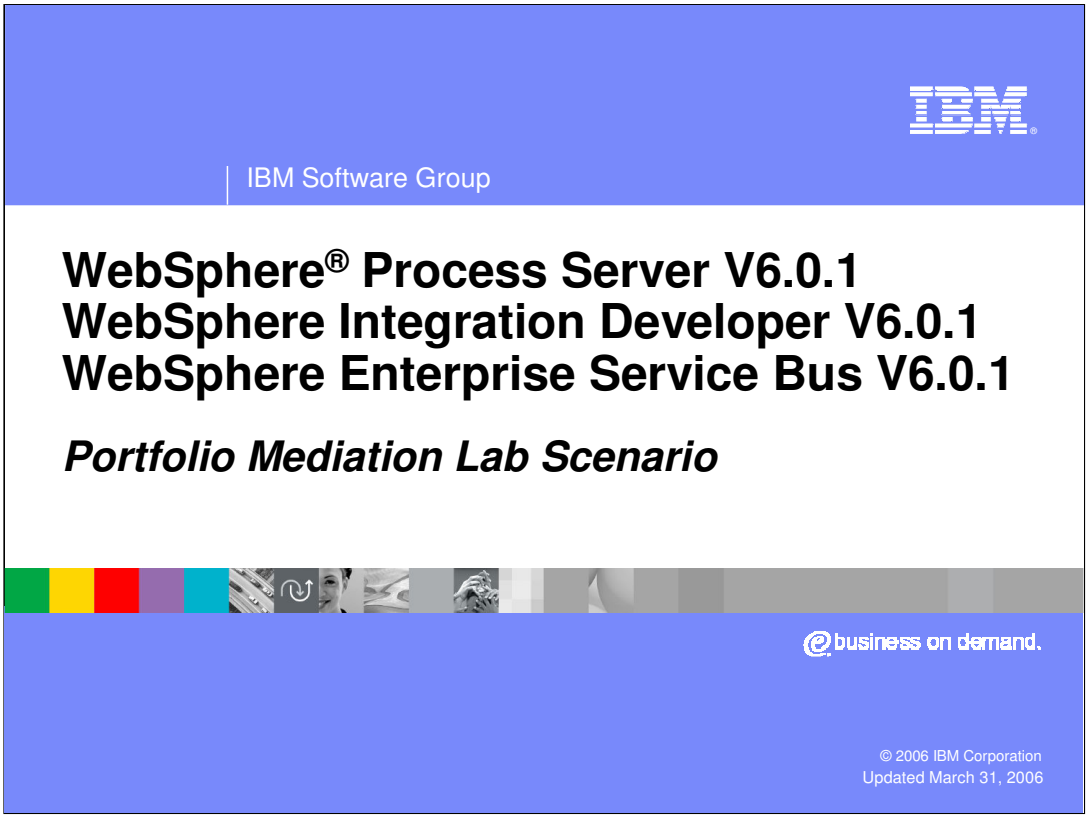


This presentation examines the Portfolio Mediation Scenario that is used by the lab exercises.

Goals

- Understand the Portfolio Mediation Scenario
  - Describe the original application prior to the mediation
  - Discuss reasons for needing a mediation
  - Describe the application with the mediation
  - Look at the flow of the mediation
- Introduce the lab exercises
  - Describe the scope of each of the labs
  - Explain the built-in test data

The goal of the presentation is to first understand the portfolio mediation scenario and then to describe the lab exercises that are based on it.

An understanding of the scenario is built by first describing the original application prior to the introduction of a mediation. The business reasons for adding a mediation are then discussed. The application containing the mediation is then examined along with the contents of the mediation itself.

Having built an understanding of the scenario, the four lab exercises built on the scenario are described.

The test data is also outlined to help you understand the expected behaviour when running the application.

Section

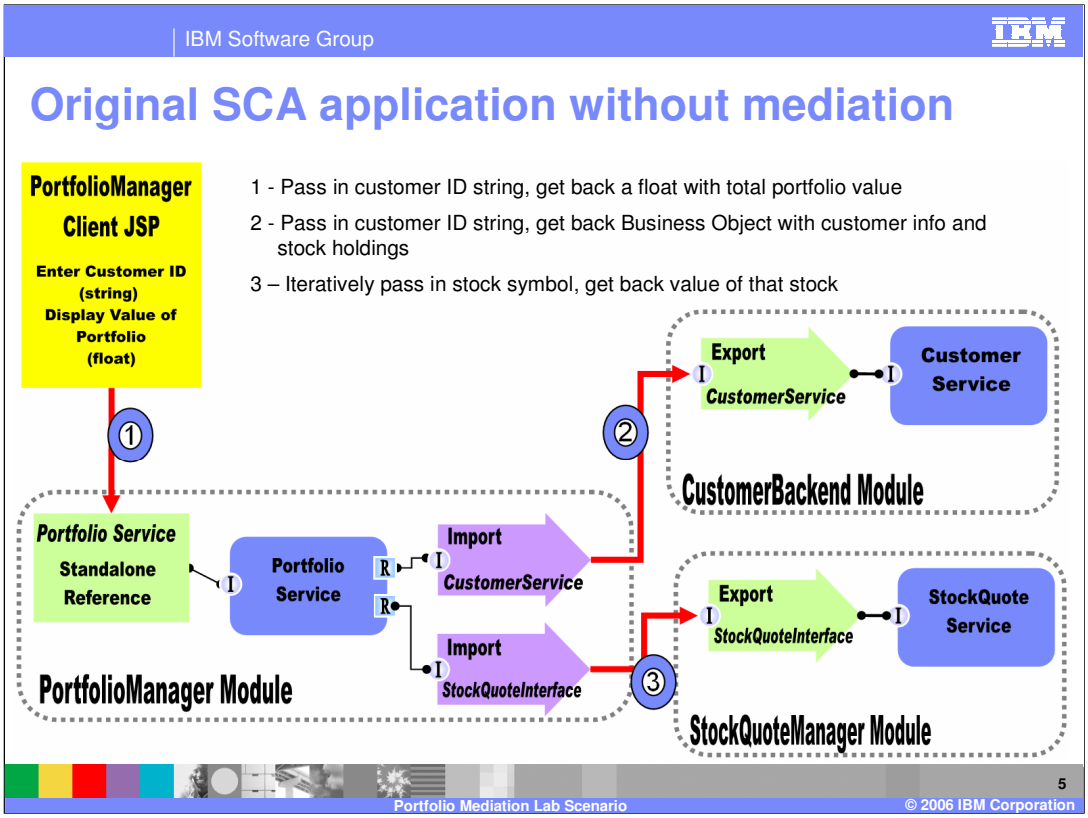
Portfolio mediation scenario description

