WebSphere. software

IBM

Redbooks Paper

Chris Nott
Peter Edwards
Andrew Humphreys
Martin Keen

Using Message Sets in WebSphere Business Integration Message Broker to Implement an ESB in an SOA

Introduction

This Redpaper shows how to build an Enterprise Service Bus (ESB) using current IBM® technology. The paper is based on a simple application scenario in which Web services invocations are used to implement a simplified supply chain process for a consumer electronics retailer. We add an ESB into the scenario between the service consumers and service providers, which provides greater flexibility and loose coupling. We explore design considerations for an ESB in a service-oriented architecture (SOA) and implement the Broker design pattern for this decoupling. We use WebSphere® Business Integration Message Broker V5 to implement the ESB with services hosted on WebSphere Application Server.

This Redpaper has three sections:

- Design guidelines
- Development guidelines
- Runtime guidelines

This paper is an extension of the IBM Redbook *Patterns: Implementing an SOA Using an Enterprise Service Bus*, SG24-6346.

Design guidelines

This section introduces the business scenario used in this paper, describes how the Patterns for e-business apply to the solution, and discusses a design alternative for building the solution.

Business scenario

The application used in this Redpaper is based on the WS-I Supply Chain Management sample application that supports a simplified supply chain for a consumer electronics retailer business scenario. This Redpaper extends the application built in the IBM Redbook *Patterns: Implementing an SOA Using an Enterprise Service Bus*, SG24-6346.

We use this business scenario to show how the Patterns for e-business, SOA, and ESB approach can be used to develop solutions to real-world business requirements that are based on interoperability principles as defined in the WS-I Basic Profile.

The scenario is based on a typical business-to-customer (B2C) model. Customers may access the retailer's Web site, review the catalog, and place orders for products such as televisions, DVD players, and video cameras. The retailer system requests fulfillment of a consumer's order from the internal company warehouses, which respond as to whether line items from the order can be filled. If stock for any line item falls below a minimum threshold in the warehouse that stocks the item, a replenishment order is sent to an external manufacturer using the business-to-business (B2B) model. The manufacturer does not immediately fulfill replenishment orders, but completes the order at some later time (possibly after completing a manufacturing run).

The company uses more than one warehouse to stock the parts that it offers to customers. However, the customer must see the order as a single transaction with the company. Therefore, aggregation of the interactions with the warehouses is required to produce a single response to a consumer request. This is shown in Figure 1.

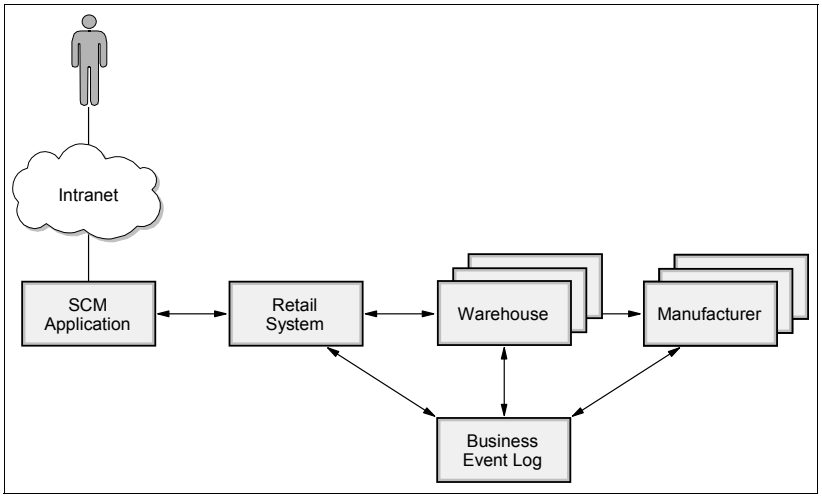

Figure 1 Business scenario

Applying the Patterns for e-business

The IBM Patterns for e-business are a group of proven, reusable assets that can be used to increase the speed of developing and deploying e-business applications. The Patterns for e-business define an SOA profile for architecting SOA implementations. This profile is defined in the redbook *Patterns: Implementing an SOA Using an Enterprise Service Bus with WebSphere Application Server V6*, SG24-6494.

The Patterns for e-business can be applied to this business scenario to determine the required architecture. Using the SOA profile of the Patterns for e-business we can determine that the Enterprise Service Bus runtime pattern is an ideal fit for this business scenario. In particular an Enterprise Service Bus runtime pattern using broker interactions is required to facilitate the interaction between the retail system and the warehouses.

The Broker application pattern is based on a 1-to-N topology that separates distribution rules from the applications. It enables a single interaction from the source application to be distributed to multiple target applications concurrently. This Application pattern reduces the proliferation of point-to-point connections. It is shown in Figure 2 on page 4.

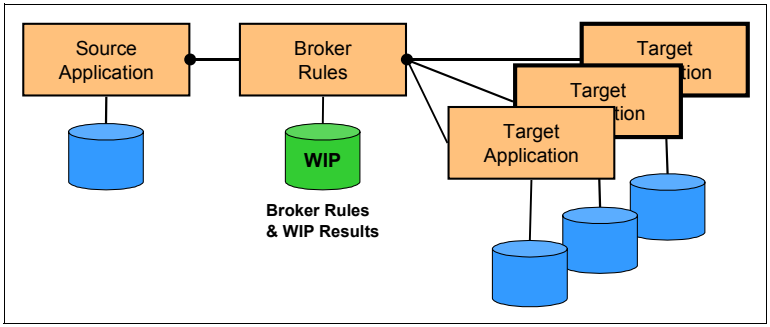

Figure 2 Application Integration::Broker pattern

An ESB is a component in an SOA that mediates interactions between service consumers and service providers so that consumers and providers are independent. It may provide additional value-added functionality within. The Broker application pattern can be applied to the Hub component of an ESB. Figure 3 shows the Enterprise Service Bus Runtime pattern.

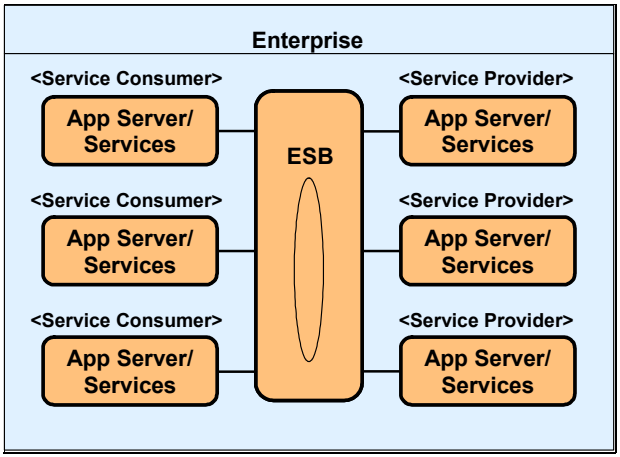

Figure 3 Enterprise Service Bus runtime pattern - level 0

The ESB is a key enabler for an SOA as it provides the capability to route and transport service requests from the service requester to the correct service provider. The ESB controls routing within the scope of a service namespace, which is indicated symbolically by the ellipse on the ESB node representation.

The true value of the ESB concept, however, is to enable the infrastructure for SOA in a way that reflects the needs of today's enterprise: to provide suitable service levels and manageability, and to operate and integrate in a heterogeneous environment.

The ESB must be centrally managed and administered and have the ability to be physically distributed.

The Runtime pattern shown in Figure 4 represents a first-level decomposition of the major components that make up an ESB. The Broker application pattern can be applied to the Hub component of an ESB.

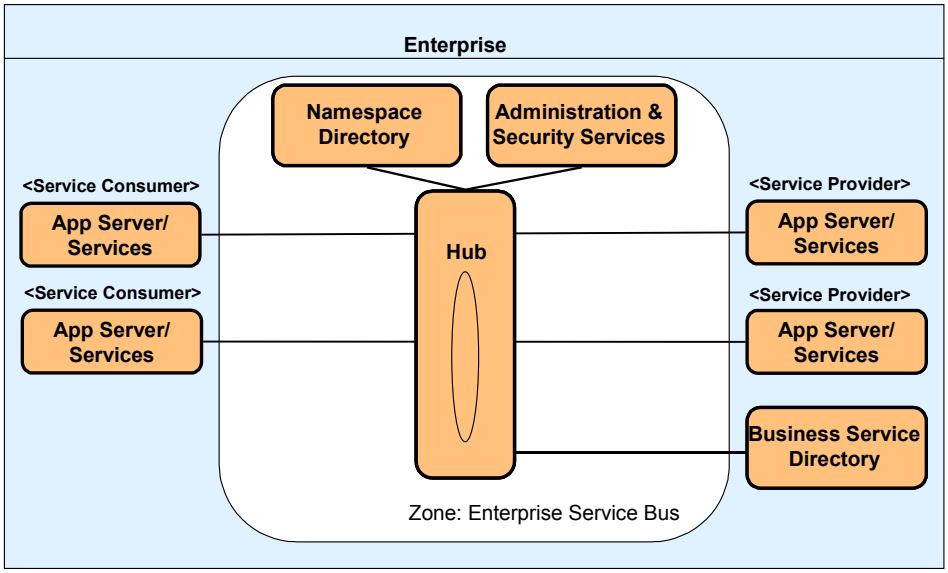

Figure 4 Enterprise Service Bus runtime pattern - level 1

The Enterprise Service Bus implementation for this business scenario is applied to the level 0 decomposition of the Enterprise Service Bus runtime pattern, as shown in Figure 5.

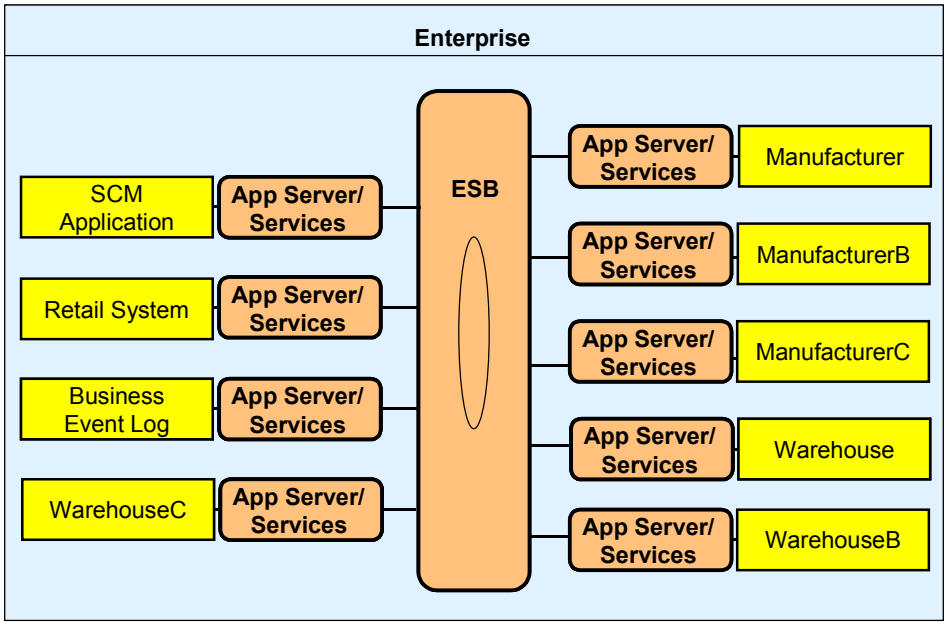

Figure 5 Enterprise Service Bus pattern applied to our scenario

The scenario implementation in this chapter requires multiple warehouses to be invoked potentially concurrently from a single retailer. This scenario requirement is met by using broker interactions in the Hub component of the Enterprise Service Bus runtime pattern.

