HP OpenView Business Process Insight

For the Windows® Operating System

Software Version: 02.10

Integration Training Guide - Modeling Flows

Legal Notices

Warranty

The only warranties for HP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HP shall not be liable for technical or editorial errors or omissions contained herein.

The information contained herein is subject to change without notice.

Restricted Rights Legend

Confidential computer software. Valid license from HP required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Copyright Notices

© Copyright 2004 - 2007 Hewlett-Packard Development Company, L.P.

Trademark Notices

Java™ is a US trademark of Sun Microsystems, Inc.

Microsoft® is a US registered trademark of Microsoft Corporation.

UNIX® is a registered trademark of The Open Group.

Windows® and MS Windows® are US registered trademarks of Microsoft Corporation.

Documentation Updates

This manual's title page contains the following identifying information:

- Software version number, which indicates the software version
- Document release date, which changes each time the document is updated
- Software release date, which indicates the release date of this version of the software

To check for recent updates, or to verify that you are using the most recent edition of a document, go to:

http://ovweb.external.hp.com/lpe/doc_serv/

You will also receive updated or new editions if you subscribe to the appropriate product support service. Contact your HP sales representative for details.

Support

Please visit the HP OpenView support Web site at:

http://www.hp.com/managementsoftware/support

This Web site provides contact information and details about the products, services, and support that HP OpenView offers.

HP OpenView online software support provides customer self-solve capabilities. It provides a fast and efficient way to access interactive technical support tools needed to manage your business. As a valuable support customer, you can benefit by using the support site to:

- Search for knowledge documents of interest
- Submit enhancement requests online
- Download software patches
- Submit and track progress on support cases
- Manage a support contract
- Look up HP support contacts
- Review information about available services
- Enter discussions with other software customers
- Research and register for software training

Most of the support areas require that you register as an HP Passport user and log in. Many also require a support contract.

To find more information about access levels, go to:

http://www.hp.com/managementsoftware/access_level

To register for an HP Passport ID, go to:

http://www.managementsoftware.hp.com/passport-registration.html

Contents

1 Introduction

This chapter looks at the overall picture that you are trying to achieve by modeling your business flow using OpenView Business Process Insight (OVBPI).

Overall Architecture

You can think of a running OVBPI system as having the following major parts:

- The Business Flow

  This is the high-level flow that represents a particular business process within your organization. This flow is created using the OVBPI Modeler and is then deployed to the OVBPI Engine.

- The OVBPI Business Impact Engine

  The Business Impact Engine runs your deployed business flow. For the Business Impact Engine to maintain anything interesting about your flow (such as: the number of orders coming through your application systems, who the customers are that are placing orders, etc.) you need to be able to feed information about the data activities happening in your underlying applications. That is, you need some "data feeds" from your actual applications that are processing and running your business.

  Remember, when you define a business flow and deploy it into the OVBPI Business Impact Engine you are not using it to run your business. This flow is being used to monitor your business, and so, to be able to monitor your business the Business Impact Engine needs to have data feeds.

- The Data Feeds

  The data feeds provide a stream of "events" to tell the Business Impact Engine what is happening in your real world.

  For example, you might have a data feed to signal whenever someone places a new order over the Web site. You might have another data feed to signal when this order has been shipped. These events then enable the OVBPI Business Impact Engine to maintain instances of your business flow for each order, and to show you where each order is within the defined flow.

  These event feeds can come from databases, files, application systems, Web sites, etc.. They need to be configured by someone who understands the underlying data sources and the OVBPI Event Handler. Configuration these event feeds is covered in detail in the *OVBPI Integration Training Guide - Business Events*.

- The IT Infrastructure Status Feeds

  Once you have configured the OVBPI Business Impact Engine to monitor your business flow and collect a whole load of information, that might be all you need. That might tell you what you need to know and help you see more clearly your business volumes, and where things are taking time within your business, etc.. But there is a further step you can take with OVBPI.

  You can configure the OVBPI Business Impact Engine to take direct "status feeds" from products such as OpenView Operations (OVO), OpenView Internet Services (OVIS) and/or Service Oriented Architecture (SOA) Manager. You can configure, for all or some of the steps within your flow, that they have dependencies on the underlying services provided by your IT. Thus if there is an IT service failure, the OVBPI Business Impact Engine is able to show you how much business is affected by this failure, and thus the cost of this failure to your business.

A running OVBPI system might look something like this:

Figure 1   High-Level View of Components
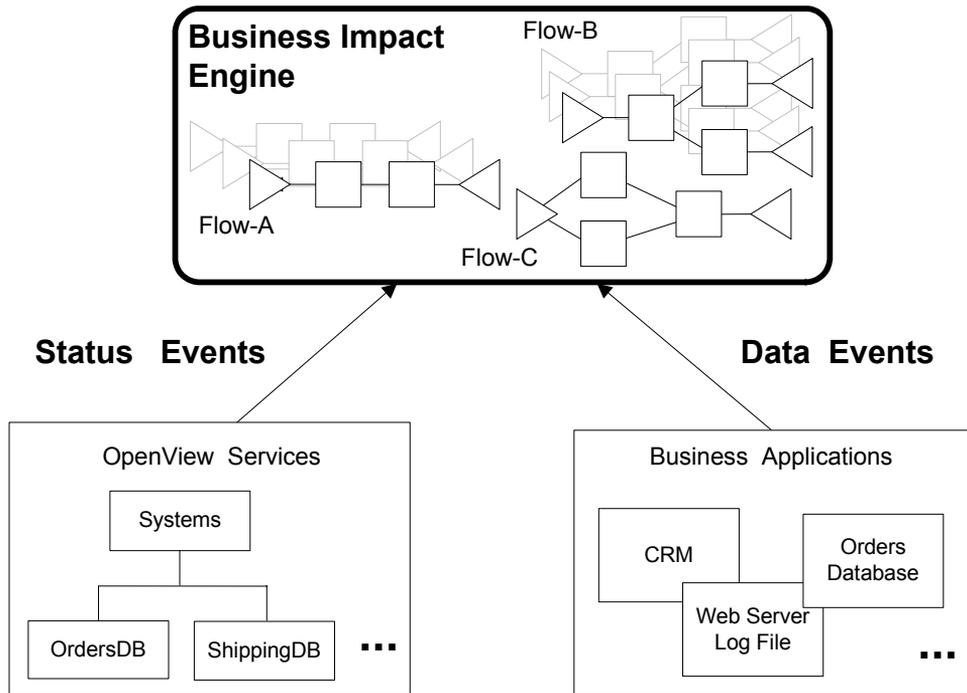
where:

- A number of different flows have been defined and deployed into the Business Impact Engine

- Status feeds are coming in from OpenView advising the Business Impact Engine of any IT failures

- Data feeds are coming in from the underlying databases and application systems advising the Business Impact Engine of the data activity happening within these applications

- Notice that, as well as showing multiple flows, the diagram shows multiple instances of Flow-A

  If Flow-A is a business flow that represents (for example) "Vendor Payments" then there is a separate instance of the flow for each active vendor payment being handled. One flow instance for each vendor payment.

  At any one time the Business Impact Engine may be maintaining many parallel instances of any deployed flow.

Let's now consider this in more detail...

What Are You Trying to Achieve?

Before jumping in and looking at the actual tools for creating flows and setting everything up, let's consider the above architecture in more detail. Specifically let's look at how things work when everything is in place and configured.

How Data Events Progress a Flow Instance

Suppose you have set up the following:

- You have defined a flow that tracks orders through your business
- For this flow you have defined the data that you would like the flow to maintain for each order

  For example:

  — `Order_ID`

     As assigned by the orders application system.

  — `Value`

     The actual dollar value of the order.

  — `Cust_ID`

     The ID of the customer who has placed this order.

  — `Credit_Status`

     Assigned true/false based on the customer's credit status.

  — `Ship_ID`

     The ID assigned by the shipping system when the order is shipped.

- You have then defined how the flow knows where it is, based on values within this set of data attributes. (These are called **progression rules**.)

  For example:

  Once the `Credit_Status` field contains a value then you must have completed the "Check Credit" step (or node) in the flow. If the `Credit_Status` value has changed to "false" then you must have entered the "Credit Check Failed" node within the flow.

- You have then deployed this all into the OVBPI Business Impact Engine.

- Someone needs to have configured the underlying application systems to send in data events whenever new orders come in from the orders application.

- And the system runs something like this...

Figure 2   Application Data Feeding the Orders Flow
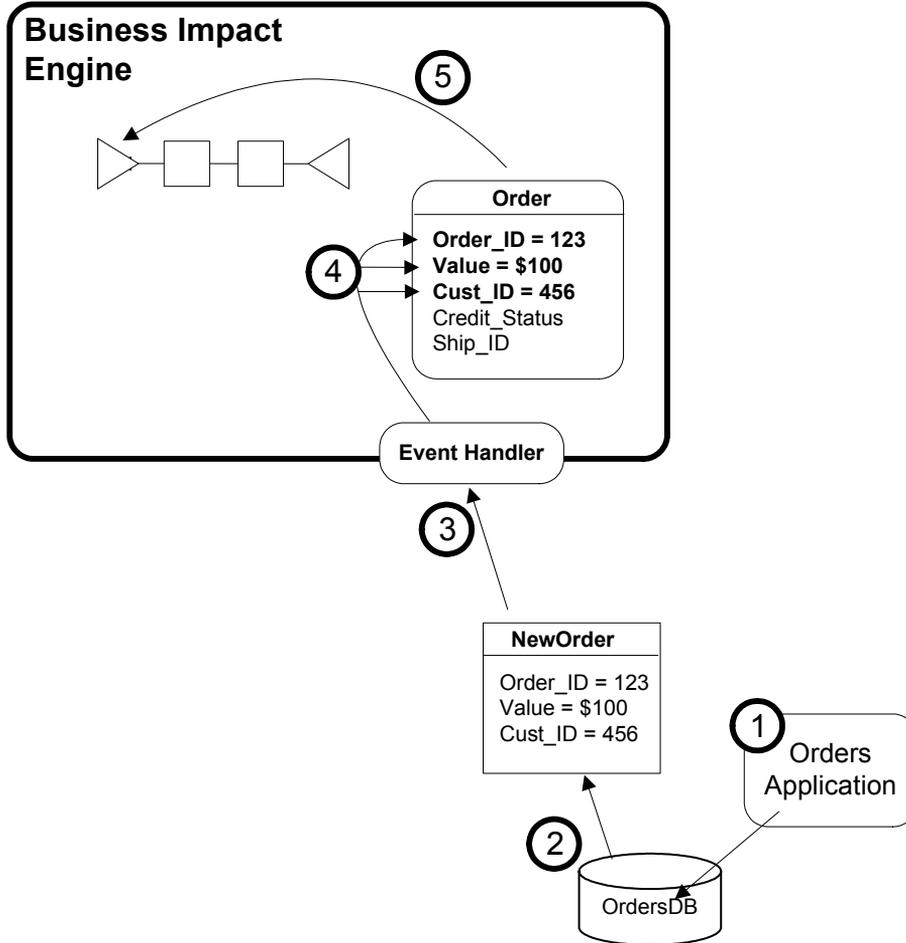
where:

1. Someone rings up and places an order with your orders department.

   The person taking the order, enters it into the orders system, using the orders application that they have always used. The system then assigns an order number - for example: order number `123`.

   The orders application then stores this order into its database.

2. Whenever your users accept a new order you want to trigger an event to be sent into the OVBPI Business Impact Engine to say that a new order has been placed.

   You would like to send an event into the OVBPI Business Impact Engine specifying the order number (in this case `123`), the dollar value for this order (for example: `$100`), and the customer ID that has placed this order (for example: `456`). You do not have to include every detail about the order - just the key details that are useful when producing reports. In this example you are mainly wanting to be able to report the number of orders and their value. By including the customer ID with each order this allows any reports to look up customer details (such as `Name`, `Address`, `Phone`...) direct from your existing customer database system.

   Triggering an event for each new order is something that needs to be configured into the orders database. This requires someone who understands the orders application and database structure such that they are able to configure in an SQL trigger on the orders table. Configuring this trigger is not difficult and should not interfere with the existing orders application system.

3. Once the trigger is configured you need to set up an Adapter to monitor this trigger output and to send the data into the OVBPI Business Impact Engine as a properly formed "event". Full details on how to configure such adapters are found in the *OVBPI Integration Training Guide - Business Events*.

   So, the phone orders clerk accepts a new order, enters it into the orders system. The orders system is configured (within its database) to produce a trigger which is monitored by an Adapter - the net result being that a data event is sent into the OVBPI Business Impact Engine (passing through the OVBPI Event Handler component).

4. When designing a business flow, you lay out the steps that your flow goes through. For example: your flow might be describing the path that your orders take as they are processed within your company. You might accept a new order, carry out a credit check and then ship the product.

   Clearly you want the OVBPI Business Impact Engine to monitor each order individually. Thus the OVBPI Business Impact Engine can instantiate a new flow instance for each order that comes into the system. To do this, when you define a business flow you define a set of data attributes that you want the OVBPI Business Impact Engine to maintain for each instance of the flow.

   In the diagram you see that, for each instance of the flow, you are maintaining the set of data attributes:

   ```
   Order_ID
   Value
   Cust_ID
   Credit_Status
   Ship_ID
   ```

   Thus when the Business Impact Engine receives the event saying that a new order has occurred, it instantiates a new instance of the flow and stores with it the data from the event.

5. Once the data event has come into the Business Impact Engine, and data for that instance has been allocated and updated, the Business Impact Engine then uses that data to decide whereabouts in the flow diagram this instance actually is.

   When you defined the flow you also defined progression rules that determine where you are in the flow diagram based on the values within the data for this flow.

   For example, in the above diagram the progression rules would simply say that when a new order comes in, the flow is positioned at the first node. In other words, when a new order comes in, the first node within the flow is considered "active".

Hopefully you can see that with OVBPI you can design a flow and the OVBPI Business Impact Engine can then monitor all instances of that flow. In this example, you are monitoring every order that is coming in - monitoring its value and the customer ID.

For this to work, someone had to configure the underlying orders application to be able to tell you when new orders have appeared.

Suppose you have also configured your shipping system to be able to tell you every time an order is shipped. When an order is shipped you would have the following scenario:

Figure 3   Shipping Updates Feeding the Orders Flow
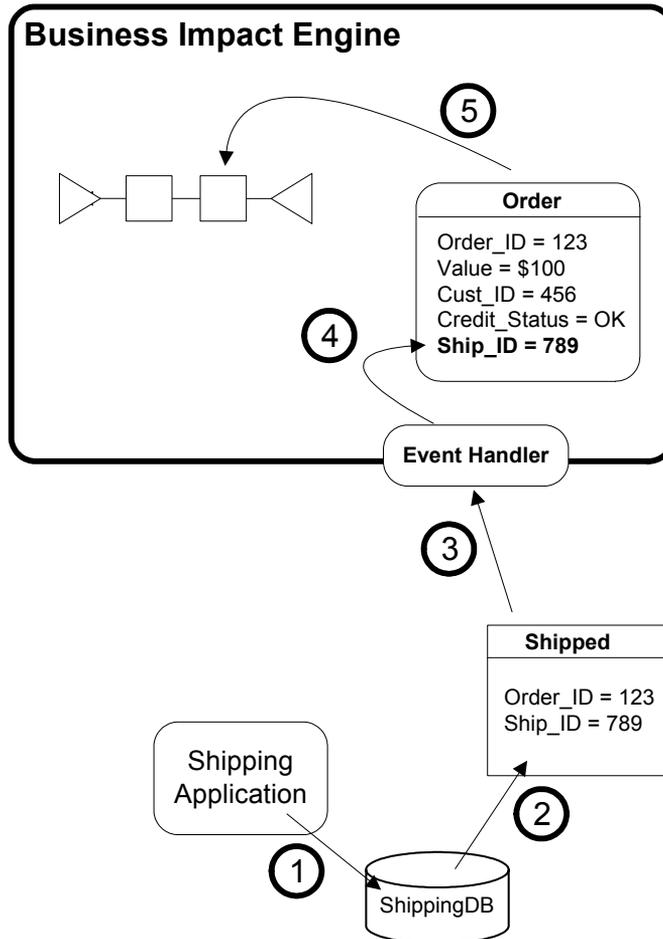
where:

1. The shipping department runs their usual application software to mark that an order has been shipped.

2. This writes to (probably) a database which you have configured to fire a trigger whenever orders are shipped.

3. You have an adapter configured to receive these triggers and this adapter sends a data event containing the order ID and the corresponding shipping ID.

4. When the OVBPI Business Impact Engine receives this event it matches it to the existing order object for `Order_ID=123` and updates the `Ship_ID` attribute to the value from the incoming event (for example: `789`).

5. The OVBPI Business Impact Engine then applies the progression rules defined for each node in the flow to see if this change to the data attributes causes the flow instance to move to a new node.

   In this case, the assignment of the `Ship_ID` would cause the flow to say that the node called "Ship Order" is now complete.

So you have the OVBPI Business Impact Engine monitoring the orders as they move through your business. This is excellent! You can now use the OVBPI Business Impact Engine statistics to generate reports to show information such as the volume of orders each day, the relative time that an order spends being shipped versus the time it takes to check the credit status of a customer...or whatever.

But you can also take this one step further...

Adding Service Status Feeds

The OVBPI Business Impact Engine is able to accept status feeds from your IT Infrastructure. So in your example, if the shipping database becomes unavailable you want the Business Impact Engine to tell you the volume of orders that are affected.

It looks something like this:

Figure 4   Receiving Service Status Feeds
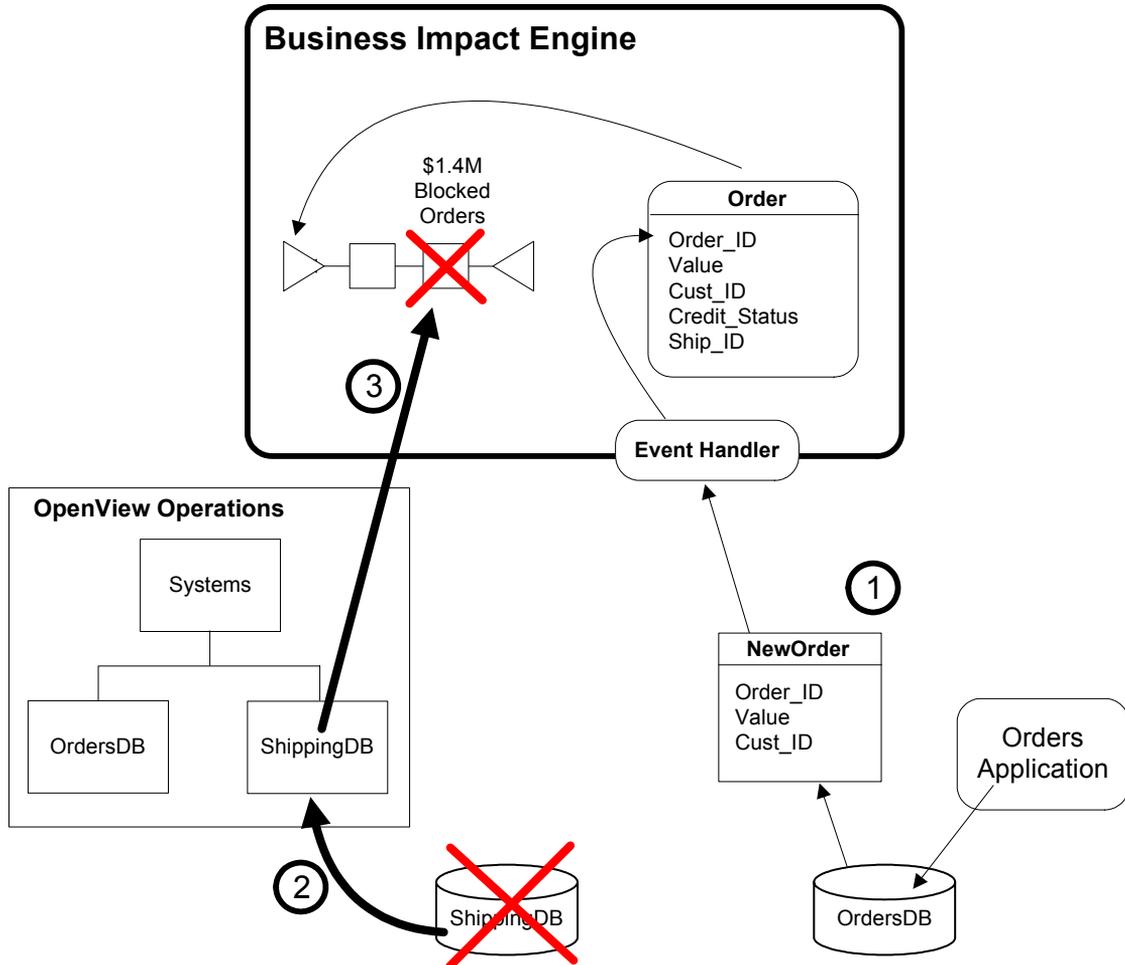
where:

1. New orders are being entered into the system and monitored by the OVBPI Business Impact Engine.

2. The shipping database is being monitored by (for example) OpenView Operations. The shipping database suddenly becomes unavailable.

3. A status event is sent into the OVBPI Business Impact Engine to say this service is unavailable. All nodes that are marked as being dependent upon that service are flagged as being unavailable.

   The OVBPI Business Impact Engine is then able to determine which flow instances are affected by this service outage and you are able to see the impact that this is having on your business. For example, you can see the volume of orders waiting to be shipped, the list of affected customers, etc..

So, by adding the service status feeds you are not only able to have OVBPI monitor your business volumes and throughput, but you are able to report actual business impact when any underlying IT services become unavailable.

The Major Steps

Let's now start to generalize the example that you have just walked through and outline the main steps that are needed to monitor your business flows.

The major steps are:

- The business flow

    — Define the flow diagram

    — Define the associated data attributes to be maintained for each instance of this flow

    — Define the progression rules that allow the flow to determine where it is, based on the state of the associated data

- The linkage to the business data - "data events"

    — Define the data events that are to be received from the underlying data applications

    — Actually configure these events into the underlying data applications such that they are being generated correctly

    — Configure adapters to monitor these events and send them into the OVBPI Business Impact Engine

- The linkage to any underlying IT infrastructure - "status events"

    — For each node in the flow, specify which IT service that node is dependent upon

- Deploy the flow to the OVBPI Business Impact Engine

- Monitor the Business Impact Engine statistics

Let's just talk a bit further about a couple of these points...

Monitoring The Business Impact Engine Statistics

The Business Impact Engine monitors your business flow. It maintains the data associated with each flow instance. It is also able to record when a node becomes impacted due to an IT service becoming unavailable. The question is: "How do you see this?"

You can use the **OVBPI Business Process Dashboard** to view these flow statistics.

OVBPI Business Process Dashboard

The OVBPI Business Process Dashboard is a set of Java Server Pages (JSPs) that present most of the Business Impact Engine statistics to the screen. The OVBPI Business Process Dashboard allows you to view each flow that is deployed to the OVBPI Business Impact Engine, and to drill down to see each instance of a flow. It also shows you the associated flow instance data and impact information. The OVBPI Business Process Dashboard is shipped as an example dashboard. You have access to all the code so you can extend/enhance the Business Process Dashboard in whatever way you like.

To run the OVBPI Business Process Dashboard, point your Web browser at the URL:

```
http://server:44080/ovbpi-bpd
```

where:

- *server* is the name of the server on which OVBPI is installed
- *44080* is the default port number on which the Tomcat servlet engine is running (on the OVBPI server)

Refer to the *OVBPI Integration Training Guide - Customizing the Business Process Dashboard* for further details about the OVBPI Business Process Dashboard and how you can customize it if you so require.

The Linkage To The Business Data - "Data Events"

This is the step where you define the set of events that are to come in from the underlying data applications. These events feed into the Business Impact Engine and provide the data to update the flow instance, and thus enable the flow instance to determine where it is within the flow diagram.

When you are creating the flow definition within the OVBPI Modeler you need to define the events that are to come into the Business Impact Engine at run time, and how you want them to affect your flow. But someone actually needs to go and implement these events. That is, someone must physically set up some adapters to listen on the underlying application systems and create these data events.

So when it comes to defining the list of events that are going to come into your flow there are two parts to it:

1. The flow needs to know the list of incoming events

   This includes things such as the event name and the attributes within the event - their name and type, etc.

2. Adapters need to be set up to produce these data events

   Someone needs to go to the underlying application systems and determine where and how to trigger the necessary information, and then set up an adapter to generate this information as an OVBPI data event.

The obvious question is: "Which do you do first?"

Do you first configure into your underlying application systems every event that you think any flow will ever need, and then simply refer to these within the flow modeler? This the "bottom-up" approach. Or do you first define the list of events that you believe you need to drive your flow and then go out to the application systems and get these events configured? This is the "top-down" approach.

The answer is that you can do it either way. However...

Bottom-Up Event Definition

The bottom-up approach probably appears to be the most logical. If you first define all the required events within the data applications, then it becomes quite easy to define the flow as you know what events you are going to receive.

However, if you just go to your application people and say "Would you please define a whole load of events and have these generated every time something important happens in our business" they will probably not know what to do. They could no doubt invest the next year configuring every event imaginable...but would that be a good use of time? Probably not.

Top-Down Event Definition

It is probably better to consider the top-down approach. This means that the business flow designer, whilst designing a specific flow, thinks carefully about the kind of data events that they would need to drive this flow. They can then go to the applications people and say "Would you please see if you can generate these events from our data systems." The application people now have a specific set of events on which to focus their attention.

It may be that the application people determine that they cannot implement every event exactly as specified - in which case the flow designer needs to sit down with them and discuss alternatives and possibly make alterations to the flow design. But this helps everybody focus on the minimum set of events required to drive the business flow.

Suggested Methodology

Here is a suggested order in which to carry out the tasks of designing a business flow:

1. **Design the business flow**

   Use the OVBPI Modeler to draw out your business flow.

2. **Define the data the flow maintains for each flow instance**

   Define the data attributes that you wish to maintain for each flow instance.

3. **Define the progression rules**

   Define the way a flow instance progresses as the data attributes change state during the life of the flow instance.

4. **Define the data events**

   Define the events that you expect to receive from the underlying application systems. Define the name of each event and its data content.

   You then need some consultation with the applications people to see if these events can be generated as you expect. For more details on how to generate these actual events refer to the OVBPI *Integration Training Guide - Business Events*.

5. **Define any IT service dependencies**

   For each node in the flow, consider whether the node is dependent upon any (zero or more) IT services.

6. **Deploy the flow and test it**

   Use the OVBPI Modeler to deploy this flow definition into the OVBPI Business Impact Engine. You can use tools such as the Generic Event Injector (contributed utility) and the OVBPI Business Process Dashboard (both described later in this manual) to test out the behavior of your flow, before then connecting in the actual application system adapters to generate the real data events.

This is just a suggested order for defining flows. You might find that your preference is to interchange steps 3 and 4, and define the events before defining the progression rules. Some people might prefer to specify the service

dependencies straight after step 2. It actually doesn't matter. Each person can develop the order that best suits them. But you can use the above steps as your starting point.

Be aware that this process is iterative. You will revisit steps (perhaps multiple times) as you discover more about your flow and as you determine more about the data that comes from the underlying application systems.

2 Flow Definition

This chapter looks at the OVBPI Modeler and the steps to defining a business flow.

Understand the Business Process

The first thing that you need to do is gain an understanding of the actual business process you wish to monitor.

Basic Analysis

Business processes are typically very complex, but your aim is to identify the main phases that occur during the life cycle of the business process.

Remember you are wanting to **monitor** the process, not **run** it.

So your aim is to try and produce a high-level summary of the underlying business process. This high-level summary is called the **business flow**.

For example, you might be wanting to monitor the company's order fulfillment process. Now there are probably many many steps and departments involved in handling an order. But for monitoring purposes you may identify that the main phases for your orders are:

- The order comes in
- A credit check is carried out
- If OK, the order is picked in the warehouse
- The order is then shipped

This then allows you to monitor things such as:
- The volume of orders for each day
- The value of your orders
- Average time to credit check clients
- Percentage of failed credit checks
- Warehouse time to pick an order
- and so on

The idea is to understand the process but express it in simplified terms.

The good thing about defining a business flow is that it is not actually running your business process, it is monitoring it. So if you create a flow and you later find that it does not capture enough monitoring statistics to really be of interest…that's OK! You can simply extend your flow definition and try that. Indeed, you might define a simple flow, deploy it to OVBPI and when you see the reports you decide that your business process is running just fine and you decide that it is not worth any more investment to monitor that particular flow, allowing you to focus your time monitoring other parts of your business. Or, you may find that the flow does indicate some issues and you decide that it is worth investing time to extend the flow definition and gather more statistics about your business.