In this section the Portfolio Mediation Scenario is described.

Original SCA application without mediation

- Application computes value of a customers portfolio
- A JSP provides a browser based interface
  - Enter customer ID
  - Total value of portfolio is returned
- Application includes three SCA modules
  - PortfolioManager module
  - CustomerBackend module
  - StockQuoteManager module
- The application flow is illustrated and described on next slide

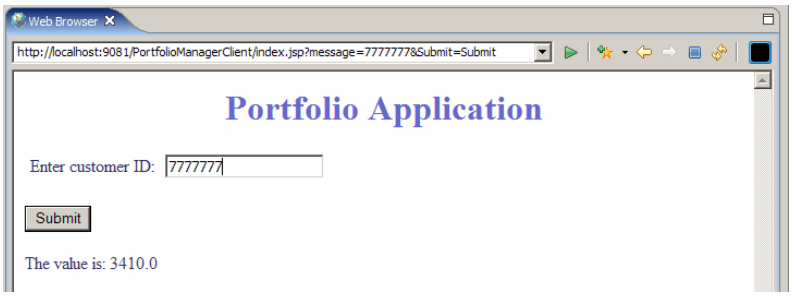
The application in its original form is used to compute the value of a customer's stock portfolio. It is composed of a JSP which provides a browser based interface where the customer enters their ID and the total dollar value of their stock portfolio is returned. The application is implemented in three SCA modules, specifically the PortfolioManager module, the CustomerBackend module and the StockQuoteManager module. The following slide shows these modules and the interactions between them.



This slide illustrates the original application showing the SCA wiring diagrams for each of the modules along with the basic flow of the application between modules.

Starting in the upper left the yellow box represents the PortfolioManager JSP which provides the browser based user interface where the customer ID is entered and the total portfolio value is displayed. The JSP uses the stand-alone reference in the PortfolioManager module to get a reference to the PortfolioService component and sends the customer ID to the PortfolioService as seen at the number 1. The PortfolioService then sends the customer ID to the CustomerService component in the CustomerBackend module as seen at number 2. The CustomerService component returns customer information to the PortfolioService, including a list of the customer's stocks and the number of shares held. The PortfolioService then calls the StockQuoteService component in the StockQuoteManager module as seen at number 3. The StockQuoteService is called once for each stock, with the stock symbol being passed in and the value of one share of that stock being returned. The PortfolioService then computes the total value of the customer's portfolio and returns that to the JSP which displays the value on the browser interface.

User interface to application



- Enter a customer ID
- Click Submit
- Total value of that customers portfolio is returned

This slide shows a screen capture of the browser interface provided by the JSP. As you can see, the interface is very basic. There is a field where the customer ID is entered, a Submit button is used to initiate the processing and the total portfolio value is then returned.

Why is a mediation needed?

- The company has two major goals
  - Provided enhanced function to the application users
  - Be able to distribute customer data across multiple new backends
- Staging implementation of upgrades to meet these goals
  - Insert mediation module between existing Portfolio Service and existing Customer Service backend
  - Incrementally introduce new Customer Service backends
    - They support an enhanced interface
    - Subset of customer data migrated to each new backend as it comes online
    - Route customer requests to correct backend and handle interface differences
  - Introduce a new enhanced Portfolio Service and associated mediation
  - When transition to all new backends complete, drop original Portfolio Service and mediation
- Only the first 2 points are addressed in these lab exercises

The requirement to add a mediation to this application is driven by a couple of goals for the application. First of all there is a need to provide an enhanced user experience with the capability to present more information to the customer than simply the total value of their stock portfolio. The second goal is related more to the underlying infrastructure. The company would like to move away from a single centralized database it now uses and make use of regional databases across which customer data would be distributed.

In order to meet these goals, a plan is put in place to stage the changes in such a way that there will be no disruption in application availability.