From this we can derive the Product mapping, which represents how the business scenario was implemented in this paper. See Figure 6.

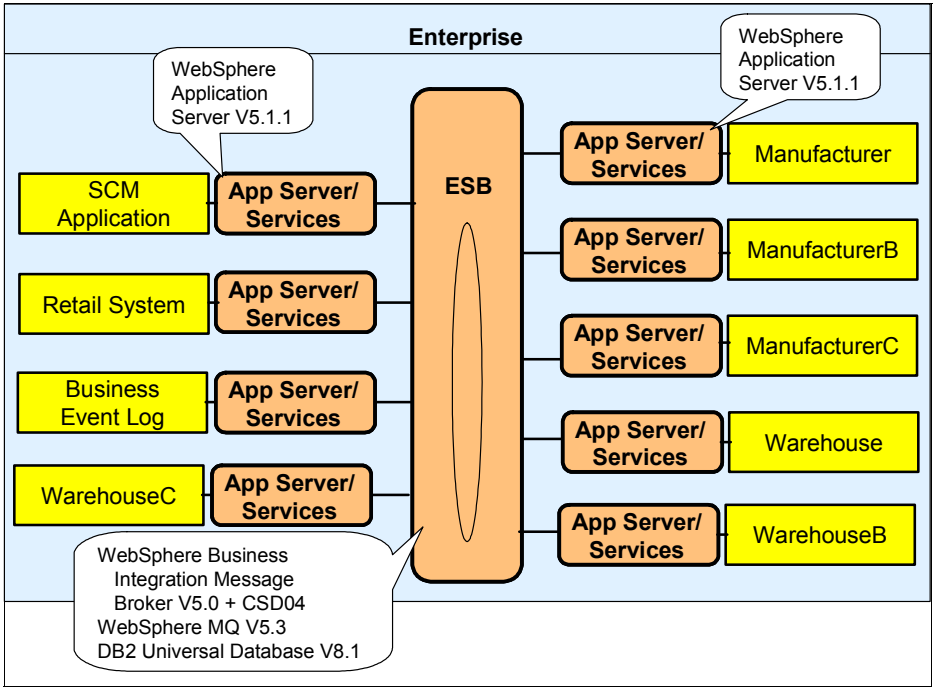

Figure 6 Product mapping

For further information about the SOA profile of the Patterns for e-business, see *Patterns: Implementing an SOA Using an Enterprise Service Bus with WebSphere Application Server V6*, SG24-6494.

For further information about the Patterns for e-business in general, see the IBM developerWorks® Web site:

`http://www.ibm.com/developerWorks/patterns`

Design alternative: SOAP message parsing

As described in the redbook *Patterns: Implementing an SOA Using an Enterprise Service Bus*, SG24-6346, a true SOAP intermediary carries out the validation and processing of SOAP headers. This Redpaper shows the development of an

example of parsing a SOAP message within the ESB. Parsing enables the validation of part or all of the SOAP message and can provide these advantages:

- Validation and processing of SOAP headers can be performed.
  - SOAP faults are handled properly so that they correctly identify the service that is causing the fault:
    - Generation of SOAP faults
    - Encoding of SOAP faults
  - Operations supported by the service can be validated.
  - The SOAP header must be properly validated and processed when the mustUnderstand attribute is set to 1.
  - Decode and encode the SOAP service requests and responses.
  - Operation name and related namespace can be set.
  - Service style can be switched from document to RPC.
- Validation and processing of the SOAP bodies, representing message content, can take place in the ESB. This includes mediation of service requests within the ESB based on the values within the SOAP body.
- Development of the intermediary logic may be eased because the message structures are defined in the tooling up front.

The simple advantage of not validating SOAP messages and processing SOAP headers is performance, because parsing of the message is not required.

In our scenario, the ESB is required to perform aggregation logic in order to ship goods on an order. By using WebSphere Business Integration Message Broker we can define message sets to validate messages prior to executing the logic. This can occur at the beginning of message flows and also within the aggregation logic as responses are returned from service providers.

We shall also see how the Message Broker Toolkit simplifies development of logic based on the content of SOAP bodies that are defined as message sets. In our scenario we create message sets for the parts and quantities on an order.

Further information about using WebSphere Business Integration Message Broker V5 in a Web services environment can be found in SupportPac™ IA81, which is available through ibm.com® at:

http://www.ibm.com/support/docview.wss?rs=171&uid=swg24006268

In particular, this SupportPac provides a subroutine library for managing the SOAP aspects of incoming, outgoing, and fault messages.

Development guidelines

This section describes how WebSphere Business Integration Message Broker V5 supports Web services within a service-oriented architecture implementation for our business scenario. It contains a step-by-step guide for building this solution.

The starting point is an application built in the redbook *Patterns: Implementing an SOA Using an Enterprise Service Bus,* SG24-6346. This application comprises WebSphere Business Integration Message Broker message flows and a number of WebSphere Application Server enterprise applications exchanging SOAP messages.

The existing aggregation message flow between the Retailer and Warehouses (shown in Figure 7) parses and builds SOAP messages; these SOAP messages are processed without reference to message sets. Notice that the flow performs protocol conversion from HTTP to JMS.

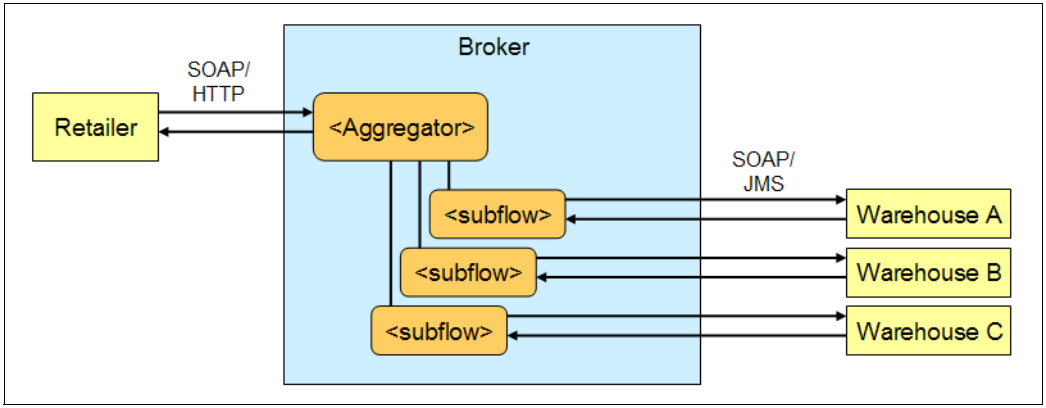

Figure 7 Aggregation design

This paper describes how to model a SOAP message in WebSphere Business Integration Message Broker message sets. It shows how both the envelope and the body of the message are modeled and how message models can be defined to support different SOAP bodies within the same message definition.

We then demonstrate techniques that show how this modeling benefits the development and execution of WebSphere Business Integration Message Broker message flows.

Model a SOAP message in a message set

In this section you create a message set and model the SOAP messages used in the ShipGoods operation of the Warehouse service.

Overview of message sets

A message set is created inside a message set project in the Message Broker Toolkit. Note the following:

- Only one message set is allowed per message set project.
- Many message set projects can be used in a system, but only one message set can be used by a Broker when parsing or writing a message instance.
- Message sets can be namespace-enabled. As SOAP messages always use namespaces, any message sets used for modelling SOAP messages *must* be namespace-enabled.
- Message sets have one or more wire formats. These represent the style of message to be input or output. Wire formats fall into three categories: Custom (mapping C or COBOL structures), XML, and Tagged/Delimited (mapping industry standards such as SWIFT). SOAP messages require an XML wire format.
- Message sets contain one or more message definition files with the suffix .mxsd. These are valid XML schema files, annotated with WebSphere Business Integration Message Broker-specific enhancements. They can be created by importing an XML schema, DTD, C header file, or COBOL copybook. There is a customized editor in the Message Broker Toolkit to manipulate the contents of message definition files.
- Message definition files contain elements (global or local), complex types, groups, and attributes (global or local), much as XML schemas do. Message definition files also contain messages, which are used by WebSphere Business Integration Message Broker as a unit of processing in a message flow.
- It is possible to INCLUDE or IMPORT one message definition file within another. These have similar functionality to the XML schema INCLUDE and IMPORT.

The Message Broker Toolkit contains a number of importers that enable messages to be constructed easily. However, you cannot directly import Web Services Description Language (WSDL) into the Message Broker Toolkit. To import a WSDL file, you must extract the XML schema to a separate file and import that file.

After an XML schema is imported, message definition files may require certain customizations, which can be made using the Message Broker Toolkit.

Message definition files also contain extra information that XML schemas do not have. One example is a keyword of complex type known as *composition*. This can indicate known XML schema properties such as *sequence* or *choice*. It also enables a WebSphere Business Integration Message Broker–specific property message, which means the same as choice, but the children of this complex type are messages rather than elements or attributes. This is useful when modeling business messages that are embedded within a standard envelope.

For example, a SOAP message comprises an envelope (XML tag name Envelope), which has a child element called Body. That Body element contains Web service payloads (application data), which are likely to be modeled in a WebSphere Business Integration Message Broker message set as messages. Therefore, the Body element is a likely candidate to have a composition message.

Note: WSDL files can be generated from message sets, though these may need customization prior to use by other Web services development tools. Within an MDF, messages can be grouped into categories. Message categories must be defined prior to WSDL generation. Categories are needed to enable WSDL generation from the Message Broker Toolkit.

Create a message set

To create a message set:

1. Start the WebSphere Business Integration Message Broker Toolkit.
2. Ensure that you are in the Broker Application Development perspective. Select **Window** → **Open Perspective** → **Broker Application Development** from the menu.
3. Create a new Message Set Project.
   a. Click **File** → **New** → **Message Set Project**.
   b. Enter `labmsp` in Project name and click **Next.**
   c. Enter `labms` in Message Set Name, check **Use namespaces**, and click **Next.**
   d. Check **XML Wire Format Name**, set the wire format name to **XML** (from XML1), and click **Finish.**

   This creates a message set within a message set project. It also creates a file called messageset.mset, which opens in the main editor of the toolkit.