The key here is that your flow should start out simple - with only a few nodes - and if it turns out that you need to gather more detailed statistics…you can! The flow can always be extended if you so desire. You do not have to get the flow right the first time!

As you are gaining an understanding of the business process, it is a good time to be finding out about the underlying application systems. The person/people responsible for setting up the OVBPI business event adapters that are to generate the data events to drive this flow, needs to know the possible places where these events might come from. It's a good idea to make notes about the possible data sources/systems for each node within your flow.

Another important step is to keep asking yourself "What statistics do you want to get from OVBPI when this flow is deployed and running?" This helps you to decide what data you want the flow to maintain for each instance. This also helps to define what data properties you need to get from the underlying application systems.

So, important steps:

- Gain an understanding of the actual business process and produce from this a high-level business flow to represent the main phases of the process.

- Ask yourself the question:

  "What statistics do I want to get from this flow?"

- List out the possible sources for data events for each node within your flow.

Business Process Execution Language (BPEL)

You may find that the customer has some, or all, of their business processes defined within a third party business process design tool. In which case they may be able to provide the process definition as a BPEL export file. The OVBPI Modeler is able to import BPEL processes and create a corresponding flow definition.

Importing the BPEL process can provide you with a good starting point when trying to create the business flow. However, the BPEL may well contain so many process steps that it might, at first, appear as an overwhelmingly large and detailed flow diagram. You typically need to go through the resultant flow definition and refine it to produce your high-level business flow.

Even if you do import the BPEL process, it only provides you with the flow diagram. You still need to define the required data feeds and data definitions for this flow to be able to monitor the underlying process. So you still need to ask yourself the questions outlined in the section Basic Analysis on page 30

For details about importing BPEL processes, refer to the *OVBPI Integration Training Guide - Working with BPEL*.

Starting The OVBPI Modeler

To run the OVBPI Modeler:

```
Start->Programs->HP OpenView->Business Process Insight->Modeler
```

or... you can run the OVBPI Modeler executable directly, by running:

```
OVBPI-install-dir\bin\biamodeler.bat
```

Logging On

When you run the OVBPI Modeler, it prompts you for a login user name and password:

Figure 5   OVBPI Modeler Logon Screen

When OVBPI is installed, a default user name and password is configured for you:

```
    User Name:   admin
    Password:    ovbpi
```

(To change the admin user password, and to configure the underlying authorization mechanism, refer to the *OVBPI System Administration Guide*.)

If you are running the OVBPI Modeler on a different machine to the OVBPI Model Repository, you are able to specify the hostname of the OVBPI Model Repository by clicking on the Options... button.

If the login fails, then the most likely causes are:

- An incorrect user name or password

- An incorrect hostname (specified under the Options button)

- The OVBPI Model Repository and OVBPI Servlet Engine components are not yet started on that hostname

  You can use the OVBPI Administration Console, on the OVBPI server, to start the Model Repository and the Servlet Engine.

- An incorrect port number (specified under the Options button)

- There is a network/security reason causing the connection to fail. For example, a firewall on the server might be disallowing you to connect.

When you enter the user name and password details, and click OK, the OVBPI Modeler runs and presents a screen something like this:

Figure 6   The OVBPI Modeler
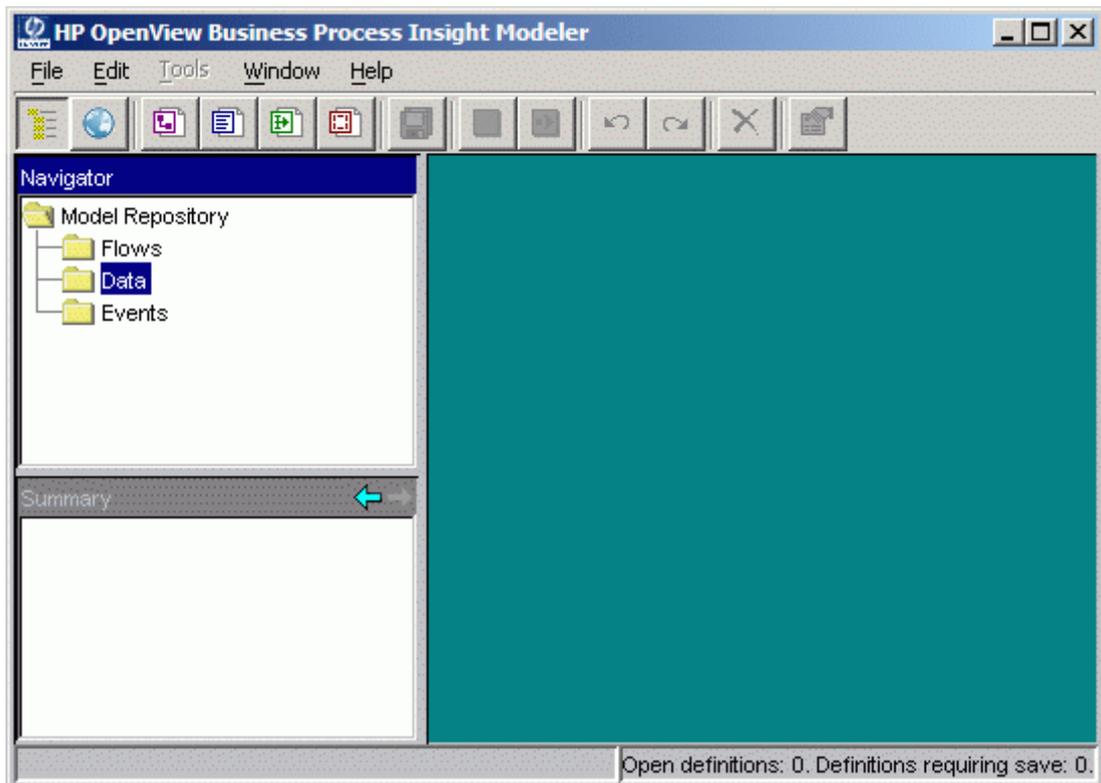
Obviously if you had previously defined some flows then these would be listed in the Navigator pane (on the left hand side).

The Model Repository

When the Modeler runs, it is connected to the OVBPI server. As you create new flows, these are saved on the OVBPI server...in the **Model Repository**.

The Model Repository is where all of your definitions are maintained. This includes your flows, all data definitions, events definitions, etc. - anything that you create in the OVBPI Modeler.

You do not actually need to know where the Model Repository is on the server, but if ever you do, it is held within the OVBPI database. The Model Repository is held in a set of data tables whose names all start with `Repos`. The main data table is: `Repos_Defns`.

Do not edit or change any of the entries in the repository data tables as you may well cause the OVBPI Modeler to crash or corrupt the repository.

Logon Users

Although you must logon to the OVBPI Modeler as a configured user, once you have logged on to the Modeler you are able to access everything within the repository.

Indeed, suppose you logon today as a user called (for example) `Fred` and create a flow definition called `MyFlow`. If you logon tomorrow as a different user (for example, `admin`) you are able to continue working on `MyFlow`.

Your login user does not restrict you within the OVBPI Modeler.

To configure the underlying authorization mechanism used by the OVBPI Modeler, refer to the *OVBPI System Administration Guide*.

Single User Repository

Although multiple developers can view the Model Repository, OVBPI currently only supports one developer actually editing the Model Repository at any one time.

If you do have multiple users writing to the OVBPI Model Repository at the same time, whoever saves their changes last overwrites the other developer's changes.

So be careful!

Drawing The Flow

Now that the Modeler is running and connected to the OVBPI server, you can start to design (model) your business flow.

Creating A New Flow

To create a flow, you can either select

```
File->New->Flow
```

or simply click on the new flow icon:

Create a new flow...

This brings up the flow properties page - in the right hand pane of the OVBPI Modeler.

You can then assign the flow a name and (optionally) a flow description.

Drawing the Nodes Of The Flow

Once the flow is named, you can switch to the flow editor pane...by clicking on the `Flow Editor` tab, as shown:

Click here to start designing your flow...
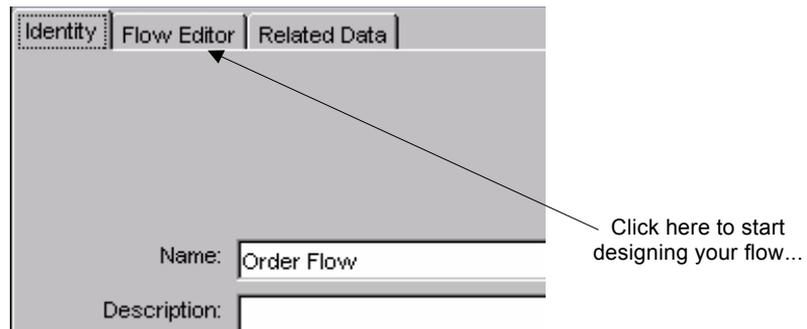
This pane then switches to become the flow editor canvas. It appears as a blank area with a selection of buttons/icons at the top, as follows:
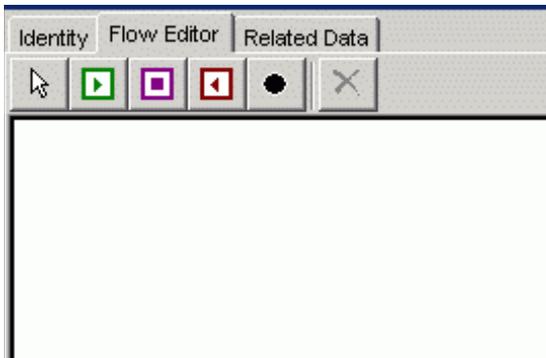
These drawing icons perform the following functions:

	Draw a Start Node To draw a start node you first click on this icon and then move the cursor to the desired position within the flow definition pane and single click. This paints a start node at that position. The cursor stays in this "start node" mode, allowing you to paint more start nodes. You select another icon to switch out of this mode.
	Draw an Activity Node To draw an activity node you first click on this icon and then move the cursor to desired location within the flow editor pane and single click. This paints an activity node at that position.
	Draw a Junction Node To draw a junction node you first click on this icon and then move the cursor to desired location within the flow editor pane and single click. This paints a junction node at that position.
	Draw an End Node To draw an end node you first click on this icon and then move the cursor to desired location within the flow editor pane and single click. This paints an end node at that position.
	The Selection Tool Icon Clicking this icon switches you out of "node drawing" mode and allows you to draw arcs between nodes and define node properties.
	Delete Nodes or Arcs Clicking this icon deletes the currently highlighted object(s).

Start Node

A start node defines where you expect this flow to start.

You can have more than one start node.

Indeed, there is nothing stopping you from having no start node at all. This is because you can configure a normal activity node to start a flow if that flow has not already been started. So a start node is really there to help you layout your flow to be as human-readable as possible.

End Node

The end node defines where you expect the flow to come to an end.

You can have more than one end node.

Although you technically do not need to have an end node, you really should! The end node is used by the OVBPI Business Impact Engine to signal that a flow instance has completed. Without an end node your flow instances can never complete.

Junction Node

A junction node can be placed anywhere within a flow diagram. You can have more than one junction node within a flow diagram. You may choose to have no junction nodes within your flow. Junction nodes are optional.

You may use a junction node to represent a step within the flow that you are choosing not to monitor. You may use a junction node to represent a decision that takes place within the underlying flow. Some people may find that using junction nodes simply helps them layout their flow diagram. Junction nodes are purely there to aid with the visual layout of your flow diagram.

An example of where you may see junction nodes used, is when you import a Business Process Execution Laguage (BPEL) process into the OVBPI Modeler. In this instance, junction nodes are used to help show the structure of the underlying business process being represented by the created OVBPI flow diagram.

Placing The Nodes For Your Flow

To create a start node:

- Click on the start node icon

- Move the cursor to the location where you wish to place this start node, and single click

To create a couple of activity nodes:

- Click on the activity node icon

- Move the cursor to the location where you wish to drop the first activity node...and single click

- Move to the location you wish to drop the second activity node...and single click

To create an end node:

- Click the end node icon

- Move the cursor to the location where you wish to place the end node...and single click

To **stop dropping nodes** (and turn your cursor back to a normal cursor):

- Click on the selection icon (the arrow head)

For example, your flow might look something like this:
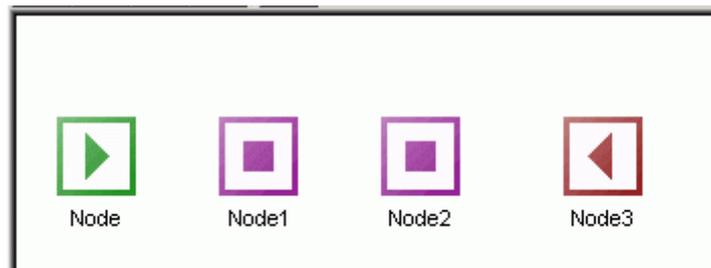
Drawing Connecting Arcs

To connect two nodes (with an arc) you need to have first **clicked the selection tool** icon (arrow head).

To draw the arc, you:

- Move the cursor to be over the center of the first node

  As you move the cursor over the node, the center of that node is highlighted. You must make sure that the cursor is then placed over that highlighted center part of the node. The cursor changes to a simple "cross".

- You then left-click-and-hold the mouse
- You then drag the mouse to be over the second node
- Then let go of the mouse, and the arc is drawn

Your flow might now look as follows:
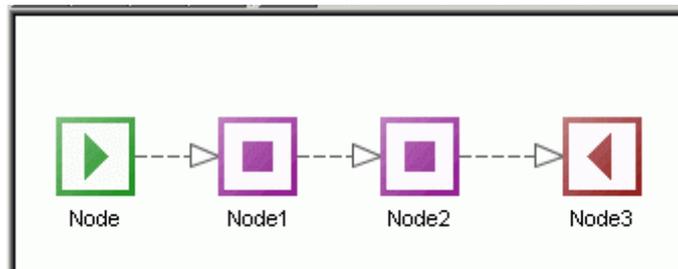
Moving Node Positions

To move a node you need to have clicked the selection tool (arrow head icon).

To move a node around on the flow picture, you:

- Move the cursor to be somewhere over the node - but **not** within that center section of the node

  As you move the cursor over the node, the center of that node is highlighted. You must make sure that the cursor is **not** placed over that highlighted center part of the node. Remember, the central part of the

node is used when you are wanting to draw an arc. For moving a node you aim for the edge of the node. The cursor changes to a cross with arrow heads.

- You then click-and-hold the mouse
- You then drag the node to its new location on the flow editor pane.
- Then let go of the mouse, and the node has now been moved

Naming Each Node

The OVBPI Modeler assigns default names for any newly added nodes.

To change the name of a node you can:

- Right-click on the node and select `Properties`

...or...

- Double-click on the node

This brings up a dialog box as follows:
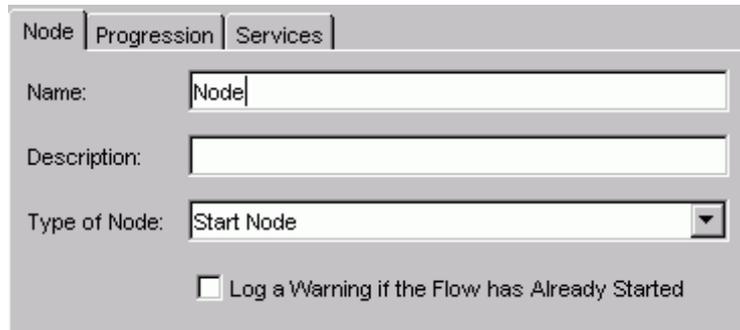
You can then enter the name of the node, and (optionally) a description.

The other tabs on this dialog allow you to set some important things such as the progression rules and service dependencies for this node.

The progression rules are where you define when a flow instance has started or completed a particular node. You define these later on but, when you are drawing out each node of your flow it is a good idea to be thinking about how you know when an instance has started or completed this node. That is, start to think and make notes to yourself about the conditions under which you

know that a flow instance has reached this node and how the instance completes this node. You define these within the OVBPI Modeler...but not just at the moment :-)

Lab - Drawing The Flow

In this lab you create a flow, draw some nodes and then link them with arcs.

Start up OVBPI

- Run the OVBPI Administration Console
- Start up **all** the components

Start the OVBPI Modeler

- Start up the OVBPI Modeler
- Create a new flow called: `Order Flow`
- Create a start node, called: `New Order`
- Create an activity node, called: `Process Order`
- Create another activity node, called: `Ship Order`
- Create an end node, called: `End`
- Now join these nodes with arcs so that your flow ends up looking something like this:
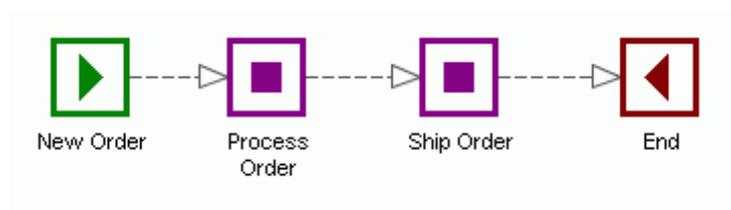
- Now save your work

  `File->Save All`

  or... press on the `Save All` icon in the OVBPI Modeler's main toolbar.

Well done! You have reached the end of the lab.

Defining The Associated Data

Once you have drawn out your flow, you need to define the data properties that you want this flow to maintain. Whilst it is easy to think of this data as being specific to your flow instance, the OVBPI Modeler is written such that the definition of the data is actually separate from the flow definition. This is done so that you can define more than one flow to be driven from the one set of data properties.

To define a new data definition you can

```
File->New->Data
```

or simply click on the new data icon:

Create a new data definition.

This opens up the data definition dialog in the right-hand pane of the OVBPI Modeler.

You specify the name of this data definition (for example: `Order Data`) and (optionally) a description.

The name of the data definition can include "/" characters to provide logical grouping. So your data definition could be called: `Orders/My Data`. This might be useful if you are planning on defining many flows all requiring data definitions, as it enables the data definitions to be listed in logical "groups".

The Data Properties

You then need to define the data properties/attributes that make up this data definition.

To do this, click on the Properties tab in the right-hand pane:
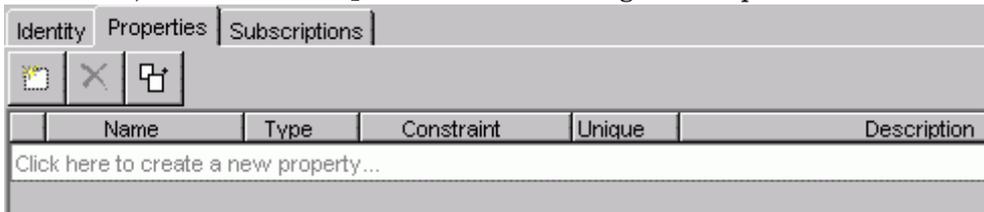
Where:

	New Data Property To create a new data property you can either click in the table where it says "Click here..." or you can click this icon.
	Copy From Existing Property This icon lets you add new properties to this data definition by copying their definitions from elsewhere.

For each data property you need to define the following:

• **Name**

This is the name of the property. **No spaces allowed!**

• **Type**

Here you select one of the following options:

— String

This says that the property contains a string value.

If you specify a string data type then you must also specify the maximum length that this string can be. You specify this length in the Constraint column for this property.

— Double

This says that the property contains a double value.

You do not need to specify any constraint for this data type.

— Integer

This says that the property contains an integer value.

You do not need to specify any constraint for this data type.

— Currency

This says that the property contains a currency (money) value. As far as the actual data inside this field, it can be any of the supported numeric types - Integer, Long or Double.

In the `Constraint` column for this property you must then specify the type of currency that this field contains. You specify the currency by typing in a valid ISO 4217 currency code. For example:

```
EUR - Euro
USD - American dollars
AUD - Australian dollars
GBP - British Pounds
etc...
```

— Date

This says that the property contains a date value.

You do not need to specify any constraint for this data type.

— Long

This says that the property contains a long value.

You do not need to specify any constraint for this data type.

— Boolean

This says that the property contains a string value containing the word "true" (in any case - upper, lower or mixed). Any other value is taken to represent "false".

You do not need to specify any constraint for this data type.

- **Constraint**

  You need to specify values in this column for the data types:

  — String

  — Currency

- **Unique**

  You can specify that a property is to be considered unique.

  You check this column if this property value is to be unique for all instances of the data definition. For example, if your data definition was holding information about specific orders then you would mark (for example) the `OrderNumber` property as being unique.

  You can specify more than one property to be unique. This simply means that each of these data properties must be unique across all instances of the data. Why would you do this? It is for situations where you need to normalize your data. Let's consider an example:

  Suppose you have a flow that is going to monitor an order as it moves across your business. The order is known in the `OrdersIn` system by the `OrderNumber` (for example: `12345`). However, when this order moves into the `Shipping` system it is assigned a different key `OrderKey`, with a value such as `ABC-XYZ`. So you would need to set up your data events to start by giving you the order details according to the `OrderNumber` as defined within the `OrdersIn` system. You would then receive data events for this order and you would identify the order by this `OrderNumber` value. As the order moves into the Shipping system, you would need to get a data event advising you that `OrderNumber 12345` was now assigned the `OrderKey` of `ABC-XYZ`. You would then begin to receive data events for the order `ABC-XYZ`. In other words, your order can be identified by (in this case) any one of two unique properties. See Multiple Unique IDs on page 172 for further discussions.

- **Description**

  You can enter a text string to record your comments about this property.

Deciding On The Data Properties

When defining the data properties it is difficult to get them all defined correctly up front. You would expect to define a list of properties now, and then add-to and subtract-from this list as you continue to develop the flow.

When you are first drawing out the flow you ask yourself:

"How do I know when the flow instance gets to this node?"

Answering this question helps you to build a list of data properties.

For example:

"When the order is first written to the orders system, an order number is assigned."

"When the shipping department assign a shipping identifier, the order has been shipped."

These questions help you realize the following things:

• The kind of data properties you need to maintain for this flow

• The kind of data events that you need to receive from the underlying application systems to enable you to monitor the flow

You also need to think about the kind of statistics that you want to get from the flow once it is deployed and running. For example, do you want to be able to report the customers who are placing orders or just the overall value of orders - or both - or more!

In other words, the data properties that you define provide information to enable the flow to know where it is at any one time, and to collect information for later reporting/statistics.

Remember, you are only defining the data properties that you need to monitor your flow - not run your process. That is, you do not need to define every data property from your application systems, only the important properties that you decide you wish to monitor.

Lab - Defining The Associated Data

Let's create a data definition and specify the data properties that it maintains:

- Create a new data definition, called: Orders/My Data
- Define the following properties for this data definition:
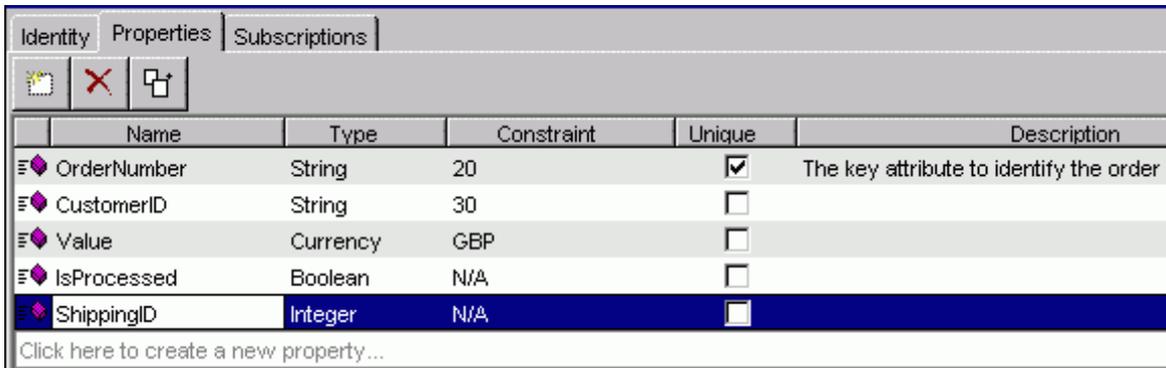
	Name	Type	Constraint	Unique	Description
	OrderNumber	String	20	☑	The key attribute to identify the order
	CustomerID	String	30	☐	
	Value	Currency	GBP	☐	
	IsProcessed	Boolean	N/A	☐	
	ShippingID	Integer	N/A	☐	
Click here to create a new property...					

- Make sure that you mark the OrderNumber as being the unique property
- Save your work

Well done! You have reached the end of the lab.

Relating The Data Definition To The Flow

Having defined the data definition, you now need to relate this definition to the flow.

To do this, you need to:

- Go to the flow definition dialog

  To do this you can either:

  — Select `Window` and then select the flow that you wish to work on

    This only works if you already have the flow editor open within the OVBPI Modeler.

  — Or simply double-click on the name of the flow in the left-hand Navigator pane

- Now click on the `Related Data` tab (in the right-hand pane)

  You should see the dialog:
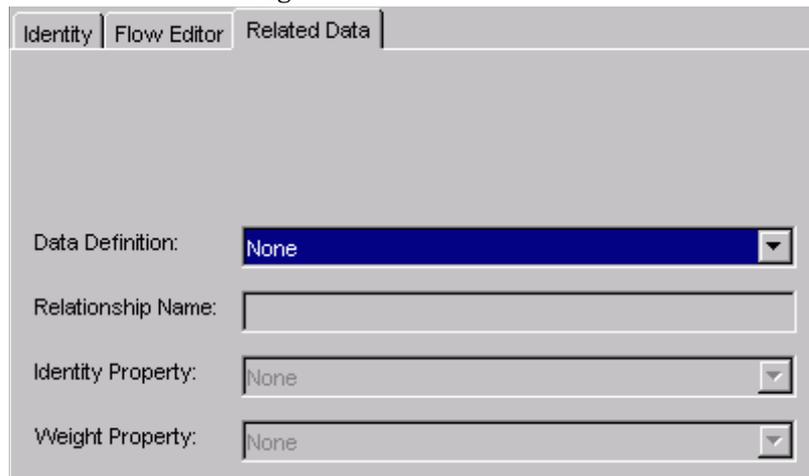
where:

- **Data Definition**

  You open the selection box and select the data definition that you wish to relate to this flow.

- **Relationship Name**

  This allows you to specify a name for this relationship. It actually is not very important so you may as well accept the default value specified.

- **Identity Property**

  You **must** assign an identity.

  This is where you choose one of the properties from this data definition to be marked as the identifier. You can choose any property you like.

  Choosing a property to be the Identity Property is only of importance when you come to run the OVBPI Business Process Dashboard to see your running flows, once everything is deployed and running. It is the property shown with each flow instance to identify it on the screen.

  Although you can choose any property you wish as the Identity, you typically choose a property that is marked as unique.

- **Weight Property** (optional)

  The weight property is like the Identity property in that it is only used when you report on the flow. The OVBPI Business Process Dashboard displays the weight property with each flow instance.

  To set the weight property, you choose one of your numeric data properties from the related data object.

  The weight property is for numeric data only.

  Non-numeric properties are grayed-out (not available) in the pull-down list of choices.

Lab - Relating The Data Definition

Let's relate the `Orders/My Data` definition to the flow `Order Flow`.

- Double click on `Order Flow` in the Navigator pane
- Select the `Related Data` tab
- Configure the relationship to be as follows:

Identity	Flow Editor	Related Data

Data Definition:	/ Data / Orders/My Data ▼
Relationship Name:	data
Identity Property:	OrderNumber : String [20] (Unique) ▼
Weight Property:	Value : Currency [GBP] ▼

Note: The dialog displays the data definition name with a `/Data/` prefix just to highlight the fact that it is a data definition :-)

- Save your work

At this point in your flow definition, you have:

- Drawn the flow
- Defined the data
- Related the data to the flow

The next step is to configure the **progression rules** for this flow...

Well done! You have reached the end of the lab.

Configuring Progression Rules

Now that the flow has an associated (related) data definition, you can go through and configure the progression rules for each node in the flow.

Progression rules are often called **start** and **complete conditions** (or start and complete criteria) because they configure the rules to say whether a node has been started or completed.

Progression rules are based solely around the values within the data definition. For example:

- This node is started when the `ShippingID` property is assigned a value for the first time

- This node is started when a `Claims_Handler_ID` has been assigned and there is a valid `Authorization_Check` and you have not yet assigned the `Claim_Value`

- This node is completed when the `CreditCheck` property is assigned `true`

Progression rules can be simple or complex. It all depends on the flow you are monitoring.

