developerWorks

# Dynamically select adapters based on context using WebSphere Business Services Fabric

Anket Jain (ankejain@in.ibm.com)
System Software Engineer
EMC

Vinay Roy Thykkuttathil (vinayroyt@in.ibm.com)
Staff Software Engineer
WSO2 Inc

Naveen Balani (banaveen@in.ibm.com)
Software Architect
WSO2 Inc

05 May 2010

Learn how to dynamically invoke resource adapters using WebSphere® Business Services Fabric's policy framework.

## Prerequisites

Before you get started, make sure you have the following software installed so that you can configure and deploy the module in this article:

- WebSphere Integration Developer V6.2 (hereafter called Integration Developer)
- WebSphere Process Server V6.2 (hereafter called Process Server)
- WebSphere Business Services Fabric V6.2 (hereafter called Fabric)

In addition, this article assumes that you're familiar with Fabric.

## Scenario overview

Company XYZ recently acquired Company ABC to expand its printing business. Company XYZ uses a DB2 Enterprise Information System (EIS) to store and retrieve customer data, but Company ABC uses a file-based data storage mechanism. The processes of the two companies need to be integrated into one system that uses both of the existing data storage mechanisms (DB2 EIS and file system).

The integrated solution design needs to be adaptable and able to retrieve or update information in the appropriate EIS, based on user log-in.

In addition, company XYZ is planning integration with other third-party vendors for printing services, in which customer data would be sourced from third-party systems. In short, the company is looking for a solution that allows generalized access to multiple systems and that enables adding new backend systems or third-party vendors incrementally without requiring changing its business processes.

For the sake of simplicity, we won’t address the overall business solution in this article, but will deal only with retrieving the appropriate customer information based on user context. We'll use WebSphere Adapters to integrate the databases, and Fabric to select the appropriate adapter implementation based on user context.

Following are the steps required to achieve the integration:

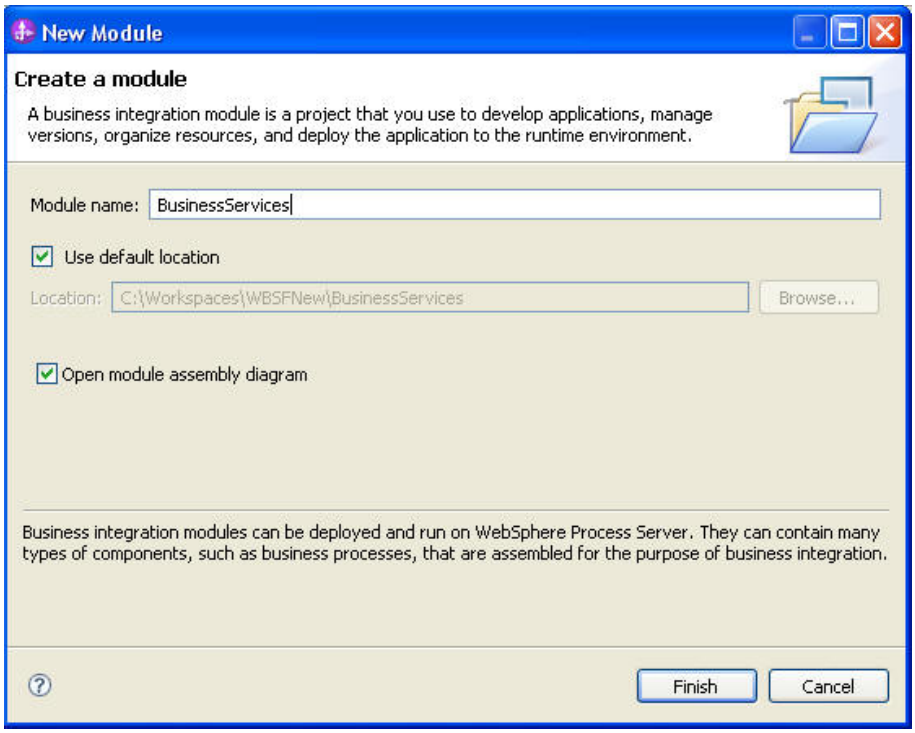
1. Create a Service Component Architecture (SCA module) based on the WebSphere Adapter for Flat Files.
2. Create an SCA module based on the WebSphere Adapter for JDBC (Java™ Database Connectivity).
3. Create a generic interface to abstract calls to the adapter implementations for Flat File and JDBC.
4. Test the Adapter for Flat File and Adapter for JDBC SCA implementations.
5. Use the Fabric policy framework to select the correct adapter implementation based on the user context.
6. Test the integrated solution.
7. Manage changes.

## Create an SCA module based on Adapter for Flat File

Using Integration Developer, you can easily create and expose SCA modules for external consumption by completing the following steps:

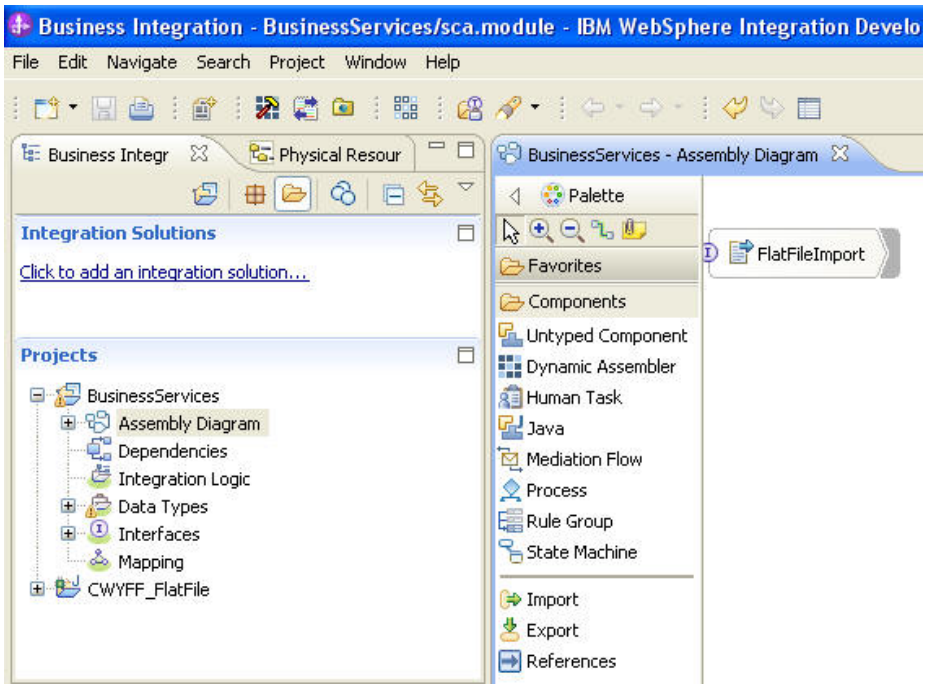
1. Open a fresh workspace in Integration Developer.
2. Select **New => Module**.
3. Specify `BusinessServices` as the module name and click **Finish**, as shown in Figure 1.