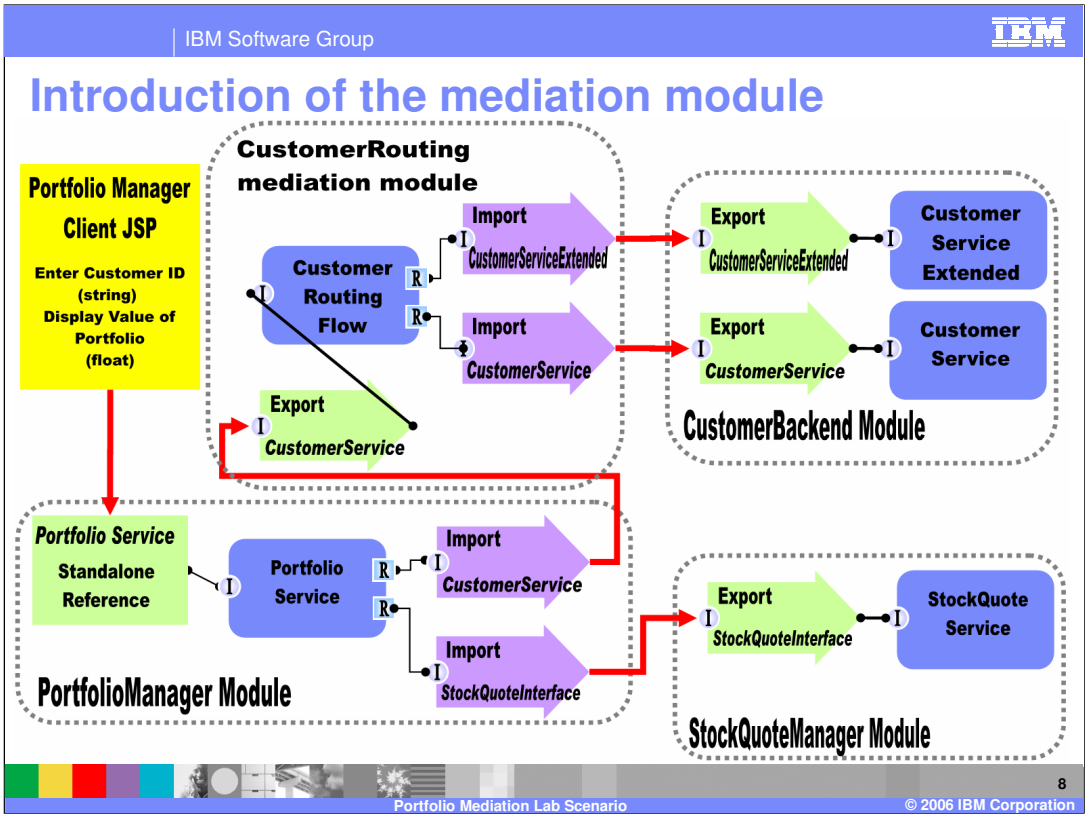
The first step is simply to insert a mediation module between the existing PortfolioService in the PortfolioManager module and the CustomerService in the CustomerBackend module. This does not change the application or the infrastructure but rather enables the possibility of mediation logic being added to the flow.

The second step is to provide an enhanced version of CustomerService and introduce that in conjunction with one of the regional databases coming online. The enhanced CustomerService implementation supports a new interface which provides the information required to support the eventual enhancement of the user interface. In addition, a subset of customer data is migrated from the central database to the new regional database. The plan is to stage in the regional databases over time. This will require the mediation to make routing decisions based on the customer ID so that the request is sent to the right backend and will also require the request to be modified to the new CustomerService interface. At this point, the user interface has not changed, but everything is staged as to allow a subset of the customers to be enabled for a new user interface implementation.

The third step would be to introduce a new PortfolioService and JSP to provide the new user experience along with a new mediation that handles requests for this new application.

The fourth stage would occur once all the customer data has been migrated to regional databases. At this point, the original JSP and PortfolioService can be removed, as can the first mediation.

The lab exercises used with this scenario will only address the first two steps in this staging. The following slide shows the structure after the second stage is implemented.



Notice that this diagram is quite similar to the original application previously presented. You can see that the interaction between the JSP and the PortfolioService is not changed. Likewise, the interaction between the PortfolioService and the StockQuoteService is also unchanged.

The following are the differences that should be noted.

First, there is now a new CustomerRoutingMediation module in the diagram.

Next, the CustomerService Import in the PortfolioManager module has been modified to connect to the CustomerService Export in the CustomerRoutingMediation module rather than the one in the CustomerBackend module.

Notice that the CustomerBackend module now has a CustomerServiceExtended component and Export in addition to the CustomerService component and Export. This is in support of the new regional database backends.

The function of the CustomerRoutingFlow mediation flow component in the CustomerRoutingMediation module is to determine the appropriate backend to which the request should be routed. Requests to the old centralized backend can be forwarded as is. Requests to the new regional backend need to have the operation modified to meet the CustomerServiceExtended interface.

The next slide examines the logic used in the CustomerRoutingFlow mediation flow component to accomplish this functionality.

The mediation flow – request flow

- Message Logger primitive
  - Log the request
- Custom Mediation primitive
  - Extract 2 digit prefix from customer ID in message body
    - These two digits from the customer ID are used to identify backend containing their information
  - Place 2 digit prefix into the transient context
- Database Lookup primitive
  - Use 2 digit prefix from transient context as key for lookup
  - Lookup returns a backend identifier, places it into the transient context

The CustomerRoutingFlow mediation component starts out with a Message Logger mediation primitive that logs the incoming request.

The next mediation primitive is a Custom Mediation. It is Java code that extracts the first two digits of the customer ID from the message body and places them into a field in the transient context. These two digits will be used to determine which backend to use for accessing this customers data.

A Database Lookup mediation primitive is then used. It takes the two digits extracted from the customer ID and uses them as a key for doing a database lookup that will identify which backend to use, the old centralized database, or a new regional database. The backend is identified with a string value that is placed into the transient context by the database lookup.

The mediation flow – request flow (cont.)