To define the progression rules for a node, you need to:

- Display the flow within the right-hand pane of the OVBPI Modeler

    You can achieve this by double-clicking on the flow (in the left-hand Navigator pane) and then selecting the `Flow Editor` tab.

- Then double-click on the node

    Or...right-click and select `Properties`.

    This brings up the **Node Properties** dialog - the one you used earlier to name the nodes.

- Within this Node Properties dialog, click on the **Progression** tab

The Rules Language

There is actually a language for writing progression rules and the grammar for this language is defined in the *OVBPI Reference Guide*.

There are also some easier ways to enter progression rules...

Style Of Progression

When developing business flows there are some common subsets of the progression rule language that people often use. These "simpler" forms of progression are available within the OVBPI Modeler as menu driven options. For more complex progression rules, you can choose to enter the progression rule by hand. So, when configuring progression rules for a node you choose your *style* of progression.

The choice of styles is as follows:

- **Complete on first assignment**

    With this style of progression you simply select the data property (from the pull down list) that you wish to monitor. The first time that this property is assigned a value, this node is marked as completed.

- **Complete on transition**

    This style lets you select a data property from the pull down list, and then you can specify the transition that you want to catch.

    You can specify the From and To values.

    If the property is defined as a String property then you need to specify the string value within double quotes ("string value").

    You can use the keyword null in either the From or To fields.

Here are some examples:

Example 1:

```
Property: MyOrder
From:      null
To:
```

Specifies that you are looking for when `MyOrder` makes the transition from being a `null` value to being any value.

Example 2:

```
Property: MyOrder
From:
To:        null
```

Specifies that you are looking for when `MyOrder` makes the transition to a null value.

Example 3:

```
Property: MyOrder
From:      null
To:        "Received"
```

Specifies that you are looking for when `MyOrder` makes the transition from a null value to the string value `"Received"`.

Example 4:

```
Property: MyOrder
From:      "Received"
To:        "Processed", "Fixed", "Solved"
```

Specifies that you are looking for when `MyOrder` makes the transition from the string value `"Received"` to one of the three specified string values `"Processed"`, `"Fixed"` or `"Solved"`.

- **Start and complete on transition**

  This style lets you configure separate start and completion transitions.

  You select a data property to signal when this node starts, and a data property to signal when this node completes. This can be the same data property or a different one.

  Here are some examples:

  Example 1:

  ```
  Start transition:
     Property: MyOrder
     From:     null
     To:

  Complete transition:
     Property: ShipID
     From:     null
     To:
  ```

  Specifies that:

  - This node starts when `MyOrder` transitions from `null` to any value.

  - This node completes when `ShipID` transitions from `null` to any value.

  Example 2:

  ```
  Start transition:
     Property: Priority
     From:     0
     To:       1

  Complete transition:
     Property: Priority
     From:     1
     To:       5,6,7,8
  ```

  Specifies that:

  - This node starts when `Priority` transitions from `0` to the value `1`.

  - This node completes when `Priority` transitions from `1` to any of the values `5`, `6`, `7` or `8`.

- **Advanced conditions**

  If your progression rules need to be more complex than the other styles available, you can select this mode and you can enter your start and completion conditions using the language as described later in Chapter 5, Advanced Progression Rules.

  For example, you would need to use the Advanced conditions option if your node start (or completion) condition was dependent upon values in two or more data properties.

Switching Styles

You can always start to define your progression rules using one of the simpler styles, and then switch to Advanced conditions to see what this actually translates to in the underlying progression rule language. You can then switch back.

When you try to switch to a progression style, the OVBPI Modeler first checks that your rules can be expressed in that style. If not, then you are warned that you could lose your progression rules if you continue with this switch. You can then choose to continue, or to cancel and leave the progression rules as they are.

No Implied Node Sequence

When defining a completion condition for a node it is sometimes easy to assume that when this completion condition is satisfied (at run time) that the flow instance automatically moves onto the next node. This is not true.

The fact that your flow diagram shows arcs from one node to the next does not guarantee any sequence or processing order. If a node's completion condition is met then that node completes. But that's it. It only enters the next node as and when that node's start condition is satisfied.

This is something that you need to think carefully about when defining your flow.

If you want to ensure that the flow instance moves from one node to the next then you might want to make sure that the start condition for the second node is the same as the completion condition for the first.

Lab - Progression Rules

Let's configure the progression rules for your `Order Flow`.

- Display the `Order Flow` in the right-hand pane and click on the `Flow Editor` tab

- Configure the `New Order` node progression rule as follow:
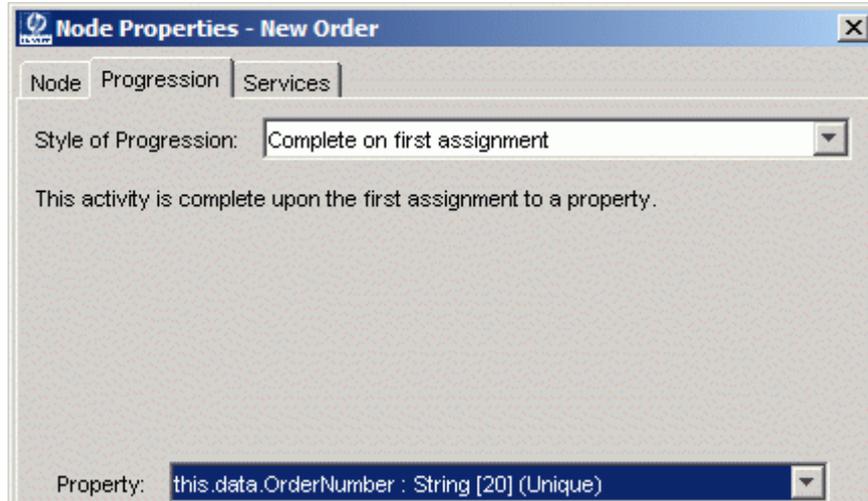
Every time a new order number comes into the OVBPI Business Impact Engine, this creates a new instance of the data definition and assigns a value to the `OrderNumber` property. The above progression rule causes a new flow instance to be allocated and the `New Order` node is marked as completed.

- Configure the `Process Order` node progression rules as follow:

This node starts when the `OrderNumber` property is assigned a value (other than null).

This node completes when the `IsProcessed` property changes to the value `true`.

- Configure the Ship Order node progression rules as follow:

This node starts when the IsProcessed property changes to the value true.

This node completes when the ShippingID property changes from null to something. (In other words, when the ShippingID property is assigned a value.)

- Configure the `End` node progression rule as follow:
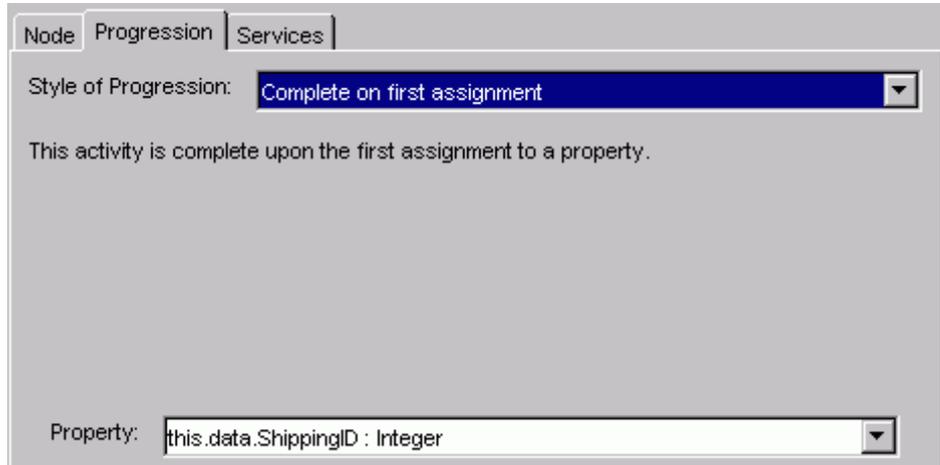
This node completes when the `ShippingID` property is first assigned a value.

- Save your work

- Click on the `New Order` node and bring up the node properties dialog
- On this node properties dialog, select the `Progression` tab
- Leave this dialog box showing on the screen, and back on the actual flow diagram, click on the `Process Order` node

Notice that the node properties dialog has now switched to represent the currently selected node.

So, when developing a flow, you can keep the node properties dialog open while you move between the nodes. There is no need to close down the node properties dialog.

Well done! You have reached the end of the lab.

Defining The Data Events

So far you have:

- Drawn the flow
- Defined the data
- Related the data to the flow
- Defined the node progression rules

You now need to define the **data events** that are needed to drive this flow...

Define Each Event

You first need to define the actual events that you expect to come from the underlying application systems.

Remember that defining these events here in the OVBPI Modeler does not mean that the application systems are set up to generate them. That is a separate step required later on when you go to run this flow for real. Refer to the *OVBPI Integration Training Guide - Business Events* for more details on how to do this.

To create an event definition, you can either select:

```
File->New->Event
```

or simply click on the new event icon:

Create a new event
definition...
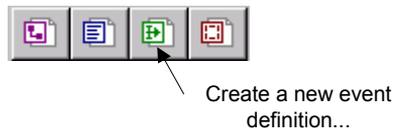
This brings up the event properties page - in the right hand pane of the OVBPI Modeler.

You specify the name of this event definition (for example: New Order) and (optionally) a description.

The name of the event definition can include "/" characters to provide logical grouping. So your event definition could be called: Orders/New. This might be useful if you are planning on defining many flows all requiring event definitions, as it enables the event definitions to be listed in logical "groups".

You then click on the `Properties` tab and you should see a dialog similar to this:
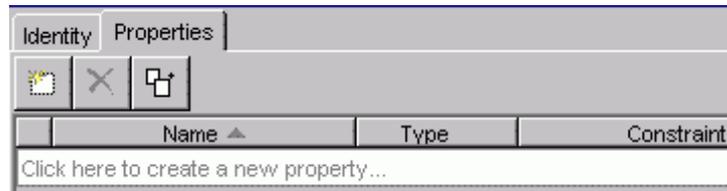
You can go through and manually create new properties for this event by clicking where it tells you to ...however... by using the `Copy Properties` icon:

you can select properties from an existing data definition and the OVBPI Modeler creates properties to match their name and definition.

This is really helpful because your events are bringing in data that you typically want to store into a data definition. So it saves you having to type the same information over and over.

When you click on the copy properties icon, you are presented with a dialog. You then:

- Select the definition from which you wish to copy

- Then select the properties within that definition that you wish to actually copy to this event definition

For example, the `Copy Properties` dialog might look something like this:

To copy properties from another definition, select the definition
and then the properties to be copied.

Definitions:
- Model Repository
  - Data
    - Orders/My Data
  - Events
    - New Order

Properties:
- ☑ CustomerID : String [30]
- ☐ IsProcessed : Boolean
- ☑ OrderNumber : String [20] (Unique)
- ☐ ShippingID : Integer
- ☑ Value : Currency [GBP]

Select All

OK   Cancel
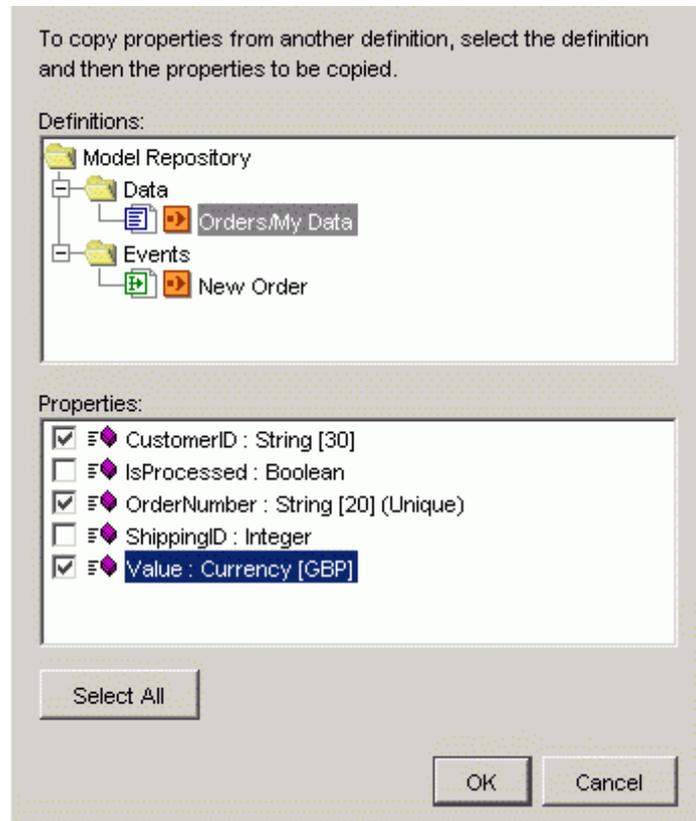
You press OK and it adds these properties to your event definition.

	Name	Type	Constraint
CustomerID	String	30	
OrderNumber	String	20	
Value	Currency	GBP	
Click here to create a new property...			
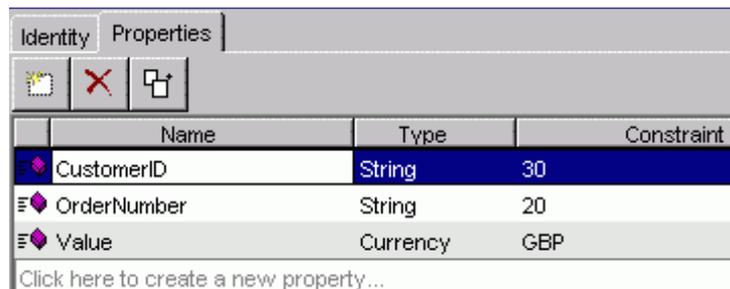
Lab - The Data Events

Let's define the events for the `Order Flow`

- Create a new event, and call it: `Orders/New`

- Click on the `Properties` tab for this event

- Using the `Copy properties...` icon, define this event to have the following properties:

	Name	Type	Constraint
♦	CustomerID	String	30
≣♦	OrderNumber	String	20
≣♦	Value	Currency	GBP

Identity / Properties

Click here to create a new property...

- Create two more events as follows:

  ```
  Event: Orders/Processed

  Properties:
          OrderNumber
  ```

  ```
  Event: Orders/Shipped

  Properties:
          OrderNumber
          ShippingID
  ```

- Save your work

Well done! You have reached the end of the lab.

Configuring The Event Subscriptions

Having defined the events, you have specified what data you expect to receive from the underlying application systems. You now need to configure how these events are going to update the `Orders/My Data` data definition.

You might be thinking:

> "But I've defined these events specifically for this flow and data definition?"

But you might use these events to update other data definitions in the future. You see, the OVBPI Modeler allows you to define your data, your events and your flows all as separate entities. It is then up to you as to how you piece them together.

Anyway, in this case, you do want to configure these events such that they update your `Orders/My Data` definition.

The way you do this is by configuring the data definition to **subscribe** to these events.

Setting Up A Subscription

A subscription is something that you set up on the data definition:

- Double click on the data definition (for example: `Orders/My Data`)
- In the right-hand pane, click on the `Subscriptions` tab
- Click to create a new subscription

The dialog that then appears, looks complicated but is actually quite straightforward. The dialog actually consists of the following four main parts:

Event and Description

Here you select the event, from the available list, to which you wish to subscribe.

You can optionally enter a description.
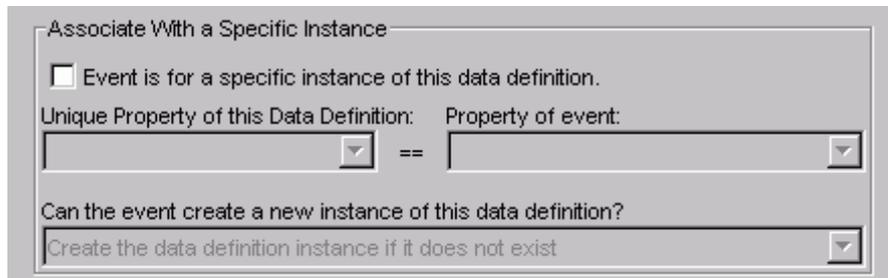
Specific or Not

Here you specify whether the event (that you are subscribing to) is destined for a specific data instance or for all instances of this data definition.

For example, you could specify that this event is for a specific instance of your data, and then specify which property in the event is to be matched with the data definition.

Or, you might not check the box and thus this event affects **all** instances of your data definition. This allows an event to update all the data instances with information.

At first, you are likely to set up events for specific instances. So typically you would check the box to say it is for a specific data instance, and then specify the event property whose value is to be used to match up with the unique property within the data definition.

If you have specified that this event is for a specific data instance, you then select the behavior for when the event comes in and there is currently no specific data instance for this event. You have a series of options available to

you where you can ask that the data instance be created (this is typically the option to select), ask for it to create the data instance but to log a warning, not create the data instance, etc.

The typical option is that you want the data instance to be automatically created the first time.

For example:

where:

- This event subscription is for a specific data instance

- Match the incoming event to the data instance by matching the data instance's OrderNumber property (this.OrderNumber) with the incoming event's OrderNumber property (event.OrderNumber)

- If there is no current data instance whose OrderNumber matches the incoming event, create a new data instance

Filtering
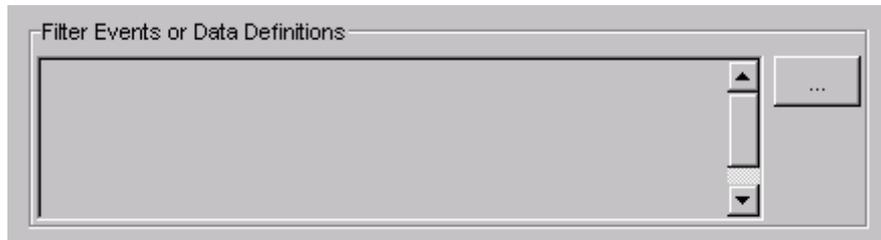
This section allows you to specify additional filtering on the incoming event. See Event/Data Filtering on page 159 for further discussions on filtering.

Actions
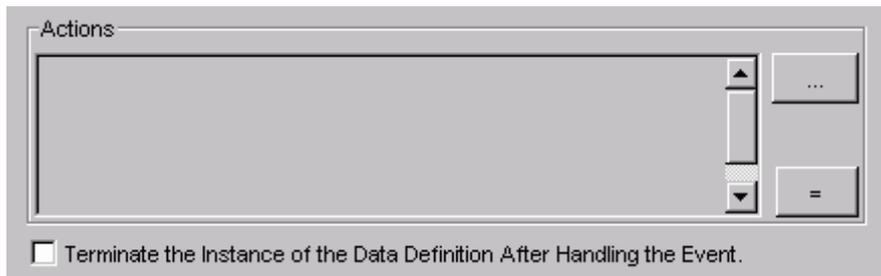
The Actions section is where you specify which properties are pulled out of the event and placed into the data definition.

The basic syntax for these actions is:

```
this.Property1 = event.EventPropertyA;
this.Property2 = event.EventPropertyB;
...
```

You are specifying which event properties to assign to data instance properties.

The " = " button is the quick way to simply say:

"Set up assignments for each property in the event that has a matching named and typed property within the data definition."

If you wish to be more selective in your assignments then you can press the
"..." button. This presents a dialog where you can select the properties that
you require. This dialog looks something like this:
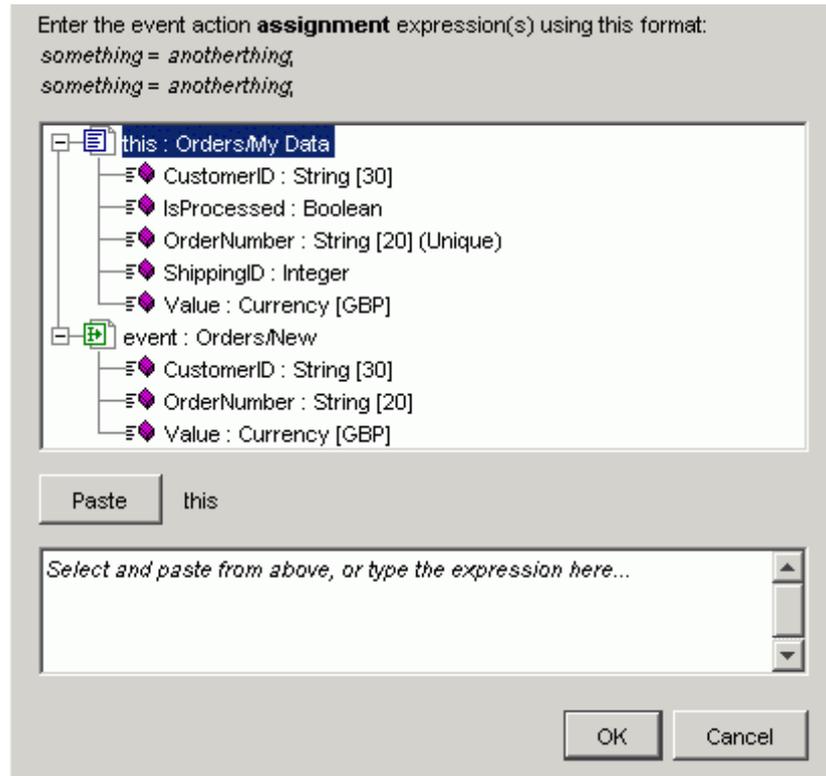
where:

- You are presented with property lists for both the data definition and the
  event definition
- You can then either type the assignments in at the bottom of the screen
  yourself ...or...
    - Select the data property
    - Click the Paste button
    - Type in =
    - Select the event property

— Click the Paste button

— Type in ;

• When you have your set of assignments...click OK

Type coercion

The assignments you specify in your actions can include type coercion. That is, you can (for example) assign an integer property to a string. The run time OVBPI Business Impact Engine is able to handle basic type coercion as described in *OVBPI Reference Guide*.

Action Order and Errors

The actions you specify are carried out in the order you specify. This is (possibly) important in the case where an error occurs. If an error occurs on an assignment then that action fails and all subsequent actions are not applied. However, all previous actions have been assigned successfully.

A possible error would be if you were assigning a string to an integer and the string happened to contain non-numeric characters.

Terminating The Data Instance

You check this box to terminate the data instance after handling this event. You would typically check this box for the event that signalled the end of the flow instance. That is, when the event arrives to end this flow instance, you want it to also terminate the data instance.

Be careful if your data instance is being shared between more than one flow. In this situation you would need to make sure that you did not terminate the data instance prematurely.

If you do not have an event subscription that terminates the data instance then the data instance remains in the OVBPI database marked as `Active`.

➤ You should always have at least one event subscription that terminates the data instance.

Lab - Event Subscriptions

Let's configure the subscriptions for your `Order Flow`:

- Double-click the `Orders/My Data` data object (in the Navigator pane)
- Click on the `Subscriptions` tab (in the right-hand pane)
- Click to create a new subscription

Orders/New

When the subscriptions dialog shows:

- Select the `Orders/New` event from the pull-down
- Check the box to say this subscription is specific for a data instance

  This pre-fills the boxes and because you have the `OrderNumber` defined in both the data and event definitions these are automatically configured for you.

- In the Actions section, click the " = " button and this sets up the necessary assignments

  Again, because you have the same named properties in both the data and event definitions the OVBPI Modeler is able to automatically configure these actions for you.

Your subscription should look something like this:

- Click OK

Orders/Processed

For this subscription you do the same as before:

- New subscription
- For `Orders/Processed` event
- Specific for a data instance
- Click the " `=` " button

    However, the " `=` " button only assigns the `OrderNumber` property and you also want this subscription to assign the data instance's `IsProcessed` property.

    This is a case where the event itself is enough to signal that the order has been processed. So just the fact that you have received this event means that the subscription can assign the `IsProcessed` property to `true`. There is no need to receive this property from the actual event - and thus no need for the underlying application systems to generate the `IsProcessed` property.

    So, to add this assignment to your event:

- Click on the "`...`" button (for the `Actions` section)
- In the text at the bottom of this dialog - the area that currently contains the assignment `this.OrderNumber = event.OrderNumber;`

    Append a carriage return after the ";" at the end of the first line. In other words, start a second line.

- Now click on the `IsProcessed` property (within the `this: Orders/My Data` section) near the top of the dialog
- Click on the `Paste` button

    You should see the text `this.IsProcessed` appear in the text area at the bottom of the dialog.

- In the text area near the bottom of the screen, click at the end of the line that has just appeared, and type in: `= true;`
- Click OK

Your subscription should now look something like this:

- Click OK

Orders/Shipped

- The `Orders/Shipped` event completes the flow instance. You want to mark that this subscription also terminates the data instance

- Go ahead and create the following subscription:

Don't forget to check the box to say that the instance of the data definition is terminated after the event is handled.

- Save your work

Well done! You have reached the end of the lab.

Summary

Let's summarize the steps (so far) to model a flow:

1. **Understand the business process**

   In the real world you would have spent time gaining an understanding of the actual business process and produced from this a high-level business flow to represent the main phases of the process

   During this step it is important to ask yourself the question:

   > "What statistics do I want to get from this flow?"

   You would also want to list out the possible sources for data events for each node within your flow

2. **Draw the flow**

   Run the OVBPI Modeler and layout the flow. Assigning meaningful names to each node. This defines the steps within the flow.

3. **Define the data definition**

   Define the data properties/attributes that are to be maintained.

4. **Relate the data definition to the flow**

5. **Configure the progression rules for each node**

   These allow the flow to know where it is at any given time.

6. **Define the data events**

   Define the expected data to be received from the underlying application systems.

7. **Configure the event subscriptions**

   This configures how the application data is assigned/mapped to the data instance.

At this stage you have a flow that is ready to be deployed to the OVBPI Business Impact Engine.

At this point you have not linked in any IT service dependencies, which would allow you to see actual business impact when/if a service goes down.

IT service dependencies can be added later on, and is covered later in Chapter 3, Services.

Let's look at how you can deploy your flow and test it out...

Deploying The Flow

Before you can deploy a flow you need to make sure that the OVBPI Business Impact Engine is up and running. You can use the OVBPI Administration Console to check this.

You deploy a flow from within the OVBPI Modeler.

You simply:

- Select the flow name in the Navigator pane
- Then either
  - — Click the green deploy icon

  - — Or select: **File->Deploy...**

  This then deploys the selected flow and any associated definitions that have changed since the previous deploy (if any).

- You see a pop-up dialog listing the definitions that are to be deployed and you can then click OK (or cancel)

The ToDo List

Before deploying your flow it is always worth checking that there are no warnings or errors listed in the ToDo list.

The ToDo list is displayed in the right-hand pane of the OVBPI Modeler, underneath the current definition pane.

For example:
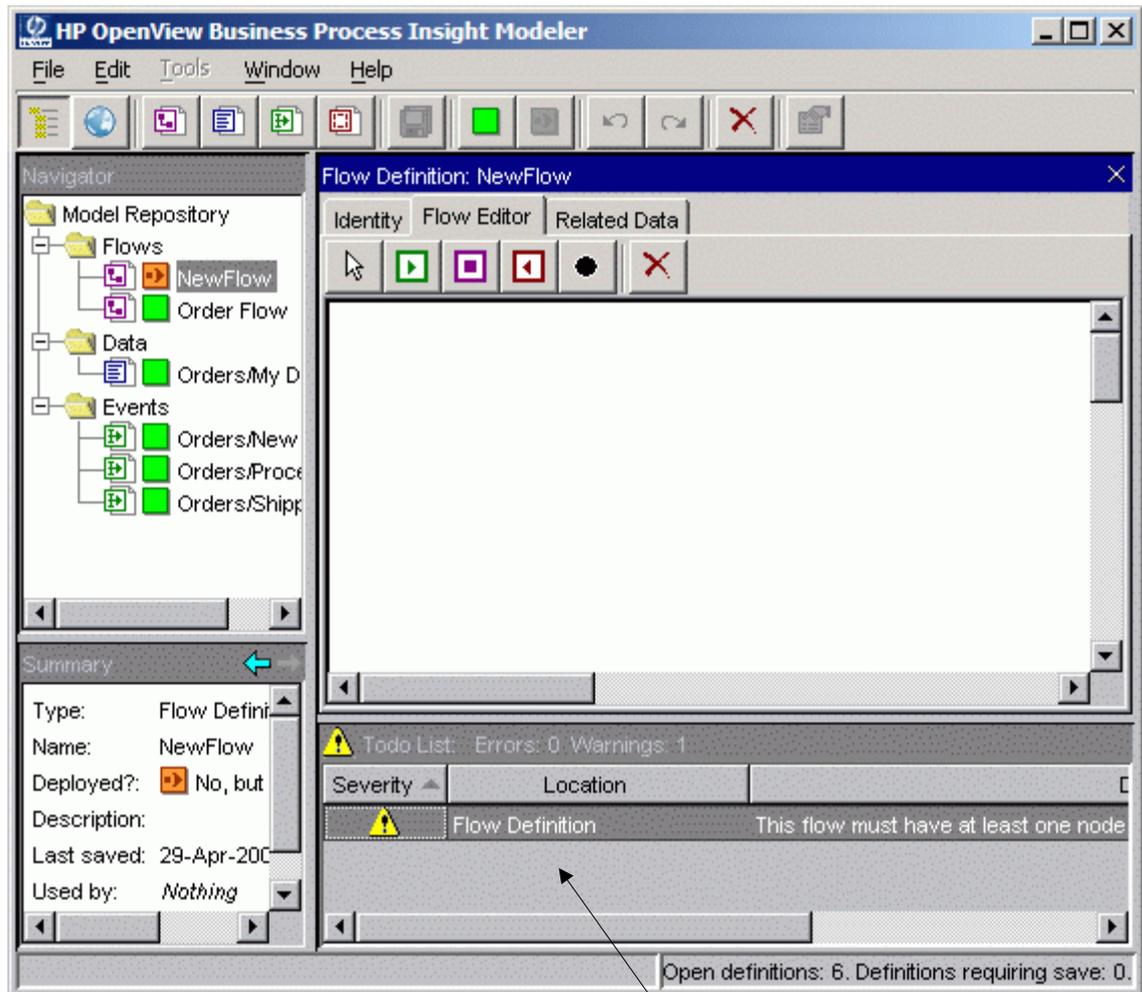
The ToDo list

If you cannot see any ToDo list area then it might be set to "zero height", in which case you simply need to expand it by using the mouse to enlarge the window.

