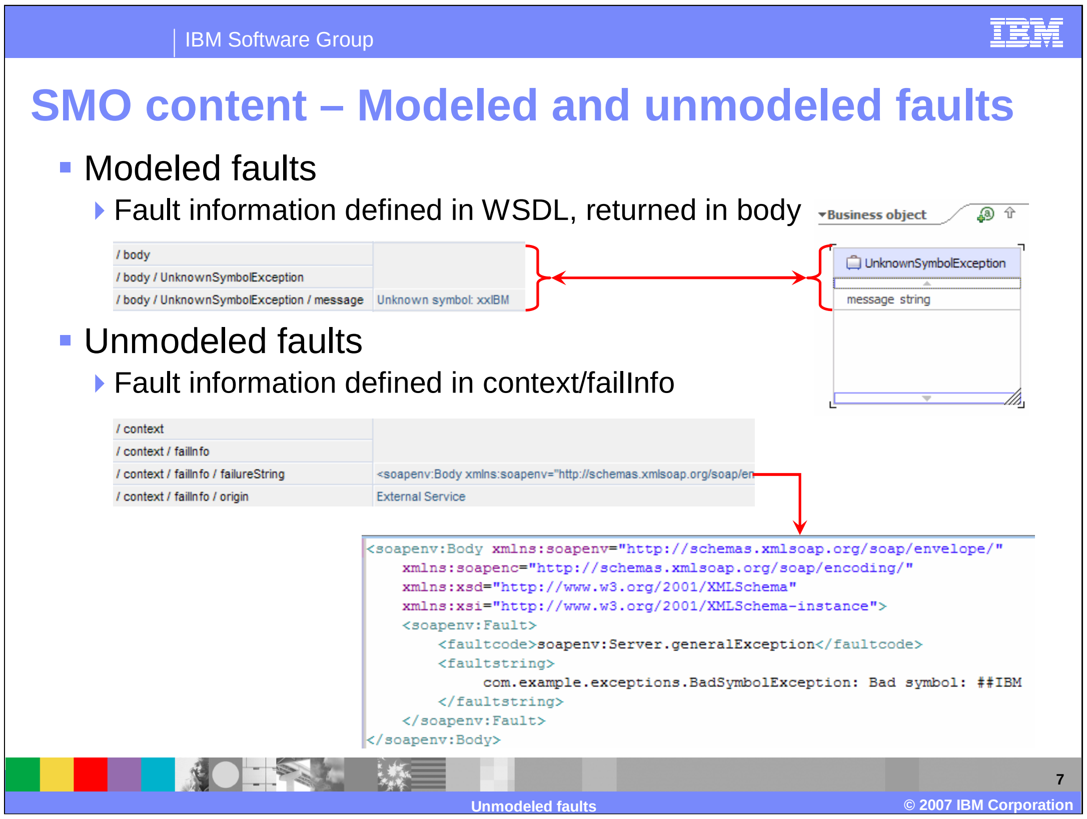
6

Unmodeled faults

© 2007 IBM Corporation

This slide shows how to have the original request message included in the response flow for an unmodeled fault. The screen capture shows the response flow on the top left, and immediately underneath it is the Details panel of the Properties view for the callout response node. It is the check box labeled Include the original request message that controls this capability. On the right is a screen capture showing the body from the original request which is included when this option is selected.

By default, this property is not selected. There is a performance implication when the entire outbound message needs to be saved across the call. Therefore, you should only make use of this capability if your flow logic for the unmodeled faults makes specific use of the outbound message data.