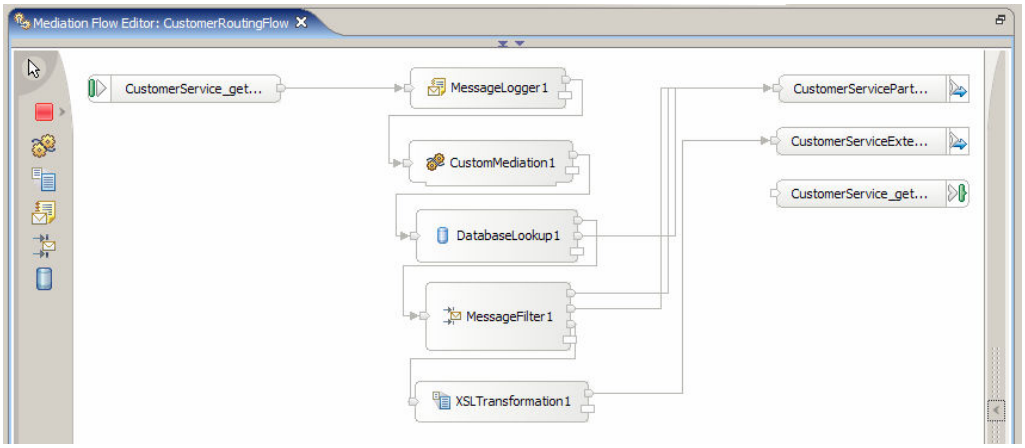
- Message Filter primitive
  - Determine routing based on backend identifier
  - Old backend – go directly to the callout for CustomerService
  - New backend – go to the XSLT
- XSLT primitive
  - Transform the CustomerServiceRequestMessage to a CustomerServiceExtendedRequestMessage
  - Go to the callout for CustomerServiceExtended

A Message Filter mediation primitive is now used to check the string value indicating the database to use and selects the appropriate path on which to propagate the message. If the message is to go to the old backend, the path goes directly from the Message Filter to the callout for the CustomerService backend.

In the case where the message is to flow to the new backend, an XSLT mediation primitive is used to transform the message from a CustomerService request to a CustomerServiceExtended request. Once transformed, the message flows from the XSLT primitive to the callout for the CustomerServiceExtended backend.

The following slide will show the mediation flow diagram for this mediation.

The mediation flow – request flow



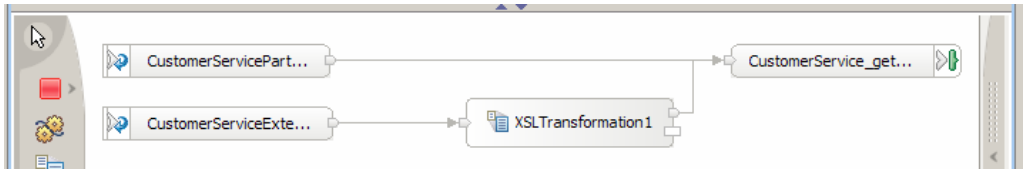
This is a screen capture of the mediation flow described on the previous slide.

On the left side you will see the input node on which the customer service request message comes in and on the right hand side you see the callouts to the CustomerService and CustomerServiceExtended backends.

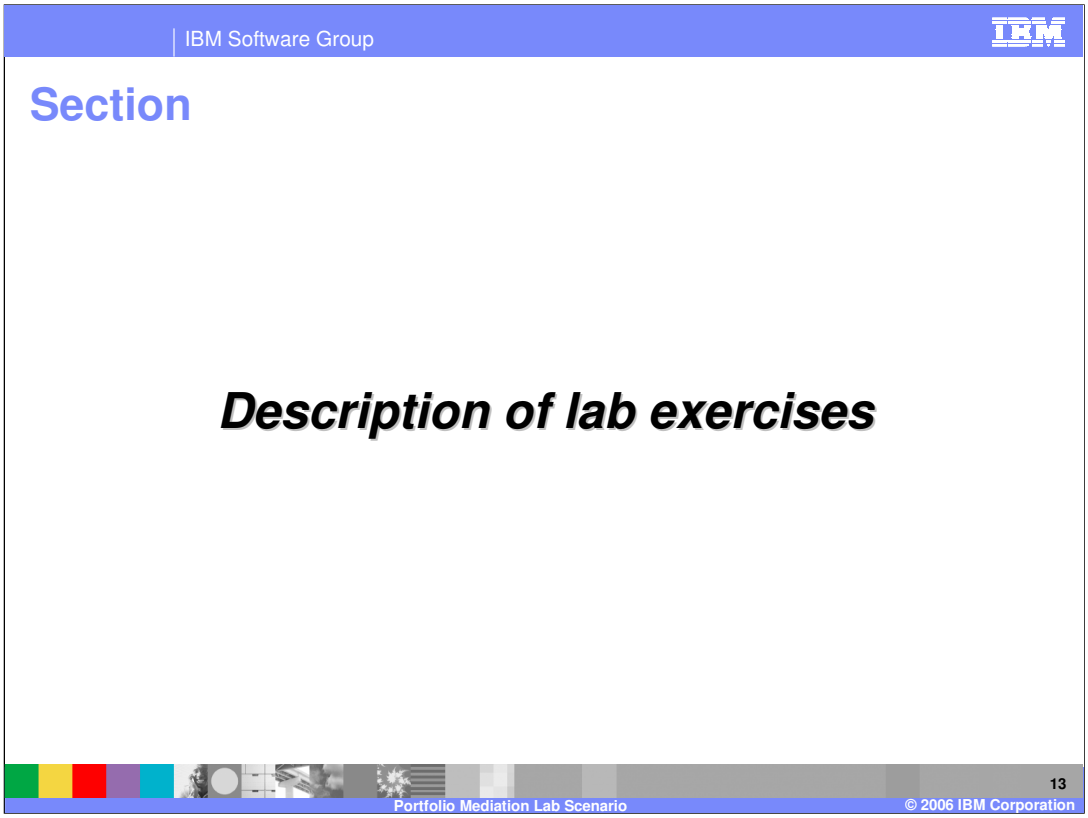
The flow starts with the Message Logger going to the Custom Mediation and then to the Database Lookup. On the Database Lookup, if the two digit key is not found, the keyNotFound flow is taken to the callout for the CustomerService backend and if the key is found the flow goes to the Message Filter. The flow from the Message Filter goes to the callout for the CustomerService backend in the case the backend is identified as the old backend or in the case where the filter does not have a matching expression for the backend value. When the backend is identified as the new backend the Message Filter flows to the XSLT to transform the message which is then passed to the callout for the CustomerServiceExtended backend.