### Figure 1. Create a new module in Integration Developer




4. Follow the instructions in the IBM WebSphere Adapter for Flat Files 6.2 Quick Start Tutorials (Tutorial 1) to create an outbound module to retrieve structured content from a filesystem-based database. Once you've complete the outbound module, the assembly diagram should look like Figure 2.

### Figure 2. SCA module based on Adapter for Flat File

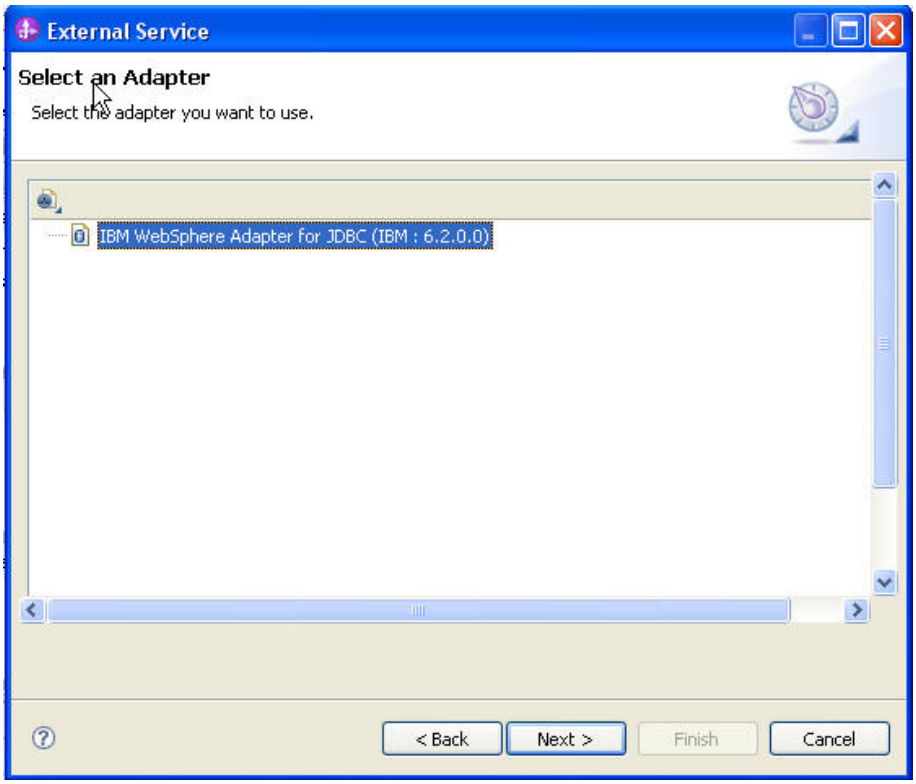



## Create an SCA module based on Adapter for JDBC

To create a JDBC module do the following:

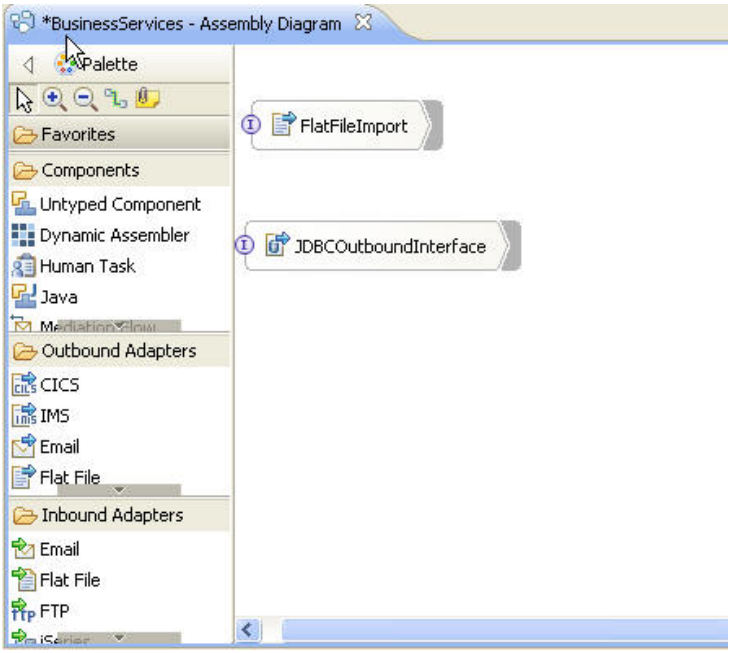
1. Open Integration Developer in a fresh workspace.
2. Select **New => Module**.
3. Specify `BusinessServices` as the module name and click **Finish**, as shown in Figure 1.
4. Follow the instructions in the IBM WebSphere Adapter for JDBC 6.2 Quick Start Tutorials (Tutorial 13) to create an outbound module create an outbound module to manipulate database table rows. Once you've completed the outbound module, the assembly diagram should look like Figure 3. Follow the instructions listed in the Quick Start Scenario (QSS) for the WebSphere Adapter for JDBC mentioned in the references section and create an outbound module to manipulate database table rows. Select **WebSphere Adapter for JDBC**, as shown in Figure 3.

### Figure 3. Select WebSphere Adapter for JDBC




5. Once you complete the outbound module, the assembly diagram should look like Figure 4.

### Figure 4. New Flat File and JDBC SCA modules



# Create a generic interface to abstract calls to the adapter implementations for Flat File and JDBC

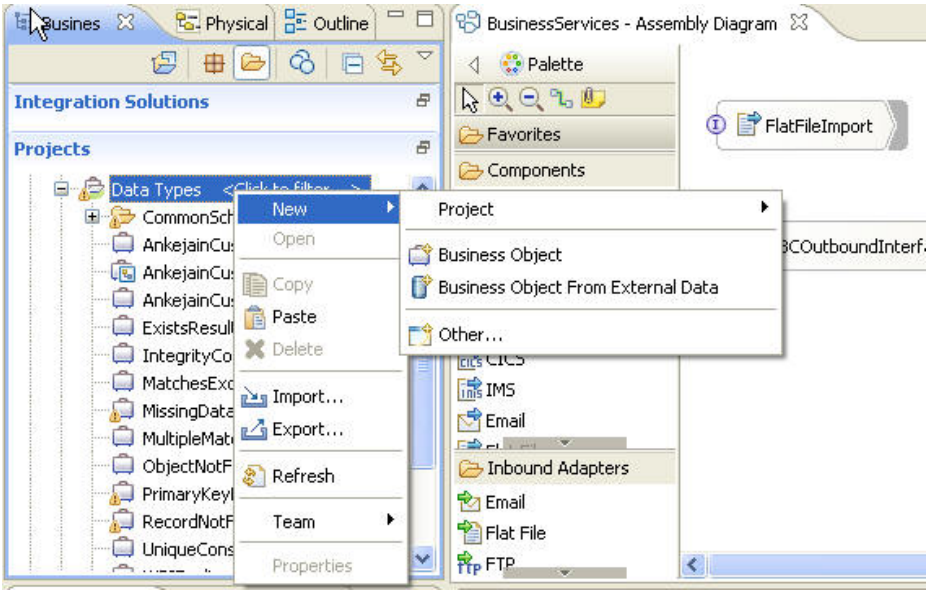
Now that you've created the required SCA modules, you need to expose these using a generic interface. To do this, you need to create an interface with generic input data types to make the outbound requests.