Viewing The Flow

"Once a flow is deployed, how can I see it?"

To view the deployed flow you can use the **OVBPI Business Process Dashboard** that comes standard with the product. It is very useful for developers when testing a flow as it allows you to view any flow and see basic statistics about it.

The Business Process Dashboard URL

For the OVBPI Business Process Dashboard to be able to work, your OVBPI administrator must have first started the **Servlet Engine** component on the OVBPI server. This is because the OVBPI Business Process Dashboard is a set of JSP pages that require a servlet engine to run them.

To run the OVBPI Business Process Dashboard, point a Web browser at the URL:

```
http://hostname:44080/ovbpi-bpd
```

where:

- *hostname* is the hostname of your OVBPI Business Impact Engine installation
- *44080* is the default port for the Servlet Engine

For example:

```
http://localhost:44080/ovbpi-bpd
```

This works if your OVBPI Business Impact Engine is installed on the same machine as your Web browser.

Logging On

When OVBPI is first installed there is no logon required to use the OVBPI Business Process Dashboard. You can simply run a Web browser, point to the OVBPI Business Process Dashboard URL, and see all of your flows and their data.

The OVBPI Administration Console allows you to specify whether you require people to give a username and password when running the OVBPI Business Process Dashboard. You then have a choice of authorization mechanisms. (See the *OVBPI System Administration* guide for more details.)

Testing The Flow

Q: "Now the flow is deployed, is there any way to test it out?"

A: "Yes!"

Obviously, for your flow to go live, someone has to work with the applications people to generate the required events that then drive the flow. This can be quite involved and it may be that your applications people are unable to generate the events exactly as you have defined, in which case you would then have to go back and alter your flow definition. For more details on how to configure the actual events from the underlying application systems you can refer to the *OVBPI Integration Training Guide - Business Events*.

But it would be good to be able to drive the flow without having to invest time developing links to the application systems. This can help you spot any errors in your progression rule logic. It can also help you demonstrate the flow to your management/customer base to get further feedback on whether the flow is right for your needs.

To test out your business flow you can use the **Generic Event Injector** contributed utility.

The Generic Event Injector is a set of JSP pages that allows you to select any of the deployed events, supply some values, and inject them into the OVBPI Business Impact Engine. This allows you to test how your flow reacts to these events.

If you wish to use the Generic Event Injector, you must first install it from the OVBPI install CD.

Installing the Generic Event Injector

- Locate the OVBPI install CD.

  The Generic Event Injector comes as a WAR file ready to install in to the Servlet Engine. The WAR file is located at:

  OVBPI-install-CD\contrib\EventSim\event-injector.war

- Copy this event-injector.war file into the directory:

  OVBPI-install-dir\nonOV\jakarta-tomcat-5.0.19\webapps

- In this jakarta-tomcat-5.0.19\webapps directory, check if there is already a subdirectory called event-injector.

  If there is a subdirectory called event-injector, remove it!

  You need to make sure that there is not already a subdirectory of this name because this is the directory into which the war file is expanded. If the subdirectory already exists then Tomcat (the servlet engine) does not expand the new war file.

- Using the OVBPI Administration Console, stop and start the Servlet Engine component.

- If your OVBPI installation directory is somewhere other than the default location, then you need to tell the Generic Event Injector the name of this install directory.

  — Edit the file:

  webapps\event-injector\WEB-INF\classes\EventConfig.properties

  — Set the OV_DATA_DIR property to point to your *OVBPI-install-dir*\data directory

  You can use either a forward slash (/) or double back slashes (\\) in the directory path. Do not use a single backslash (\) as this causes errors at run time.

The Generic Event Injector contributed utility is now installed.

Using the Generic Event Injector

To use the Generic Event Injector, you need to:

- Ensure that the following components are running on the OVBPI server:

  — The Business Event Handler

  — The Servlet Engine

  You can use the OVBPI Administration Console to verify/start these components.

- Point a Web browser at the URL:

  ```
  http://hostname:44080/event-injector
  ```

  where:

  — *hostname* is the hostname of your OVBPI Engine installation

  — *44080* is the default port for the servlet engine

  For example:

  ```
  http://localhost:44080/event-injector
  ```

  This works if your OVBPI Engine is installed on the same machine as your Web browser.

- You are required to enter a user name and password to access the event injector JSP pages. The default user/password details:

  ```
  User:     admin
  Password: ovbpi
  ```

  To change the admin user password, and to configure the underlying authorization mechanism, refer to the *OVBPI System Administration Guide*.

- You are then able to select the event you wish to send into the OVBPI Business Impact Engine, and provide test data for this event

Lab - Testing The Order Flow

Let's deploy the `Order Flow` and then test it out by sending in some data events and seeing what happens...

Deploying The Flow

- Make sure that the `Business Impact Engine` component is up and running on the OVBPI server
- Using the OVBPI Modeler, deploy the flow: `Order Flow`

  You should see that it deploys:

  — The three data events

  — The data definition

  — The orders flow

Viewing The Deployed Flow (OVBPI Dashboard)

- Make sure that the `Servlet Engine` component is up and running on the OVBPI server
- Run a Web browser and point it to the URL:

  `http://hostname:44080/ovbpi-bpd`

  where:

  — `hostname` is the hostname of your OVBPI server

  — `44080` is the default port for the OVBPI Servlet Engine component

- Click on the flow name and it should then show the flow diagram for your `Order Flow`

  This details page also shows that there are currently no active instances.

Let's create some flow instances...

Generating Data Events (Event Injector)

The Generic Event Injector is a contributed utility and so the first thing you need to do is make sure that it is installed on the OVBPI server.

Installing the Generic Event Injector

- Install the Generic Event Injector following the instruction given in Installing the Generic Event Injector on page 89

Using the Generic Event Injector

- Make sure that the components:

  ```
  Servlet Engine
  Business Event Handler
  ```

  are up and running on your OVBPI server

- Start up a new Web browser, and point it at the URL:

  ```
  http://hostname:44080/event-injector
  ```
  where:

  — *hostname* is the hostname of your OVBPI Business Impact Engine installation

  — *44080* is the default port for the OVBPI Servlet Engine

- By default, you are required to enter a password to access the event injector JSP pages. The default user/password details:

  ```
  User:     admin
  Password: ovbpi
  ```

  When the Generic Event Injector main page is displayed it gives you a screen something like this:
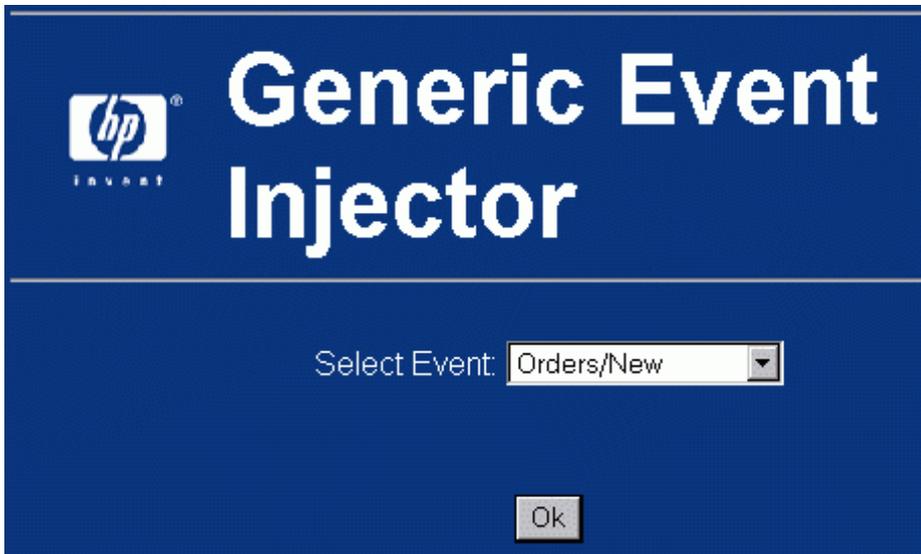
If this screen is shown, but the `Select Event:` pull-down is empty, then your OVBPI installation is probably installed in a non-default location. That's OK, but you need to go back to the installation instructions for the Generic Event Injector and look at the step that talks about a non-default installation of OVBPI. The instructions explain how to get the Generic Event Injector working with an installation of OVBPI that is installed under a non-default directory.

- Select the `Orders/New` event and click OK.

  This brings up a screen that shows the definition of this event:

Orders/New

Property	Type	Value	Constraint
CustomerID	String	Modify me	30 characters
OrderNumber	String	Modify me	20 characters
Value	Currency	10.99	GBP

Submit

Default values are provided - although you would be wise to modify them to be values you need for your test conditions

You are able to see the properties of the event and any constraints, however the injector does not know which of the properties is defined as being "Unique". So you need to find out this information and make sure that you always specify unique values for that field(s) with each event. You can easily check the flow definition to see which property(s) are specified as unique.

- Set the properties to the following values:

```
OrderNumber  = 123
CustomerID   = 456
Value        = 10.99
```

Note: Make sure you enter these into the correct fields on the screen

- Click `Submit`
- Click `OK` to acknowledge the confirmation page

Back In The OVBPI Business Process Dashboard

In the OVBPI Business Process Dashboard you can now:

- See that you have one active instance of the `Order Flow` flow

- Drill down into the `Healthy Instances` selection, and then drill into your flow instance. You should be able to see the actual flow diagram for this instance and see the associated (related) data instance

  You should see a flow diagram similar to this:
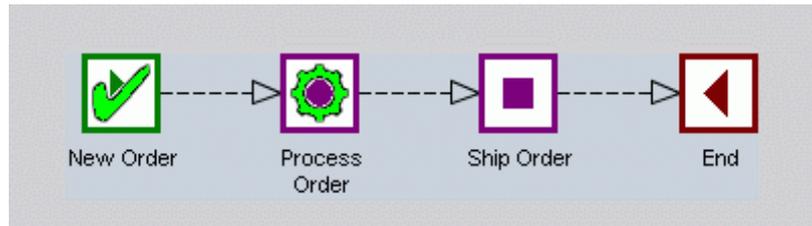
The `New Order` node has completed and the flow instance is currently sitting active in the `Process Order` node

Injecting More Events

You can now send in further events for this instance and see how your flow progresses:

- Send in the Event:

  Event Name: Orders/Processed

  OrderNumber: 123

- Back in the OVBPI Business Process Dashboard, refresh your browser page and you should see the flow instance has moved on to the `Ship Order` node

  On the flow instance page, you can also scroll the page down to see the associated data values - so you can see the `OrderNumber`, `CustomerID`, `IsProcessed`, etc. properties and their values.

- Send in the `Orders/Shipped` event for OrderNumber `123`

  Your flow instance should now show as completed.

- If you move back to the `Business Flow & Resource Summary` page you should now see that your instance is showing as a completed instance

  Note: It is best to use the navigation bar ("bread crumbs") across the top of the page to move backwards. If you just use the Web browser's "back" button the page is shown but not refreshed. By using the bread crumbs to move back, the page is rerun and shows you the latest information

---

Well done! You have reached the end of the lab.

---

3 Services

In the last chapter you defined a flow that monitors your orders as they move through your business. This enables you to see and measure things, such as:

- Order volumes
- Backlogs at any particular nodes
- Time taken for each node
- etc.

Now let's link this to your IT services.

OVBPI allows you to specify, for each node, the underlying service(s) for which that node is dependent. So for your `Order Flow` flow, you could specify that the `Process Order` node is dependent on (for example) the Orders Oracle database. That is, if the Orders Oracle database goes down then you are unable to process orders and hence you have a business impact. OVBPI would then be able to show you the volume of orders currently waiting to be processed and the rate at which they are continuing to arrive.

This chapter looks at how to link your business flows to your IT service hierarchy.

Service Status Events

If you remember back to the overall architecture of OVBPI, the Business Impact Engine is able to receive two main types of events:

- Business Data Events

  These contain business data from underlying applications, databases, files, etc.

- Service Status Events

  These signal the current status of the underlying servers, application systems, databases, etc. For example, a status event might tell OVBPI that the Web Server has just gone down, or that the CRM system has just gone off line.

Figure 7   The Two Types of Event Feeds
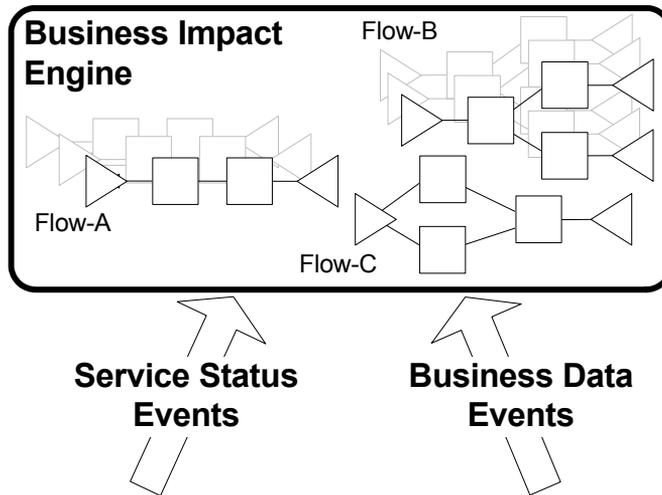
Service status events can be sent to OVBPI from any source. Whilst this technically allows you to link in any 3rd party service monitoring system, OVBPI provides out-of-the-box integration for the following service monitoring products:

- OpenView Operations (OVO)

- OpenView Internet Services (OVIS)

- Openview Service Oriented Architecture (SOA) Manager

OVO Integration

The OVBPI/OVO integration basically works as follows:

- Within the OVBPI Modeler, you are able to specify the OVO services on which your business flow depends

- When you come to deploy this business flow, OVBPI tells OVO that it now wants to receive on-going status change information for these particular services

- As and when these services change state, OVO sends this status change information to OVBPI and your flow reflects any resultant business impact

Architecture

For OVO to be able to communicate with OVBPI you need to install and run the OVBPI OVO Adapter component on the OVO server. Refer to the *OVBPI Installation Guide* for more details.

Let's consider the architecture where you have installed OVBPI and OVO on different servers:

Figure 8   OVO Integration Architecture
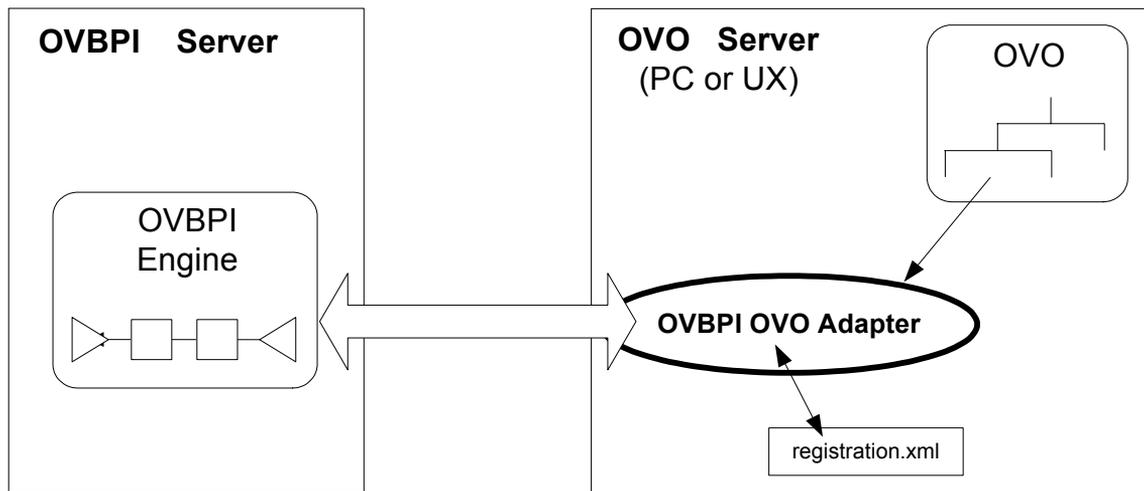
where:

- OVO can be installed on either UX or Windows

- All communication between OVBPI and OVO goes through the OVBPI OVO Adapter

- Service Definition Enquiry

  When defining a flow within the OVBPI Modeler you specify the names of the OVO services that you wish to use. The OVBPI Modeler uses the OVBPI OVO Adapter to determine whether these services exists within OVO.

- Service Registration

  When you deploy a flow that contains references to OVO services, the OVBPI deployer registers these service names with the OVBPI OVO Adapter. These registrations are held in the file (on the OVO server):

  OVBPI-install-dir/misc/bia/registration.xml

  This enables the OVBPI OVO Adapter to know what status change events it needs to send back to OVBPI.

- Service Status Events

  Once the OVBPI OVO Adapter has a list of registered OVO services, any status changes for these services are passed through to OVBPI.

Configuring The OVBPI OVO Integration

Configuring the OVBPI OVO integration requires some set up on both the OVO server and the OVBPI server. Even if you have installed OVBPI on the same server as OVO, these steps are still required:

- On the OVBPI Server

    You configure the OVO connection details using the OVBPI Administration Console.

    — In the left-hand navigation pane, select the option:

        Operational Service Sources

    — In the right-hand pane, double click the option:

        OVO interoperability

    The main piece of information that the OVBPI server needs to know is the host name of the server running OVO.

    — Accept the other defaults and click the OK button.

    — Ensure that your new adapter has a tick in the Enabled? column.

    — Click Apply to save these configuration details.

    You can refer to the *OVBPI System Administration Guide* for more details.

    After configuring your OVO connection you **need to restart all OVBPI components.**

- On the OVO Server

    You need to install the OVBPI OVO Adapter on this OVO server. Refer to the *OVBPI Installation Guide* for more details.

    Once the OVBPI OVO Adapter is installed on the OVO server, it is ready to be used.

    Use the OVBPI Administration Console (on the OVO server) to start up the OVBPI OVO Adapter.

The OVBPI Modeler

The OVBPI Modeler lets you import service definitions from OVO. This then allows you to define the dependencies between the nodes in your flows and the underlying IT services.

The services you import into the OVBPI Modeler are effectively just pointers (or "links") to the services defined and managed within your OVO. That is, the services are still maintained and owned by OVO.

Linking OVO Services

OVO service hierarchies are typically quite large, and your business flows are typically going to depend on a small'ish subset of the full OVO hierarchy. Hence the OVBPI Modeler does **not** import the full OVO hierarchy. Instead, the flow developer specifies the individual services by name.

Let's walk through an example...

The steps to link OVO service definitions are as follows:

1. Within the Modeler, select: File->Link to Services...

   This brings up the following dialog:
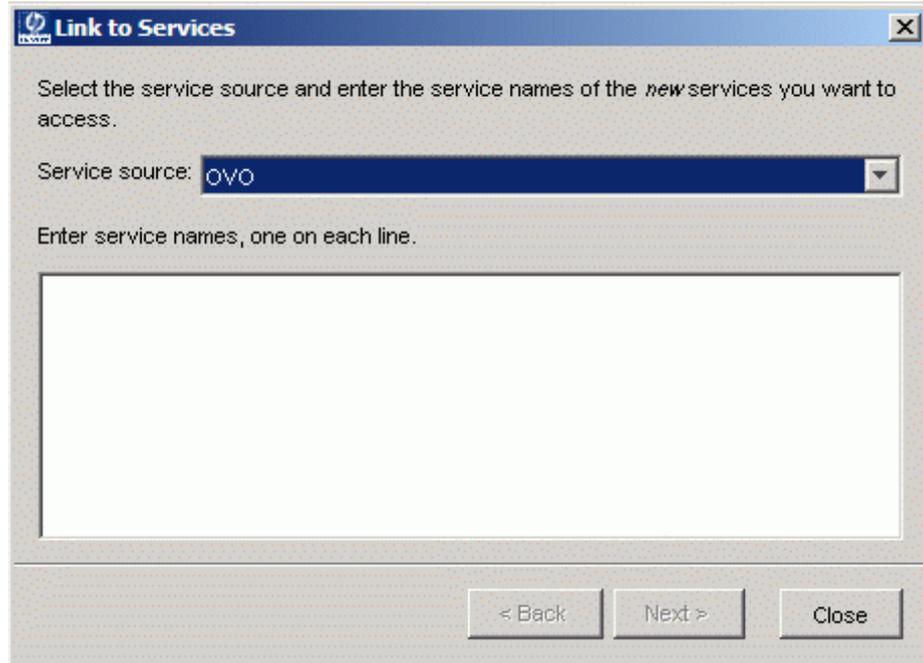
2. When the dialog appears, select the option OVO from the Service source drop down menu.

3. Type in the name(s) of the OVO services you wish to link to the OVBPI Modeler

For example, to link the service definition `OrdersDB`, the dialog would look as follows:
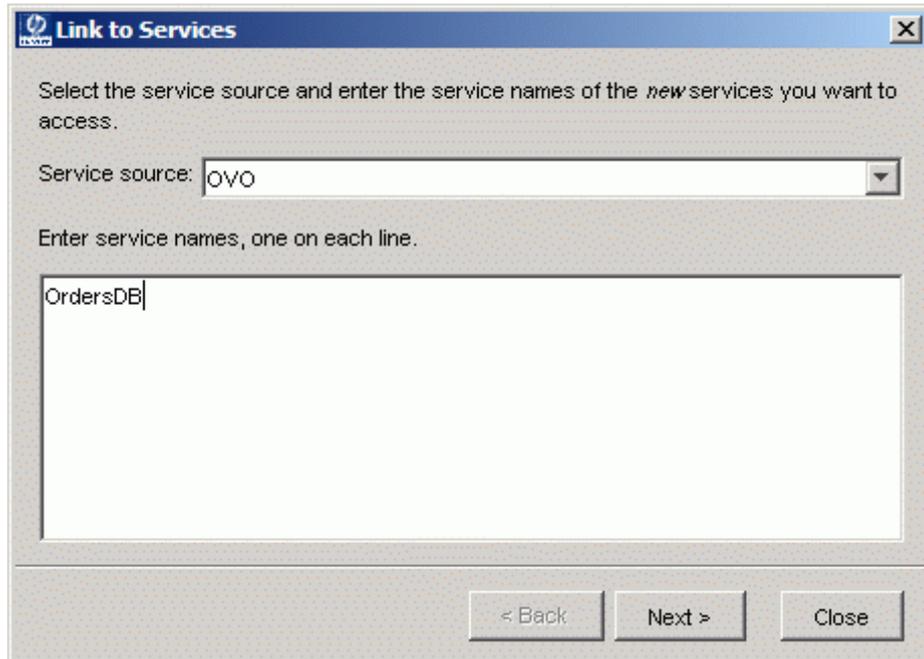
The name(s) that you type here must match the OVO service name(s). These are the unique names assigned when you created the service within your OVO hierarchy. It is not the display label for the OVO service.

If you want to link in more than one service, simply type each name on a new line.

Be careful linking to OVO services whose names differ only by case. OVBPI running on an Oracle database handles this, however, OVBPI running on a MSSQL database is not able to distinguish between them.

The OVO service names that you are linking to must not exceed 255 characters.

4. Click on the `Next` button

It is at this point that the OVBPI Modeler goes out and talks to OVO (using the OVBPI OVO Adapter) to locate the service definition for the name(s) specified.

If the OVBPI Modeler is able to communicate with OVO then you see the following dialog:
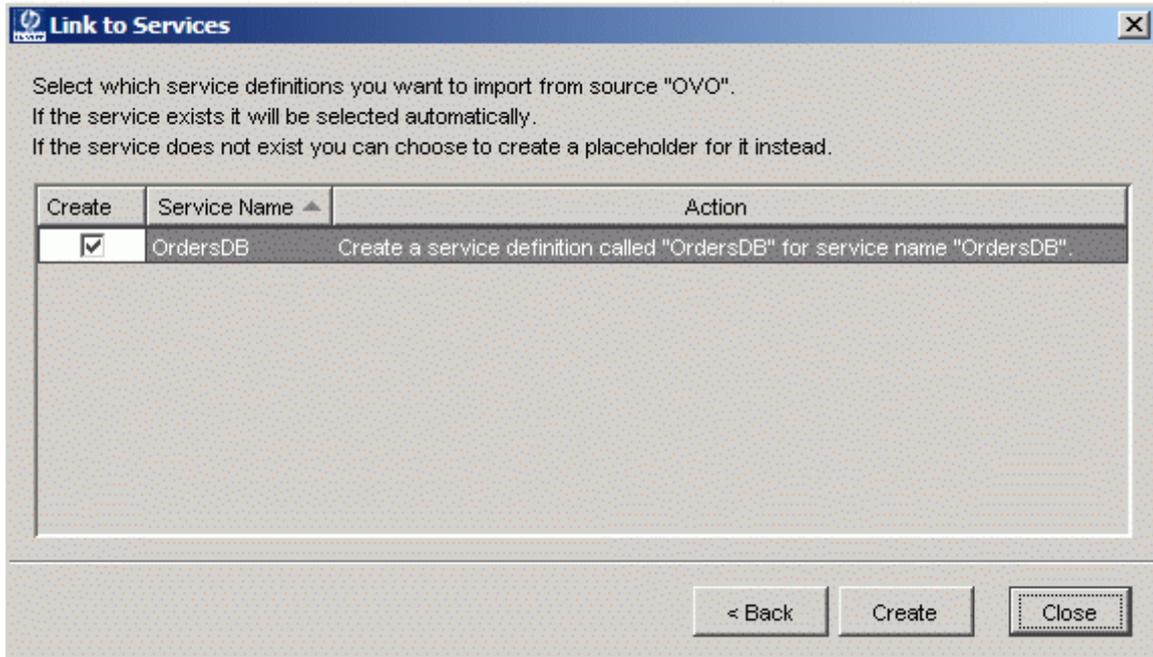
If the service name exists in OVO it is listed here with the `Create` option checked - all ready for you to accept.

If the service name does not exist in OVO, it is listed here with the `Create` option left un-checked. This occurs for two main reasons:

— Either you mis-typed the service name

In which case you click the `Back` button and re-type the correct name.

...or...

— You have specified a name that you know is going to exist within OVO some time in the future and you want the OVBPI Modeler to create a place holder

By creating a placeholder within the OVBPI Modeler, this allows you to define business flows that use this service. When this service is eventually created within OVO, any deployed flows start receiving status change events for this service.

5. Once the correct services are listed and you have correctly set the Create option for each one, simply press the Create button

You see a progress dialog. When this is completed, you can close this progress dialog.

You are now able to see your newly linked OVO services. They are listed in the Navigator pane on the OVBPI Modeler, under a folder called OVO Services.
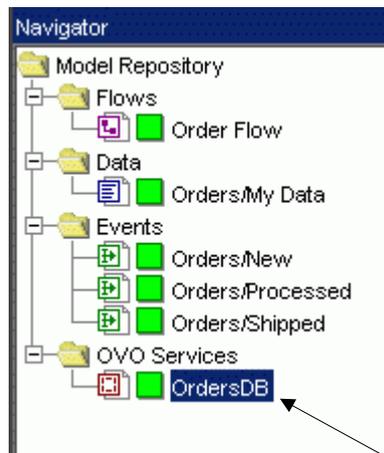
Notice that the newly linked service OrdersDB is marked as deployed (with a green icon next to its name). This does not mean that it is deployed within the OVBPI Business Impact Engine, it means that this service corresponds to an active OVO service definition.