4. In the main editor, select **Properties** → **Physical properties** → **XML** and customize the wire format (Figure 8):
   a. In the Properties Hierarchy section, select **Physical Properties** → **XML**.
   b. Check **Suppress DOCTYPE** under XML document type settings. DTD declarations are not needed, as the system is schema rather than DTD-based.
   c. Set **Root Tag Name** to blank by deleting the default value MRM. Web services use SOAP messages, where the root tag name must be Envelope. Also, the system uses embedded messages. It is not sensible to have a hard-coded root tag name when embedded messages are used.
   d. Close the messageset.mset window and save the contents.

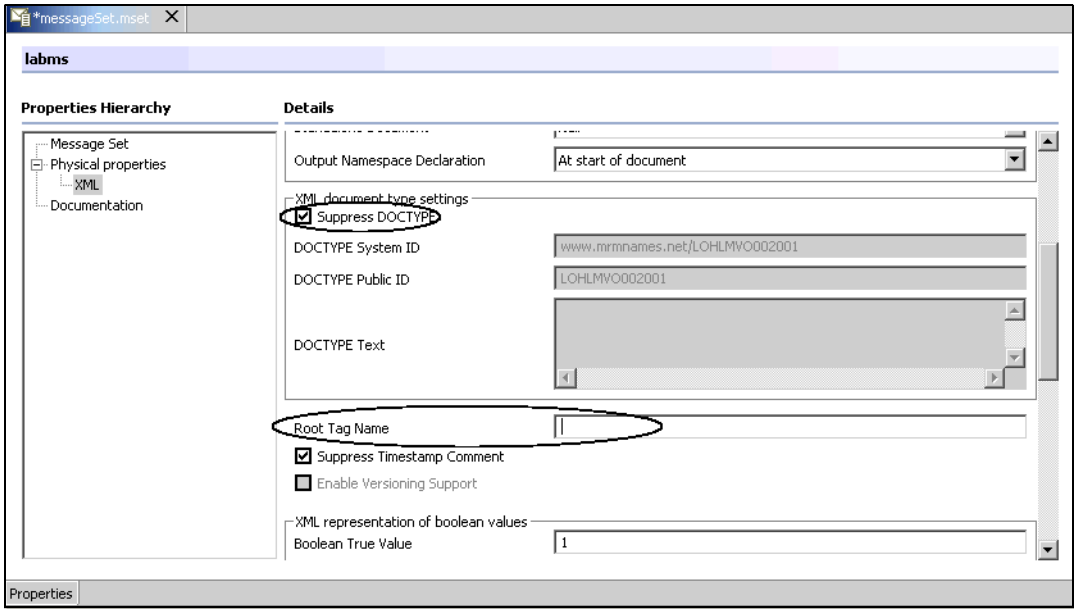

Figure 8 Message set properties

Create message definition files for SOAP messages

The SOAP V1.1 XML schema must be imported into the message set so that it can be used to model SOAP messages. This schema can be found at:

`http://www.w3.org/TR/2000/NOTE-SOAP-20000508/envelope-2000-04-18.xml`

You also need to import the XML schemas that define the SOAP Header and SOAP Bodies of the messages we will model.

Import the soap11.xsd, Configuration.xsd, Warehouse.xsd and Warehousesg.xsd files into message set project labmsp (see the additional material supplied with this redpaper for these files):

1. Select the **labmsp** project in the Resource Navigator.
2. Select **File** → **Import** from the menu.
3. Select **File system** and click **Next**.
4. Click the **Browse** button to the right of the Directory field and navigate to the directory where you downloaded the additional material supplied with this redpaper. Click **OK**.
5. Check **soap11.xsd**, **Configuration.xsd**, **Warehouse.xsd**, and **Warehousesg.xsd**. Click **Finish**.

Next, create an MDF based on the soap11.xsd file you just imported:

1. Select **File** → **New** → **Message Definition File**. This starts the new message definition file wizard.
2. In the window Select the message definition source, click **XML Schema file** and click **Next**.
3. In the window Select a schema file, select file **soap11.xsd** from project labmsp and click **Next.**
4. In the window Select a message set from the list, select **labms** in project labmsp and click **Next.**
5. In the window Select global elements from which to create messages, click **Envelope** and **Fault** and click **Finish.**

During this import, the global element Envelope is defined as a message. This means that message flows can nominate this message as the one to be processed. The global element Fault is also defined as a message. This is because Fault is a child of the element Body. In the following customization, the complex type of Body will be amended to be of composition message. This enables children of Body to be defined as messages.

Several warnings are generated in the task list as a result of importing this schema. These can be eliminated by measures described below, or they can be suppressed as shown in “Setting Message Broker Toolkit preferences” on page 31.

Before any further customization of the SOAP message definition file (MDF) occurs, you must create new message definitions based on the other XML schemas you have imported:

1. Select **File** → **New** → **Message Definition File**. This starts the new message definition file wizard.

2. In the window Select the message definition source, click **XML Schema file** and click **Next.**
3. In the window Select a schema file, select file **Warehouse.xsd** from project labmsp and click **Next.**
4. In the window Select a message set from the list, select **labms** in project labmsp and click **Next.**

> **Note:** Do not select ItemListElement to be a global element because it is not an immediate child of the SOAP Body as defined in the WSDL.

5. Click **Finish.**

You have already copied Warehousesg.xsd into message set project labmsp. This is an XML schema which contains elements referenced in the WSDL for the Web service but are missing from the Warehouse XML schema we have imported to represent the Web service. The elements are missing because the XML schema required amendment to support a document-literal Web service. Warehousesg.xsd contains global elements ShipGoods and ShipGoodsResponse.

Perform the following:

1. Select **File** → **New** → **Message Definition File**.
2. In the window Select the message definition source, click **XML Schema file** and click **Next**.
3. In the window Select a schema file, select file **Warehousesg.xsd** from project labmsp and click **Next.**
4. In the window Select a message set from the list, select **labms** in project labmsp and click **Next.**
5. In the window Select global elements from which to create messages, click both **ShipGoods** and **ShipGoodsResponse** and click **Finish.** These global elements describe the SOAP Body input and output messages that we use in our scenario.

In this example, you also need to import a schema that represents the SOAP header content. Perform the following:

1. Select **File** → **New** → **Message Definition File**. This starts the new message definition file wizard.
2. In the window Select the message definition source, click **XML Schema file** and click **Next**.

3. In the window Select a schema file, select file **Configuration.xsd** and click **Next**.
4. In the window Select a message set from the list, select **labms** in project labmsp and click **Next.**
5. Do not select any global elements.
6. Click **Finish.**

Now customize the SOAP message definition files:

1. Import the message definition files representing the Web services into the SOAP message definition file. This is the initial step in building a message hierarchy. At the top of this hierarchy is SOAP, and the Web services are subordinate.
   a. Open the SOAP message definition file **soap11.mxsd**.
   b. In the outline view, click **soap11.mxsd**.
   c. In the main editor pane, click the **Properties** tab.
   d. In the main editor, right-click **Imports** → **Add import** in Properties Hierarchy (Figure 9). A wizard to select a message definition file appears.

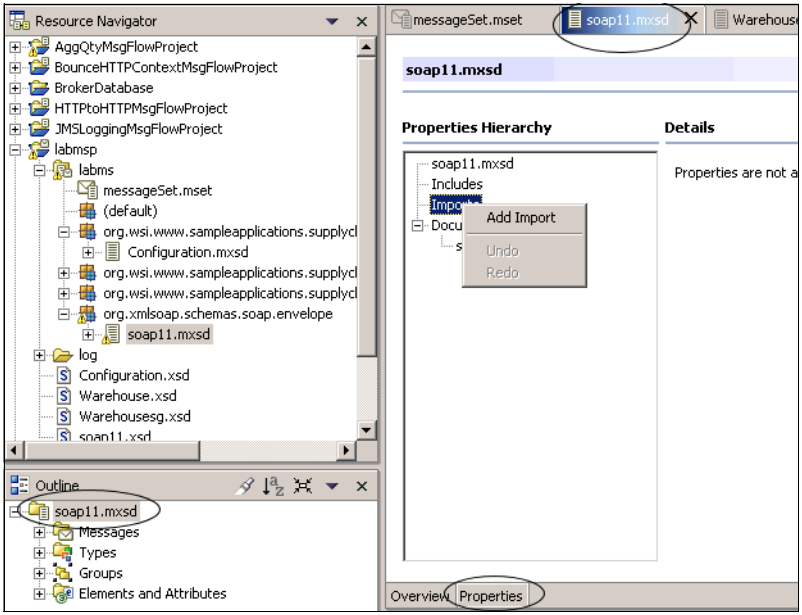

Figure 9 Add import

e. Select **Warehousesg.mxsd** from the hierarchy (**labmsp** → **labms** → **org** → **wsi** → **www** → **sampleapplications** → **supplychainmanagement** → **_200208** → **warehousewsdl**) as shown in Figure 10 and click **Finish**.

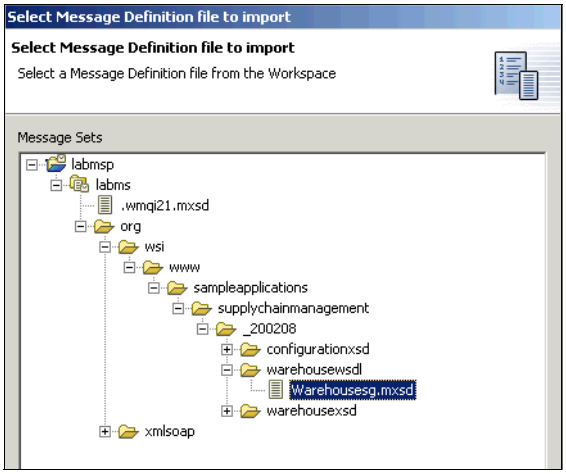

Figure 10 Selecting the message definition file

Note: Ensure that you select Warehousesg.mxsd, not Warehouse.mxsd. You do not have to import the Warehouse schema Warehouse.mxsd because we have not created messages based on global elements in it.

2. Repeat this process for the Configuration schema in the soap11.mxsd message definition file. In the main editor, right-click **Imports** → **Add import** in Properties Hierarchy. A wizard to select a message definition file appears. Select **Configuration.mxsd** from the hierarchy (**labmsp** → **labms** → **org** → **wsi** → **www** → **sampleapplications** → **supplychainmanagement** → **_200208** → **configurationxsd)** and click **Finish**.
3. As it is likely that a Broker may try to process the payload of a SOAP message as a message in its own right, change Body's complex type to Type Composition Message. This means that one and only one of Body's child elements must be present, and that each possible child must be defined as a message.
   a. In the Outline view, expand **Elements and Attributes** and select **Body**.
   b. In the main editor view, click the **Properties** tab.

c. In the main editor view, click the **Goto Type Definition** link at the top. You are presented with details of Body's complex type (Figure 11).