## Create the GenericRequest and GenericResponse data objects

To create the `GenericRequest` and `GenericResponse` data objects, do the following:

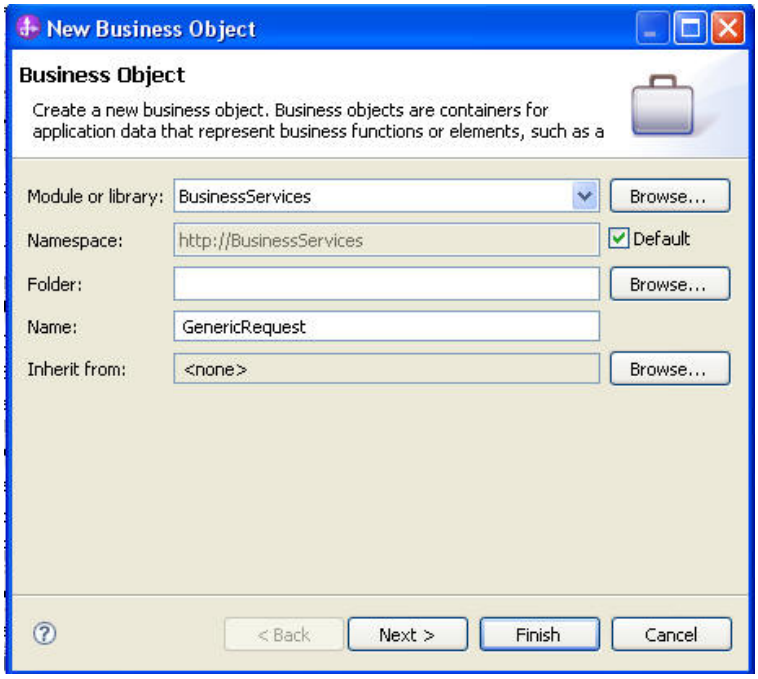
1. Right-click on **Data Types** and select **New => Business Object**, as shown in Figure 5.

   **Figure 5. Create a new business object**




2. In the New Business Object dialog, specify `GenericRequest` as the **Name** and keep the default values for everything else, as shown in Figure 6.

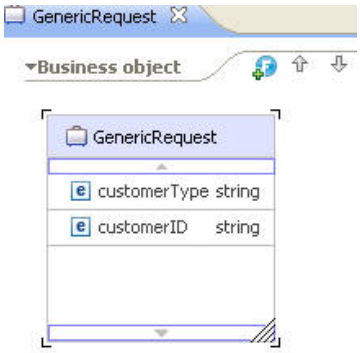
### Figure 6. Create GenericRequest business object




3. Create two attributes of string type `customerType` and `customerID` where:
   - `customerType` can either be `NEW` or `EXISTING`. If the customer type is `NEW`, details are stored in a database table. If the customer type is `EXISTING`, the details are stored in a filesystem-based database.
   - `customerID` is a unique field that is used to retrieve the customer details.

After the attributes are added, the `GenericRequest` business object should look like Figure 7.

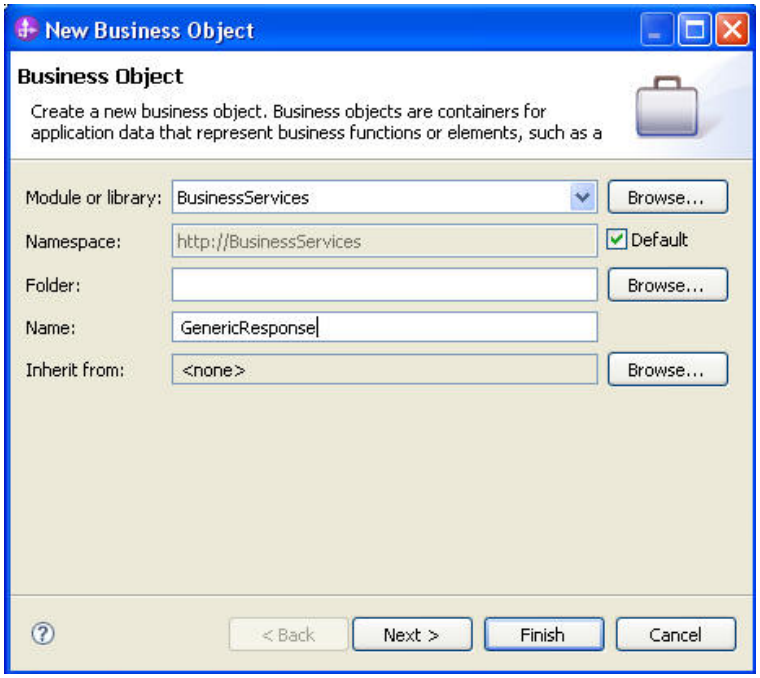
### Figure 7. GenericRequest business object




Based on the value of the `CustomerType` field in the GenericRequest Business Object, you can make a decision about which adapter interface to invoke.

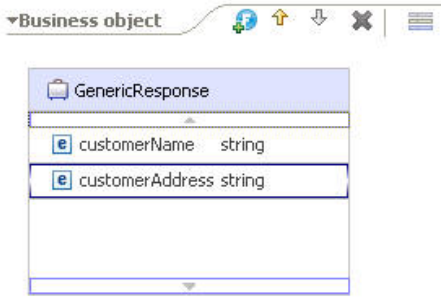
4. Now create a `GenericResponse` business object to be returned back to the caller, as shown in Figure 8.

### Figure 8. Create GenericResponse business object




The `GenericResponse` business object must contain the **Customer Name** and **Customer Address** information retrieved from the appropriate store (JDBC table or Flat File database). The final business object structure will look like Figure 9.

### Figure 9. GenericResponse business object




The Adapter for Flat Files will be used to retrieve customer details stored in a flat file database with structured content, as shown in Listing 1. The `CustomerID` field is used to look up the details of the customer.