Placeholder Services

Suppose you know that a service is going to be defined in OVO sometime next week, but you want to start defining the business flow right now. You want to define a placeholder OVO service.

Consider the example where you want to define a placeholder service called: `ABC Service`:

- As described in the previous section () you select:

    `File->Link to Services...`

- Select the Service source called: `OVO Services`

- Enter the name: `ABC Service`, and click `Next`

    Notice that the `Create` option is not checked, and the `Action` column explains that if you do check the create option then this creates a placeholder for this service definition.

- Check the `Create` option for this service

- Now click the `Create` button

    This creates a placeholder in the OVBPI Modeler for this service.

When you look in the Navigator pane of your Modeler, you see your service as follows:
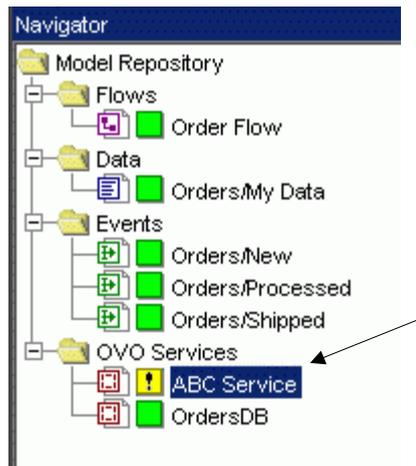
Notice it is listed with an icon that indicates that this service definition does not currently exist in OVO.

You can quite happily refer to a placeholder service definition within your flow and then deploy this flow to the OVBPI Business Impact Engine. It just means that your flow does not receive any service status change events as the service does not yet exist within OVO. Once the service has been created within OVO your deployed flow automatically starts receiving service status change events and your flow reflects the business impact.

Synchronizing Services

Within the OVBPI Modeler, there is the option to synchronize your service definitions. This tells the OVBPI Modeler to synchronize its view of the OVO services.

To synchronize service definitions, you:

1. Select the menu option:

   `File->Synchronize Services...`

   You see a dialog similar to the following:
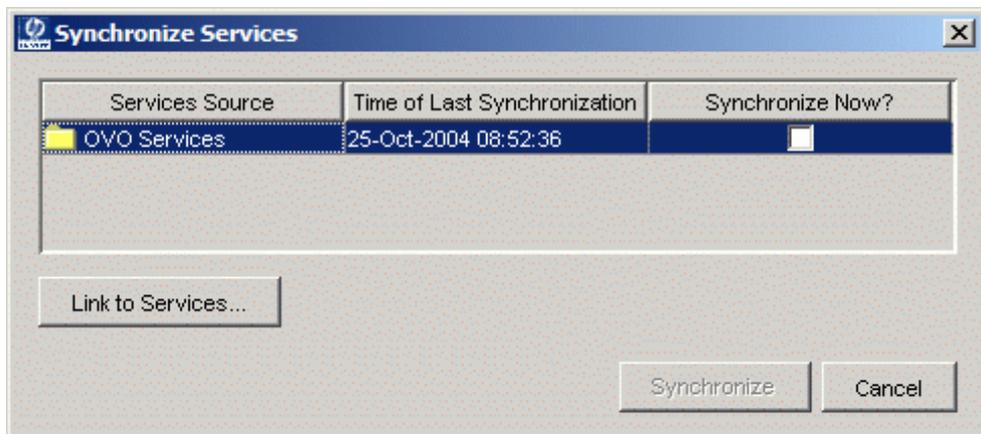
If your OVBPI installation has other service integrations configured (such as OVIS or SOA Manager) then you also see thoses listed in the `Services Source` column.

2. You simply check the `Synchronize Now?` checkbox and click on the `Synchronize` button.

   This tells the OVBPI Modeler to check the existence of all the references it has to OVO services and see if they exist in the OVO hierarchy:

   — Any placeholder services become marked as deployed (green icon) if the service definition is found to exist in OVO

   — Any currently deployed services become placeholders if (for some reason) they have been removed from OVO

In the example above, with the placeholder called `ABC Service`, the icon changes to green once a service of this name is created within OVO and you have run a synchronize.

Notice that the Synchronize Services screen also has a button for linking to the services. This button takes you to the same place as the `File->Link to Services...` menu option.

Using Services Within A Flow

Once the OVBPI Modeler knows about some OVO services, you can use these within your business flows.

Let's walk through an example:

Within the OVBPI Modeler you have linked in some OVO services.

In the `Order Flow` flow, you want the node called `Process Order` to be dependent on the OVO service called `OrdersDB`.

- In the flow editor, double-click on the `Process Order` node to bring up the `Node Properties` dialog

- Click on the `Services` tab and you are presented with a new dialog as follows:
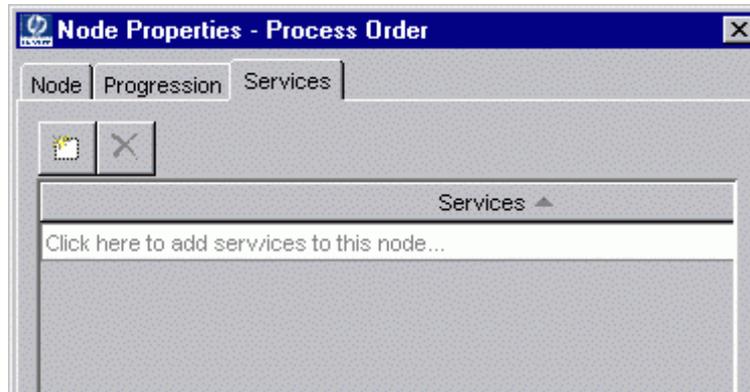
- Click the icon

  ...or... click where it says to add new services.

- This displays the list of available services

  This includes all services known to the OVBPI Modeler, not just the OVO services.

For example, the service list might look as follows:
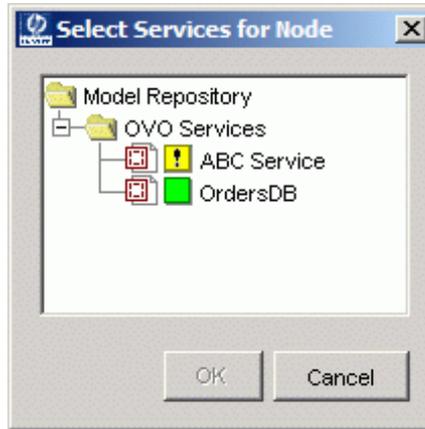
- You then select the `OrdersDB` service and press OK

  You can select both services. A node can be dependent on more than one service and they can be a mixture of OVO and other services.
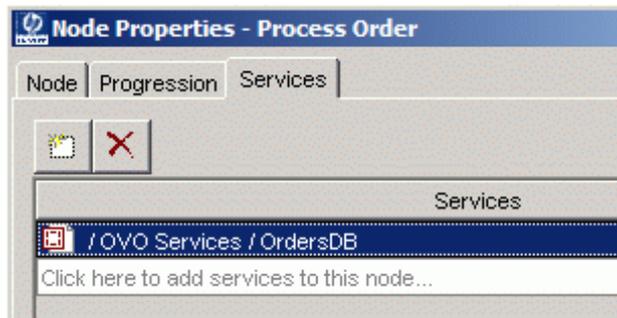
- The Node Properties dialog now displays the chosen service, as follows:

- You then click OK and the `Process Order` node is now marked as being dependent on the `OrdersDB` OVO service

  The node's tooltip within the OVBPI Modeler Flow Editor pane shows this dependency. Just hold the cursor over the node image and wait for the tooltip to appear.

OV Service Navigator (OVSN) Simulator

There is a contributed utility called the OVSN Simulator. It simulates the part of OVO that manages the OVO Service hierarchy. The OVSN Simulator can be used in situations where you do not have a link to a real OVO installation. This is obviously very useful when giving demonstrations.

Installing The OVSN Simulator

- Locate the OVBPI install CD

  The files are located in the `contrib\OVSNSim` directory.

- Copy this `contrib\OVSNSim` directory, and its contents, to your OVBPI installation to create the directory:

  OVBPI-install-dir`\contrib\OVSNSim`

- If your OVBPI installation is not in the default location then you need to edit the following two files:

  — `OVSNSim\ovsnsim.bat`

    Locate the line that sets the environment variable `BIA_ROOT`. It is currently set as follows:

    `set BIA_ROOT=C:\Program Files\HP OpenView\OVBPI`

    You need to set this to point to you OVBPI install directory.

  — `OVSNSim\conf\bia\OVSNAdaptorServer.cfg`

    Locate the line:

    ```
    RegistrationURL=file:///C:/Program%20Files/
          HP%20OpenView/OVBPI/misc/bia/registration.xml
    ```

    You need to replace the reference to:

    `C:/Program%20Files/HP%20OpenView/OVBPI`

    with the directory in which you have installed OVBPI.

The OVSN Simulator is now installed and ready for use.

Using The OVSN Simulator

- Run a command window

- cd to the directory:

  OVBPI-install-dir\contrib\OVSNSim

- Run the command:

  ovsnsim.bat runsimulator

  This first runs the OVBPI OVO Adapter (in a separate command window) and then runs the OVSN Simulator front-end GUI.

  With the Simulator GUI you can:

- Create new services (using the File menu)

  This allows you to link these into the OVBPI Modeler as OVO services. You can then make the nodes within your business flows dependent on these services.

- See which services are being used within currently deployed flows

  When you deploy a flow from the OVBPI Modeler, any dependent services appear within the simulator in the right-hand pane - listed as "Registered Services".

- Set the status of a service

  You can click on any service within the right-hand pane of the OVSN Simulator and set the current status for that service. This is then fed to the OVBPI Business Impact Engine and reflected in any dependent flows.

Simulator Commands Options

There are a number of different command options that you can use with the `ovsnsim.bat` command:

- `runsimulator`

  This starts up both the OVBPI OVO Adapter, and the OVSN Simulator front-end GUI.

- `stopsimulator`

  This stops the OVBPI OVO Adapter. You must then **manually** stop the OVSN Simulator front-end GUI.

- `gui`

  This starts just the OVSN Simulator front-end GUI. This requires that you have already started up the OVBPI OVO Adapter using the `runsimulator` option.

Lab - Using OVO Services

Let's add some OVO service dependencies to your `Order Flow` flow that you defined in the previous chapter.

The `Order Flow` flow is defined as follows:
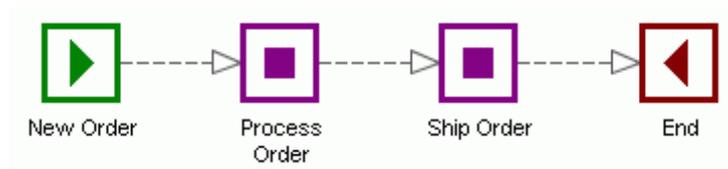
Your task is to set up the following OVO Service dependencies:

- The node `Process Order` is to be dependent on the OVO service called `OrdersDB`

- The node `Ship Order` is to be dependent on the same OVO service called `OrdersDB` as well as being dependent on the OVO service called `ShippingDB`

Enable OVBPI/OVO Integration

You need to make sure that your OVBPI installation has the OVO integration enabled. This is configured using the OVBPI Administration Console.

Let's check that the OVO integration is enabled:

- Run the OVBPI Administration Console

- In the left-hand pane, click on the option:

  `Operational Service Sources`

- In the right hand pane, double-click on the option:

  `OVO interoperability`

- You need to set the hostname field to the host name that is running your OVBPI OVO adapter

    If you are going to be running the OVSN Simulator on your local host then enter the hostname `localhost`.

- Accept the other defaults and click the `OK` button.

- Now, back in the main Administration GUI:

    — Ensure that the `Enabled?` checkbox is checked.

    — Click the `Apply` button.

Now RE-START all the OVBPI components!!!

OVO Setup

The exact way that you carry out this lab depends on whether you have access to a real OVO installation or not. Here are some setup guidelines for using OVO or the OVSN Simulator...

Using An Actual OVO Installation

- On the OVO server make sure that you have installed the OVBPI OVO Adapter and it is configured and running. Refer to the *OVBPI Installation Guide* for more details

- Within OVO, create two service definitions as follows:

```
    Display Label            Service Name
    -------------            ------------
    Orders Database          OrdersDB
    Shipping Database        ShippingDB
```

These two services can be defined at any level within your OVO hierarchy.

Using The OVSN Simulator

- Configure the OVSN Simulator on your OVBPI server as described in the section OV Service Navigator (OVSN) Simulator on page 112

- Create two services (using the `File` menu) called:

    OrdersDB
    ShippingDB

Link To The OVO Service Definitions

You first need to link the service definitions into the OVBPI Modeler. Once you have linked the service definitions, they are then available for inclusion in as many flows as you require. For this lab you just use them within the one flow:

- Within the OVBPI Modeler, select: `File->Link to Services...`
- Select the Service source called: `OVO Services`
- Type in the name of the two services (one to a line)

  ```
  OrdersDB
  ShippingDB
  ```

  ...and click the `Next` button.

  It should find these two services within OVO and list them on the screen with the `Create` option checked.

- Click on the `Create` button
- Close the dialog

You should now see your two services listed in the OVBPI Modeler - in the Navigator pane - under the heading `OVO Services`.

Add Node Dependencies

Now you can specify the nodes within your business flow that are dependent on these services:

- Within the OVBPI Modeler, open the flow editor for the `Order Flow` flow
- Open up the node properties for the node: `Process Order`
- Click on the `Services` tab
- Click to list the available service definitions
- Expand the `OVO Services` section
- Select the service `OrdersDB` and click OK

  You have now assigned the `Process Order` node to be dependent on the OVO service `OrdersDB`.

- Click on the `Ship Order` node

- Set up the `Ship Order` node to be dependent on both the `OrdersDB` and the `ShippingDB` services

  (You can use ctl-click to select more than one service from the list.)

Deploy the New Flow Definition

Now that you have added the OVO service dependencies you need to re-deploy your `Order Flow`:

- Select your `Order Flow`, and deploy it

The OVBPI Business Impact Engine is now ready to receive service status change events from OVO to indicate the health of the `OrdersDB` and `ShippingDB` services.

If you are using the OVSN Simulator for this lab you are now able to see the `OrdersDB` and `ShippingDB` services listed within the OVSN Simulator, under the `Registered Services` heading, and there status shows as normal (green).

▶ If you do not see these services appear under the `Registered Services` heading of the OVSN Simulator, then you have forgotten to restart all OVBPI components after enabling the OVO Interoperability. Restart all OVBPI components, rerun the Modeler and OVSN Simulator, and then redeploy your `Order Flow`.

Testing The Order Flow

- Using the OVBPI Business Process Dashboard you should be able to see your newly deployed `Order Flow`

- Select the `Order Flow` and display the flow details (so you can see the flow definition diagram)

- Use the Generic Event Injector (contributed utility) to send in a new order with the Order ID of `999`

  You see that this new instance is now active and is at the `Process Order` node.

Let's now go to the OVO server and set the `OrdersDB` service to be critical:

- If you are running the OVSN Simulator you can set the status of a service as follows:

  — In the OVSN Simulator GUI, click on the `OrdersDB` service in the right-hand pane - where the `OrdersDB` service is displayed as a green rectangle.

    A new dialog appears allowing you to select the new status for this service.

  — Select `Critical` from the `Set Status` pull-down

  — Now click OK

    The `OrdersDB` service is now set to the state of `Critical`.

- If you are connected to a real OVO installation then the quickest way to set the Orders Database (OrdersDB) service is by sending in an opcmsg for this service. This could be achieved as follows:

On the OVO server:

— Open a command window

— Type in the command:

```
opcmsg service_id=OrdersDB sev=Critical
        object="My Test" application="My Appl"
        msg_text="Test message for the OVBPI Lab"
```

Back in the OVBPI Business Process Dashboard:

- If you refresh the Order Flow diagram you should see that both the Process Order and the Ship Order nodes are showing that their underlying services are in a critical state...and your order 999 is blocked!

If the Business Process Dashboard does not show the services as being in a critical state, there may be a problem with the OVSN Simulator contributed utility. You could try stopping and then restarting the OVSN Simulator utility and this might sort out any problems.

Back in OVO

- Set the status of the OVO service OrdersDB to normal
- Now set the status of the OVO service ShippingDB to critical

Back in the OVBPI Business Process Dashboard:

- Refresh the Order Flow diagram you should see that the Ship Order node is now impacted...and your order 999 is shown as being at risk!

Your business flow is now linked to OVO and your service hierarchy.

Well done! You have reached the end of the lab.

OVIS Integration

The OVBPI/OVIS integration is more than just a way of receiving service status events from OVIS. The integration also includes some OVIS probes allowing OVIS to monitor OVBPI. The main parts of the integration are as follows:

- Within the OVBPI Modeler, you are able to import the OVIS customer and service group hierarchy

  You can then use these within the nodes of your business flows.

  Once deployed, these flow(s) receive service status feeds from OVIS, and OVBPI then monitors any resultant business impact.

- OVBPI provides a series of OVIS probes that you can configure into your OVIS installation

  These probes are able to monitor OVBPI flows and metrics, and can be configured with OVIS objectives.

- OVBPI can monitor OVIS for any SLO/SLA violations and these can be configured to be sent to the OVBPI Notification component

  This allows OVIS SLO/SLA violations to be emailed to you or sent into OpenView. Refer to *OVBPI System Administration Guide* for more details about the OVBPI Notification component.

So, with the OVBPI/OVIS integration you can have your business flows reflecting business impact from OVIS services. You can also have OVIS monitor the behavior of your business flow(s). Indeed, you could put this together and have a business flow depend on an OVIS service that was monitoring that business flow. For example, you could set up your business flow to be impacted when the backlog of instances at a particular node within this flow reaches a certain threshold.

Architecture

OVIS Service Status Feeds

For OVIS to provide service status feeds to OVBPI you need to configure and enable the OVBPI/OVIS interoperability. This is configured using the OVBPI Administration Console; see the *OVBPI System Administration Guide*.

When configured, the sending of OVIS service status feeds works as follows:
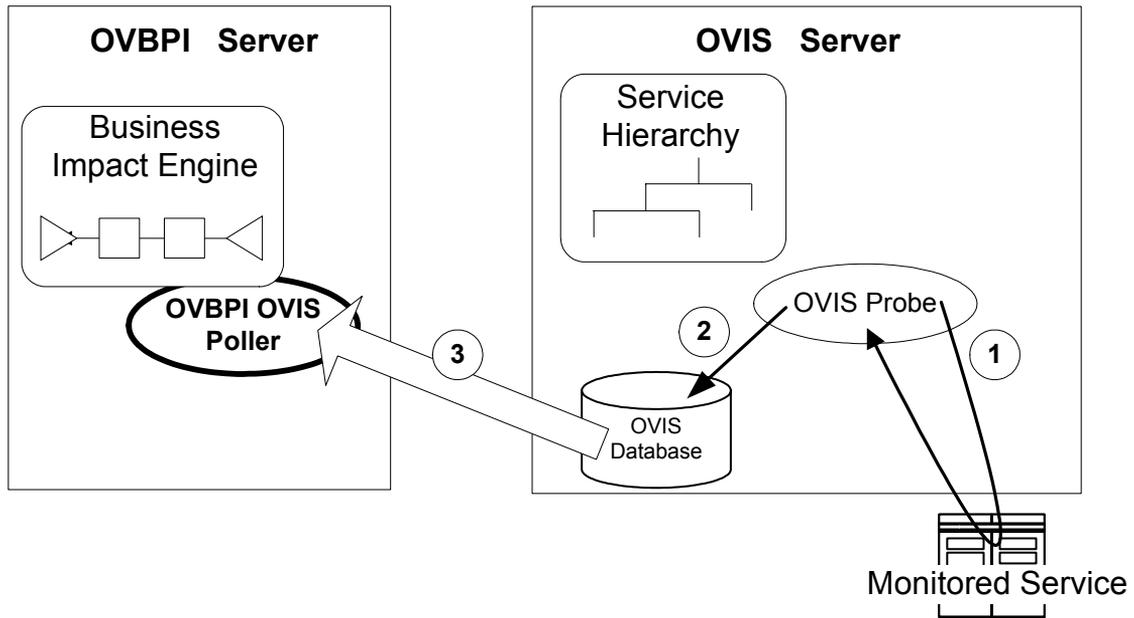
where:

- The OVIS server is configured with a hierarchy of services

  In OVIS terms, these are a series of Customers, with Service Groups, Targets, Objectives, etc..

- For a particular service being monitored by OVIS, there is an OVIS probe periodically probing this service (1)

  The resultant state of this service is stored in the OVIS database (2).

- On the OVBPI server, if the OVBPI/OVIS interoperability is configured, the OVBPI Business Impact Engine starts up an OVBPI OVIS poller thread

  This OVBPI OVIS poller periodically polls the OVIS database to retrieve the status of the monitored service (3). If the service goes critical then the OVBPI Business Impact Engine reflects the business impact in any dependent flows.

OVIS Monitoring OVBPI

For OVIS to monitor your OVBPI flows you need to install the OVBPI probes (that come with the OVBPI installation) onto your OVIS installation. Refer to the *OVBPI Installation Guide* for more details.

There are OVBPI probes available to collect information on:

- Flow level statistics

  This can signal when the number of flow instances is over a set threshold, or the flow throughput rate is too low, etc..

- Node level statistics

  This can signal when the backlog at a particular node is too large, or when the throughput at that node is too high, etc..

- OVBPI Metric statistics

  This can signal when the time taken between two nodes in a flow is too long, etc.

Refer to the *OVBPI Reference Guide* for more details.

The architecture diagram for OVIS monitoring OVBPI looks as follows:
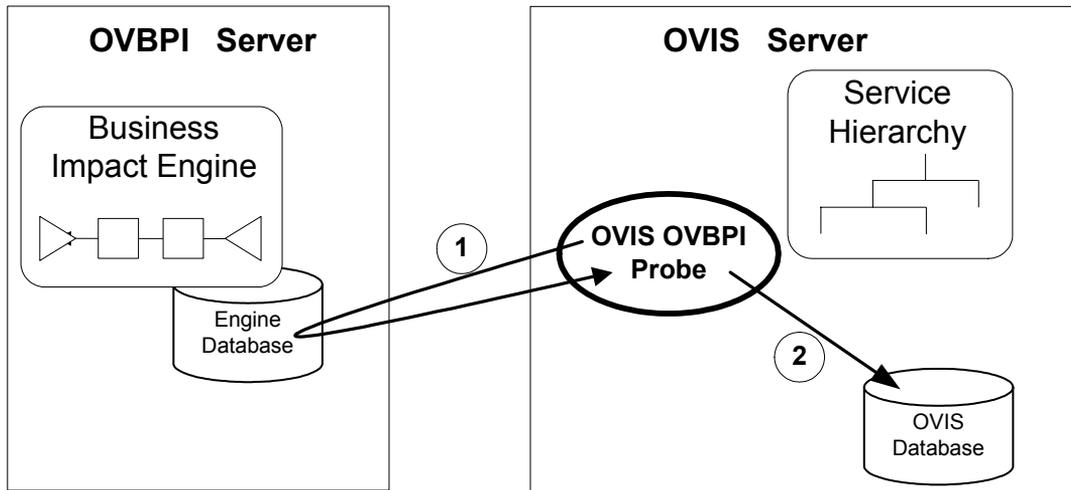
where:

- You configure the OVBPI probe within OVIS

  You specify the OVBPI details (flow/node/metric) that you want collected.

- The probe periodically accesses the OVBPI Business Impact Engine database to collect the statistics (1)

- The results are then stored in the OVIS database (2)

Configuring The OVIS Integration

Configuring the OVIS integration depends on how much of the integration you want to use.

If you want be able to receive OVIS service status feeds then you need to configure the OVIS interoperability within OVBPI. This can be configured using the OVBPI Administration Console. The main details you specify are things such as the OVIS database details and the frequency of polling this database.

If you want to be able to monitor OVBPI flows from within OVIS then you need to install the OVBPI probes onto your OVIS server installation. You can then configure these probes to monitor the flows/nodes/metrics that you require.

Refer to the *OVBPI System Administration Guide, OVBPI Installataion Guide* and *OVBPI Reference Guide* for more details.

The OVBPI Modeler

The OVBPI Modeler lets you synchronize (import) service definitions from OVIS. You simply synchronize the OVBPI Modeler's view of the OVIS service hierarchy in one step, and the services are then available to your flow developers.

To say that the OVBPI Modeler synchronizes the entire OVIS service hierarchy is not strictly correct. Only the top two levels of the hierarchy are synchronized - the Customers and their Service Groups. The OVBPI Modeler does not synchronize any details below the Service Group level.

Synchronizing OVIS Services

The steps to synchronizing the OVIS service hierarchy are as follows:

1. Within the Modeler, select: `File->Synchronize Services...`

   This brings up the following dialog:
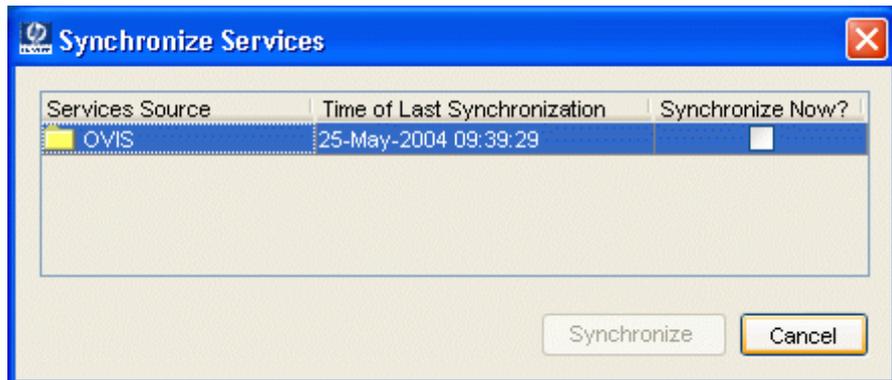
   If your OVBPI installation has other service sources configured then you also see those services listed in the `Service Source` column.

2. You simply check the `Synchronize Now?` box that corresponds to the OVIS Service source, and click on the `Synchronize` button

3. Your OVIS service hierarchy is now synchronized and displayed within the Navigator pane of the OVBPI Modeler.

For example, it might look something like this:
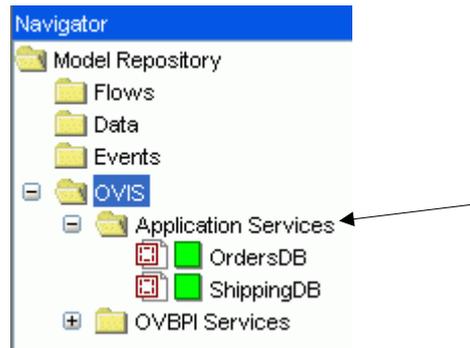
Within the OVBPI Administration Console you can configure the Model Repository to carry out periodic service synchronizations such that your OVBPI Modeler always has an up-to-date list of valid OVIS services.

Using Services Within A Flow

Once the OVBPI Modeler knows about some OVIS services, you can use these within your business flows.

Let's walk through an example:

Within the OVBPI Modeler you have synchronized your OVIS service hierarchy.

In the `Order Flow` flow, you want the node called `Process Order` to be dependent on the OVIS service called `OrdersDB`.

• In the flow editor, double-click on the `Process Order` flow to bring up the `Node Properties` dialog

• Click on the `Services` tab and you are presented with a new dialog as follows:
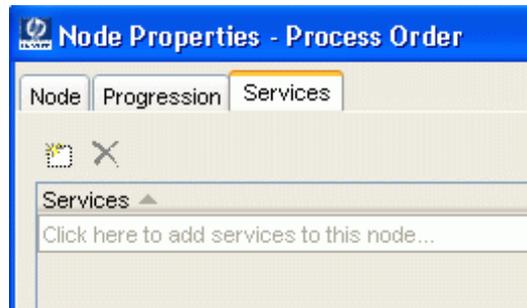
• Click the icon

...or... where it says to click to add services.

• This displays the list of available services

This includes all services known to the OVBPI Modeler, not just the OVIS services.