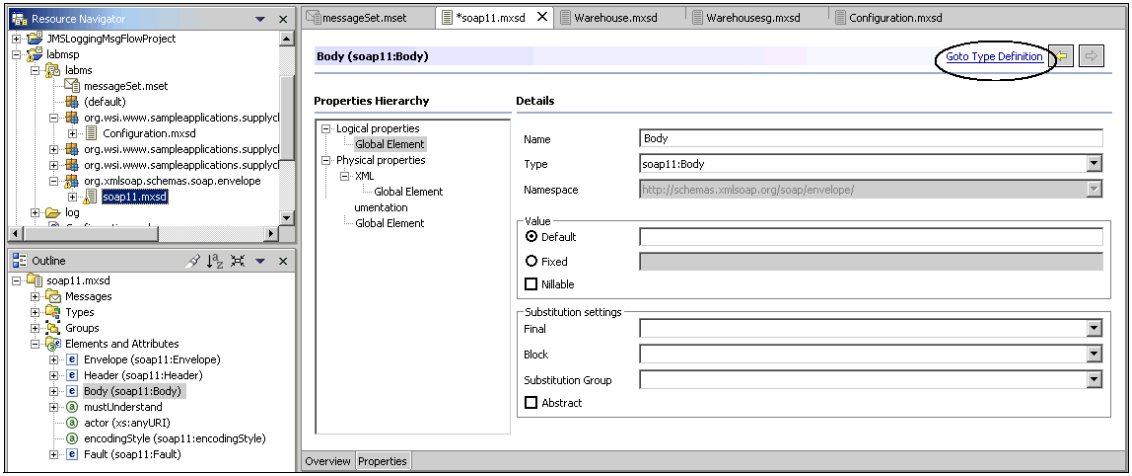

Figure 11 Customized the message body

d. Click the Composition pull-down menu and select **message** (changing it from sequence).

e. Note that the Content validation setting is **Closed.** This will affect later customization (Figure 12).

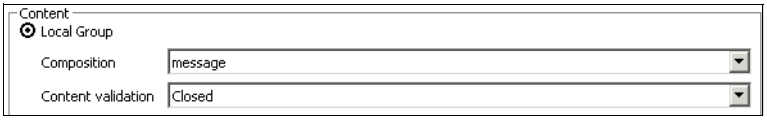

Figure 12 Body settings

f. Save soap11.mxsd by pressing Ctrl+S.

4. Note that there are now errors in the Task List, so correct them. A restriction of complex type Composition's value message is that wildcard elements and attributes are not allowed for elements of this type. So remove the Wildcard attribute and the Wildcard element from the Body element. To do this, perform the following:

   a. In the **Outline** view, expand **Elements and Attributes** and **Body**.

   b. Select the **Wildcard Attribute** and press Delete.

   c. Select the **Wildcard Element** and press Delete.

In our scenario, the SOAP Body may contain a ShipGoods message, a ShipGoodsResponse message, or a SOAP Fault. These must be added as children to the SOAP Body element because:

- The Content validation setting of Body's type is set to `Closed`. This means that only children explicitly defined to Body are allowed.
- Having explicitly defined children enables the message bodies to be used in the ESQL editor's Content Assist when developing message flows.

To add the elements as children to the SOAP Body element:

1. In the **Outline** view, expand **Elements and Attributes** and select **Body**.
2. In the main editor view, click the **Overview** tab.
3. In the main editor, right-click **Body** and select **Add Element Reference**.
4. From the pull-down menu, select **wh:ShipGoods**.
5. Repeat the above two steps, selecting **wh:ShipGoodsResponse** and **soap11:Fault** (Figure 13).

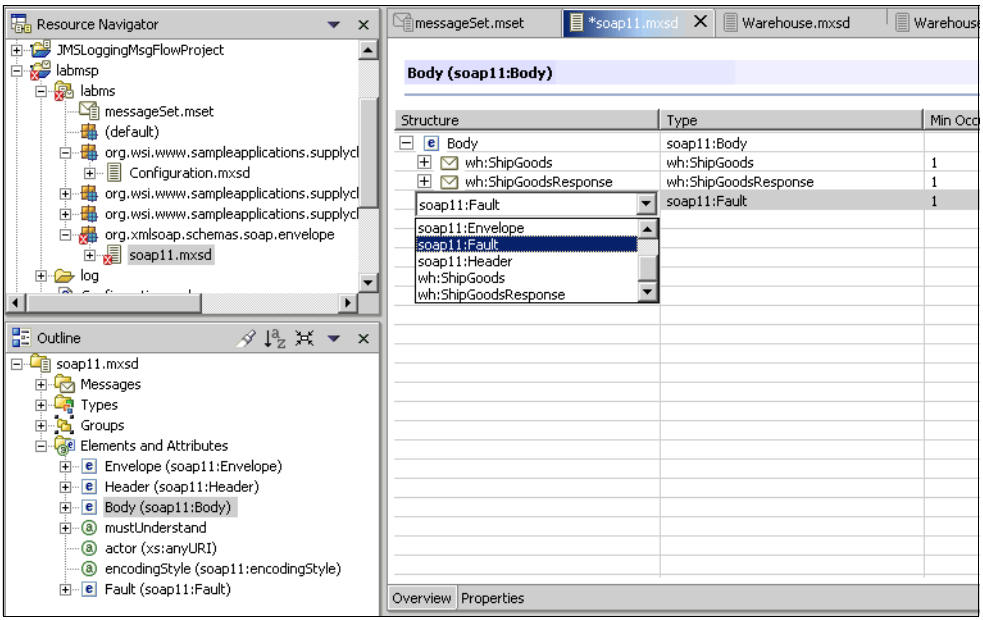

Figure 13 Add element references

Also in this system, the SOAP Header may contain a Configuration element. This must be added as a child to the SOAP Header element. To do this:

1. In the **Outline** view, expand **Elements and Attributes** and select **Header**.

2. In the main editor view, click the **Overview** tab.
3. In the main editor, right-click **Header** and select **Add Element Reference**.
4. From the pull-down menu, select **tns:Configuration**.
5. Save **soap11.mxsd**. Note that there are now no errors in the Task List.

This is all the mandatory customization needed for the message set. However, you will still have a number of task list warnings in your toolkit (unless you have suppressed these warnings in the Message Broker Toolkit). To get rid of these warnings:

1. Delete Wildcard Elements (but not the Wildcard Attributes) from elements **Envelope**, **Header**, and Fault **detail** (note detail is a child of the Fault element) under **Elements and Attributes** in the Outline view. Select the Wildcard Element in each case and press the delete key. In their place, we amend the complex type definitions. Doing this removes the Wildcard Element warnings generated by the Message Broker Toolkit. After completing this step you should have four warnings left in the task list.
2. Set the Content Validation of the Envelope complex type to **Open**, from its original setting of Closed. This means that any elements and attributes are allowed as children of Envelope. Envelope still has explicitly defined children Header and Body. This closely matches the business requirement for the SOAP Envelope while still satisfying the wildcard element removal. Do this as follows:
   a. Select **Envelope** under Elements and Attributes.
   b. Click the **Properties** tab in the main editing window.
   c. Click the **Goto Type Definition** link at the top.
   d. Change the Content Validation from Closed to **Open** using the pull-down menu.
3. Similarly set the Content Validation of the **Header** complex type and the **Fault.detail** (note that detail is a child of the Fault element) complex type to **Open**, from its original setting of Closed. This means that any elements and attributes are allowed as children of Header and detail. This closely matches the business requirement for the SOAP Header and Fault detail while still satisfying the wildcard element removal.
4. For each of the elements **Envelope**, **Header**, and Fault **detail** under Elements and Attributes in the Outline view, remove the namespaces from **Wildcard**

Attributes. This removes warnings generated by the Message Broker Toolkit. To do this:

a. Select the **Wildcard Attribute** in the Outline view.

b. Remove the entry in the Namespace field in the main editing view (Figure 14).

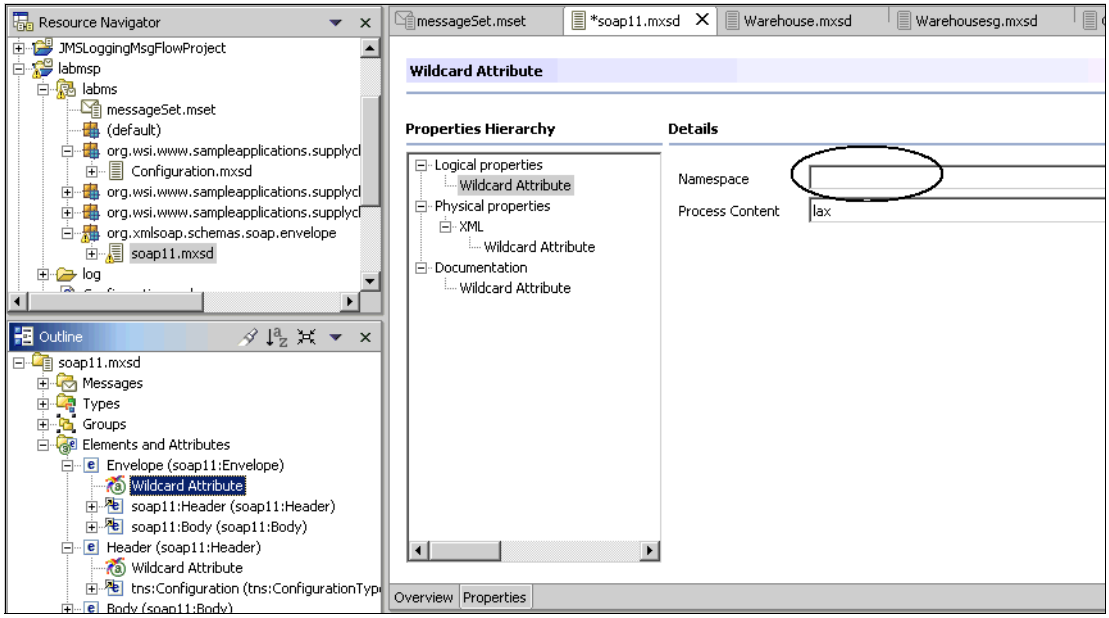

Figure 14 Wildcard attribute

You should now have only one warning left in the task list. To remove this:

1. Remove the pattern facet of the **mustUnderstand** global attribute as follows. It is not required and its deletion removes a warning generated by the Message Broker Toolkit.

   a. Select **mustUnderstand** in Outline view.

   b. Click the **Goto Type Definition** link in the main editing window and select **Value Constraints** in Property Hierarchies.

c. Select **0/1** in Patterns and click **Delete** (Figure 15).