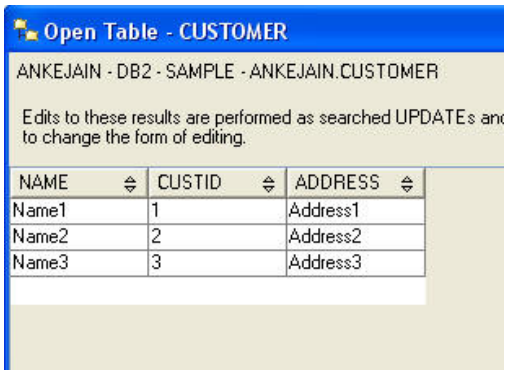
## Listing 1. Flat File Structured Content

```
  <?xml version="1.0" encoding="UTF-8"?>
<p:Customer xsi:type="p:Customer"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:p="http://BusinessServices">
  <customerID>1</customerID>
  <customerName>NDI-1</customerName>
```

```
  <customerAddress>France</customerAddress>
</p:Customer>
##
<?xml version="1.0" encoding="UTF-8"?>
<p:Customer xsi:type="p:Customer"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:p="http://BusinessServices">
  <customerID>2</customerID>
  <customerName>NDI-2</customerName>
  <customerAddress>France</customerAddress>
</p:Customer>
##
<?xml version="1.0" encoding="UTF-8"?>
<p:Customer xsi:type="p:Customer"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:p="http://BusinessServices">
  <customerID>3</customerID>
  <customerName>NDI-3</customerName>
  <customerAddress>France</customerAddress>
</p:Customer>
##
```

The Adapter for JDBC will be used to retrieve customer details from a pre-populated database table, as shown in Figure 10.

### Figure 10. Customer record for JDBC

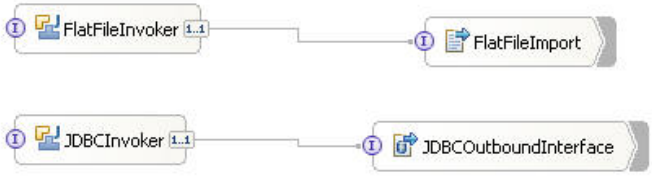



## Create POJO components to invoke the adapter modules

Now you need to create POJO components to invoke the outbound Flat File and JDBC adapter modules you've created. To do this, complete the following steps:

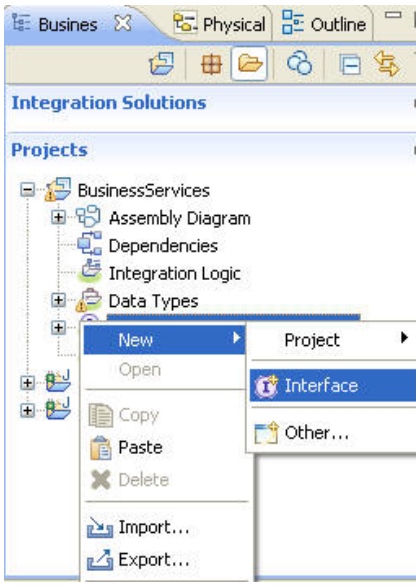
1. Click **Java** in the **Components** folder and drag it onto the palette. Repeat this operation and wire one **Java** component to the **FlatFileImport** component and the other to the **JDBCOutboundInterface** component. Your assembly diagram should look like Figure 11.

   ### Figure 11. POJO components wired to the imports



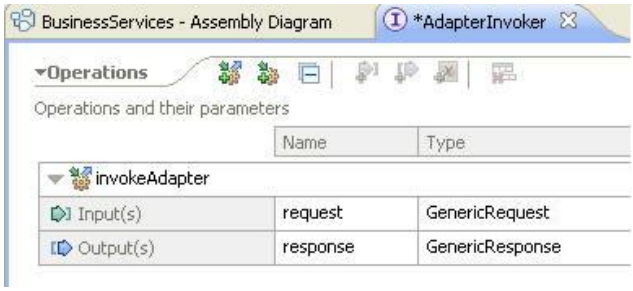
2. For easier invocation of the adapter components, create a generic interface by right-clicking **Interface** and selecting **New => Interface**, as shown in Figure 12.

### Figure 12. Create a generic interface



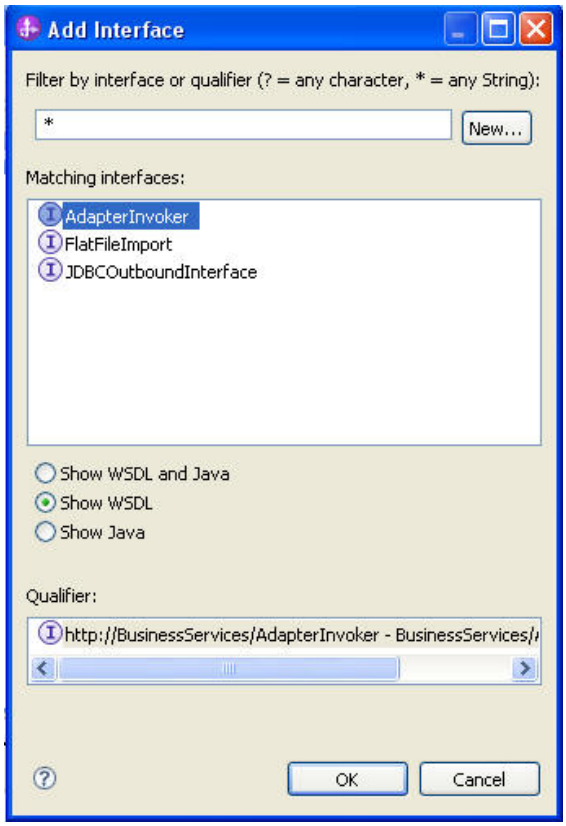

3. Specify `AdapterInvoker` as the name of the interface and add a two-way method called `invokeAdapter`.
4. Add `GenericRequest` and `GenericResponse` as the input and output types of the `invokeAdapter method`, as shown in Figure 13.

### Figure 13. AdapterInvoker interface




5. Add the newly created `AdapterInvoker` interface to both the Java components (`FlatFileInvoker` and `JDBCInvoker`) by right-clicking on the respective Java components and selecting **Add => Interface**. You'll see the Add Interface dialog, as shown in Figure 14.

### Figure 14. Add AdapterInvoker interface




6. After adding the newly created interface, you need to implement the two Java components. Double-click on the specific Java component to implement it.
   The default implementation for the `invokeAdapter` method of the POJO components should look like Listing 2.

### Listing 2. Default implementation for the invokeAdapter method

```
            public DataObject invokeAdapter (DataObject request) {

return null;

}
```

7. Change the default implementation for the `invokeAdapter` method in the FlatFile component to the content shown in Listings 3 and 4.