In this example, the service list might look as follows:
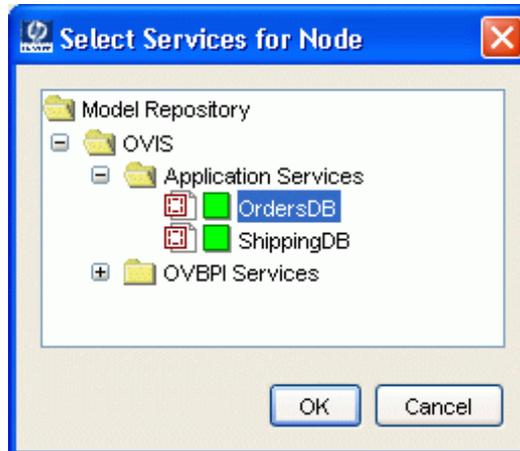
- You then select the OrdersDB service and press OK

  You can select more than one service. A node can be dependent on more than one service and they can be a mixture of other services.

- The Node Properties dialog now displays the chosen service, as follows:
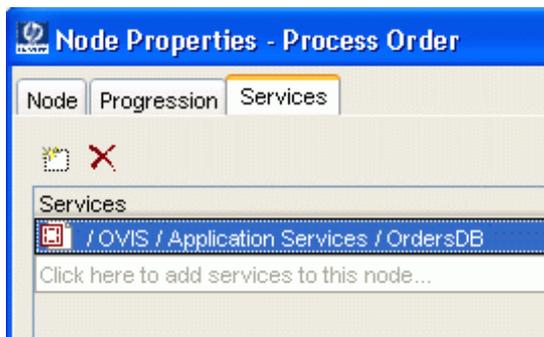
- You then click OK and the Process Order node is now marked as being dependent on the OrdersDB OVIS service

  The node's tooltip now shows this dependency.

Lab - Using OVIS Services

If you have access to an OVIS installation than you may want to try and work through this lab.

Let's add some OVIS service dependencies to your Order Flow flow that you defined in the previous chapter.

The Order Flow flow is defined as follows:
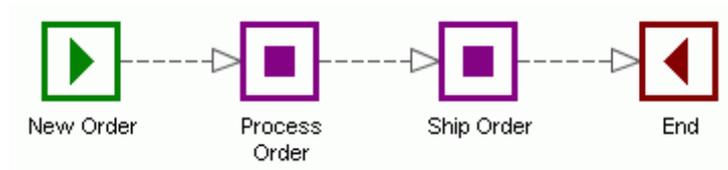
Your task is to set up the following scenario:

- The node Ship Order is to be dependent on an OVIS service called ShippingBacklog

- The ShippingBacklog service is configured within OVIS to monitor the backlog of node instances at the Ship Order node within the Order Flow flow

- If the number of node instances waiting to be handled by the Ship Order node is greater than 5 then you have a blockage and the ShippingBacklog service should be set to a critical state - which in turn means that the Ship Order node should be displayed as impacted within the OVBPI Business Process Dashboard

On The OVIS Server

- Install the OVBPI probes onto your OVIS installation

  Refer to the *OVBPI Installation Guide* for details about this.

- Configure a new OVIS Customer, called: `OVBPI Services`

- Configure a Service Group, called: `ShippingBacklog`, where the monitored service type is the OVBPI custom probe called: `C_OVBPI_NODE`

- Configure a Service Target for this service group

  How to complete this screen is described in the *OVBPI System Administration Guide* - but here are some tips:

  — `Target Host` and `Port`

    The standard OVIS screen handling requires that you provide values for these fields...but they are ignored for all the OVBPI probes.

    So, you can enter anything you like for these two fields. For example, enter `localhost` and `80` respectively.

  — `OVBPI_Identifier`

    When you installed the OVBPI OVIS probes you specified some basic details about your OVBPI server - such as: DB type, DB schema name, hostname. These details are saved in the configuration file:

    `OVIS-install-dir\probes\OvbpiProbe.cfg`

    The details are saved in this file within a section heading of (by default) `OVBPI`. If you were to configure OVIS to talk to more OVBPI servers then you would need to manually edit this `OvbpiProbe.cfg` file and add in new sections - with unique names.

    The `OVBPI_Identifier` is the name of the section heading within this `OvbpiProbe.cfg` configuration file. If you like, the OVBPI identifier describes the OVBPI server being probed.

    So, by default, you set the `OVBPI_Identifier` to the string `OVBPI`.

  — `OVBPI_DB_User_Name`

    Enter the user name to be used when the probe logs on to the OVBPI database. By default, this would be: `ovbpiuser`

— OVBPI_Password

   Enter the password to be used when the probe logs on to the OVBPI
   database. By default, this would be: `ovbpi`

— OVBPI_Flow_Name

   Enter the flow name to be monitored. For this lab, enter: `Order Flow`

— OVBPI_Node_Name

   Enter the node name within this flow to be monitored. For this lab
   enter: `Ship Order`

- Configure Service Objectives for this service such that the service goes
  critical when the number of node instances at `Ship Order` goes above 5

  Choose the `ACTIVE_INSTANCES_HIGH` as the OVIS `Metric`.

  You could set the following table of objectives:

  ```
      Normal   < 3
  3 < Minor    < 4
  4 < Major    < 5
      Critical > 5
  ```

- Set the `Alarm Duration` to be low - set it to one (1) minute.

  This is just so as to help you see quick feedback from OVIS during the
  labs.

- Configure Probe Locations for this service

  Set the host name to `localhost`.

  Set the `Measurement Interval` down to 60 seconds (the minimum) -
  again, this is just to try and speed up things during the lab.

- Save these settings

OVIS is now configured.

On The OVBPI Server

You need to make sure that your OVBPI installation has the OVBPI/OVIS integration enabled:

- Run the OVBPI Administration Console.

- In the left-hand pane, click on the option:

  `Operational Service Sources`

- In the right hand pane, double-click on the option:

  `OVIS interoperability`

- You then need to fill-in the configuration details:

  — Event poll intervals

    For the purposes of this lab you might want to set these all down to 1 minute.

  — OVIS database settings

    You need to set these to correctly point to your OVIS database.

  — Service Import settings

    You need to specify the hostname for your OVIS installation.

- Now, back in the main Administration GUI:

  — Ensure that the `Enabled?` checkbox is checked.

  — Click the `Apply` button.

Now RE-START all the OVBPI components!!!

Synchronize The OVIS Service Definitions

You need to synchronize the service definitions between the OVBPI Modeler and OVIS. Once you have synchronized the service definitions, they are then available for inclusion in as many flows as you require. For this lab you are using them within the one flow.

- Within the OVBPI Modeler, select: File->Synchronize Services...

- Check the Synchronize Now? column in the OVIS Services row

- Click the Synchronize button

You should now see your OVIS service hierarchy listed in the OVBPI Modeler - in the Navigator pane - under the heading OVIS Services.

Add Node Dependencies

Now you can specify the nodes within your business flow that are dependent on these services:

- Within the OVBPI Modeler, open the flow editor for the Order Flow flow

- Open up the node properties for the node: Ship Order

- Click on the Services tab

- Click to list the available service definitions

- Select the OVIS service ShippingBacklog and click OK

- Then click OK to close the Node Properties dialog

  You have now assigned the Ship Order node to be dependent on the OVIS service ShippingBacklog.

Deploy the New Flow Definition

Now that you have added the OVIS service dependencies you need to re-deploy your Order Flow:

- Select your Order Flow, and deploy it

The OVBPI Business Impact Engine is now ready to receive service status change events from OVIS to indicate the health of the ShippingBacklog services.

Testing the Order Flow

- Using the OVBPI Business Process Dashboard you should be able to see your newly deployed `Order Flow`

- Select the `Order Flow` and display the flow details (so you can see the flow definition diagram)

- Use the Generic Event Injector (contributed utility) to send in 6 new orders, using the Order IDs:

```
881
882
883
884
885
886
```

  Progress each of these orders along so that they are all in the `Ship Order` Node.

Back in the OVBPI Business Process Dashboard:

- If you refresh the `Order Flow` diagram you should see the `Ship Order` node showing that its underlying service is in a critical state...and your orders are all showing as blocked!

▶ You may have to wait for a minute or two for the OVIS status to come through to the OVBPI Business Process Dashboard. This is because OVIS needs to detect the state change across a couple of polling periods before it triggers a status alarm. This is all down to how you configure OVIS.

If you have problems... then you could try checking that the OVIS probe is actually running correctly. Some things worth checking in OVIS are:

— In the OVIS Configuration Manager, select `Status` in the left-hand navigation pane. Then in the right-hand pane, double-click on the customer you have configured: `OVBPI Services`. If you scroll down to the latest entry, and then double-click this entry, you should see the results returned from the latest probe. In particular, you can see the value returned for the `ACTIVE_INSTANCES_HIGH` field.

— You could enable OVIS tracing. In the OVIS Configuration manager, select: `File->Configure->Tracing`. Set the `Probe` trace level to be `9`. This is the level that the OVBPI custom probes log trace information. Then you can view the trace file at: *OVIS-install-dir*`\Data\log\probe\trace.log`.

Once your `Order Flow` is showing as blocked, and the `Ship Order` node is showing the service as critical:

- Use the Generic Event Injector to complete one of these orders. Notice the state of the `ShippingBacklog` service drops down to Major. This is due to your Service Objectives as configured in OVIS

- Continue to "ship" orders and notice the state of the `ShippingBacklog` service as it continues to improve

  As for when your overall `Order Flow` flow state returns to "healthy" depends on how you have your OVBPI Business Impact Engine configured. By default, the flow returns to a healthy state only once the `ShippingBacklog` service has returned to a state of `Normal`. This is because the OVBPI Business Impact Engine is configured (by default) to mark flows as blocked if there services are Warning or above (Minor, Major, Critical) - but you could alter this using the OVBPI Administration Console.

Your business flow is now linked to OVIS and your service hierarchy.

<div style="border:1px solid black; text-align:center; padding:20px;">

Well done! You have reached the end of the lab.

</div>

SOA Manager Integration

The OVBPI/SOA Manager integration basically works as follows:

- Within the OVBPI Modeler, you are able to specify the SOA Manager Business Services on which your business flow depends.

- You then deploy your flow to the OVBPI Business Impact Engine.

- As these Business Services change state, this status information is collected from SOA Manager and your OVBPI flow reflects any resultant business impact.

Architecture

For SOA Manager to be able to communicate with OVBPI you need to install and run the OVBPI SOA Manager Adapter component. This SOA Manager Adapter component can be installed on any machine that has access to your network, but typically you would install the SOA Manager Adapter component on the SOA Manager server. Refer to the *OVBPI Installation Guide* for more details.

Let's consider the architecture where you have installed OVBPI and the SOA Manager adapter on different servers:

Figure 9   SOA Manager Integration Architecture
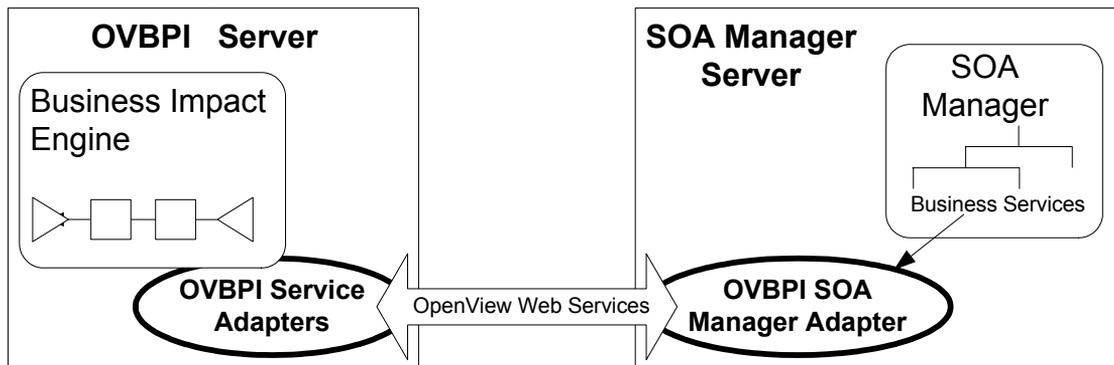
where:

- You install and configure the OVBPI SOA Manager Adapter to talk to your SOA Manager installation.

- Within the OVBPI Administration Console, you configure a connection to your OVBPI SOA Manager Adapter.

- You make sure the OVBPI Service Adapters component is running, as this is the component that handles the OVBPI side of the SOA Manager connection. This communication uses the OpenView Web Services interface.

- When defining a flow within the OVBPI Modeler you specify the names of the SOA Manager Business Services that you wish to use. The OVBPI Modeler calls the OVBPI Service Adapters component (which, in turn, calls the OVBPI SOA Manager Adapter) to determine whether these services exist as Business Services within SOA Manager.

- You deploy your flow to the OVBPI Business Impact Engine.

- The OVBPI Service Adapters component polls the OVBPI SOA Manager Adapter to get status details about the SOA Manager Business Services. Any status changes for these Business Services are passed through to the OVBPI Business Impact Engine.

Configuring the OVBPI SOA Manager Integration

Configuring the OVBPI SOA Manager integration requires some set up on both the SOA Manager server and the OVBPI server. Even if you have installed OVBPI on the same server as SOA Manager, these steps are still required:

- On the OVBPI Server

  You configure the SOA Manager connection details using the OVBPI Administration Console:

  — In the left-hand navigation pane, select the option:

    ```
    Operational Service Sources
    ```

  — Click the `Add...` button and configure a connection to a SOA Manager adapter.

    The main pieces of information that OVBPI needs to know are the name of the server running the OVBPI SOA Manager adapter, and the port number for the Web services communication.

    You give your SOA Manager connection a name. For example: `My SOAM Adapter`. This name is then used to create the service folder within the OVBPI Modeler. Once set, you cannot modify this folder name.

  — Ensure that your new adapter has a tick in the `Enabled?` column.

  — Click `Apply` to save these configuration details.

  You can configure connections to more than one SOA Manager adapter, you just need to give each connection a unique name.

  You can refer to the *OVBPI System Administration Guide* for more details.

  After configuring your new SOA Manager adapter(s) you **need to restart all OVBPI components.**

- On the SOA Manager Server

  You need to install the OVBPI SOA Manager Adapter and configure it to talk to your SOA Manager installation.

  Once the OVBPI SOA Manager Adapter is installed and running, it is available to OVBPI.

  Refer to the *OVBPI Installation Guide* for more details.

The OVBPI Modeler

The OVBPI Modeler lets you import Business Service definitions from SOA Manager. This then allows you to define the dependencies between the nodes in your flows and the underlying SOA Manager Business Services.

The services you import into the OVBPI Modeler are effectively just pointers (or "links") to the Business Services defined and managed within your SOA Manager. That is, the services are still maintained and owned by SOA Manager.

Linking SOA Manager Services

The OVBPI Modeler does **not** import the full SOA Manager Business Service hierarchy. Instead, the flow developer specifies the individual services by name.

Let's walk through an example...

The steps to link SOA Manager Business Service definitions are as follows:

1. Within the Modeler, select: `File->Link to Services...`

   This brings up the dialog:
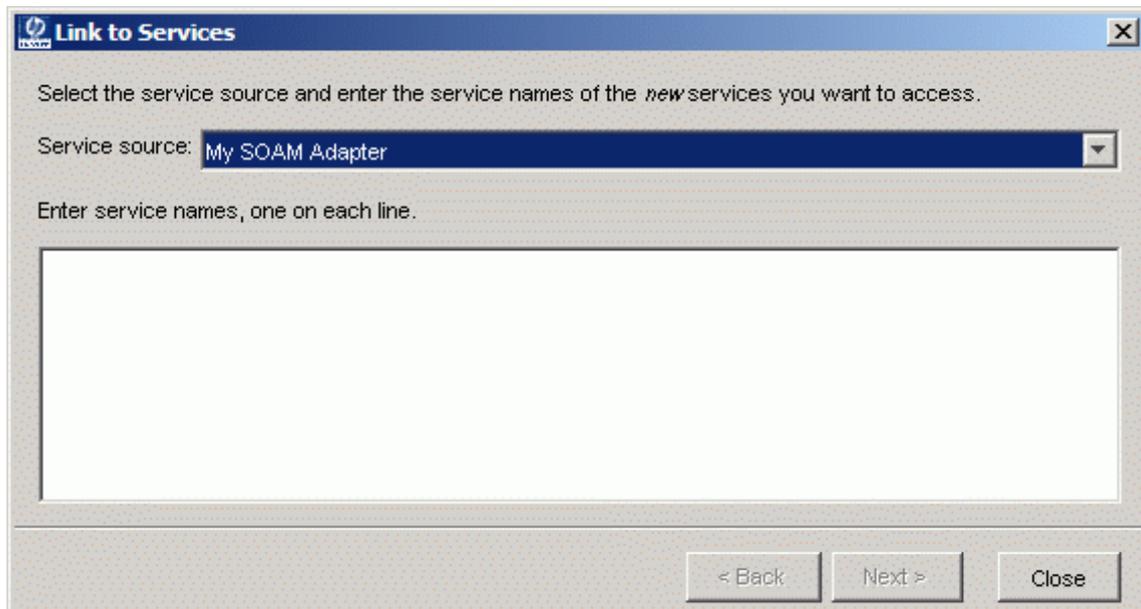
2. When the `Link to Services` dialog appears, you select the appropriate `Service source`. In this example, you would select the SOA Manager service source called: `My SOAM Adapter`. (You had previously configured this service source using the OVBPI Administration Console. See Configuring the OVBPI SOA Manager Integration on page 139)

3. Type in the name(s) of the SOA Manager services you wish to link to the OVBPI Modeler. The service names must be Business Services as defined within SOA Manager. (Within SOA Manager, you can use the Business Service Explorer (BSE) to list the available Business Services.)

For example, to link the SOA Manager Business Service definition `Finance`, the dialog would look as follows:
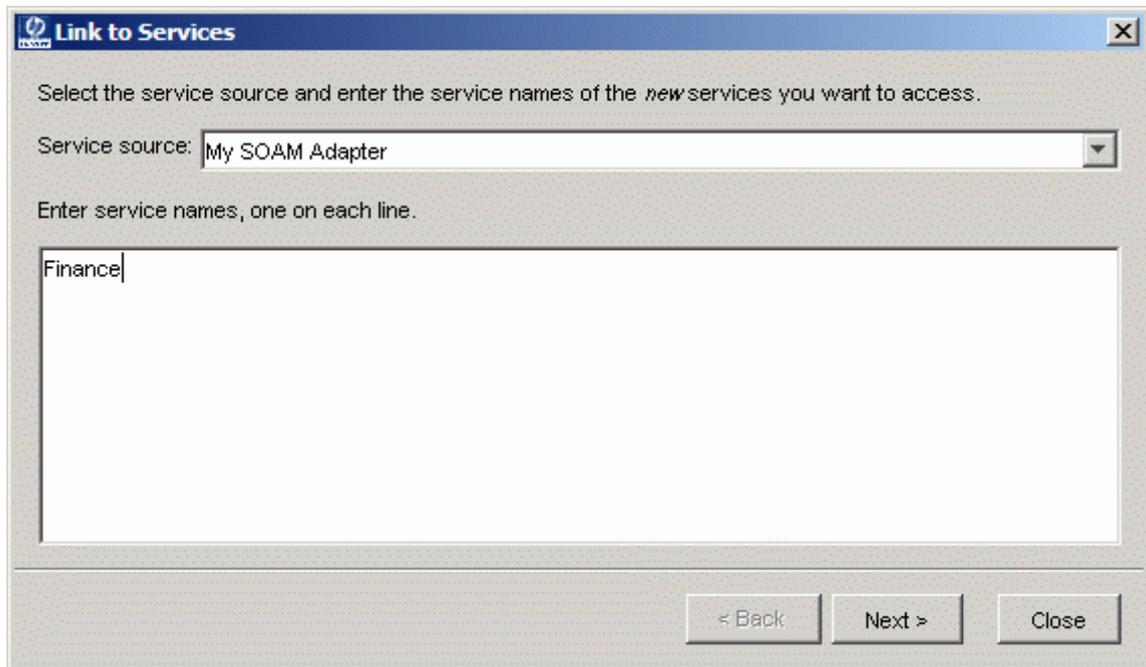
The name(s) that you type here must match the name of the Business Service(s) within SOA Manager.

If you want to link in more than one Business Service, simply type each name on a new line.

4. Click on the `Next` button

   It is at this point that the OVBPI Modeler goes out and talks to the SOA Manager adapter to locate the Business Service definition for the name(s) specified.

   If the OVBPI Modeler is able to communicate with the SOA Manager Adapter then you see the following dialog:
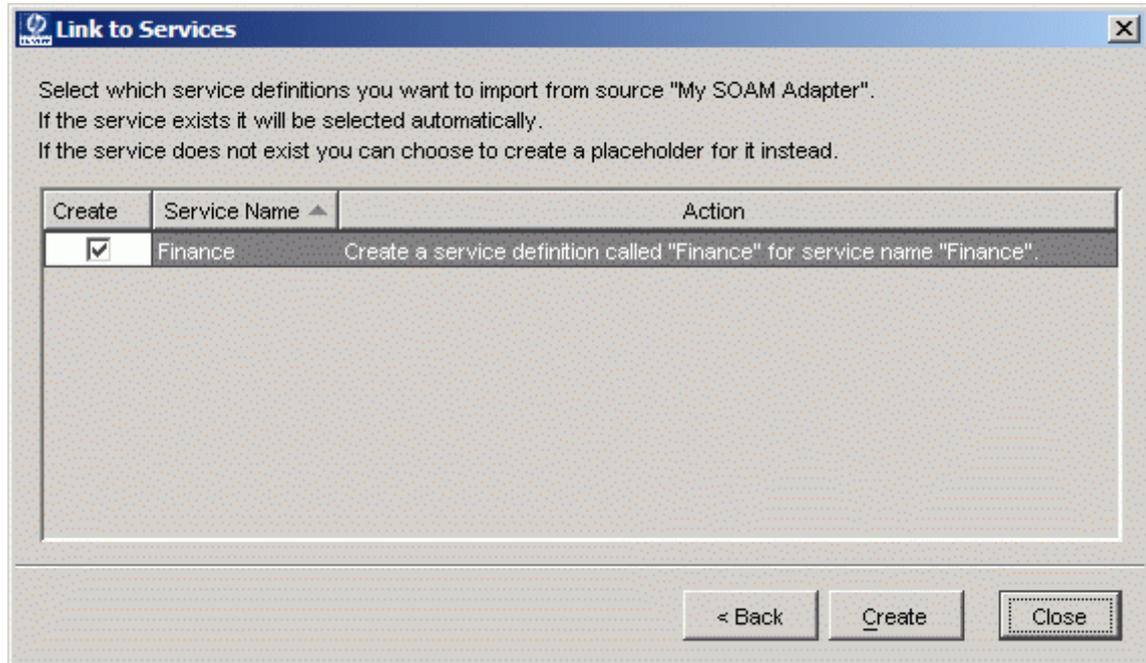
   If the service name exists in SOA Manager as a Business Service, it is listed here with the `Create` option checked - all ready for you to accept.

   If the service name does not exist in SOA Manager, it is listed with the `Create` option left un-checked. This occurs for two main reasons:

   — Either you mis-typed the service name

   In which case you click the `Back` button and re-type the correct name.

   ...or...

— You have specified a name that you know is going to exist within SOA Manager some time in the future and you want the OVBPI Modeler to create a place holder.

By creating a placeholder within the OVBPI Modeler, this allows you to define business flows that use this service. When this service is eventually created within SOA Manager, any deployed flows start receiving status change events for this service.

5. Once the correct services are listed and you have correctly set the Create option for each one, simply press the Create button.

You see a progress dialog. When this is completed, you can close this progress dialog.

You are now able to see your newly linked SOA Manager Business Services. They are listed in the Navigator pane of the OVBPI Modeler, under a folder called: My SOAM Adapter - which is the name of the service source you used to import this service.
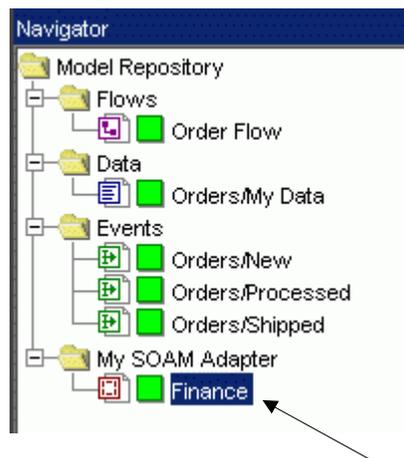
Notice that the newly linked service Finance is marked as deployed (with a green icon next to its name). This does not mean that it is deployed within the OVBPI Business Impact Engine, it means that this service corresponds to an active SOA Manager Business Service definition.

Placeholder Services

Suppose you know that a service is going to be defined in SOA Manager sometime next week, but you want to start defining the business flow right now. You want to define a placeholder SOA Manager Business Service.

Consider the example where you want to define a placeholder service called: ABC Service:

• As described in the previous section (Linking SOA Manager Services on page 140) you select:

    File->Link to Services...

• Select the appropriate service source. For this example, you select the SOA Manager adapter configured as: My SOAM Adapter.

• Enter the name: ABC Service, and click Next

    Notice that the Create option is not checked, and the Action column explains that if you do check the create option then this creates a placeholder for this Business Service definition.

• Check the Create option for this service

• Now click the Create button

    This creates a placeholder in the OVBPI Modeler for this Business Service.

When you look in the Navigator pane of your Modeler, you see your service as follows:
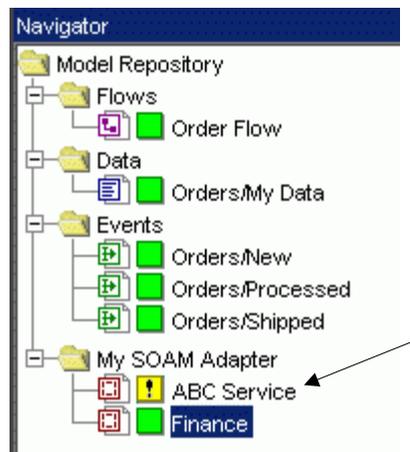
Notice it is listed with an icon that indicates that this service definition does not currently exist in SOA Manager.

You can quite happily refer to a placeholder service definition within your flow and then deploy this flow to the OVBPI Business Impact Engine. It just means that your flow does not receive any service status change events as the Business Service does not yet exist within SOA Manager. Once the Business Service has been created within SOA Manager your deployed flow automatically starts receiving service status change events and your flow reflects the business impact.

Synchronizing Services

Within the OVBPI Modeler, there is the option to synchronize your service definitions. This tells the OVBPI Modeler to synchronize its view of the SOA Manager Business Services.

To synchronize service definitions, you:

1. Select the menu option:

   `File->Synchronize Services...`

   You see a dialog similar to the following:
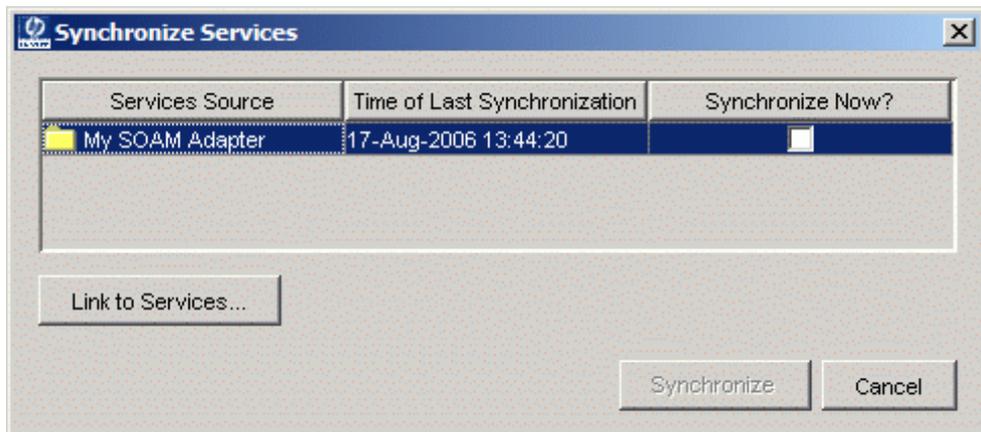
If your OVBPI installation has other service integrations configured (such as OVO or OVIS) then you also see thoses listed in the `Services Source` column.

2. You simply check the `Synchronize Now?` checkbox and click on the `Synchronize` button.

   This tells the OVBPI Modeler to check the existence of all the references it has to SOA Manager Business Services and see if they exist in the SOA Manager hierarchy:

   — Any placeholder services become marked as deployed (green icon) if the Business Service definition is found to exist in SOA Manager

   — Any currently deployed services become placeholders if (for some reason) they have been removed from SOA Manager

In the example above, with the placeholder called `ABC Service`, the icon changes to green once a Business Service of this name is created within SOA Manager and you have run a synchronize.

Notice that the Synchronize Services screen also has a button for linking to the services. This button takes you to the same place as the `File->Link to Services...` menu option.