The mediation flow – response flow

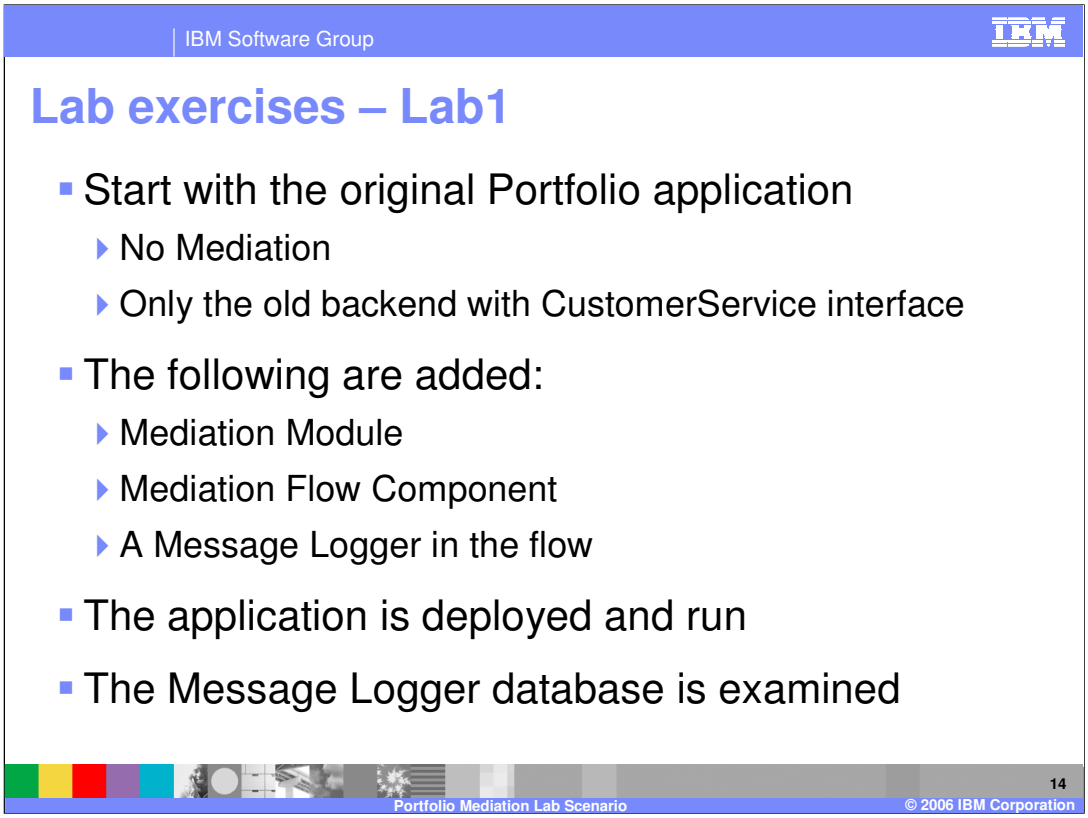
- CustomerService callout response
  - Go directly to the CustomerService response node
- CustomerServiceExtended callout response
  - XSLT transforms from CustomerServiceExtendedResponseMsg to CustomerServiceResponseMsg
  - Go to CustomerService response node



This is the response flow. When the call went to the CustomerService backend, there is nothing special to do on the return, so the callout response node is wired directly to the input response node. In the case where the call went to the CustomerServiceExtended backend an XSLT mediation primitive is required to transform the response message prior to returning.



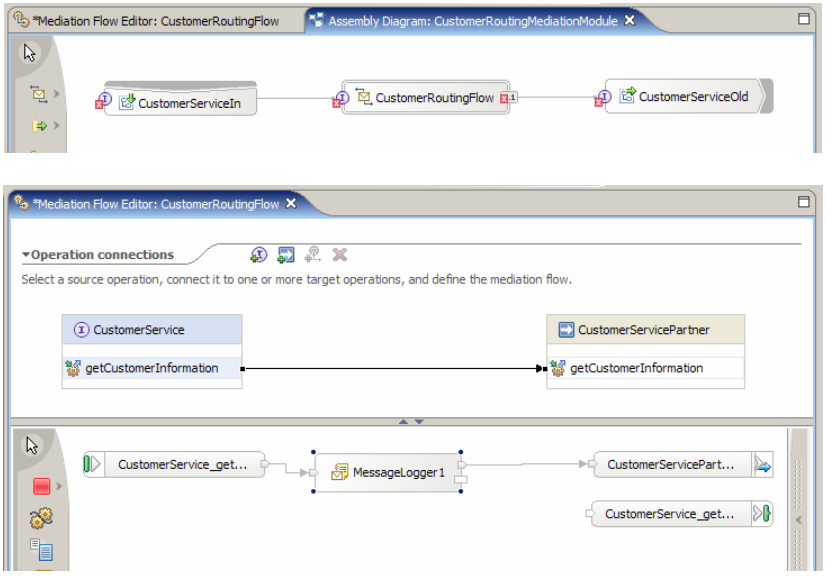
Now that a description of the application and mediation has been provided, this section will describe each of the four lab exercises that have been provided.