### Listing 3.Implementation for the invokeAdapter method for the FlatFile component: invoking the FlatFile outbound interface.

```
          public DataObject invokeAdapter(DataObject request) {
  String customerID = request.getString("customerID");

  Service ffService = locateService_FlatFileImportPartner();
  ServiceManager serviceManager = new ServiceManager();
  BOFactory bof = (BOFactory)
serviceManager.locateService("com/ibm/websphere/bo/BOFactory");
```

```
  DataObject flatFile =
bof.create("http://www.ibm.com/xmlns/prod/websphere/j2ca/flatfile/flatfile",
"FlatFile");
  DataObject genericResponse =
bof.create("http://BusinessServices", "GenericResponse");

  flatFile.setString("fileName", "FF.xml");
  flatFile.setString("directoryPath", "C:\\FF\\out");
  flatFile.setString("splitFunctionClassName", "com.ibm.j2ca.utils.
filesplit.SplitByDelimiter");
  flatFile.setString("splitCriteria", "##;\r\n");

  DataObject response = (DataObject)
ffService.invoke("retrieveFlatFile", flatFile);
```

## Listing 4. Implementation for the invokeAdapter method for the FlatFile component: parsing the response

```
Iterator customers = ((List)
response.getDataObject("retrieveFlatFileOutput").getDataObject
("CustomerRetrieveWrapper").getList("Content")).iterator();

  while(customers.hasNext()) {
   DataObject customer = (DataObject) customers.next();

 if(customer.getString("customerID").compareToIgnoreCase(customerID)
== 0) {
    genericResponse.setString("customerName",
customer.getString("customerName"));
    genericResponse.setString("customerAddress",
customer.getString("customerAddress"));
   }
  }

  return genericResponse;
 }
```

8. Change the default implementation for the `invokeAdapter` method in the JDBC component to the content shown in Listings 5 and 6.

## Listing 5. Implementation for the invokeAdapter methods for the JDBC component: invoking the JDBC outbound interface

```
public DataObject invokeAdapter(DataObject request) {
  Service jdbcService =
   locateService_JDBCOutboundInterfacePartner();
  ServiceManager serviceManager = new ServiceManager();
  BOFactory bof = (BOFactory) serviceManager.
  locateService("com/ibm/websphere/bo/BOFactory");
  DataObject ankejainCustomerBG = bof.create("http://" +
    "www.ibm.com/xmlns/prod/websphere/j2ca/jdbc/" +
    "ankejaincustomerbg", "AnkejainCustomerBG");
  DataObject ankejainCustomer = bof.create("http://www" +
    ".ibm.com/xmlns/prod/websphere/j2ca/jdbc/" +
    "ankejaincustomer",
    "AnkejainCustomer");
  DataObject genericResponse = bof.create("http://" +
    "BusinessServices",
    "GenericResponse");

  ankejainCustomer.setString("custid", request.getString
```

```
    ("customerID"));
  ankejainCustomerBG.setDataObject("AnkejainCustomer",
    ankejainCustomer);

  DataObject response = (DataObject) jdbcService.invoke
  ("retrieveAnkejainCustomerBG", ankejainCustomerBG);
```

### Listing 6. Implementation for the invokeAdapter methods for the JDBC component: parsing the response to retrieve the customer name and address

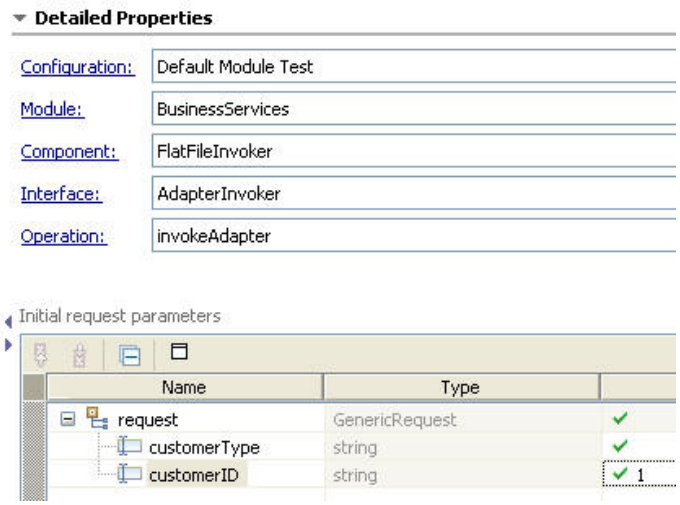
```
genericResponse.setString("customerName", response.
    getDataObject("retrieveAnkejain" +
      "CustomerBGOutput")
    .getDataObject("AnkejainCustomer")
    .getString("name"));
  genericResponse.setString("customerAddress",
    response
    .getDataObject("retrieveAnkejain" +
      "CustomerBGOutput")
    .getDataObject("AnkejainCustomer")
    .getString("address"));

  return genericResponse;
 }
```

## Test the Flat File and JDBC adapter SCA module implementations

To test the implementations, complete the following steps:

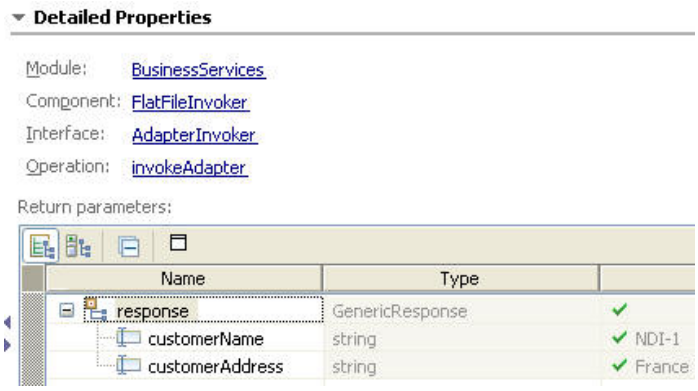
1. After generating the assembly diagram and implementing the components, save the module and deploy it to your WebSphere Process Server instance.
2. After successful deployment, invoke the test client for the **BusinessServices** module.
3. To invoke the Flat File module, select the **FlatFileInvoker** component and specify a value for the **customerID** field, as shown in Figure 15.