IBM Software Group

SMO content – Modeled and unmodeled faults

- Modeled faults
  - Fault information defined in WSDL, returned in body

/ body	
/ body / UnknownSymbolException	
/ body / UnknownSymbolException / message	Unknown symbol: xxIBM

Business object
UnknownSymbolException
message string

- Unmodeled faults
  - Fault information defined in context/failInfo

/ context	
/ context / failInfo	
/ context / failInfo / failureString	<soapenv:Body xmlns:soapenv="http://schemas.xmlsoap.org/soap/en
/ context / failInfo / origin	External Service

```
<soapenv:Body xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soapenv:Fault>
        <faultcode>soapenv:Server.generalException</faultcode>
        <faultstring>
             com.example.exceptions.BadSymbolException: Bad symbol: ##IBM
        </faultstring>
    </soapenv:Fault>
</soapenv:Body>
```

7

Unmodeled faults

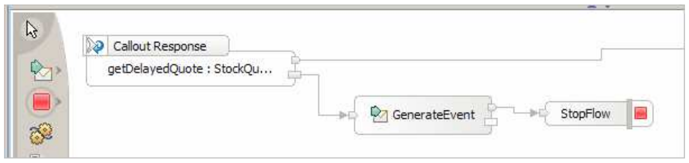
© 2007 IBM Corporation

Both unmodeled and modeled fault data are placed into the SMO, but where they are placed is different. For a modeled fault, there is a WSDL definition that defines what is returned with the fault. In the upper screen capture, on the right, is a business object which has been defined to be returned for a modeled fault. On the upper left you can see that the information is placed into the body of the SMO.

For unmodeled faults, there is no WSDL definition and therefore no definition of what could be placed into the message body. The lower screen capture shows that the unmodeled fault is returned in the failInfo element within the context of the SMO. You can see that in this case, since the callout was for a Web Service, the unmodeled fault information is returned as a soap fault that is placed into the failureString of the failInfo element.

Usage scenario – Generate event

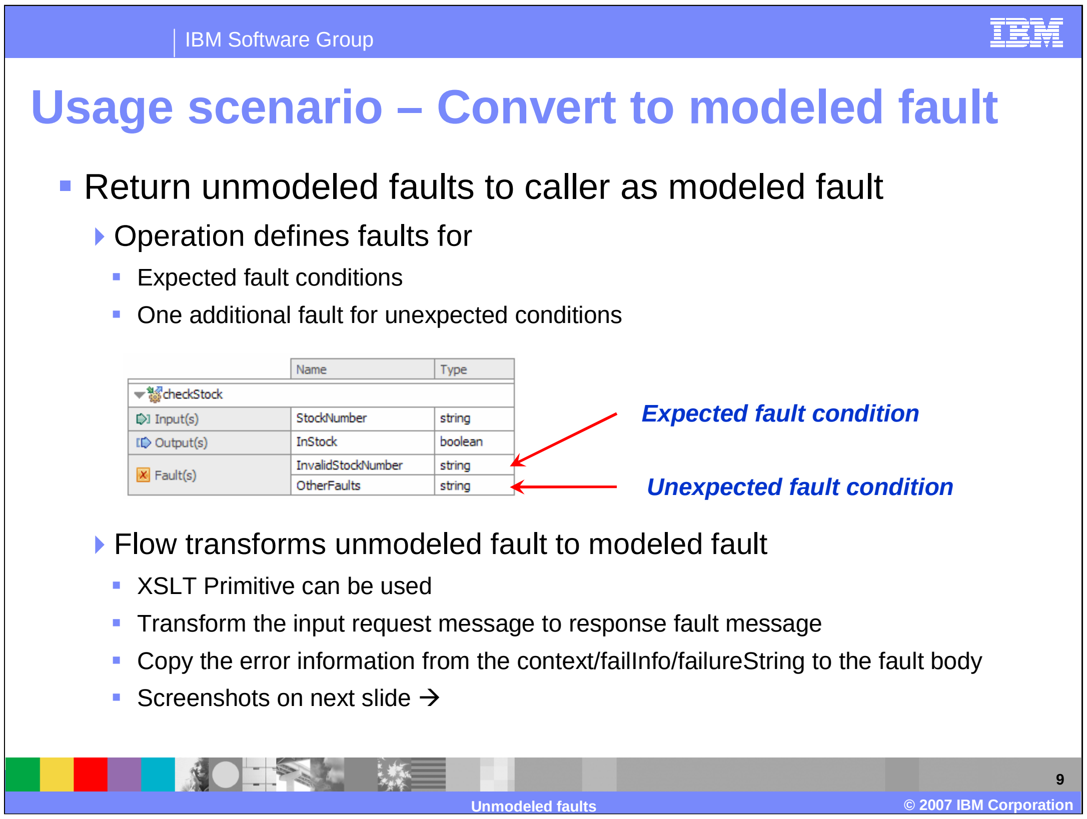
- Unmodeled fault flow generates event
  - Generated with event emitter primitive
    - Handled by common event infrastructure (CEI)