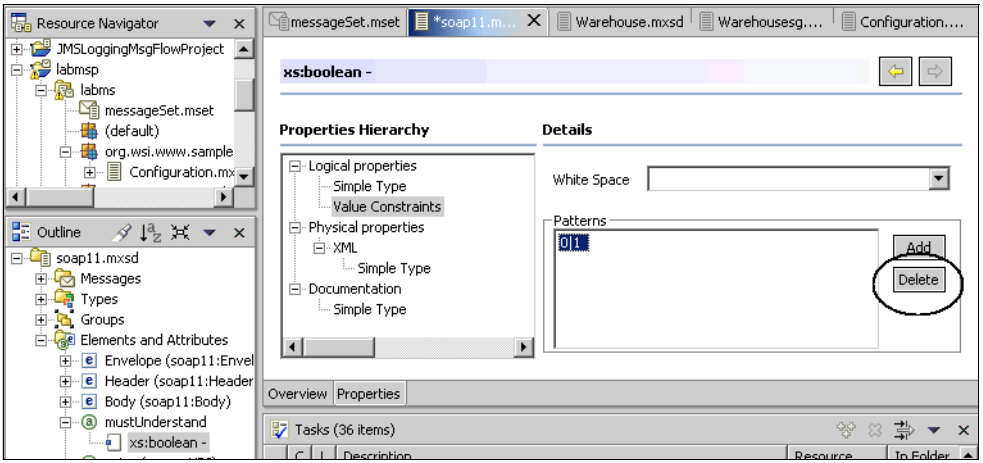

Figure 15 Remove pattern facet

2. Save your changes to soap11.mxsd by pressing Ctrl+S. You should not see any warnings or errors in the Tasks window.

SOAP message flow development

Here you will enhance the aggregation message flow built as part of the WebSphere Business Integration Message Broker scenario application in the redbook *Patterns: Implementing an SOA Using an Enterprise Service Bus,* SG24-6346, to use the message definitions you have just created.

Create a message flow project

Ensure you are in the Broker Application Development perspective, and perform the following:

1. Select **File** → **New** → **Message Flow Project**. (Make sure you create a Message Flow Project.)
2. Enter **labmfp** in project name and click **Next.**
3. In Referenced projects, click the message set project **labmsp** and click **Finish.**

This creates a message flow project with a link to the message set project you have just created. Create a broker schema for your message flow:

1. Select the message flow project **labmfp.**
2. Select **File** → **New** → **Broker Schema**.
3. Enter **labschema** into the Name and click **Finish**.

Create a message flow

The message flow has already been developed for a non–message set environment. Take the existing message flow and customize the ESQL so that it uses your new message set.

In the Broker Application Development perspective:

1. Copy files **AggQtyMsgFlow.msgflow** and **AggQtyMsgFlow.esql** from project AggQtyMsgFlowProject into the **labschema** subdirectory of Message Flow Project labmfp.
   a. Expand the **default** broker schema.
   b. Select **AggQtyMsgFlow.esql** and **AggQtyMsgFlow.msgflow**.
   c. Right-click and select **Copy.**
   d. Click on the new Broker Schema, **labschema.** Right click and select **Paste.**
   e. Copy the six Data XMI objects also in the AggQtyMsgFlowProject and paste them into the labschema. These Data XMI files describe the structure of the database tables referenced in the flows. (Select the top object, hold down the Shift key, and select the bottom object.)
2. Expand **labschema** in **lapmfp** and double-click **AggQtyMsgFlow.msgflow.** The message flow should be shown in the main editor**.**

Set node properties to use the message set

The message flow was originally developed to use XML messages without a message set. In WebSphere Business Integration Message Broker terms, this means using a domain (or parser) called **XMLNS**. As we are now using a message set, the parser name must be changed to **MRM.** We also need to identify to the message flow:

- The message set to be used. This is the message set name (**labms**).
- The message inside that message set. This is **Envelope.**
- The Wire Format. This is **XML.**

These values must be set on the HTTP node called **HTTPInput** in the message flow (Figure 16 on page 23):

1. Right-click node **HTTP Input** and select **Properties.**
2. First, notice that on the Basic panel, the URL Selector is set to /ShipGoods. WebSphere Business Integration Message Broker listens to HTTP URL requests ending with this selector and uses this to initiate this flow. In our

application, the Warehouse_Impl.wsdl definition in the Retailer service contains the following element to invoke this flow:

```
<wsdlsoap:address
location="http://appsrv1a.itso.ral.ibm.com:7080/ShipGoods"/>
```

3. Select **Default**.
4. Set Message Domain to **MRM.** This should be a value in the pull-down menu.
5. Set Message Set to **labms.** This should be a value in the pull-down menu.
6. Set Message Type to **Envelope.** There is no pull-down menu, so you must type this. Note that Message Type is case-sensitive.
7. Set Message Format to **XML.** This should be a value in the pull-down menu.

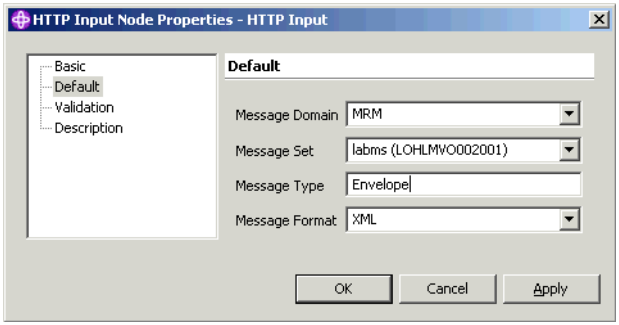

Figure 16 HTTP input node properties

We also want the Broker to validate that input messages conform to the XML schema definition (Figure 17 on page 24). To do this:

1. Click **Validation.**
2. Set Validate to **Content and Value.** This enables both content validation (type content and composition checks) and value validation (value data types checks, null permitted checks, length checks, range checks, enumeration checks, and so on).
3. Set Failure Action to **User Trace.** This writes all validation errors to usertrace and continues processing.
4. Set Timing to **Complete.** This validates the entire message, immediately overriding the partial parsing used by default in the MRM.
5. Click **OK.**

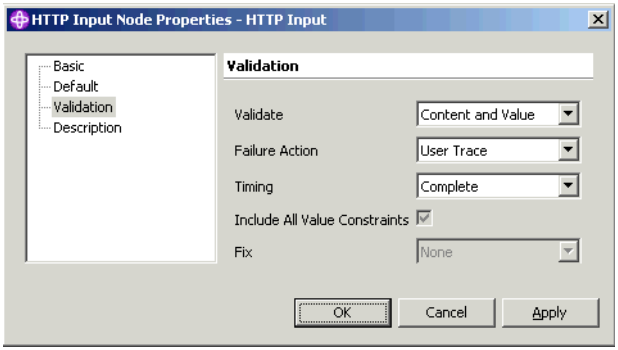

Figure 17 HTTP input node validation

These values also must be set on the MQInput node in the message flow. Repeat the two sets of bullets above for the node called **Replies**.

We also want the Broker to validate that output messages conform to the XML schema definition. To do this, we update the properties of each **Compute** node nearest to the output nodes. (The output nodes are the Aggregation Request nodes and the HTTPReply node.) These are:

- Whse A Request
- Whse B Request
- Whse C Request
- Consolidate Response

For each of these nodes, right-click **Properties**:

1. Click **Validation**.
2. Set Validate to **Content and Value**.
3. Set Failure Action to **User Trace**.
4. Click **OK**.

Amend ESQL to use MRM parser

We now amend the ESQL to use the new MRM parser. Every statement that currently contains XMLNS potentially must be changed.

Right-click the Compute node **Whse A Request** and select **Open ESQL. AggQtyMsgFlow.esql** opens and the ESQL for this node is highlighted.

The ESQL editor contains a feature called Content Assist. This is where the editor is sensitive to the position of the cursor and can make suggestions for what should be input. Content Assist can be invoked by pressing Ctrl+Spacebar (or by selecting **Edit** → **Content Assist** from the menu).

For example, the statement below is from the **AggQtyMsgFlow_WhseAReq** module in **AggQtyMsgFlow.esql**:

```
DECLARE CRD INTEGER
CARDINALITY(InputRoot.XMLNS.SoapNS:Envelope.SoapNS:Body.*:ShipGoods.*:ItemL
ist.*:Item[]);
```

If you place the cursor after `InputRoot.` and press Ctrl+Spacebar, a list of keywords to follow InputRoot is displayed.

To demonstrate this:

1. Copy the statement above into the editor and paste it directly below the original.
2. Comment out the first statement (by placing two dashes at the beginning of the line).
3. Erase all of the second statement after `InputRoot.`

You should see:

```
--DECLARE CRD INTEGER
CARDINALITY(InputRoot.XMLNS.SoapNS:Envelope.SoapNS:Body.*:ShipGoods.*:ItemL
ist.*:Item[]);
DECLARE CRD INTEGER CARDINALITY(InputRoot.
```

Note that two dashes signify a comment for the remainder of the line. The text above is wrapped onto two lines only because of the length of the statement and the page width in this document.

1. Place the cursor after `InputRoot.` and press Ctrl+Spacebar. A list of keywords is displayed. In this case, they are the parsers available in WebSphere Business Integration Message Broker. Select **MRM** (the WebSphere Business Integration Message Broker parser that uses message sets). The statement now reads:

   ```
   DECLARE CRD INTEGER CARDINALITY(InputRoot.MRM
   ```

2. Enter a dot after MRM and press Ctrl+Spacebar again. This time you are presented with a list of global elements from the known message sets. Known message sets are from those message set projects that have been defined as referenced projects to this message flow project. You previously defined **labmsp** as a referenced project.
3. Double-click **SoapNS:Body**. When using message sets, you do not qualify the top-most part of the message in ESQL; instead, it is identified by the Message Type property. On input, you specified the Message Type property to be **Envelope**, so **Envelope** is not needed in this ESQL statement. Note

that if you are not using message sets, you have to code Envelope in the ESQL. The statement now reads:

```
DECLARE CRD INTEGER CARDINALITY(InputRoot.MRM.SoapNS:Body
```

4. Enter a dot after Body and press Ctrl+Spacebar again. A list of elements defined as children of Body are defined. You defined these previously. Fault has a namespace prefix called SoapNS, but ShipGoods and ShipGoodsResponse have a fully qualified namespace.
5. Double-click **ShipGoods**. The statement now reads:

```
DECLARE CRD INTEGER CARDINALITY(InputRoot.MRM.SoapNS:Body.wh:ShipGoods
```

6. Note that the fully qualified namespace is not present in the ESQL; instead there is a prefix `wh:`. Scroll to the top of the ESQL file to see that this statement has been added:

```
DECLARE wh NAMESPACE
'http://www.ws-i.org/SampleApplications/SupplyChainManagement/2002-08/Wa
rehouse.wsdl';
```

7. Now that this namespace declaration is present, you should see this rather than the fully qualified namespace in any future Content Assist executions that reference this namespace.
8. Enter a dot after ShipGoods and press Ctrl+Spacebar. A list of child elements of ShipGoods is displayed. Double-click **ItemList**. Note that this has no namespace prefix. This is because the XML schema specified that the `form` for this element is `unqualified`**.**
9. Enter a dot after ItemList and press Ctrl+Spacebar. A list of child elements of ItemList is displayed. Double-click **Item**. Note that this has a different namespace to ShipGoods. You may see a fully qualified namespace or you may see a prefix, depending on whether this namespace has been used before.
10. Complete the statement so that it reflects the commented statement directly above it. Add `[]);` at the end of the statement. The completed statement should read as:

```
DECLARE CRD INTEGER
CARDINALITY(InputRoot.MRM.SoapNS:Body.wh:ShipGoods.wh:ItemList.wh1:Item[
]);
```

(`wh1` is the namespace prefix associated with the Item element.)