   ### Figure 15. Test FlatFileInvoker component



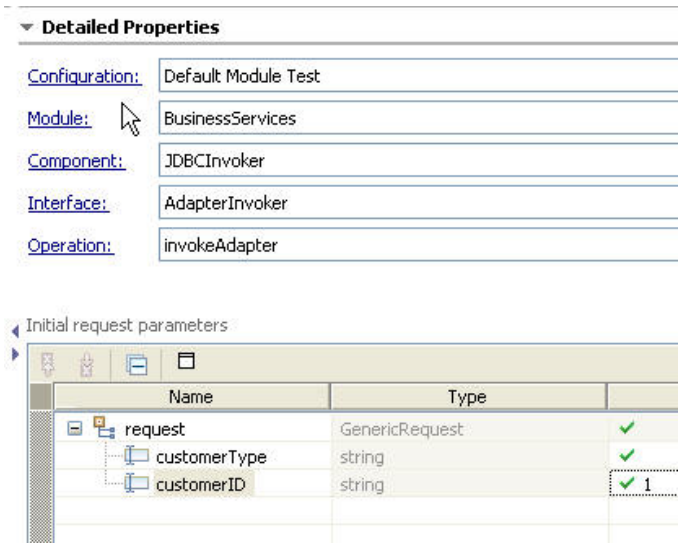

4. Click **Execute** to run the test client. You'll see that the customer details are retrieved, as shown in Figure 16.

### Figure 16. FlatFileInvoker response



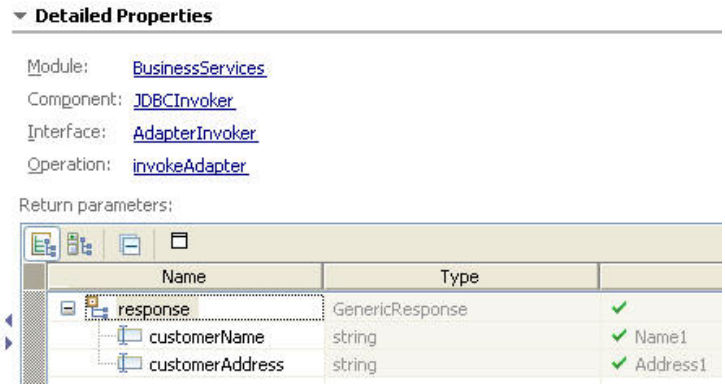

5. To invoke the JDBC module, select the **JDBCInvoker** component and specify a value for the **customerID** field, as shown in Figure 17.

### Figure 17. Test JDBCInvoker component




6. Click **Execute** to run the test client. The customer details are retrieved from the database table, as shown in Figure 18.

### Figure 18. JDBCInvoker response




## Use the Fabric policy framework to select the appropriate adapter based on the user context

Now that you've created the generic interface and implemented the corresponding flat file and JDBC components, you now need to configure Fabric's policy framework to select the appropriate adapter implementation based on the customer type.

Customer type is one of the points of variability for the customer service that drives selection of a particular type of service implementation. This point of variability is modeled as an ontology extension (prior to Fabric V6.2) or vocabularies (Fabric V6.2 or later) in Fabric, which provides the required context for the service that is later used by business services and policies.

Vocabulary concepts can be created with the Fabric authoring template using Business Space, which provides a Web-based user interface to create concepts. The concepts define the terms used by business services and policies.

For this solution, we'll assume that the ontology extensions and Business Services project are already created and available.

For detailed instructions on how to create ontology extensions and a Business Services project, refer to the article series Creating flexible service-oriented business solutions with WebSphere Business Services Fabric.

To import the ontology extensions and Business Services project for the Customer Application project, complete the following steps:

1. Log on to Fabric administration console.
2. Select **Governance Manager => Import/Export**.
3. On the Import dialog, select the CustomerOntologyModel.zip file provided for download with this article, and click **Import File**. This imports the Customer Ontology model ontology extensions, which include the `CustomerTypeAssertion`.

4. Next, import the CustomerApplicationProject.zip provided for download. This is the Fabric Business Services project that contains the business service metadata for the customer solution.
5. Replicate the Customer Application project in Composition Studio and analyze the business service metadata, as described in the next section.

## Analyze the Business Services project

Complete the following steps to replicate the Customer Application Business Service project into Composition Studio:

1. In Integration Developer, switch to the **Business Service** perspective.
2. Select **Window => Open Perspective => Other**, and select **Business Service**.
3. Select **File => New => Project => Business Services Fabric => Fabric Project**, then click **Next**.
4. Specify the project name as `CustomerApplicationProject` and click **Next**.
5. Click **Configure**, and specify the Business Service repository connection information, as follows:
   - **Hostname:**`localhost`. The host where Fabric is deployed.
   - **Port:**: `portnumber`. The port on which the Fabric UTE server is running (for example, 9081).
   - **Username:**`admin`
   - **Password:**`admin`
6. Click **OK**.
7. Click **Next**, and select **Customer Application Project**, then click **Finish**.
   This new Fabric project will hold the composite service and associated interfaces for the Customer Application project.
8. Add the Dynamic Assembler component to the existing SCA module and then create the business service metadata for the Customer Application project from the SCA module.

## Wire the Fabric Dynamic Assembler to the SCA module

So far, you've created and tested the Customer Flat file and JDBC exports, but you haven't yet added any dynamic binding support. Fabric doesn't know which set of services to call if the customer type is `NEW` or `EXISTING`. You can use the Fabric Dynamic Assembler to provide this dynamic binding support.

The Dynamic Assembler is a highly scalable engine that enables dynamic policy assembly and service selection based on content, context, and contract. It enables business agility through policy-driven run-time assembly of business services. The Dynamic Assembler links service consumers to service providers. Think of it as a smart proxy that determines which endpoints to use based on requests. Rather than invoking your endpoint directly, consuming applications invoke Fabric proxy URIs. The Dynamic Assembler then redirects the requester to the appropriate endpoint.

To assemble the components, do the following:

1. Open the assembly diagram editor for the module.
2. Add a **Dynamic Assembler** component for the **AdapterInvoker** service by dragging it onto the editor. Label it `CustomerDetailsDA`.
3. Right-click the **Dynamic Assembler** component and select **Add => Interface**, then select **AdapterInvoker**.
4. Double-click the **Dynamic Assembler** component to implement it.
5. Specify the folder where you want to generate the implementation file.
6. In the dynamic assembly editor, check **Enable Verbose Logging**.
7. Right-click **CustomerDetailsDA** and select **Generate Export => SCA Binding** (or **Web Service Binding**).

The Dynamic Assembler needs to extract information from the Context in order to select the appropriate endpoints. However, the Dynamic Assembler cannot directly act on the body of the request, it can only use the data in the Context. The Dynamic Assembler provides a set of plug-ins that are invoked at certain defined times in the life cycle of the request-response pair. One of these plug-ins is the ContextExtractor. This plug-in is invoked early in the life cycle of a Dynamic Assembler request. Its job is to look through the body of a request message and insert (or update) data items in the Context based on what it finds. In order to use the ContextExtractor, you need to add the Dynamic Assembler plug-ins to the build path of the project by doing the following:

1. Right-click the project, and select **Properties => Java build path**.
2. On the **Libraries** tab, select the **WebSphere Process Server v6.2** server run-time library, and click **Remove**.
3. Select **Server Runtime**, then click **Next**.
4. Select **WebSphere Business Services Fabric Server v6.2** from the list of run-time libraries.
5. Select a Java component from the palette and drag it onto the assembly editor so that you can add an interface to the component. Rename it to `CustomerContextExtractor`.
6. Wire the `CustomerContextExtractor` component to the `CustomerDetailsDA` component.
7. Right-click the `CustomerContextExtractor` component and select **Add => Interface**, then select **ContextExtractor**.
8. Double-click on the Java component to generate skeleton Java code that implements the `ContextExtractor` interface you added. Add the code shown in Listing 7 in the `extractContext()` method . As you see, we've added `CUSTOMER_TYPE_ASSERTION` (the required context) to the Fabric context. `CUSTOMER_TYPE_ASSERTION` is the ontology extension that you imported earlier in CustomerOntologyModel.zip.