  - Event can be used to document occurrence of the fault
  - Monitoring application can take action on event
    - Use CEI to filter events
    - Obtain this event as a JMS message
    - Read this event from the event database
    - Initiate action to debug or retry operation based on event

In this example an unmodeled fault causes an event to be generated and then the flow terminates without an exception being thrown. You can see that the fail terminal is wired to an event emitter primitive and then it is wired to a stop primitive.

Because event emitter primitives generate common base events handled by the common event infrastructure, or CEI, there are several options regarding how the event is handled. The CEI server provides filtering capabilities which are used to determine what it does with the event. For example, the event can be added to the CEI event database and it can also be sent as a message to a JMS queue or published to a JMS topic. This enables a monitoring application that is searching the database or receiving the JMS message to act on the event appropriately, such as initiating debugging activities or retrying the operation.



IBM Software Group

Usage scenario – Convert to modeled fault

- Return unmodeled faults to caller as modeled fault
  - Operation defines faults for
    - Expected fault conditions
    - One additional fault for unexpected conditions

	Name	Type
checkStock		
Input(s)	StockNumber	string
Output(s)	InStock	boolean
Fault(s)	InvalidStockNumber	string
	OtherFaults	string

Expected fault condition

Unexpected fault condition

  - Flow transforms unmodeled fault to modeled fault
    - XSLT Primitive can be used
    - Transform the input request message to response fault message
    - Copy the error information from the context/failInfo/failureString to the fault body
    - Screenshots on next slide →