Using Services Within A Flow

Once the OVBPI Modeler knows about some SOA Manager Business Services, you can use these within your business flows.

Let's walk through an example:

Within the OVBPI Modeler you have linked in some SOA Manager Business Services.

In the `Order Flow` flow, you want the node called `Process Order` to be dependent on the SOA Manager Business Service called `Finance`.

- In the flow editor, double-click on the `Process Order` node to bring up the `Node Properties` dialog

- Click on the `Services` tab and you are presented with a new dialog as follows:
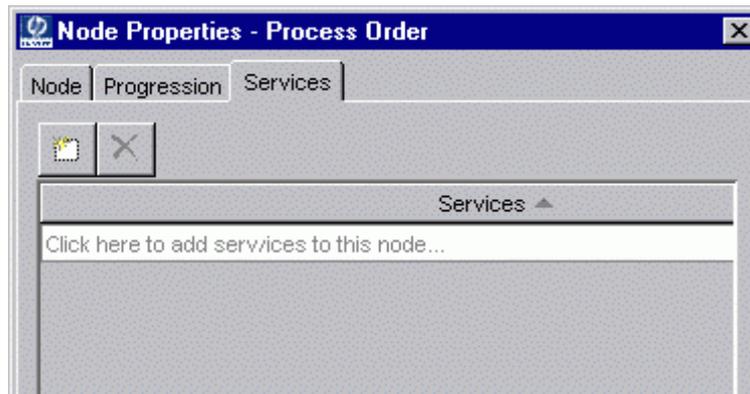
- Click the icon

  ...or... click where it says to add new services.

- This displays the list of available services

  This includes all services known to the OVBPI Modeler, not just the SOA Manager Business Services.

For example, the service list might look as follows:
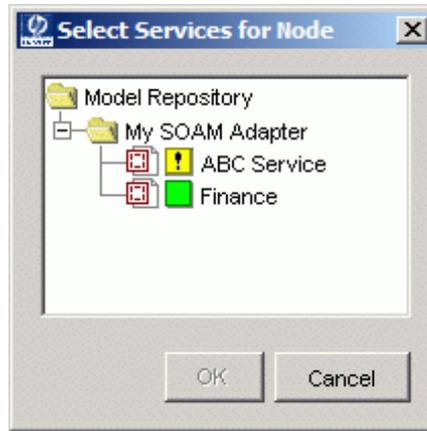
- You then select the `Finance` service and press OK

  You can select both services. A node can be dependent on more than one service and they can be a mixture of SOA Manager and other services.

- The Node Properties dialog now displays the chosen service, as follows:
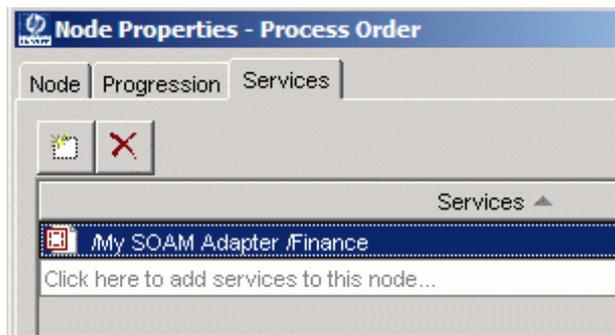
- You then click OK and the `Process Order` node is now marked as being dependent on the `Finance` SOA Manager Business Service

  The node's tooltip within the OVBPI Modeler Flow Editor pane shows this dependency. Just hold the cursor over the node image and wait for the tooltip to appear.

Standalone Services

What if you have services that are not monitored by any of the standard integrations offered by OVBPI? That is, you have services that are monitored by some 3rd party system or a new monitoring system that you are yet to write(!). How can you set these up and define flows that depend on their status?

The OVBPI Modeler lets you define Standalone Services. That is, services that are not monitored by any of the standard integrations offered by OVBPI.

Defining a New Service

Within the OVBPI Modeler, you can define a standalone service by either selecting:

```
File->New->Standalone service
```

or simply click on the new standalone service icon:

Create a new
standalone service...

This brings up the service definition page - in the right hand pane of the Modeler.

You can then assign the standalone service a `name` and (optionally) a `description`.

Your new service is then listed in the Navigator pane, under the `Standalone Services` folder. For example, if you create a standalone service called `MyService`, it might appear as follows:
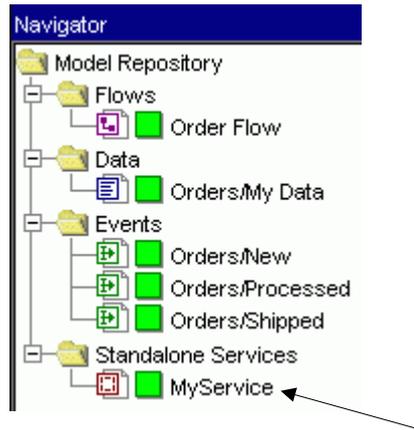
Using a Standalone Service

Once your standalone service is defined within the OVBPI Modeler, you are able to specify your nodes as being dependent upon this service in exactly the same way as you would for any service.

You can then deploy the flow.

For the OVBPI Business Impact Engine to receive service status events for this standalone service you must set up some form of monitoring system (external to OVBPI) that monitors the actual service and then sends in service status change events through the OVBPI Event Handler. Refer to the *OVBPI Integration Training Guide - Business Events* for details on how to set this up.

4 Advanced Flow Definition

This chapter covers some more advanced topics of flow definition.

Actual Node Ordering

Suppose you have defined the following flow segment:

```
- - - - - - - ►  ┌───┐  - - - - - - - - - - - ►  ┌───┐   - - - - - - - ►
                 │ A │                           │ B │
                 └───┘                           └───┘
```
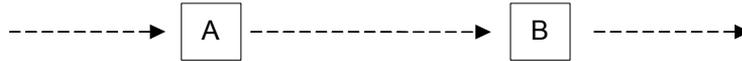
It is important to realize that there is no guarantee that the nodes activate in the order as shown in the diagram.

The flow diagram is what you believe is going to happen. At run time, the OVBPI Business Impact Engine actually tells you what has happened and the order in which the nodes actually occurred.

The connecting arcs are just there for you to indicate the ordering that you expect to occur. The actual node progression is dictated by your node progression rules and the order in which the work actually occurs in your underlying application systems.

Trapping Nodes Out Of Sequence

If you do wish to dictate the ordering of some, or all, of the nodes, and trap any times when this ordering is not obeyed...you can.

When you define a connecting arc in the OVBPI Modeler, you can right-click on the arc and select that you wish to **Check Sequence**.

This changes the arc to be a solid line (rather than a dashed line).

For example:

```
- - - - - - - ►  ┌───┐  ─────────────►  ┌───┐   - - - - - - - ►
                 │ A │                   │ B │
                 └───┘                   └───┘
```

This tells the OVBPI Business Impact Engine (at run time) to trap any situation where node B is started before node A. If node B does start before node A then OVBPI issues a `Flow Out Of Sequence` notification. (The destination of this notification can be configured within the OVBPI Notification component.)

➤ Note that the out of sequence condition is only tested when node `B` starts. That is, if node `B` completes (without having first started) before node `A` has started or completed, then this does not cause an out of sequence trap to be caught.

Nodes With No Arcs

There is nothing to stop you designing a business flow where you have one or more nodes that are not connected by arcs to any other nodes.

For example:
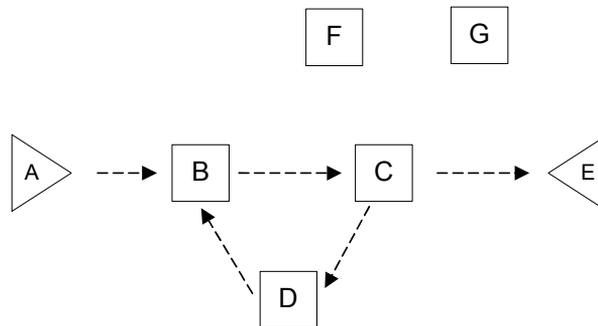
where:

- The flow should normally progress along the path: A, B, C, D, B, C, E

- But at any stage in this progression the flow could switch out to either Node `F` or `G`

    Rather than trying to draw lines from every node to `F`, and then every node to `G`, just show them as "stand alone" nodes within the flow.

A stand alone node is started/completed according to its start/complete progression rules - just like any other node.

Active Flows With No Active Node(s)

It can happen that you deploy a flow and then, when monitoring it with the OVBPI Business Process Dashboard, you notice that you have an active flow instance however it does not show as being active within any of the nodes of that flow.

For example, you might have one active flow instance, but when you list the number of flow instances at each node you see all nodes showing that they have no active instances.

This can seem very confusing at first sight. It can also make you feel that perhaps OVBPI has got it wrong. No...it is all to do with your node progression rules.

Let's consider this example flow:

Node B: Complete
Prop1.after() == "Yes"

A ⟩ ---▶ B ------▶ C -----▶ ⟨ E

Node C: Start
Prop2.after() == "Ready"
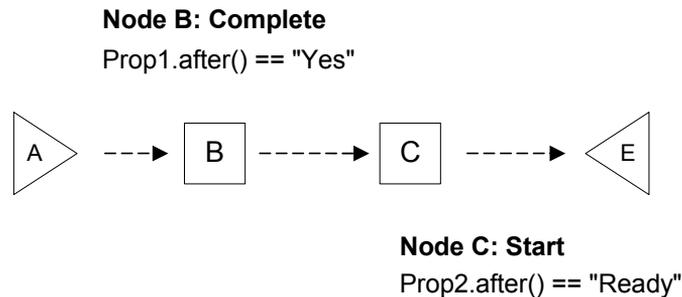
- When the flow enters node B the properties are in the state:

  ```
  Prop1 = "No"
  Prop2 = null
  ```

- When the event comes in to set Prop1 to "Yes" you then have the following property values:

  ```
  Prop1 = "Yes"
  Prop2 = null
  ```

  Which means that the node B completion rule is satisfied - so the flow instance moves out of node B.

  However, Prop2 did not change to "Ready" so the flow instance does not move in to node C.

Thus the OVBPI Business Impact Engine has an active flow instance - and this shows in the OVBPI Business Process Dashboard allowing you to see its existence - however the OVBPI Business Process Dashboard is not able to show you where the flow is at the moment.

This situation normally occurs because you have "gaps" in your flow. That is, you are not representing all the stages/transitions of the flow. There is nothing wrong about this. OVBPI quite happily monitors your flow and reports what it can. It just might surprise you when/if it happens.

If you do have this situation where you have active flow instances that are not showing up in your node instance list, then it might mean that you need to model more of your business flow. Or it might mean that you have made an error in one or more of your progression rules. Of-course, it might be exactly what you want :-)

Activity Nodes Starting Flows

You would normally expect to design a flow with a start node and then have that start node actually start the flow instance. And, by default, that is the normal behavior - where a flow instance is only created when the start node's start condition is satisfied.

However, you can also design your flow to allow a normal activity node(s) to start your flow instance.

If you open up the properties dialog for an activity node you see two check boxes available to you:

where:

- The "OK to Start the flow..." check box

  This applies in the situation where the start condition for this node has been satisfied, however, there is no current flow instance.

  This tells the OVBPI Business Impact Engine that if this start condition is satisfied, and there is no current flow instance, then go ahead and instantiate one.

  In other words, this activity node is able to start the flow instance if the flow instance has not already been started.

- The "Log a Warning..." check box

  You can check this only if you have checked the "OK to Start..." check box.

  This tells the OVBPI Business Impact Engine that you would like to log a warning message whenever this node does actually instantiate a new flow instance.

  The log message is logged to the OVBPI Business Impact Engine's log file, which can be viewed using the OVBPI Administration console.

  The message is logged as a warning. For example:

  ```
  WARNING: Entering a flow instance at node "Assigned" when
           the instance did not already exist.
  ```

Junction Nodes

A junction node can be placed anywhere within a flow diagram. You can have more than one junction node within a flow diagram. You may choose to have no junction nodes within your flow. Junction nodes are optional.

So why does OVBPI have Junction nodes?

You may use a Junction node to represent a step within the flow that you are choosing not to monitor. You may use a Junction node to represent a decision that takes place within the underlying flow. Some people may find that using Junction nodes simply helps them layout their flow diagram. Junction nodes are purely there to aid with the visual layout of your flow diagram.

For example, suppose you were modeling an order flow system. You might draw the flow as shown here in Figure 10:

Figure 10 Order Flow Diagram
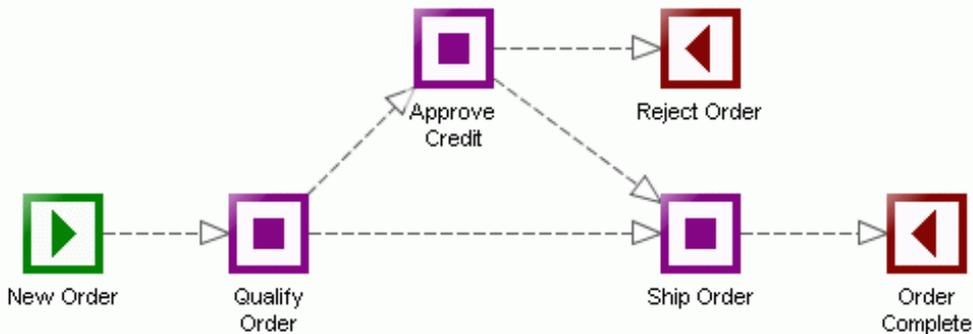
The above flow shows that when an order comes in, you first qualify the order to see if the customer is a special customer or not. In the case of a special customer's order, you choose to expedite this order as there is no need to aprove their credit. You can go straight ahead and ship the order. For all non-special customers, you first approve their credit, and if they are approved, you ship the order, else you reject the order.

Some people might feel that the diagram in Figure 10 does not fully show these decision points. That is, how do you know, simply from looking at the diagram, that the Approve Credit and the Ship Order node are not caried

out in parallel? Also, the diagram does not obviously show why an order might take the path through the `Approve Credit` node as opposed to the `Ship Order` node.

You can use Junction nodes to add this additional information. For example, you could redraw the order flow as shown in Figure 11:

Figure 11 Order Flow Diagram with Junction Nodes
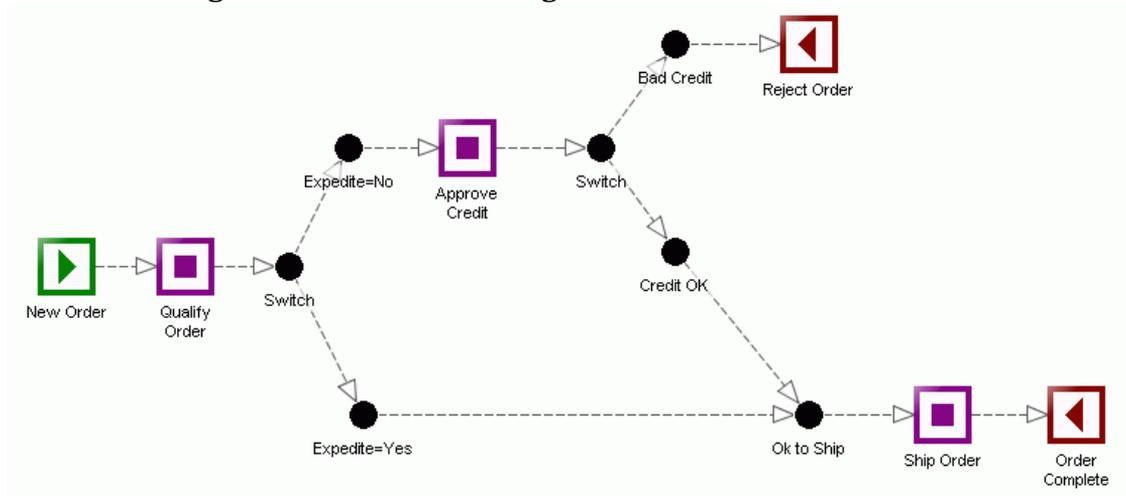
Now you can see that after the order is qualified, there is a decision which sends the order one way or the other. If the order is marked to be expedited, then the order goes through one path, else it goes through the other path. The Junction nodes are there purely to help show the structure and decision points within your flow diagram.

A Junction node has no progression rules and has no dependencies. It is simply a node that you can add to the flow diagram to help you represent the layout of the flow.

The name of a Junction node does not need to be unique within the flow diagram. You can have duplicate Junction node names.

Event Subscription

Event/Data Filtering

When you set up an event subscription, you typically check the box to specify that the incoming event is to be matched to a specific data instance.

Of-course, you don't have to specify this. By not ticking that box you can allow the incoming event to affect all data instances. This might be appropriate if the incoming event was an update of a general nature.

Regardless of which option you have selected (specific or non) you can set up additional filtering to be applied to the incoming event. You specify this in the Filter section of the subscriptions dialog:
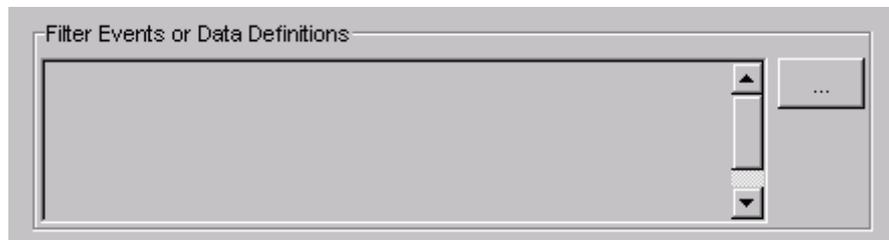
The idea is that you can specify (by clicking on the " ... " button) additional filtering. This filtering must be any expression that results in a boolean true or false.

Example 1:

```
event.CustomerType == "Gold"
```

Applies this incoming event if the event property `CustomerType` is set to the string `Gold`.

Example 2:

```
    event.CustomerType == "Silver"
&& event.Value > 1000
```

Applies this incoming event if the event property `CustomerType` is set to the string `Silver` and the event property `Value` is set to a value greater than `1000`.

Your filters can be based on the property values within the incoming event and/or the current data instance values.

The Model Repository

When you are running the OVBPI Modeler you are connected to the Model Repository.

Repository Startup

The Model Repository is started/stopped by the OVBPI Administrator console.

At startup, the Model Repository performs a number of consistency checks that are worth understanding about.

When the Model Repository starts up it tries to check the deployed state of all its definitions - Flows, Data, Events. To do this it needs to connect and talk to the OVBPI Business Impact Engine. Hence it is always a good idea to start the OVBPI Business Impact Engine component before starting the Model Repository.

The Model Repository connects to the OVBPI Business Impact Engine and finds out whether the definitions that it thinks are deployed are indeed still deployed to the Engine. If the OVBPI Business Impact Engine has (for example) undeployed a flow, the Model Repository is able to flag that this flow is no longer deployed.

Thus, if you restart the Model Repository and then re-run the OVBPI Modeler you see a clean indication of which flow/data/etc. definitions are actually deployed to the OVBPI Business Impact Engine.

However, be careful because if the Model Repository is started when the OVBPI Business Impact Engine is not running then, when you run the OVBPI Modeler, you see **all** your flows and data definitions appear to be **undeployed**!

Importing And Exporting Definitions

The OVBPI Modeler allows you to import and export definitions.

The most obvious example is where you have defined a flow - and associated data, events, etc. - on a test machine. You now want to export this all to a file, take this file to your production server, import it and deploy it.

Exporting

To export a definition from within the OVBPI Modeler:

- Click on the definition you wish to export (in the Navigator pane)

- Then select:

  ```
  File->Export Definition...
  ```

- Then specify the file to which you want the export saved

  This exports the selected definition **and all related definitions**.

  The definition(s) are saved to a zip file.

For example, if you export a flow, this exports the flow definition and any associated data definitions, event definitions, etc. It exports everything required by that flow.

You can also export the entire Model Repository in one go, using the OVBPI Repository Explorer - see Exporting Definitions on page 165 for more details.

Importing

To import definitions into the OVBPI Modeler:

- Select:

    ```
    File->Import Definitions...
    ```

- Then choose the input file to import.

    You can select either a previously exported (.zip) file, or a file containing a Business Process Execution Language (.bpel) file.

    If you have selected a ZIP file:

    — You are then presented with a list of all the objects to be imported. If there are any clashes with current definitions you have the ability to decide whether to overwrite/rename/ignore these definitions.

    If you have selected a BPEL file:

    — You are presented with a range of import options which allow you to affect how the import is performed.

- When you are happy with your import options...click the `Import` button

Any definitions that you import, come into the OVBPI Modeler as being **undeployed**. It is then up to you to deploy this definition to the OVBPI Business Impact Engine.

Repository Explorer

As well as the OVBPI Modeler, there is an additional tool that allows you to view the Model Repository, it is the OVBPI Repository Explorer.

The OVBPI Repository Explorer is a set of JSP pages that display the contents of the Model Repository within a Web browser window. You cannot create or modify definitions from within the OVBPI Repository Explorer, but there are a number of features that are quite useful.

Running the Repository Explorer

There are two ways to run the OVBPI Repository Explorer:

1. Standalone
2. From within the Modeler

Standalone

To run the OVBPI Repository Explorer standalone, run a Web browser and go to the URL:

```
http://hostname:44080/ovbpirepositoryexplorer
```

where:

- *hostname* is the hostname of your OVBPI Model Repository
- *44080* is the default port for the OVBPI Servlet Engine

You are prompted for a user name and password. These are the same user name and passwords as required for the OVBPI Modeler. The default user name and password details are:

```
User Name:  admin
Password:   ovbpi
```

(To change the admin user password, and to configure the underlying authorization mechanism, refer to the *OVBPI System Administration Guide*.)

From Within the Modeler

You can launch the OVBPI Repository Explorer from within the OVBPI Modeler.

You can either select:

        Window->Repository Explorer

from the menu item, or simply click on the OVBPI Repository Explorer icon:

Choosing either of these options launches a Web browser with the correct URL specified. You still need to logon as a valid repository user.

View Details

The OVBPI Repository Explorer lets you view your definitions. The details are displayed for each object all on a single screen. This means that you can see a flow's definition, including all the progression rules, on a single screen. The same is true for viewing data definitions. You can view all the event subscriptions together on one screen. This can be very useful when developing or troubleshooting a definition.

The OVBPI Repository Explorer also lets you view the details of **superseded** definitions. Select a definition, and then click the History tab (in the right-hand frame). For example, this allows you to see the flow diagram of a superseded flow, or the data properties of a superseded data definition. You can also walk through the links from a flow definition to its corresponding data definition.

Recycle Bin

When you delete definitions within the OVBPI Modeler these definitions are not physically deleted. They are simply marked as deleted within the Repository. Once marked as deleted, the definition are no longer displayed in the OVBPI Modeler.

To see the definitions currently marked as deleted, you use the OVBPI Repository Explorer and look in the Recycled bin.

Cleanup Recycled

You can individually remove items from the Recycled bin, or you can ask to remove all items from the Recycled bin.

To remove all items from the Recycled bin, you:

- Click on the `File` icon within the `Navigator` pane of the OVBPI Repository Explorer

- Then select the `Cleanup Recycled` option

Restore

If you have accidently deleted a definition from within the OVBPI Modeler, you can go into the OVBPI Repository Explorer and restore that definition from the Recycled bin.

Exporting Definitions

The OVBPI Repository Explorer lets you export a definition hierarchy, or the entire Repository (excluding the Recycled bin).

Export

To export a single definition hierarchy:

- Select the definition (in the Navigator frame) you wish to export

- Click on the `File` icon, within the Navigator frame.

- Select `Export`

  In the right-hand frame you see the list of definitions that are to be exported.

- Click the `Download` button in the right-hand frame

Export All

To export the entire contents of the OVBPI Model Repository - excluding items in the Recycled bin:

- Click on the `File` icon, within the Navigator frame.

- Select `Export All`

  In the right-hand frame you see the list of definitions that are to be exported.

- Click the `Download` button in the right-hand frame

Definition Change Hierarchy

The OVBPI Modeler keeps track of what definitions have been changed since they were last deployed. Indeed, the OVBPI Modeler only lets you deploy definitions that have changed since the last time they were deployed.

If you change a flow definition, the OVBPI Modeler marks that flow definition as having been changed, and marks it available to be redeployed.

However, if you were to make a change to a data definition, the OVBPI Modeler marks the data definition as changed **and** all flow definitions that have it as a related data definition. That is, the OVBPI Modeler has a system for marking what has changed. The dependencies are as follows:

Item Changed	Item(s) Marked as Changed
Flow definition	Flow definition
Data definition	Data definition and all related Flow definitions
Event definition	Event definition, all data definitions that subscribe to this event, and all flow definitions that are related to this(these) data definitions.

Deployment

What Gets Deployed?

When you select a definition to deploy, the OVBPI Modeler deploys the selected definition and any dependency. This is why, when you deploy a flow, the OVBPI Modeler deploys all the parts necessary to enable that flow to be deployed and run.

There is nothing stopping you deploying an event definition. In which case, only that event definition is deployed. You can deploy just a data definition - and the OVBPI Modeler deploys that data definition and any as yet undeployed event definitions that this data definition is subscribed to.

But don't forget that the OVBPI Modeler only deploys definitions if they have been changed since the last time they were deployed. Thus, the first time you deploy a flow, the OVBPI Modeler deploys everything. You then make a change to the flow definition (maybe move the position of a node, or change the description text). You then re-deploy this flow definition...and the OVBPI Modeler only deploys the flow definition. This is because the related data and event definitions have not changed and thus there is no need for them to be redeployed. There is nothing wrong with this behavior - it just surprises some people when they first notice it.

Superseded Flows

When you redeploy a flow, the OVBPI Business Impact Engine wants to delete the previously active version of the flow. And if there are no flow instances (active or completed) for this old flow definition...the OVBPI Business Impact Engine deletes it.

However, what happens when the old version of the flow still has some active (or completed) flow instances? The OVBPI Business Impact Engine cannot simply delete them - the active flow instances are still running and still receiving data events. So, the OVBPI Business Impact Engine marks this old flow definition with a status of "Deleted" - which actually means "Pending Delete". What this means is that, as and when this flow definition no longer has any active or completed flow instances, it is then deleted from the OVBPI Business Impact Engine.

These "Deleted" flows are called **Superseded** Flows.

Suppose your flow name is: FlowA.

- At any one time there can be many versions of FlowA deployed to the OVBPI Business Impact Engine

- Only one of these (the most recently deployed) is marked as the "Active" flow definition. All the rest are superseded (marked as "Deleted")

- When an event comes in to start a new instance of FlowA, the OVBPI Business Impact Engine always starts the version marked as "Active" - the most recently deployed version

- All instances of superseded flow definitions run through to completion as normal

  As events come in they are routed to the correct flow instance and run according to that flow definition. Redeploying a flow definition does **not** affect running instances.

- Over time, as flow instances complete, these may be removed from the OVBPI Business Impact Engine by the "Instance Cleaner" (refer to the *OVBPI System Administration Guide* for details about the instance cleaner).

  Once all instances of a superseded flow definition have been removed, the actual flow definition is then removed.

Superseded Definitions

The idea of a flow becoming superseded when a newer version is deployed applies to some of the other definitions that you can define within the OVBPI Modeler.

Data Definition

When you redeploy a data definition, the previous deployed one is superseded.

Events

Events behave differently.

When you redeploy an event definition, the previous definition is removed from the system.

Event definition are not superseded, they are replaced.

Intervention Client

If you have superseded definitions (flows/data) and you would like to remove them from the OVBPI Business Impact Engine...you can.

You use the **OVBPI Intervention Client**.

The OVBPI Intervention Client is a set of JSP pages (and java classes) that allow you to view the deployed definitions and instances within the OVBPI Business Impact Engine...and update/delete them.

▶ You need to be **very careful** when using the OVBPI Intervention Client. If you use it incorrectly you can seriously mess up the OVBPI Business Impact Engine and OVBPI Modeler.

To run the OVBPI Intervention Client, you point a Web browser at the URL:

```
http://hostname:44080/ovbpiintclient
```

where:

- *hostname* is the hostname of your servlet engine. This is usually the same as the OVBPI server hostname
- *44080* is the default port of the servlet engine

By default these JSP pages are password protected. The default login details are:

```
User:      admin
Password:  ovbpi
```

(To change the admin user password, and to configure the underlying authorization mechanism, refer to the *OVBPI System Administration Guide*.)

The OVBPI Intervention Client is a useful utility for flow developers. It allows you to deploy your flows and data as often as you need to get them right, and then you can use the OVBPI Intervention Client to remove all the superseded definitions and thus clean up your OVBPI Business Impact Engine. Very useful!