The first lab starts with the original application which uses the old backend that supports the CustomerService interface and there is no mediation. In the lab you will create the Mediation Module with corresponding Mediation Flow Component. The mediation flow will only contain a Message Logger that will log the message as it flows through the mediation. This is the equivalent to stage 1 of the staged approach previously described. The application function has not changed at all but there is now a mediation inserted into the flow.

The application with this mediation will be deployed to the test server in the WebSphere Integration Developer and run by using the browser interface to initiate a request for a customer. The database used by the Message Logger is then examined to see what has been logged.

Lab exercises – Lab1

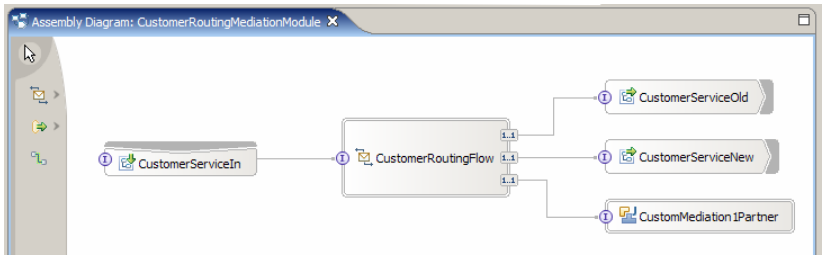


The top screen capture shows the assembly diagram for the mediation module. It contains an Export supporting the CustomerService interface wired to the Mediation Flow Component. This in turn is wired to an Import which also supports the CustomerService interface.

The bottom screen capture shows the Mediation Flow Editor with both the operation connections panel and the canvas containing the mediation flow. You can see that the flow simply contains a Message Logger.

Lab exercises – Lab2

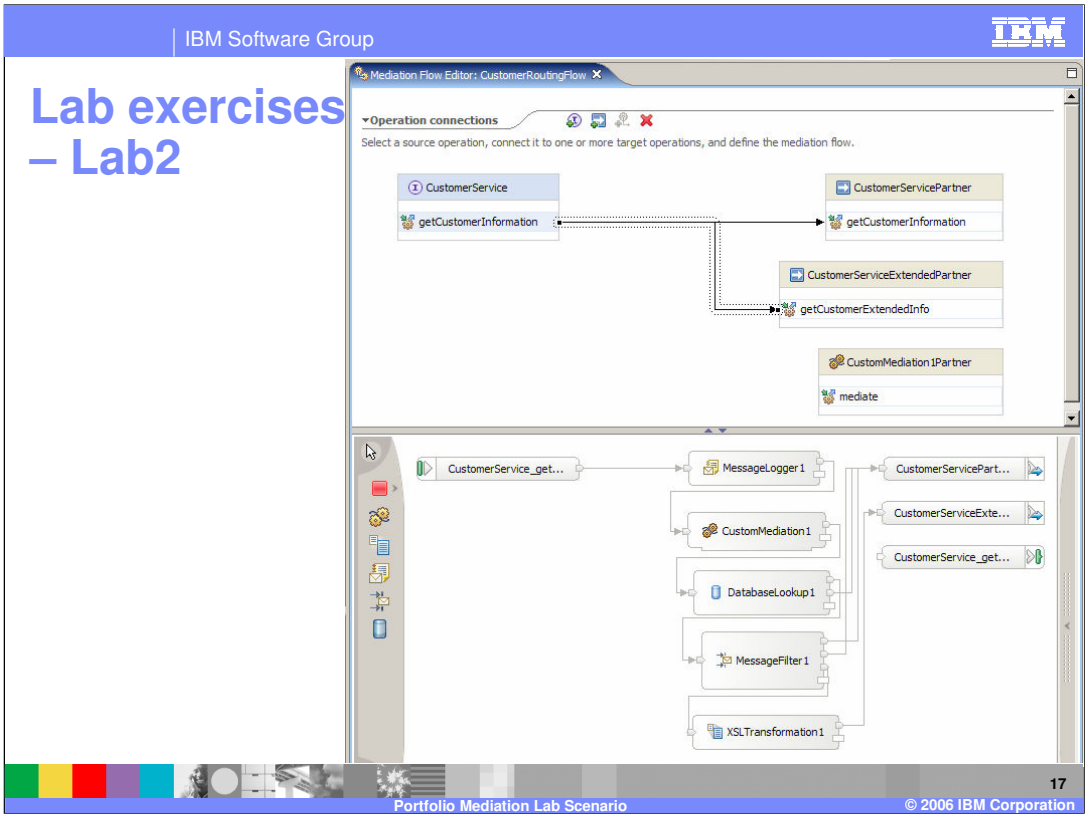
- Start with the results of Lab1
- The following are added:
  - Import to access the new CustomerServiceExtended backend
  - Expand Mediation flow by adding and configuring
    - Custom Mediation (including Java™ SCA component on the assembly)
    - Database Lookup
    - Message Filter
    - XSLT (in both the request flow and the response flow)
- The application is deployed and run



Lab 2 uses the results of Lab 1 as a starting point.

First an Import is added to the assembly diagram which supports the CustomerServiceExtended interface as is shown in the screen capture. The mediation flow is enhanced to include the Custom Mediation, Database Lookup, Message Filter and XSLT. There is an XSLT also added for the response flow. These flows will be seen in screen captures on the next slide.

The Custom Mediation will make use of a Java SCA Component containing the custom code. This Java SCA component can be seen in the screen capture on this slide.