9

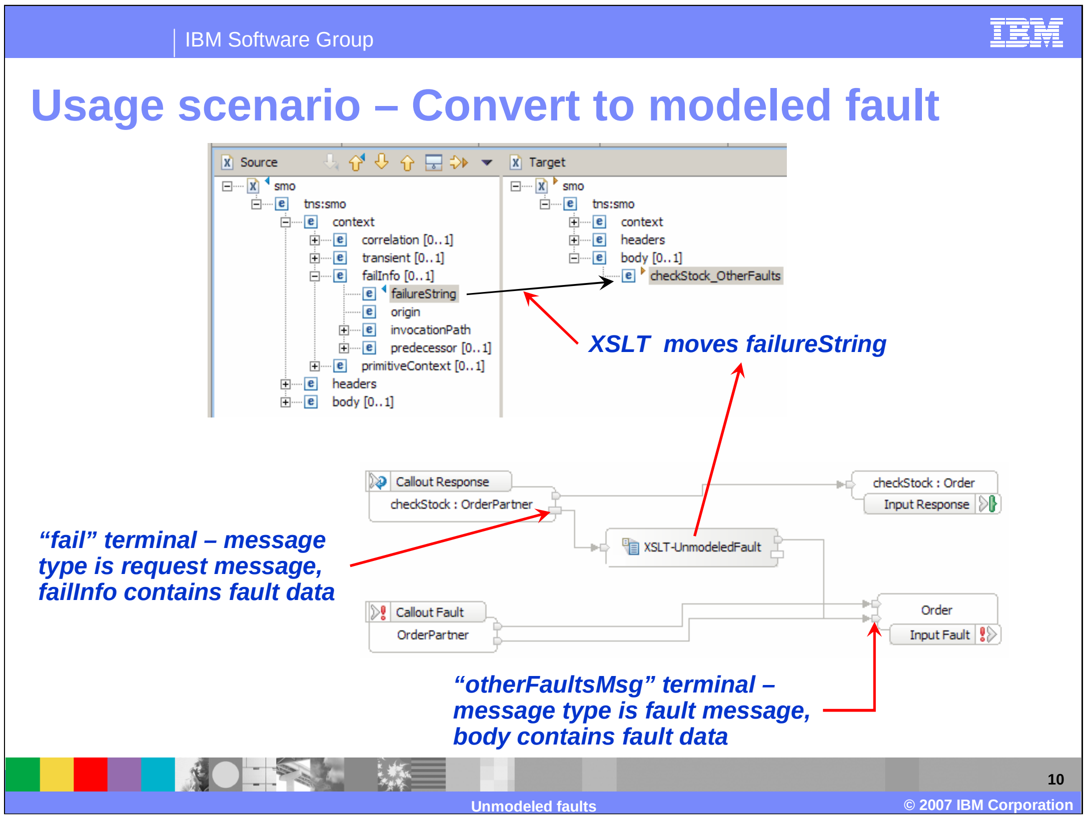
Unmodeled faults

© 2007 IBM Corporation

This next example is presented over two slides. This is the case of converting an unmodeled fault to a modeled fault so that it can be returned to the original caller.

Looking at the screen capture, you can see an interface with an operation that has an input, an output and two faults. There is an InvalidStockNumber fault that represents an expected fault condition that the service would return. In addition, there is an OtherFaults fault that is used to return any unmodeled faults that occur. An XSLT primitive is used to transform the message from the request flow message type to the OtherFaults fault message type. The transformation includes copying the failureString from the failInfo section of the message context to the body of the fault message.

The next slide provides an illustration of how this is handled.


IBM Software Group

Usage scenario – Convert to modeled fault

XSLT moves failureString

"fail" terminal – message type is request message, failInfo contains fault data

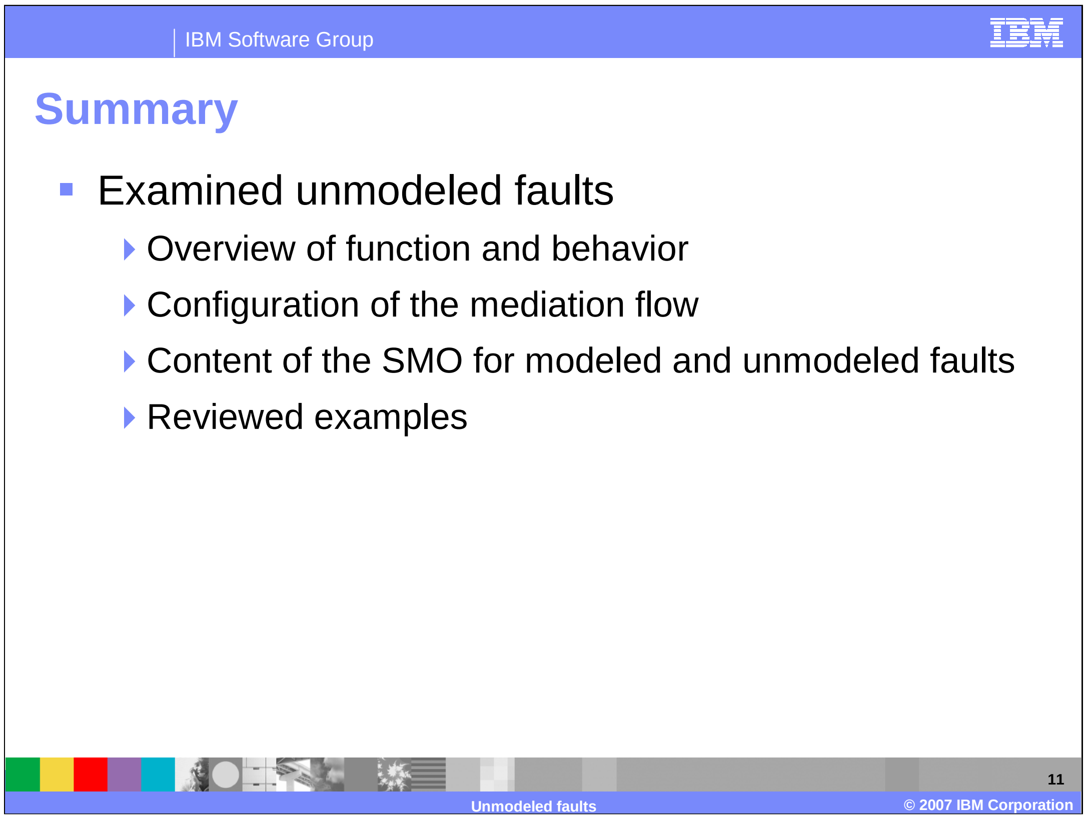
"otherFaultsMsg" terminal – message type is fault message, body contains fault data