11. Amend every statement in this module that contains **XMLNS** in its message path. (The statements are highlighted with a comment beginning: `-- LAB:`.) In each case, XMLNS should be changed to **MRM**, and **Envelope**, if present, should be removed. Then use Content Assist to generate the rest of the statement. It is recommended that you duplicate each such statement and

comment out one copy, so that you can see the field names that are required. For example:

```
--SET
OutputRoot.XMLNS.SoapNS:Envelope.SoapNS:Body.*:ShipGoods.*:ItemList.*:It
em[J].*:ProductNumber = NULL;
SET
OutputRoot.MRM.SoapNS:Body.wh:ShipGoods.ItemList.wh1:Item[J].wh1:Product
Number = NULL;
```

12. You should have amended five statements in AggQtyMsgFlow_WhseAReq.
13. Repeat for the **AggQtyMsgFlow_WhseBReq** and **AggQtyMsgFlow_WhseCReq** modules. Note that you can copy and paste the statements you have just updated (but not the entire Compute Module) from the ESQL for **AggQtyMsgFlow_WhseAReq**.
14. Module **AggQtyMsgFlow_Consolidate** is more complex, but the principle is the same. When messages are input to an Aggregate Reply node, their message tree changes from the form `InputRoot.XMLNS...` to:

    ```
    InputRoot.ComIbmAggregateReplyBody.WhseB.XMLNS...
    ```

    Content Assist does not currently recognize the Aggregate message tree (parser ComIbmAggregateReplyBody), so these statements may require more manual coding. However, you coded the lower message hierarchy in earlier modules, so you can paste the lower hierarchy of these statements from these previous modules.

    Here is an example of the required coding in this module. The original code is commented out:

    ```
    --SET OutputRoot.XMLNS.*:Envelope =
    InputRoot.ComIbmAggregateReplyBody.WhseA.XMLNS.*:Envelope;
    SET OutputRoot.MRM = InputRoot.ComIbmAggregateReplyBody.WhseA.MRM;
    ```

15. Because you are using message sets, ESQL to create a namespace in the output message is not required. For example, in module **AggQtyMsgFlow_Consolidate,** the following statements are not needed, so should be deleted or commented out:

    ```
    --CREATE FIELD
    OutputRoot.XMLNS.*:Envelope.*:Body.*:ShipGoodsResponse.*:Response.*:Item
    Status[C] TYPE Name NAMESPACE ns2;
    --SET
    OutputRoot.XMLNS.*:Envelope.*:Body.*:ShipGoodsResponse.*:Response.*:Item
    Status[C].(XML.NamespaceDecl)xmlns = ns2;
    --CREATE LASTCHILD …
    ```

    The first `CREATE FIELD` statement should also be removed or commented out.

16. Module **AggQtyMsgFlow_Consolidate** builds output messages from scratch (rather than copy the entire input message), so needs to set output properties

relating to the message set. Add the following statements at the beginning of this module:

```
SET OutputRoot.Properties.MessageSet = 'labms';
SET OutputRoot.Properties.MessageType = 'Envelope';
SET OutputRoot.Properties.MessageFormat = 'XML';
```

17. Module SetHTTPcontext must be amended to enable the HTTP context to be retained throughout the message flow. Currently this is achieved with this statement:

    ```
    SET OutputRoot.XMLNS.HTTPcontext = Environment.Variables.context;
    ```

    Change it to:

    ```
    SET OutputRoot.MRM.soap11:Header.HTTPcontext =
    Environment.Variables.context;
    ```

    (assuming the namespace prefix for SOAP is `soap11`).

18. Comment out the CREATE NEXTSIBLING statement that precedes it.
19. Finally, check that there are no remaining instances of XMLNS in the entire ESQL module file.
20. Save the changes you have made to the ESQL and Message Flow files by pressing Ctrl+S.

Note:

- If you use Usertrace to trace the message flow execution, you will see BIP5374 warnings. This is because the validation of HTTPcontext fails as it is not part of the message set. This is used to pass the requester's HTTP context from the first part of the aggregation message flow to the second part via WebSphere MQ. It is internal to the ESB and not seen by service requesters or service providers.
- Projects can choose to use message sets for development only. This enables developers to gain the advantage of using ESQL Content Assist in compute nodes and mapping nodes. However, it avoids incurring runtime overhead. To do this, set the message set runtime parser to XMLNS, not MRM. You must also develop ESQL using XMLNS rather than MRM, so the decision must be made on the project before development begins.

Removing Message Broker Toolkit warning tasks

In creating the message set and message flow, you have worked through the warnings that have been generated in the Tasks view of the Message Broker Toolkit. This section describes two approaches to removing these warnings.

Importing database table definitions

First, we look at removing the warning task messages associated with the ESQL references to database tables in message flow projects. This can be done by importing table definitions from the underlying database into the Message Broker Toolkit using these steps:

1. From the Message Broker Toolkit menu, open the Data perspective by selecting **Window** → **Open Perspective** → **Other** and selecting **Data**.
2. From the menu select **File** → **New** → **Project**. Select **Simple** → **Project** from the dialog window and click **Next**. Enter `BrokerDb` as the project name and click **Finish**.
3. In the DB Servers window, right-click and select **New Connection**. Enter the connection name (`BrokerDb`), the database password, and other properties as shown in Figure 18. Click **Finish**.

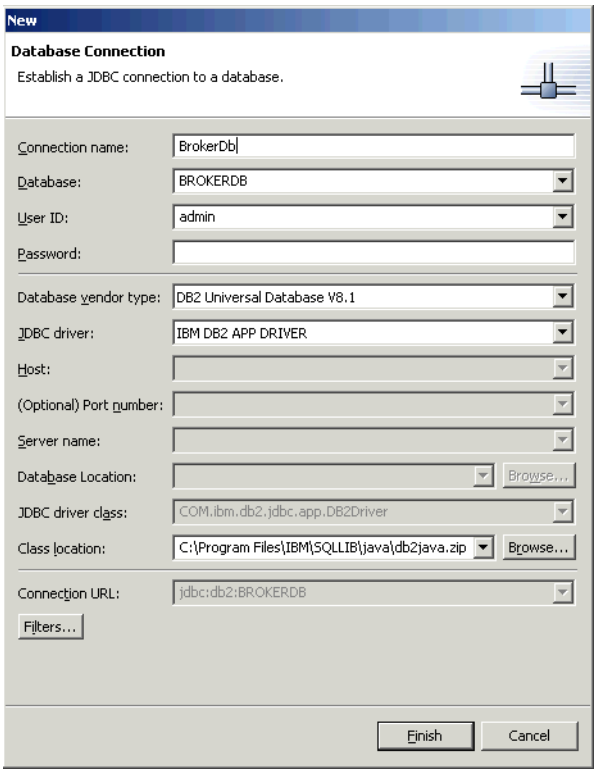

Figure 18 Setting the database connection properties

4. Expand the structure in the DB Servers panel to see the tables used by the message flows as shown in Figure 19.

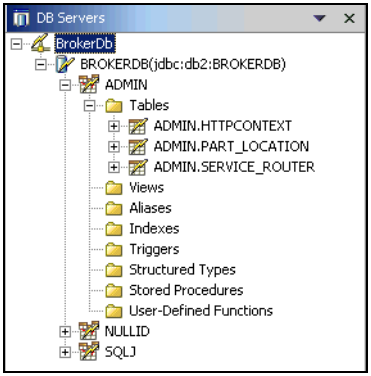

Figure 19 The ADMIN schema for connection BrokerDb in the Message Broker Toolkit

5. Perform these steps to import the table definitions into the BrokerDb project:

   a. Right-click the table **ADMIN.HTTPCONTEXT** in the DB Server panel and select **Import to Folder**. Use the Browse button to select the **BrokerDb** folder. Click **Finish**. Click **Yes** to confirm the creation of the entry for database BROKERDB and then **Yes** again to confirm the creation of schema ADMIN.

   b. Right-click the table **ADMIN.PART_LOCATION** in the DB Server panel and select **Import to Folder**. Use the Browse button to select the **BrokerDb** folder. Click **Finish**.

   c. Right-click the table **ADMIN.SERVICE_ROUTER** in the DB Server panel and select **Import to Folder**. Use the Browse button to select the **BrokerDb** folder. Click **Finish**.

   Figure 20 shows how the Navigator panel should look.

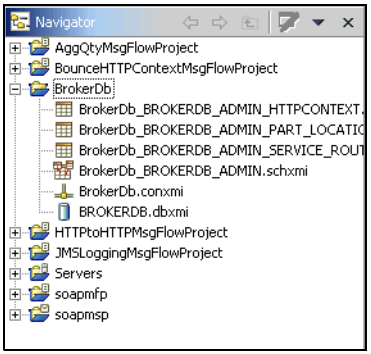

Figure 20 The imported ADMIN schema in the Message Broker Toolkit

6. Switch to the Broker Application Development Perspective by selecting **Window** → **Open Perspective** → **Broker Application Development**.
7. Select the six artifacts in the BrokerDb project and select **Edit** → **Copy**. Right-click a message flow project (such as labmfp) that references the database tables and select **Paste**. Right-click the message flow project and select **Rebuild Project**.

Setting Message Broker Toolkit preferences

Next, we look at suppressing warning tasks more generally in the Tasks view of the Message Broker Toolkit. This can be done by changing the ESQL validation preferences:

1. Select **Window** → **Preferences** from the menu.
2. Expand **ESQL and Mapping** and select **Validation**.
3. Change the values for the following settings from Warning to **Ignore** as shown in Figure 21.
   - Message references mismatch message definition

     This will suppress the warnings associated with the message set development in “Model a SOAP message in a message set” on page 10.
   - Database reference mismatch database schema

     This will suppress the warnings generated by the references to the underlying DB2® tables used in the ESQL in “SOAP message flow development” on page 21.