## Listing 7. Implementation of extractContext() method

```
public Context extractContext(PendingRequest arg0)
   throws UnexpectedContentException {
  //TODO Needs to be implemented.
  /**
   * URI for state based assertion.
   */

  Context context = arg0.getContext();

      DataObject body = arg0.getMessageBody();
  DataObject request =
((DataObject)body.getSequence(0).getValue(0));
```

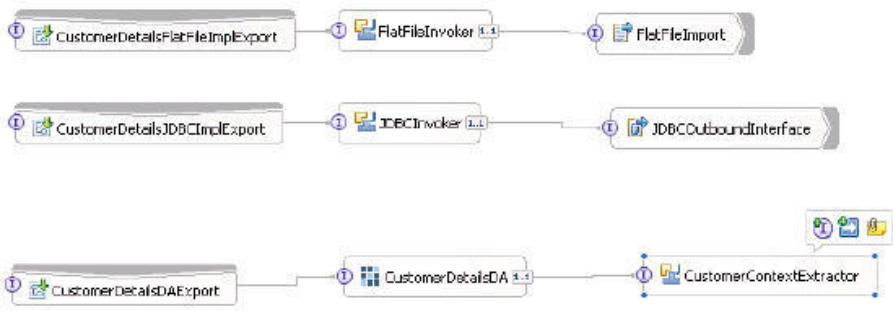
```
  //Print out the content of data object via SdoUtil provided
by Fabric
  System.out.println("request"+SdoUtil.printTree(request));


     DataObject genericRequest = (DataObject)
request.getSequence(0).getValue(0);

     System.out.println("genericRequest"+SdoUtil.printTree(genericRequest));

     //Retrieve customerType
  String input = (String) genericRequest.getString("customerType");

  System.out.println("Input received in extractContext is"
+ input);


  System.out.println("Context received is " + context);

  TypedValue assertionValue = new TypedValue(input);
  context.setSelectionProperty(CUSTOMER_TYPE_ASSERTION,
assertionValue);

  return context;


 }
```

For more information about extending the Dynamic Assembler using plug-ins like `ContextInjector` and `ContextExtractor`, refer to the Websphere Business Services Fabric Version 6.2 Information Center.

9. Publish the updated SCA module to the WebSphere Business Services Fabric unit test server.

The completed assembly diagram should look like Figure 19.

### Figure 19. Complete assembly diagram



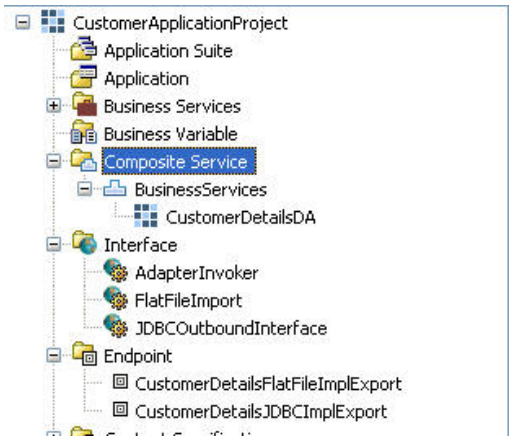

## Create the Business Service metadata for the CustomerApplication project

In this section, you'll create the composite service definition and policies for the Customer Application project.

1. Switch to the Business Services perspective. Right-click on the **CustomerApplication** Business Service project and select **New => Composite Service**.

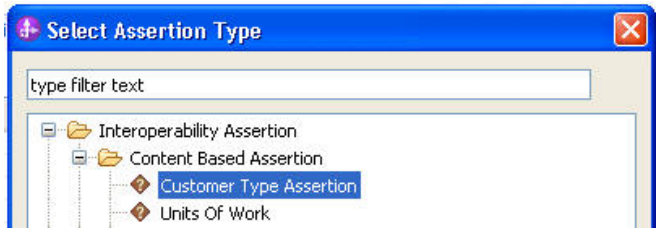
2. Specify `BusinessService` as the **SCA Project** and click **Finish**. The wizard creates metadata definitions for the composite service , atomic services and endpoints invoked by the composite service, as shown in Figure 20.

### Figure 20. Composite service definition




3. Add the points of variability supported by the endpoints. The points of variability were modeled as ontology extensions and imported in Fabric by importing the CustomerOntologyModel.zip. Double-click the **CustomerDetailsFlatFileImplExport** endpoint, then click on the **Assertions** tab and click **Add**.
4. In the Assertion Type dialog, select **Customer Type Assertion** and click **OK**, as shown in Figure 21.

### Figure 21. Select Customer Type Assertion

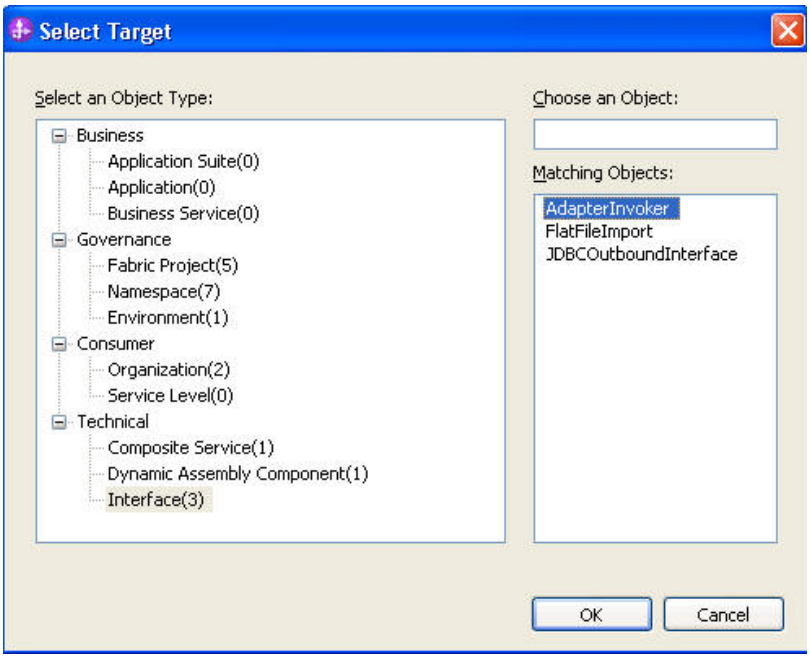



5. In the Customer Type Assertion dialog, check **Required** for **Assertion Options** and **EXISTING** for **Customer Type**, then click **OK**, as shown in Figure 22.