10

Unmodeled faults

© 2007 IBM Corporation

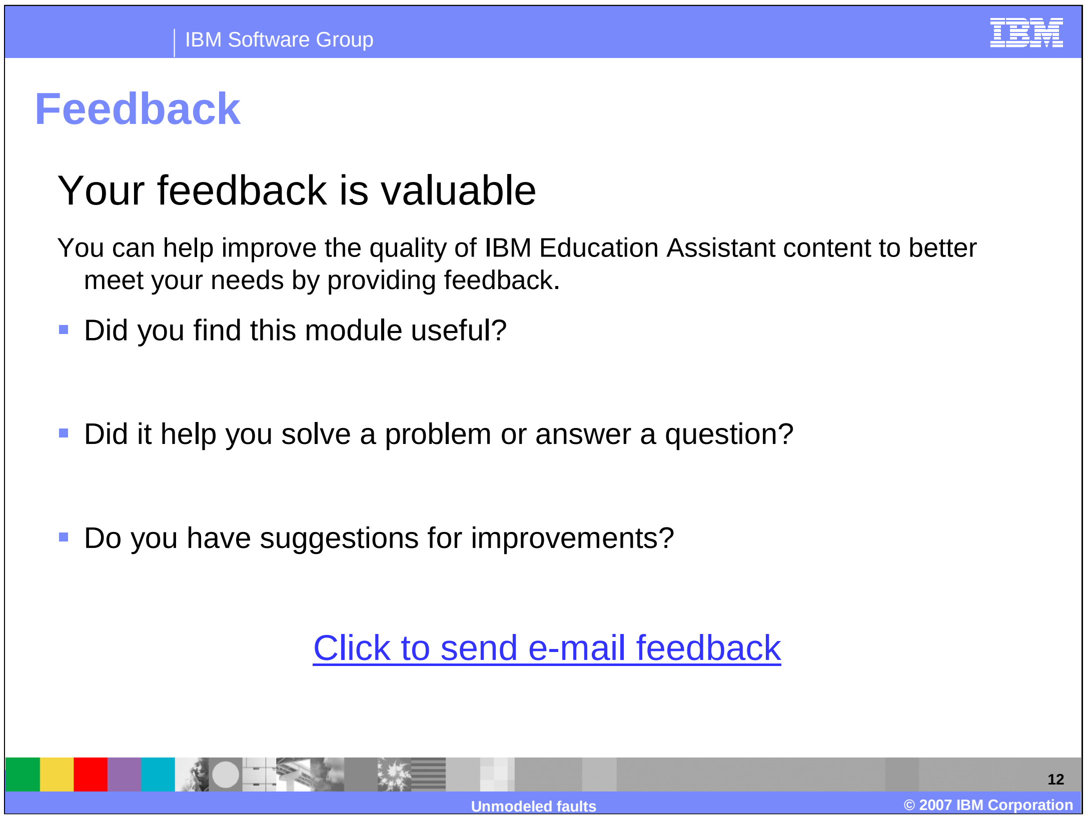
On the bottom portion of the slide is the response flow. On the bottom of the flow you can see that the callout fault node has two terminals, one for each of the defined faults. These are wired to the two terminals on the input fault node which are for the same two WSDL defined faults. These wires represent the flow that occurs if the service explicitly returns either of these two faults. In addition, the fail terminal of the callout response node is wired to an XSLT primitive which is then wired to the OtherFaults fault terminal of the input fault node. The unmodeled faults flow through here. The upper portion of the slide shows the XML map that is used to copy the unmodeled fault information from the failureString in the context to the body of the fault message.