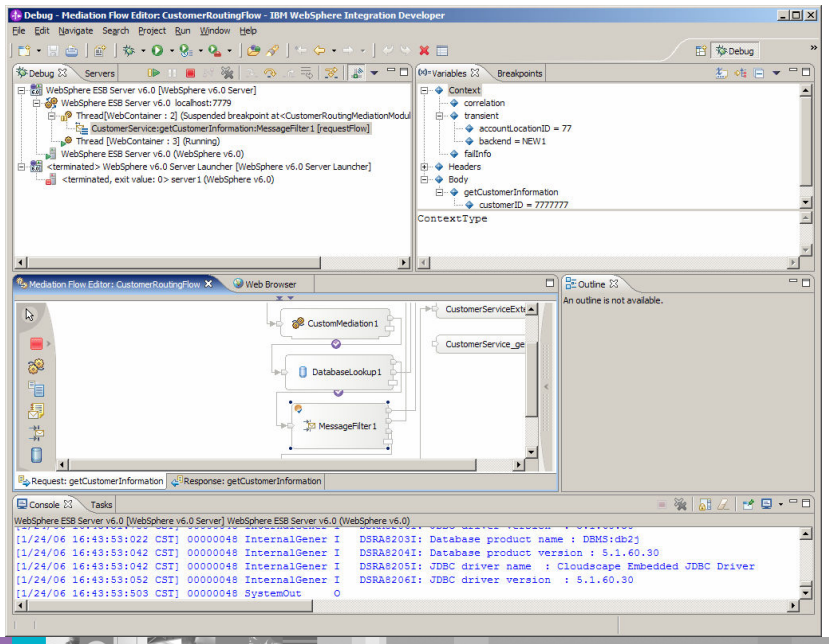
The mediation flow editor for Lab 2 is shown here. In the Operation Connections panel you can see that the input operation on the CustomerService interface is wired to two possible output operations, one on the CustomerService interface and the other on the CustomerServiceExtended interface. The lower portion shows the canvas with the mediation flow which is identical to the flow described earlier.

Lab exercises – Lab3

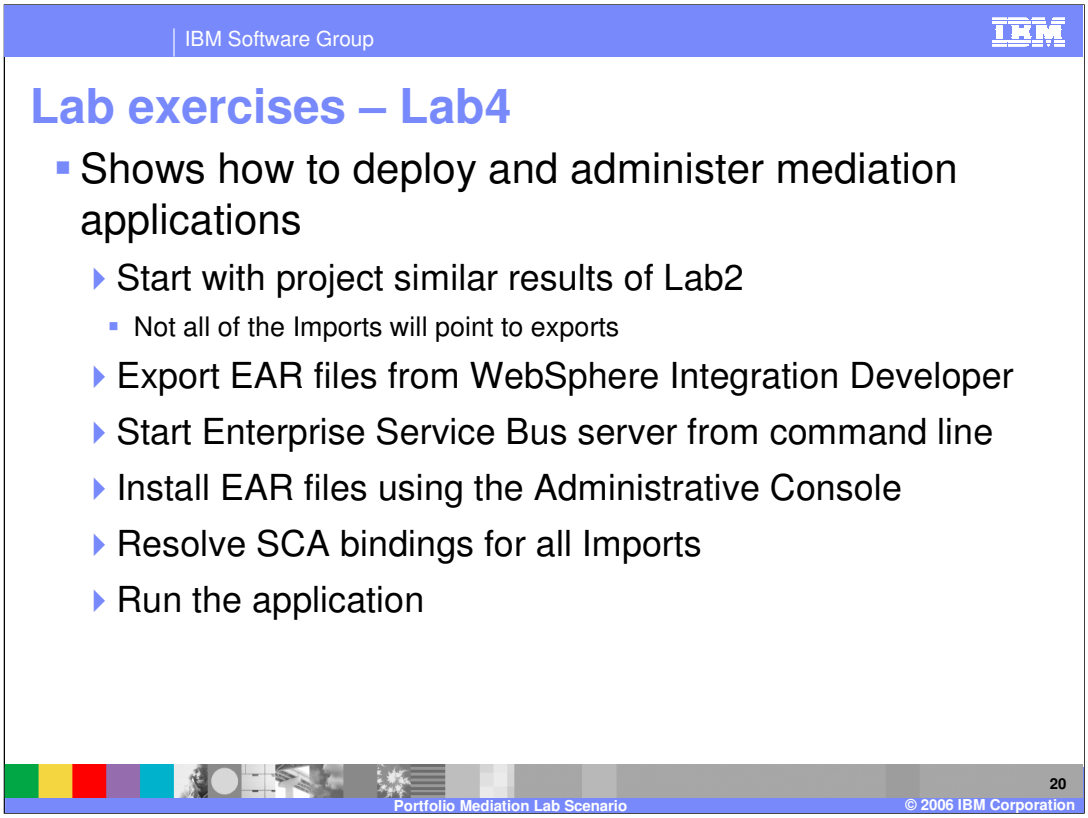
- Start with the results of Lab2
- Use the Visual Debugger to examine the flow
  - Add breakpoints throughout the mediation flow
  - Deploy and run on server in debug mode
  - Examine the Service Message Object (SMO) at various places in the flow

Lab 3 starts using the end result from Lab 2. In this lab you will use the Visual Debugger to examine the flow of a message through the mediation. This will be done by adding breakpoints on the mediation primitives, running the server in debug mode and examining the Service Message Object at various places in the flow. This will allow you to see the changes made to the Service Message Object by the Custom Mediation, Database Lookup and XSLT mediation primitives.

Lab exercises – Lab3



In this slide you see a screen capture of the visual debugger. The panel in the middle shows the mediation flow including the icons for breakpoints and icons on the wires to show through which wires the flow has already passed. The upper right panel shows the Service Message Object. You can see that the operation contains a customerID of 7777777. The transient context has been updated with the first two digits of this customerID and the backend has been identified as the new backend. The upper left panel shows exactly where you are in the flow and the lower panel is the console log from the server.