### Figure 22. Select assertion options and customer type

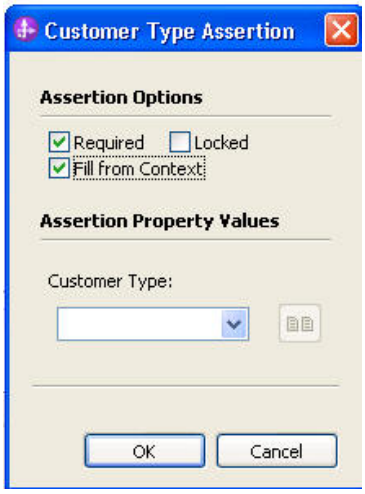

6. In the same way, specify the **Customer Type** value as**NEW** for the **CustomerDetailsJDBCImplExport** endpoint.
7. So far, you've assigned some capabilities to service endpoints. Next you need to defined policies to enable service endpoints to be selected based on the points of variability. A policy is nothing but a rule in the form *if{condition} then {expression}*. You want to define a policy such that, depending upon the customer type , the appropriate adapter implementation endpoint is invoked. To define a policy to accomplish this, do the following:
   1. Right-click **Policy** and select **New => Policy**.
   2. Specify `CustomerTypePolicy` as the name and click **Browse**.
   3. Click **Browse** next to **Target**, and select **Interface** and **AdpaterInvoker**, as shown in Figure 23, then click **Finish**.

### Figure 23. Select AdapterInvoker interface




4. After you've created the policy, select the **Policy Expression** tab,and click **Add**. In the Select Assertion Type dialog, select **Interopability Assertion => Content Based Assertion => Customer Type**, and click **OK**.
5. In the**Customer Type Assertion** dialog, check **Required** and **Fill from Context**, then click **OK**, as shown in Figure 24. If **Fill from Context** is checked and a value is supplied for an assertion property, that value is used as as the default if the property does not appear in the context.

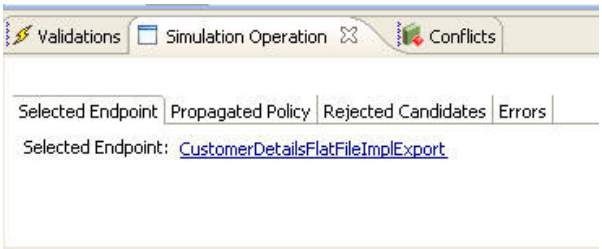
### Figure 24. Select assertion options




8. You can similuate the policy prior to deployment to ensure that the right service implementations are being picked up based on the context. To create a policy simulation, follow these steps:

1. Right-click **CustomerApplicationProject** and select **New => Simulation**.
2. Specify a name for the simulation and select the **CustomerDetailsDA**from the list of Dynamic Assembly components, then click **Finish**.
3. Click **Add Content-Based dimension** and select **Customer Type Assertion**.
4. Click **Edit** on **Customer Type** to specify the customer type value as **EXISTING**, then click **Run**. You'll see **CustomerDetailsFlatFileImplExport** is selected, as shown in Figure 25.

   **Figure 25. Simulation operation**

   Similarly, if you specified the **Customer Type** value as **NEW**, **CustomerDetailsJDBCImplExport** would be selected.

9. Now that the Business Service metadata is created, you need to submit the changes made in the workspace to the Business Service registry. Since you're using the Fabric UTE environment , these changes are automatically approved and published to the repository.

## Test the integrated solution

To test the solution, do the following:

1. Switch to the Business Integration perspective and right-click **CustomerDetailsDAExport**, then select **Test component**.
2. Specify the **NEW** for the **Customer Type** and **1** for the **Customer ID**. You'll see that **CustomerDetailsJDBCImplExport** is invoked. If you specified **EXISTING** for the **Customer Type**, **CustomerDetailsJDBCImplExport** would be invoked.

You have now successfully enabled adapter implementations that can be selected dynamically based on user context.

## Manage changes

A major requirement for this solution was the ability to seamlessly add new backend systems and integrate with third-party systems without changing the overall process. For instance, take an example where Company XYZ acquires another firm that uses an Oracle-based system for customer access. Following are the steps that would be required to integrate the Oracle backend with the existing solution:

1. Create a new Adapter for JDBC component to access the Oracle database. The JDBC Oracle adapter will implement the generic `AdapterInvoker` interface.
2. Add or extend the customer context.

3. Modify or create Fabric policies to specify the business context to use.
4. Publish the changes.

As you can see, using the Fabric policy-driven approach, you can introduce additional components without changing the core process.

Similarly, if the company needs to integrate with third-party printing systems that provide customer details, you would need to do the following:

1. Create a new component (such as a Web service or Java component) to access the third-party print service systems. The new component will implement the generic `AdapterInvoker` interface.
2. Add or extend the customer context, for instance, specify **XZYPrint** for the **Customer Type**.
3. Modify or create Fabric policies to specify the business context to use.
4. Publish the changes.

## Summary

In this article, you learned how to enable adapter implementations to be invoked dynamically using Fabric. Using the Fabric policy-driven approach, we provided an integrated solution that enables additional components to be introduced over time without changing the core process.

# Downloads

| Description | Name | Size |
| --- | --- | --- |
| Project files | CustomerApplicationProject.zip | 7KB |
| Project files | CustomerOntologyModel.zip | 4KB |

## Resources

- Business Process Management Samples & Tutorials Version 7.0: To build modules using WebSphere Integration Developer V6.2, refer to the Quick Start Scenario tutorials for WebSphere Adapters.
- Configuring and using adapters: This WebSphere Integration Developer V6.2 describes the adapters for Enterprise Information Systems (EIS) that can be configured to work with WebSphere Integration Developer.
- Getting Started with IBM WebSphere Business Services Fabric V6.1: This Redbook provides a complete overview of Fabric, from an architectural introduction, to an installation guide, and a step-by-step scenario that describes how to model, assemble, deploy, and manage composite business applications.
- Creating flexible service-oriented business solutions with WebSphere Business Services Fabric: This series of articles introduces you to WebSphere Business Services Fabric and shows you how to use it to build composite business services.
- WebSphere Business Services Fabric Version 6.2 Information Center: Get complete product information.
- developerWorks BPM zone: Get the latest technical resources on IBM BPM solutions, including downloads, demos, articles, tutorials, events, webcasts, and more.

## About the authors

### Anket Jain

**Anket Jain** is a Software Developer at the IBM India Software Lab and is currently working on development and customer support for file-based resource adapters. He has four years of experience working with various Java technologies including JCA.

---

### Vinay Roy Thykkuttathil

**Vinay Roy Thykkuttathil** works on the WebSphere Business Integration team at the IBM India Software Lab. He works on WebSphere JCA adapters and has more than two years of experience in the Business Integration domain.

---

### Naveen Balani

**Naveen Balani** works as a Software Architect on WebSphere Business Services Fabric. He likes to research on upcoming technologies and is a Master Author with IBM developerWorks, who has written on topics such as Web services, ESB, JMS, SOA, architectures, open source frameworks, semantic Web, J2ME, pervasive computing, Spring, Ajax, and various IBM products. Naveen is also a co-author of Apache CXF Web Service Getting Started (Packt Publishing, 2009), Beginning Spring Framework 2 (Wiley, 2007), and multiple IBM Redbooks on WebSphere Business Services Fabric and BPM V6.2 products.