Summary

- Examined unmodeled faults
  - Overview of function and behavior
  - Configuration of the mediation flow
  - Content of the SMO for modeled and unmodeled faults
  - Reviewed examples

In this presentation you were provided with an introduction to unmodeled faults. The presentation began with an overview of the function and behavior of unmodeled faults and then looked at how to configure your mediation flow in WebSphere Integration Developer. A description of how fault information is represented in the SMO for both modeled and unmodeled faults was provided. Finally, there were a couple of examples to show how you can make use of the unmodeled fault capabilities within your mediation flows.



Feedback

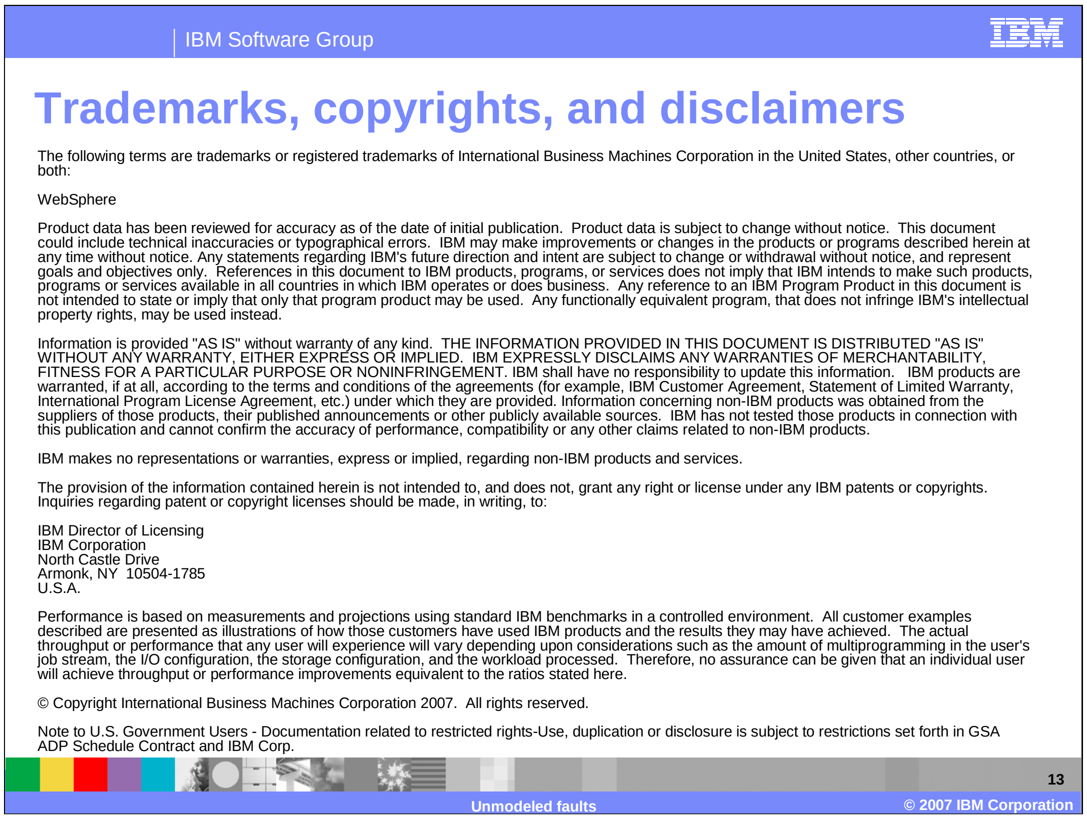
Your feedback is valuable

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

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



Trademarks, copyrights, and disclaimers