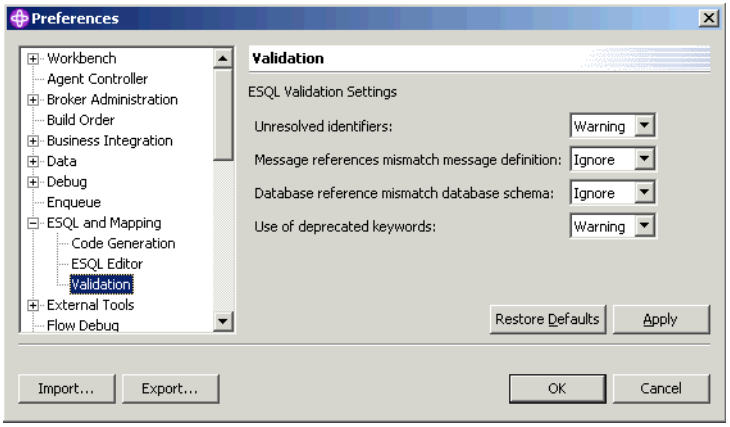

Figure 21 Setting Message Broker Toolkit preferences

4. Click **OK** to confirm your changes and close the Preferences window.

Runtime guidelines

This section describes how to deploy and test the scenario in this paper. We show how to deploy the message set and message flow to WebSphere Business Integration Message Broker, how to deploy the J2EE enterprise applications to WebSphere Application Server, and how to test the scenario.

Deploy the message set and message flow

Deploy the **labmsp** message set and **labmfp** message flow projects.

Important: If you have already deployed the original AggQtyMsgFlow to the broker, you must first delete it from both the Mediations.bar file and the execution group in your broker before proceeding with the deployment instructions in this section.

1. Open the Broker Administration perspective by selecting **Window** → **Open Perspective** → **Broker Administration** from the menu.
2. Open the **Mediations.bar** file under **Broker Archives** → **Servers** by double-clicking on it.
3. Click the **Add** icon button in the main editing window (circled in Figure 22).

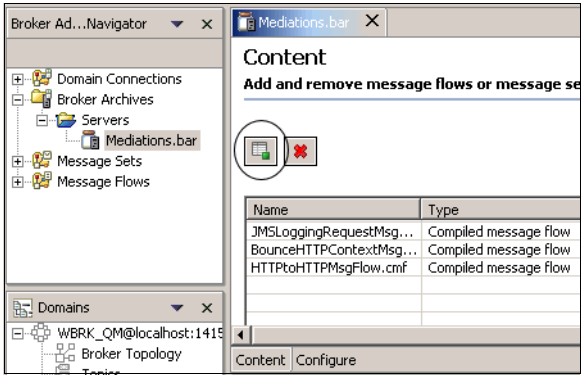

Figure 22 Add flow to bar file

4. Check the **labmsp** message set project and **labmfp** message flow project and click **OK** to add them to the Mediations.bar file. Click **OK** in the response dialog window.
5. Save the Mediations.bar file by pressing Ctrl+S.

6. In the Domains view, connect to the Broker by right-clicking and selecting **Connect** (Figure 23).

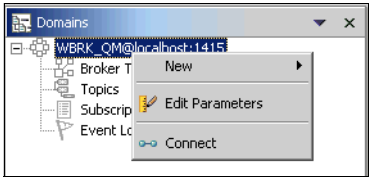

Figure 23 Connect to Broker

7. Deploy the **Mediations.bar** file by dragging it from the Broker Administration Navigator view to the execution group ESBExGp2 in broker WBRK_BROKER in the Domains view (Figure 24).

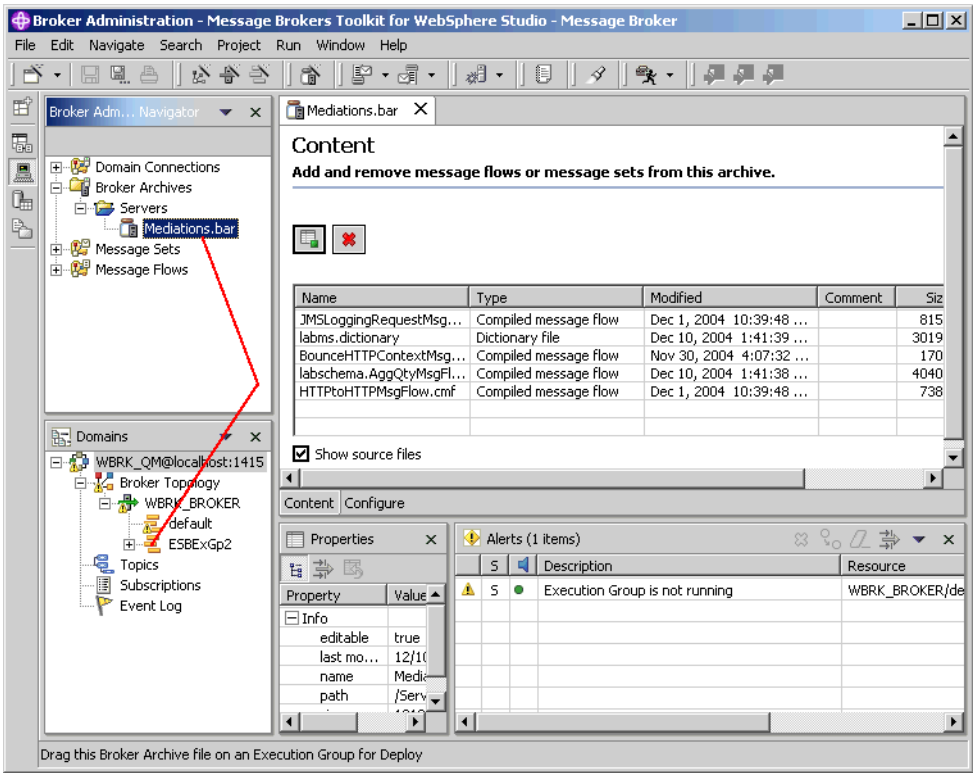

Figure 24 Deployment of Mediations.bar

8. Click **OK** to acknowledge the response message from the Configuration Manager.

9. Double-click the **Event log** in the Domains view, and watch for the two successful messages with a current timestamp.

The message set and message flow are now deployed and ready for use.

Install and start the J2EE sample application

The Web service enterprise applications must be installed into WebSphere Application Server V5.1.1 or higher, and then started.

Install WebSphere Application Server and configure a server. Then perform these steps:

1. Start WebSphere Application Server.
2. Launch a Web browser and navigate to the WebSphere Application Server Admin console at:

   ```
   http://localhost:9090/admin
   ```

3. The user ID is not validated, so enter anything you like (for example, `sa`) for the user ID or leave it blank, and click **OK**.
4. Click **Applications** → **Install New Application**. Use the installation wizard to install the following EAR files (these files are supplied in the additional material accompanying this Redpaper):
   - SCMSampleUIMB
   - RetailerMB
   - LoggingFacilityMB
   - WarehouseMB
   - WarehouseBMB
   - WarehouseCMB
   - ManufacturerMB
   - ManufacturerBMB
   - ManufacturerCMB

5. Start each of these enterprise applications. These applications should now appear with a status of Started in the **Applications** → **Enterprise Applications** window (Figure 25).

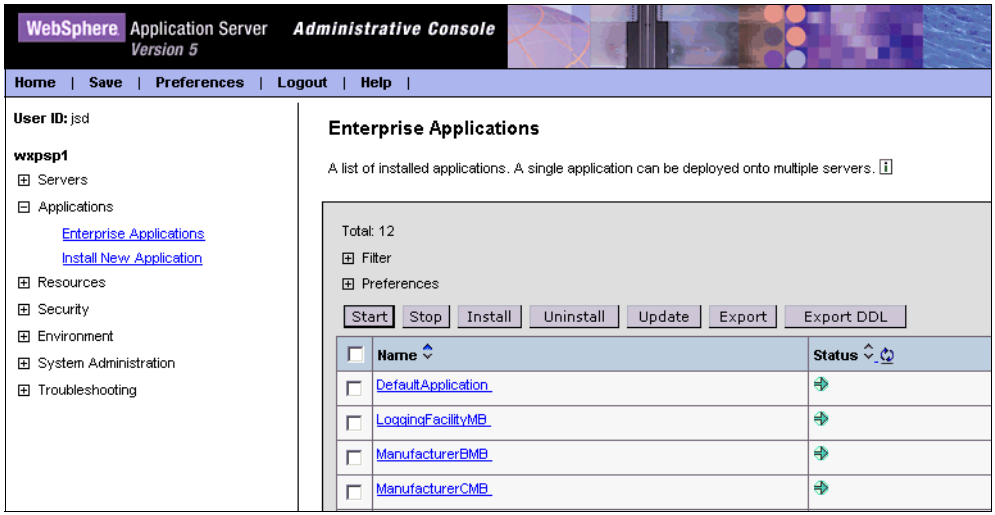

Figure 25 Started enterprise applications

Note: Refer to Appendix B and Section 9.4 of the redbook *Patterns: Implementing an SOA Using an Enterprise Service Bus*, SG24-6346, if necessary for further details.

Run the application

Start the Supply Chain Management sample application by entering in a Web browser:

```
http://localhost:9080/SCMSample/
```

This should start the Supply Chain Management sample application, as shown in Figure 26 on page 36.

Figure 26 Supply Chain Management sample application

The following steps describe how to use the SCM sample application:

1. To retrieve a list of products, click **Place New Order**. This displays a list of 10 products, as shown in Figure 27.

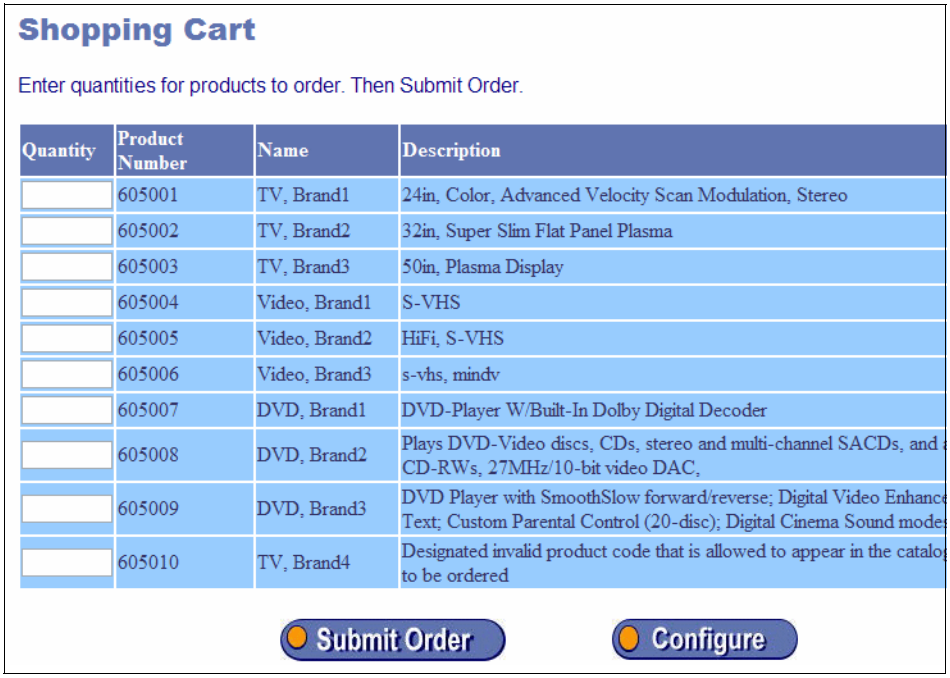

Figure 27 Shopping cart screen

2. Each warehouse stocks only certain products. Warehouse A stocks products 605001, 605004, and 605007; Warehouse B stocks products 605002, 605005, and 605008; and Warehouse C stocks products 605003, 605006, and 605009.
3. Test that each warehouse is working correctly:
   a. Enter 1 as the quantity for 605001 and select **Submit Order**. This places an order for Warehouse A. The order status window shown in Figure 28 should be displayed.