Lab 4 will be used to show you how to export mediation applications from WebSphere Integration Developer and then to deploy and administer them within the WebSphere Enterprise Service Bus. The lab starts with a project similar to the final result of Lab2, except that not all of the Imports have their SCA bindings resolved to the corresponding SCA Exports. You will be shown how to export the EAR files for the modules that make up the application. The next step will be to stop WebSphere Integration Developer and start the WebSphere Enterprise Service Bus server from the command line. Using the Administrative Console you will install the EAR files and then modify the SCA bindings for the Imports to be associated with the appropriate SCA Exports. Finally you will test the application.

Lab exercises – Test data

- The data used in the exercise is simply hard coded in the java code in:
  - CustomerBackend module
  - StockQuoteManager module
- The table shows the pre-defined customer IDs and portfolio values
- All other customer IDs return $50

Customer ID	Backend	Portfolio Value
1111111	CustomerService (old)	$16,850
2222222	CustomerService (old)	$28,500
3333333	CustomerService (old)	$6,880
4444444	CustomerService (old)	$81,000
5555555	CustomerServiceExtended (new)	$32,550
6666666	CustomerServiceExtended (new)	$86,250
7777777	CustomerServiceExtended (new)	$3,410
8888888	CustomerServiceExtended (new)	$101,400
9999999	CustomerServiceExtended (new)	$3,650

Understanding the test data that has been built into the lab exercises will help you predict and understand the behavior you will see when running the application using different customer IDs. The test data has been hard coded into the CustomerBackend module and the StockQuoteManager module. The only customer IDs that will be recognized are those shown in the table. The table also shows for each ID which backend will be used to process the request and the total portfolio value that will be computed. A customer ID that does not exist in this table will return a portfolio value of 50 dollars.

Lab exercises – Test data

- The Database Lookup determines the backend to use
  - Prefixes directed to old backend: 11, 22, 33, 44
  - Prefixes directed to new backend: 55, 66, 77, 88, 99
  - A prefix not found in the database (e.g. 14, 73, 42) will go to the old backend through the keyNotFound terminal
- How to determine which backend was used:
  - There is no way to tell from the browser interface
  - There are messages written to the console that show this:

```
O Entering getPortfolioValue() with 881111
O Getting customer info...
O Entering getCustomerExtendedInfo() with id: 881111
O Exiting getCustomerExtendedInfo() with id: 881111
O Invoking getStockPrice() with SBC
O Portfolio value is 50.0
O Value: 50.0
```

```
O Entering getPortfolioValue() with 112345
O Getting customer info...
O Entering getCustomerInformation() with id: 112345
O Exiting getCustomerInformation() with id: 112345
O Invoking getStockPrice() with DELL
O Portfolio value is 50.0
O Value: 50.0
```


The database lookup determines which backend will be used. The database has been initialized so that customer IDs with prefixes of 11, 22, 33 or 44 will all go to the old backend whereas customer IDs with a prefix of 55, 66, 77, 88 or 99 all go to the new backend. Any other prefixes will not be recognized resulting in the keyNotFound terminal being used with the result of the flow going to the old backend.

Unfortunately, it is not possible to tell which of the backends has been used from the browser interface used to drive the application. To determine that you must look at the output in the Console, specifically for the lines circled in red. On the left is output from the new backend showing the entry and exit of the getCustomerExtendedInfo operation whereas on the right is output from the old backend showing the entry and exit of the getCustomerInformation operation.

Summary

- The Portfolio Mediation Scenario was described, including:
  - The original application prior to the mediation
  - Reasons for needing a mediation
  - The application containing the mediation
  - The flow logic within the mediation
- The lab exercises were introduced
  - The scope of each of the labs was described
  - The built-in test data was explained.

In this presentation the Portfolio Mediation Scenario used by the lab exercises was described. This included looking at the original application and then examining the business reasons for wanting to introduce a mediation. The application with the mediation and the mediation flow were then examined.

With that as a base, the individual lab exercises were introduced. The test data used by the labs was also explained.

Trademarks, Copyrights, and Disclaimers

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

IBM	CICS	IMS	MQSeries	Tivoli
IBM(logo)	Cloudscape	Informix	OS/390	WebSphere
e(logo)business	DB2	iSeries	OS/400	xSeries
AIX	DB2 Universal Database	Lotus	pSeries	zSeries

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

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

Intel, ActionMedia, LANDesk, MMX, Pentium and ProShare 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 registered trademark of Linus Torvalds.

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

Product data has been reviewed for accuracy as of the date of initial publication. Product data is subject to change without notice. This document could include technical inaccuracies or typographical errors. IBM may make improvements and/or changes in the product(s) and/or program(s) described herein at any time without notice. Any statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business. Any reference to an IBM Program Product in this document is not intended to state or imply that only that program product may be used. Any functionally equivalent program, that does not infringe IBM's intellectual property rights, may be used instead.

Information is provided "AS IS" without warranty of any kind. THE INFORMATION PROVIDED IN THIS DOCUMENT IS DISTRIBUTED "AS IS" WITHOUT ANY WARRANTY, EITHER EXPRESS OR IMPLIED. IBM EXPRESSLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IBM shall have no responsibility to update this information. IBM products are warranted, if at all, according to the terms and conditions of the agreements (e.g., IBM Customer Agreement, Statement of Limited Warranty, International Program License Agreement, etc.) under which they are provided. Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. IBM makes no representations or warranties, express or implied, regarding non-IBM products and services.

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents or copyrights. Inquiries regarding patent or copyright licenses should be made, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.