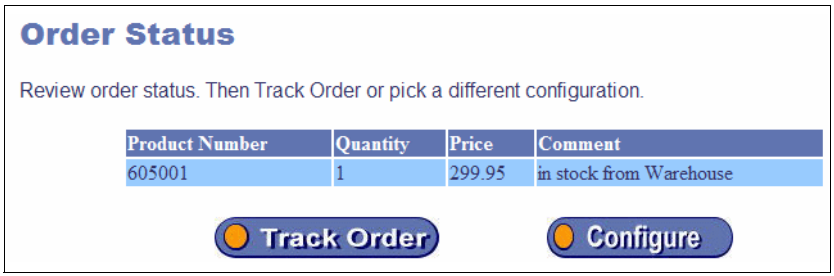

Figure 28 Order status

   b. Click **Track Order**. The order status screen shows which orders were placed and which were not placed due to insufficient stock. Track Order shows entries that were written to the LoggingFacility. As new entries are added to the Logging Facility, you must refresh this window by clicking **Order Status** and then clicking **Track Order** again.

      You should see the window shown in Figure 29.

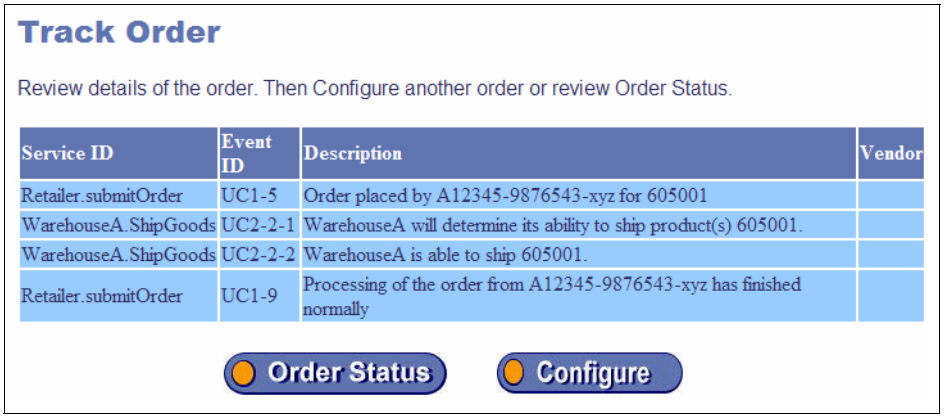

Figure 29 Track order

c. Repeat this test for Warehouse B and Warehouse C by ordering 1 unit each of 605002 and 605003.

4. Confirm that the aggregation is working correctly. Order 1 unit of 605001, 605002, and 605003 in the same order. Review the Order Status and Track Order screens.

5. Try ordering multiple quantities of each product. If the warehouse has sufficient stock for the product, an order will be placed. If the placement of the order causes the warehouse's stock level of that product to drop below a certain threshold, a reorder request is sent to the manufacturer of the product.

   If the current stock level falls below the minimum stock level, the stock is reordered so that, after the reorder has arrived, the stock will be at maximum stock level. For example, you order six items of 605001. This reduces the current stock level to 4 (10 - 6). A reorder will be made for 21 new items.

   Place orders for multiple products in the SCM sample application by entering quantities and clicking **Submit Order**. For example, order three items of product 605001 and six items of product 605002. This triggers a reorder of product 605002 with Manufacturer B.

6. The order status screen shows which orders were placed and which orders were not placed due to insufficient stock. Click **Track Order** to see the entries that were written to the LoggingFacility. As new entries are added to the Logging Facility, you must refresh this screen by clicking **Order Status** and then clicking **Track Order** again. Figure 30 on page 39 shows the results of an order in which products 605001 and 605002 were shipped and a reorder for 19 units of product 605002 was placed with Manufacturer B.

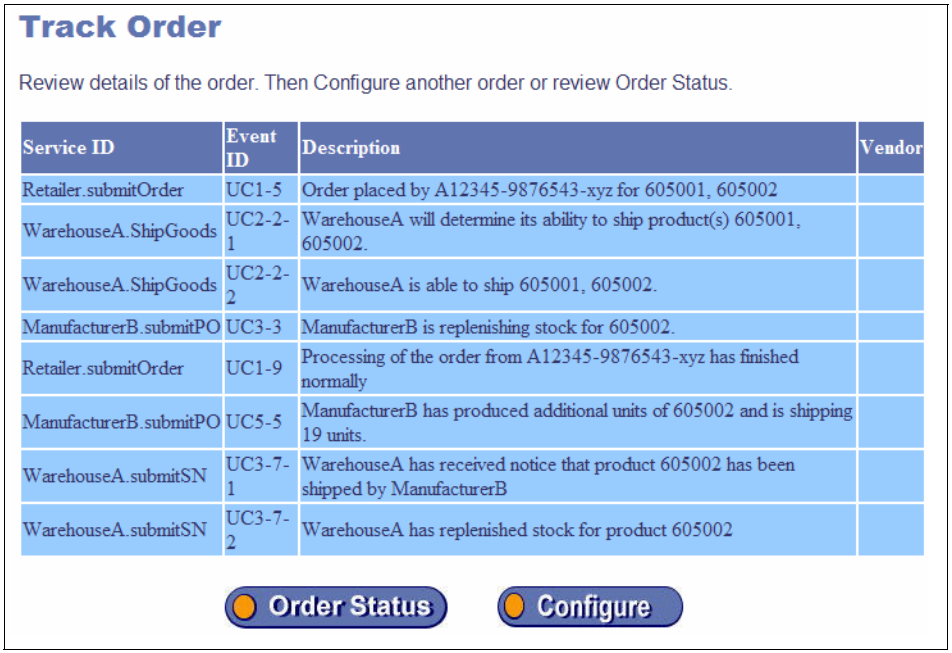

Figure 30 Track order for an order fulfilled by Manufacturer B

7. To start a new order, click **Configure**. At this point, all state is lost, and the warehouse stock levels return to their default values.

Additional material

The Web material associated with this Redpaper is available in softcopy on the Internet from the IBM Redbooks™ Web server. Point your Web browser to:

ftp://www.redbooks.ibm.com/redbooks/REDP3978

Alternatively, you can go to the IBM Redbooks Web site at:

ibm.com/redbooks

Select **Additional materials** and open the directory that corresponds with the Redpaper form number, REDP3978.

Using the Web material

The additional Web material that accompanies this Redpaper includes the following file:

File name	*Description*
REDP3978.zip	Zipped EAR files

System requirements for downloading the Web material

The following system configuration is recommended:

- Microsoft® Windows® 2000 or XP
- WebSphere Business Integration Message Broker V5

How to use the Web material

To extract the files, right-click the zip file and select **Extract all**. This extracts the files into the following two folders:

Redpaper Materials	Contains code snippets and XSD files required to complete the scenario.
SG24-6346 Broker EARs	Contains EAR files taken from the redbook SG24-6346. These EAR files are required for the scenario.

The team that wrote this Redpaper

This Redpaper was produced by a team of specialists working at the IBM Hursley Laboratory in the United Kingdom with help from others around the world.

Chris Nott is a Consulting IT Specialist with IBM Software Group in the UK. He has 14 years of IT experience in software development, technical pre-sales consulting, and solution architecture. His current area of expertise is business integration and he has a strong background in relational systems design and database technology. He has written extensively about business integration and the Enterprise Service Bus. He holds a BSc degree in Mathematics from the University of Durham and is registered in the UK as a Chartered Engineer.

Peter Edwards has worked in IT for nearly 30 years, in industries as diverse as steelmaking, telecommunications, and banking, before joining IBM in 1995. Currently he is based at the UK laboratories in Hursley, working in the Solution Test Department. Here, he ensures that products such as WebSphere Business Integration Message Broker integrate well with other IBM products.

Andrew Humphreys is a Senior IT Specialist working for Software Group Lab Services concentrating on helping customers design service-oriented architectures and implement Enterprise Service Buses. He is recognized by IBM customers, the IBM services community, and the Hursley development laboratory as a worldwide expert on WebSphere Business Integration Message Broker, and he has extensive experience in architecting ESB-style solutions. He has used these skills at customer sites around the world working as a consultant providing expertise in solution architecture and design application development, migration services, performance evaluation, problem determination, project specification, and sizing. His customer-facing work has led to an understanding of the problems customers face and the requirements they have for implementing ESBs.

Martin Keen is an Advisory IT Specialist at the International Technical Support Organization, Raleigh Center. He writes extensively about WebSphere products and Patterns for e-business. He also teaches IBM classes worldwide about WebSphere and business process management. Before joining the ITSO, Martin worked in the EMEA WebSphere Lab Services team in Hursley, UK. Martin holds a bachelor's degree in Computer Studies from Southampton Institute of Higher Education.

Thanks to the following people for their contributions to this project:

Matthew Courtois, Customer Solutions Development
IBM Software Group

Stephen Cox, WebSphere MQ Software Services
IBM Software Group

Kirk Davis, Customer Solutions Development
IBM Software Group

David Hardcastle, Consulting IT Specialist
WebSphere Technical Sales

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

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 and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:
This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

This document created or updated on April 13, 2005.

Send us your comments in one of the following ways:

- Use the online **Contact us** review redbook form found at:
  ibm.com/redbooks
- Send your comments in an e-mail to:
  redbook@us.ibm.com
- Mail your comments to:
  IBM Corporation, International Technical Support Organization
  Dept. HYJ Building 662, P.O. Box 12195
  Research Triangle Park, NC 27709-2195 U.S.A.

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

DB2®	ibm.com®	SupportPac™
developerWorks®	Redbooks™	WebSphere®
IBM®	Redbooks (logo) ™	

The following terms are trademarks of other companies:

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

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

Intel, Intel Inside (logos), MMX, and Pentium are trademarks of Intel Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

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

Other company, product, and service names may be trademarks or service marks of others.