▶ If you are wanting to clean up your development system using the OVBPI Intervention Client, then it is recommended that you use the OVBPI Intervention Client to remove the definitions and instances from the running OVBPI Business Impact Engine **and then re-deploy** the flow(s) that you wish to use afresh.

Multiple Unique IDs

When defining a data definition, you normally set up the definition to have a single unique ID attribute. This means that each data instance is identified by a single unique ID for the life of the data instance. For example, each order is known by its `OrderID`.

But how do you handle the situation where you are monitoring (for example) an order as it moves across disparate application systems? For example:

The order flows across both applications...
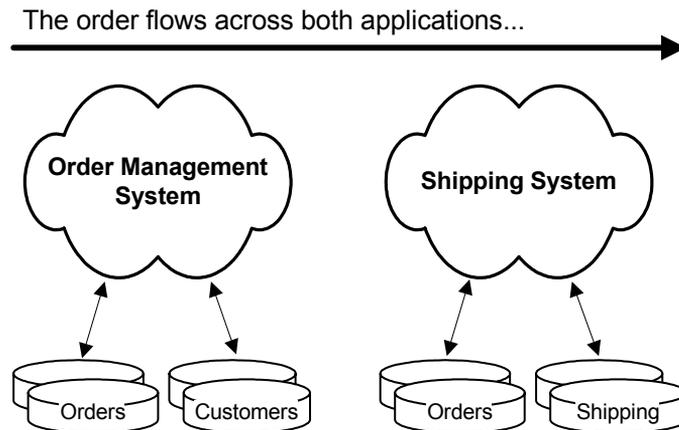
While the order moves around the Order Management System, it may be uniquely identified by an `OrderID` attribute. But when the order moves across to the Shipping System, the order may well be known by not just a different attribute name, but also a different unique ID value.

For example, an order within the Order Management System known by the key `OrderID = 123`, might become known within the Shipping System by the key `ShippingOrderNum = ABC-987`. While this order is in the Order Management System you receive events identified by the `OrderID` (123). When this order moves around the Shipping System, you receive events identified by the `ShippingOrderNum` (ABC-987). You need a way for the OVBPI Engine to know that `OrderID=123` and `ShippingOrderNum=ABC-987` are indeed the same data instance.

To be able to track this order as it moves across these two applications, you need to:

- Create your data definition to have two unique ID attributes

  Create two attributes, OrderID and ShippingOrderNum, and mark each one as being unique. This says that a particular data instance can be identified by either a unique OrderID value or a unique ShippingOrderNum value.

- Get the applications people to provide a mechanism for sending an event into the OVBPI Business Impact Engine that maps between these two key values

  When an order moves from the Order Management System across into the Shipping System, you need to receive an event containing both key attributes. This allows the OVBPI Business Impact Engine to identify that the data instance identified by OrderID=123, now has the ID ShippingOrderNum=ABC-987.

  Setting up this mechanism to send an event when an order moves across to the Shipping System may take some time and effort, but you need some way to send an event that contains both the OrderID value and its new ShippingOrderNum value.

So, for example, your event sequence might become:

```
New Order            (OrderID=123, ...)
Update Order         (OrderID=123, ...)
X-Application Send   (OrderID=123, ShippingOrderNum=ABC-987)
Update Shipping      (ShippingOrderNum=ABC-987, ...)
Shipping Complete    (ShippingOrderNum=ABC-987, ...)
```

where:

- The first two events (New Order and Update Order) are uniquely identified by the OrderID attribute, and the value is 123.

- When the order is sent from the Order Management System to the Shipping System, the event X-Application Send provides the mapping that OrderID 123 is now known as ShippingOrderNum ABC-987.

- The last two events provide information about the order as it moves through the Shipping System. Each event is identified by the ShippingOrderNum attribute, whose value is ABC-987.

Handling Out of Sequence Events

When working with events from across multiple application systems, the ordering of the events becomes crucial.

For example, with the event sequence:

```
New Order           (OrderID=123, ...)
Update Order        (OrderID=123, ...)
X-Application Send  (OrderID=123, ShippingOrderNum=ABC-987)
Update Shipping     (ShippingOrderNum=ABC-987, ...)
Shipping Complete   (ShippingOrderNum=ABC-987, ...)
```

It is the `X-Application Send` event that tells the OVBPI Business Impact Engine that, from now on, events for `ShippingOrderNum=ABC-987` are to be linked to the same data instance identified by `OrderID=123`.

However, what happens if the events arrive in this order:

```
New Order           (OrderID=123, ...)
Update Order        (OrderID=123, ...)
Update Shipping     (ShippingOrderNum=ABC-987, ...)
X-Application Send  (OrderID=123, ShippingOrderNum=ABC-987)
Shipping Complete   (ShippingOrderNum=ABC-987, ...)
```

When the `New Order` event arrives, a new data instance is created for `OrderID=123`. When the `Update Order` event arrives, this updates the same data instance where `OrderID=123`. When the `Update Shipping` event arrives the OVBPI Business Impact Engine tries to locate an existing data instance where `ShippingOrderNum=ABC-987`. It does not find one...so it creates a new data instance setting `ShippingOrderNum=ABC-987`. You now have two data instances - one with `OrderID=123` and another with `ShippingOrderNum=ABC-987`. Unfortunately, this is not what you wanted! Because the `X-Application Send` event had not made it through to the OVBPI Business Impact Engine in time, the OVBPI Business Impact Engine had no way to know that `ShippingOrderNum=ABC-987` should have been tied to the existing data instance where `OrderID=123`.

But you can solve this issue by correctly setting your event subscriptions. See

Fail If Data Instance Doesn't Exist

In the case where a flow is monitoring across more than one application system, and therefore the item being monitored is known by more than one unique ID, you need to configure your flow model not to automatically create new data instances all the time.

For the example where the events are:

```
New Order            (OrderID=123, ...)
Update Order         (OrderID=123, ...)
X-Application Send   (OrderID=123, ShippingOrderNum=ABC-987)
Update Shipping      (ShippingOrderNum=ABC-987, ...)
Shipping Complete    (ShippingOrderNum=ABC-987, ...)
```

Within the OVBPI Modeler, you set up the data definition subscriptions for each of these events.

For the New Order, Update Order and X-Application Send events, you can accept the default setting:

```
Create the data definition instance if it does not exist.
```

However, for the remaining Update Shipping and Shipping Complete events, you set the creation option to be:

```
Fail if the data definition instance does not exist.
```

The subscription page looks something like this:

Figure 12 Fail the Event Subscription

where:

1. The event subscription is for a specific instance of the data definition

2. You select that the event **fails** if the specific data instance does not yet exist

However, setting the option to Fail does **not** mean that the event fails and is then lost. When the event arrives at the OVBPI Business Impact Engine, if there is no current data instance, the event is "failed" and therefore not applied by the OVBPI Business Impact Engine. The OVBPI Business Impact Engine then marks this event to be re-submitted, and hands this event back to the OVBPI Event Handler. The event is periodically re-sent over a configurable time period. The idea is that during this retry time the missing event (that sets up the data instance) arrives.

So by marking an event subscription to "Fail" if the data instance does not yet exist, allows you to receive the events out of order but the OVBPI Business Impact Engine processes them in the correct order.

Event Subscription - MSSQL

When you create a data definition with multiple unique IDs, there is unfortunately a problem at run-time if your OVBPI Business Impact Engine database is using MSSQL. The problem is that you get a duplicate key violation error generated by the OVBPI Business Impact Engine when you try to create the second data instance.

This problem does **not** happen when using an Oracle database.

Suppose you have the following data definition:

```
OrderID           (Unique)
ShippingOrderNum (Unique)
OrderValue
ShipDate
```

where there are two unique key attributes, OrderID and ShippingOrderNum.

When the OVBPI Business Impact Engine receives the event:

```
New Order(OrderId=123, OrderValue=100.00)
```

a new data instance is created as follows:

```
OrderID           = 123
ShippingOrderNum = null
OrderValue        = 100.00
ShipDate          = null
```

Suppose another `New Order` event arrives, as follows:

```
New Order(OrderId=456, OrderValue=230.00)
```

The OVBPI Business Impact Engine wants to create a new data instance as follows:

```
OrderID          = 456
ShippingOrderNum = null
OrderValue       = 230.00
ShipDate         = null
```

However, this creates two instances within the database where the `ShippingOrderNum` attribute is set to `null`. Within Oracle this is fine, however, MSSQL does not allow two rows to have the same value (`null`) for an attribute that is marked as being unique. MSSQL throws a "duplicate key violation" error (logged in the OVBPI Business Impact Engine logs) and rejects the event.

So, if you are running OVBPI on an MSSQL database and you want to use multiple unique IDs within a data definition you need to make sure that when each data instance is first created, all unique ID attributes are assigned unique values, and not left to default to `null`.

Let's look at an example...

If you consider the example data definition:

```
OrderID           (Unique)
ShippingOrderNum  (Unique)
OrderValue
ShipDate
```

When the `New Order` event comes through and starts a new data instance, this assigns the `OrderID` value. You need to also assign the other unique attribute (`ShippingOrderNum`) to some temporary unique value. So why not assign it to the `OrderID` value. Your event subscription actions become:

```
this.OrderID = event.OrderID;
this.ShippingOrderNum = (this.ShippingOrderNum==null) ? event.OrderID
                                            : this.ShippingOrderNum;
```

where the assignment of `this.ShippingOrderNum` uses the single-line-if-statement format. This says:

If the `ShippingOrderNum` attribute is currently set to `null`, then assign the `event.OrderID` value to it. Otherwise, leave it with its current value.

This means that when the data instance is first created, both the `OrderID` and `ShippingOrderNum` attributes are set to the same value.

This cures the problem that your ID attributes are not left being `null`, however you then need to ensure your progression rules can cope with the fact that an ID being set doesn't necessarily mean that it is real. That is, when the above event subscription actions set both `OrderID` and `ShippingOrderNum` to the same value, this does not mean that the order has entered the Shipping System.

You need to adjust your **progression rules** so they handle the fact that it is only when the `ShippingOrderNum` does not equal the `OrderID` that you have a valid shipping order number.

For example:

To test when the `ShippingOrderNum` is assigned, you might have tried the progression rule:

```
this.ShippingOrderNum.after() != null
```

However, this is no longer correct, because you are setting the `ShippingOrderNum` the minute the `OrderID` is set.

So you would need to rewrite this progression rule to be something like this:

```
this.ShippingOrderNum.after() != OrderID
```

This is true the moment `ShippingOrderNum` is assigned a value that is different to `OrderID`.

Lab - The Flow Seems To Stop Working

This lab looks at a typical developers problem where you have a flow that
works, you make some minor changes, and suddenly your flow appears to
have stopped working.

The Flow

Let's create a simple flow:

- Create a flow called: `Simple Flow`
- Draw it out to look like this

Start   Middle   End
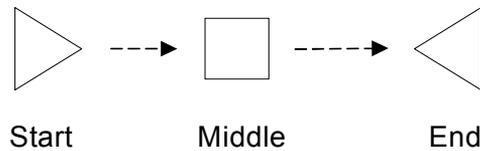
- Create a data definition called: `Simple/Data`
- Define the data properties as follows:

Name	Type	Constraints	Unique
Key_Field	String	10	Yes
Value_Field	Integer		

- Relate `Simple/Data` to `Simple Flow`
- Set the `Identity` to be `Key_Field`
- Set the `Weight` to be `Value_Field`

- Set the progression rules as follows:

  — `Start`

    ```
    Complete on first assignment:

    Property:  Key_Field
    ```

  — `Middle`

    ```
    Start and complete on transitions:

    Start Transition:
          Property:  Value_Field
          From:
          To:          15

    Complete Transition:
          Property:  Value_Field
          From:        15
          To:          66
    ```

  — `End`

    ```
    Complete on transition:

          Property:  Value_Field
          From:
          To:          66
    ```

These progression rules simply mean that when a key comes in, it moves into the `Middle` node when its value field is set to `15`. It then moves into the `End` node when the value field is set to `66`.

Nice and simple :-)

- Define an event called: `Simple/Event`

- Define the properties for this event to be both the `Key_Field` and `Value_Field` as defined in the `Simple/Data` definition

- Configure `Simple/Data` to subscribe to this `Simple/Event` as follows:

  — It is a specific subscription matching on `Key_Field`

  — The subscription assigns:

  ```
  this.Value_Field = event.Value_Field;
  this.Key_Field   = event.Key_Field;
  ```

- Check your ToDo lists to make sure that nothing is missing

- Deploy `Simple Flow`

  You should see it deploy:

  — `Simple/Event`

  — `Simple/Data`

  — `Simple Flow`

Testing The Flow

- Run the OVBPI Business Process Dashboard

  You should be able to see that your `Simple Flow` is deployed and currently has no active flow instances.

- Using the Generic Event Injector contributed utility, inject the following event:

  ```
  Event name:    Simple/Event

  Key_Field:     123
  Value_Field:   15
  ```

- Now in the OVBPI Business Process Dashboard you should hopefully be able to see this new flow instance and see that it is indeed sitting in the `Middle` node.

Wonderful! Your flow is working.

Altering The Flow Definition

Let's suppose that you are happy with your flow and how it all works, but you are about to give a demonstration to your manager and you would like to make some alterations to the flow's appearance.

So...let's go back into the OVBPI Modeler and make some changes...

- In the OVBPI Modeler, edit the `Simple Flow` definition and simply move the nodes about on the screen so that they are now positioned as follows:
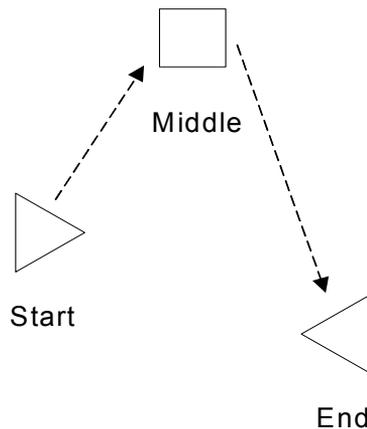
That's right. All you do is to alter the positions of the nodes on the page.

The important thing is that you are updating the flow definition.

- Save your changes

  Notice that this causes the icon next to `Simple/Flow` (in the Navigator pane) to change color.

- Now select `Simple/Flow` in the Navigator pane...and deploy the flow

  Notice that it only redeploys the flow definition, as that is the only thing that has changed.

Testing The New Flow

Ok, you have redeployed the flow. You know that it worked before so you are all ready to do your demonstration for your manager. Good luck!

Let's do the demonstration...

- Run the OVBPI Business Process Dashboard

  If you refresh the flows page, you should now see that `Simple/Flow` is listed in both the active flows table and the superseded flows table.

  And your new active version of `Simple/Flow` has no flow instances as yet.

- Using the Generic Event Injector, you inject the following event:

  ```
  Event name:     Simple/Event

  Key_Field:      123
  Value_Field:    15
  ```

- You then look in the OVBPI Business Process Dashboard and you don't see a new instance for `Simple Flow`!!!

At this point you look very embarrassed and you explain to your manager that "It worked this morning...honest!"

The Explanation

The problem is very simple.

When you first deployed your flow and tested it, you used a key value of `123`. This created a data instance in the OVBPI Business Impact Engine for the unique key of `123`. This was then connected to a flow instance of `Simple Flow`.

When you redeployed `Simple Flow`, you only redeployed the flow definition.

Now when you inject an event with `Key_Field` of `123`, the OVBPI Business Impact Engine sees that it is for the data instance whose unique field (`Key_Field`) is `123`. And guess what? You already have one of those.

Updates to this data instance `123` affect both the superseded and the newly deployed `Simple Flow`, however it depends on the progression rules for the new `Simple Flow` as to what behavior you see. In this case, sending in the event to set the `Value_Field` to `15` does not cause a new flow to be instantiated.

If instead you had injected an event with a new unique key value (for example: `Key_Field = 456`) you would have seen a new instance of your newly deployed `Simple Flow` and everything would have been fine!

There are a couple of things to realize here:

- Firstly, when injecting your own "made up" events you need to be **very** careful about your unique key fields

  If you are injecting events that require new unique values, make sure that they are unique, otherwise you may not see new flow instances as you expect.

- If you really do wish to ensure that you can reuse previously tested unique keys you have an option available to you:

  If you change the flow definition, make sure you also change the data definition (just changing the description should be enough).

  This causes the data definition to be redeployed as well as the flow definition. This means that events received instantiate new data instances and you are able to see new flow instances created as you expect.

However, be aware that these events with previously used unique key values also affect the superseded flow instances as well. So it is not necessarily a good thing to do on a production system.

You might want to consider using the **OVBPI Intervention Client** contributed utility to cleanup superseded flow and data definitions - and then redeploying the flow. Then when you give your demonstration everything works correctly because there is no "old data" left hanging around.

Well done! You have reached the end of the lab.

---

5 Advanced Progression Rules

When configuring your node progression rules you have various *styles* available to you. There are a collection of "typical" (or easy) styles and then you have the `Advanced conditions` style.

The set of typical/easy styles are fine for developing your flows and the use of them is certainly encouraged. However, depending on the complexity of your flow, you may need to configure advanced progression rules for some, or all, of your nodes.

You typically need to write advanced progression rules when your start (or complete) condition for a node relies on testing the values of more than one data property.

So let's take a look at how you write advanced progression rules.

The Language

When writing progression rules you are writing the evaluation(s) that you want the OVBPI Business Impact Engine to perform for your node each time a data event comes in that affects your flow's related data definition.

So, as you write each progression rule, you must always be asking yourself:

"When the event comes in, does this rule evaluate to true or false."

Your start condition, if it evaluates to true, means that the node is marked as started. Your complete condition, if it evaluates to true, means that the node is marked as completed.

The Grammar

The details of the actual grammar for the rules language is all specified in the *OVBPI Reference Guide*.

When writing progression rules you are writing the "test condition" part of an "if" statement. The elements that you can test are the state of the data definition instance itself and/or the state of the properties of this related data instance.

To write these tests there are a set of available methods that you can use. The trick is knowing what each method does and how to use it...

Data Definition Level Methods

You can test the state of the related data instance.

The important methods available to you are:

- **.created()**

  This evaluates to boolean true when the data instance has just been created.

  This might typically be used as the entry condition for a start node.

  For example:

  Have a start node whose entry condition says:

  ```
  this.data.created()
  ```

  This starts the node when an event comes in that creates an instance of the related data definition.

- **.terminated()**

  This evaluates to boolean true when the data instance has just been terminated.

  This can occur if the event subscription has been configured to terminate the data instance on receipt of the event. (There is a check box on the event subscription dialog.)

Data Property Level Methods

You have various methods available to test the values of the properties of the data instance.

The important methods available are:

- **.changed()**

  This method evaluates to boolean `true` if the value of the data property has changed with this event.

  For example:

  ```
  this.data.Call_Status.changed()
  ```


- **.before()**

  This method is for when a data event has come in and changed the value of the data property. By using `.before()` you can test the old value of the property.

  For example:

  ```
  this.data.Call_Status.before() == "Assigned"
  ```

  Note that `.before()` only evaluates if the data property actually changed on receipt of this event. If this event did not change `Call_Status`, then this expression evaluates to false. Indeed, if the event did not change the property then any expression containing the use of `.before()` on this property evaluates to false...no matter what you are comparing it to.

  So:

  ```
  this.data.Call_Status.before() == "Assigned"
  ```

  is the same as:

  ```
    this.data.Call_Status.changed()
  &&
    this.data.Call_Status.before() == "Assigned"
  ```

- **.after()**

  This method is for when a data event has come in and changed the value of the data property. By using `.after()` you can test the new value of the property.

  Note that `.after()` only evaluates if the data property actually changed on receipt of this event. If the event did not change the property then any expression containing the use of `.after()` on this property evaluates to false.

  So:

  ```
  this.data.Call_Priority.after() > 3
  ```

  is the same as:

  ```
    this.data.Call_Priority.changed()
  &&
    this.data.Call_Priority.after() > 3
  ```

- **.contains()**

  This method allows you to test whether the data property contains a specified string.

  This only works on string properties.

  For example:

  ```
  this.data.property1.after().contains("AbC")
  ```

  This evaluates to true if `property1` has changed and the after value (the new value) is a string that contains the characters `AbC` (case sensitive).

  ```
  this.data.property1.contains("AbC")
  ```

  This evaluates to true if `property1` is a string that contains the characters `AbC`.

- **.starts()**

  This method allows you to test whether the data property starts with a specified string.

  This only works on string properties.

- **.ends()**

  This method allows you to test whether the data property ends with a specified string.

  This only works on string properties.

- **.in()**

  This method allows you to test whether the data property is set to any one of a list of values.

  This method works on string/non-string properties.

  For example:

  ```
  this.data.property1.after().in(1,2,3)
  ```

  This evaluates to true if the numeric property `property1` has changed value and is now set to either numeric `1` or `2` or `3`.


  ```
  this.data.property2.after().in("E103","E203")
  ```

  This evaluates to true if the string property `property2` has changed value and is now set to any of the strings `E103` or `E203`.

Comments Within Progression Rules

The logic within advanced progression rules can sometimes get quite complex. To help you document what the progression rule is doing, you can place comments within the progression rule.

There are two types of comments:

1. /*   */

   These allow you to mark the beginning and end of a comment. These comments can extend over multiple lines.

2. //

   This allows you to mark the start of a comment. This comment ends when a new line is encountered.

   If you place one of these comments at the end of a progression rule, there must be a carriage-return after the end of the comment. If you forget to do this, the OVBPI Modeler shows a pop-up dialog to remind you.

For example, you could create a progression rule as follows:

```
// This is my progression rule
/*
 * This tests whether a claim status starts with
 * the letter D
 */
   this.data.Claim_State.starts("D")  // A comment
&&
   (   this.data.Claim_Letter_Status.after() == "Drafted"
// another comment
    &&
        this.data.Claim_Letter_Type.after() == "Dispute"
/* Some
   more
 comments */
   )
// A comment on the last line
```

where:

- The actual progression rule code is shown here in bold type-face.

- The comments are there purely for your own documentation - they do not affect the execution of the progression rule.

- The comment on the last line is followed by a carriage return.

Single Start/Complete Condition

Sometimes you want a node to simply signal that something has happened.

For example, you might have a node called: "Purchase Order Received", which simply completes the moment you receive the purchase order from the customer.

You could set the node progression rules as follows:

Start Condition:

```
this.data.POReceived.after() == true
```

Complete Condition:

```
this.data.POReceived.after() == true
```

This means that the node both starts and completes the moment the POReceived property becomes true.

To save you having to specify the same rule for both the start and the completion conditions, if a node has only an entry condition, then when that condition is satisfied the node is said to start **and** complete at the same time. So too, if a node only has a complete condition, then when that condition is satisfied the node is said to both start and complete at the same time.

Nodes Re-Starting/Completing Too Often

The way node-start and node-complete progression rules work is easy to understand:

A node is started each time its start condition evaluates to true.

A node is completed each time its complete condition evaluates to true.

Sometimes you deploy a flow and observe that one or more of your nodes seem to be started (or completed) more than once. Now this might be perfectly expected behavior, however, this can be confusing if you expected the node to start (or complete) only once.

Clearly, if a node shows as starting (or completing) more than once then it's start (or complete) condition was satisfied more than once. If you only expected the node to start (or complete) the once then you have a problem with the logic in your progression rules.

A common error that can cause this behavior is where you have forgotten to use the `.after()` or `.before()` methods of a data property. Let's consider an example:

You have a node where the start condition is when the property `MyProp` transitions to be not null. However, you write the progression rule as follows:

```
this.data.MyProp != null
```

As each event comes in for this data instance, whilst `MyProp` is still `null` the node does not start.

The first time a non-null value is assigned to `MyProp`, the above progression rule evaluates to `true` and the node starts. So far so good!

However, for **every** event thereafter that affects this data instance, the above progression rule evaluates to `true`! That is, this rule is evaluated and the Engine says that `MyProp` is indeed still not equal to `null`.

So the progression rule evaluates to true the first time `MyProp` transitions to `non-null`, and for every event thereafter - assuming that `MyProp` stays set to a non-null value.

The correct progression rule is:

```
this.data.MyProp.after() != null
```

This only evaluates to true on events where `MyProp` has actually been changed and the new value is not null.

So be careful. Remember that **all** node progression rules are evaluated for each event that affects the flow's related data instance.

Flow Splitting

Designing a flow that "splits" does not require the use of advanced progression rules. You can use them, but you can also split a flow based on a single data property.

For example:

this.data.MyProp.after().contains("E")

this.data.MyProp.after().in("V12", "F55")

this.data.MyProp.after().starts("W")

where the flow instance splits out to the appropriate node based on the value of the `MyProp` data property.

(The above progression rules can all be created using the simple progression rule styles.)

Flow Merging

Designing a flow that merges two, or more, paths together typically requires the use of advanced progression rules. It depends on how you want the flow to behave.

No-Wait Merging

This is the situation where you have two (or more) paths that converge into one, however you do not need to wait until all incoming paths have completed before continuing on.

For example:
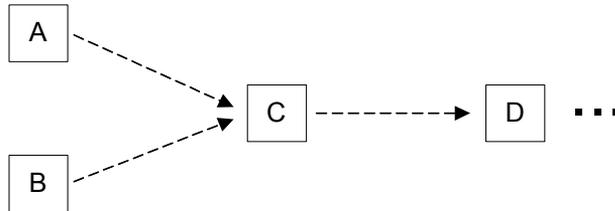
where, the nodes might complete in this order:

```
Node B
Node C
Node D
Node A
. . .
```

In such a scenario, your rules do not need to force any ordering, and it's quite possible that you could set this up using only the simplified progression rule styles.

Wait Merging

This is perhaps the more interesting style of flow merge, where you want your progression rules to ensure that all the incoming paths have completed before the flow instance is able to continue on.

For example:
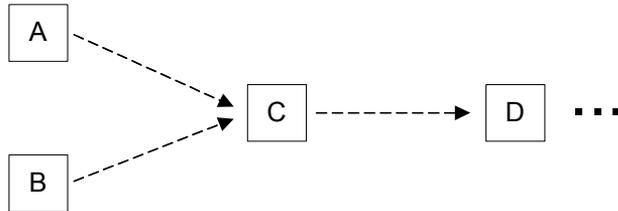
where:

- You wish to ensure that both `Node A` and `Node B` have both completed before the flow instance is able to progress into `Node C`

This style of progression more than likely requires you to write advanced progression rules. Let's consider some examples...

2-Node Merge Example

Suppose your flow looks as follows:

sets **ValidPONumber** to be true
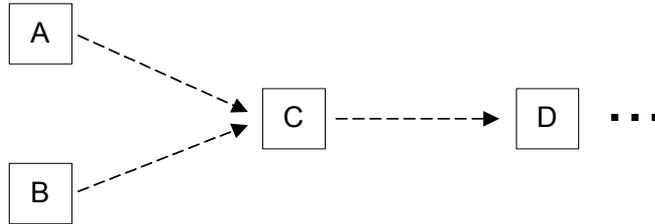
sets **VendorID** to be non-null

where:

- Node A **completes when** ValidPONumber **is set to true**
- Node B **completes when** VendorID **becomes a non-null value**

You want Node C to start only when **both** ValidPONumber has transitioned to true and VendorID has transitioned to be non-null.

Most people would start by writing the following progression rule to start Node C:

```
    this.data.ValidPONumber.after() == true
&&
    this.data.VendorID.after() != null
```

and then wonder why Node C never starts :-(

Indeed, the above progression might work...but only if the one event caused both the ValidPONumber and the VendorID to transition.

Remember that the above progression rule is actually saying:

```
    (   this.data.ValidPONumber.changed()
     && this.data.ValidPONumber.after() == true
    )
&&
    (   this.data.VendorID.changed()
     && this.data.VendorID.after() != null
    )
```

So this only evaluates to true if both properties have been changed by the one event.

Instead, you need to write a progression rule that caters for the situation where the event has either updated the `VendorID` or the `ValidPONumber`. The progression rule should be written as follows:

```
(    this.data.ValidPONumber.changed()
 || this.data.VendorID.changed()
)
&&
(    this.data.ValidPONumber == true
 && this.data.VendorID      != null
)
```

where this tests:

- That at least one (or both) of the attributes `ValidPONumber` and `VendorID` have just been changed by the event

- And the attributes have the set of values that you are testing for:

```
    this.data.ValidPONumber == true
 && this.data.VendorID      != null
```

- This progression rule also works in the situation where both `VendorID` and `ValidPONumber` are set within the one event

3-Node Merge Example

Suppose you have the following flow definition:

sets **ValidPONumber** to be true

sets **VendorID** to be non-null

sets **ValidClaim** to be true
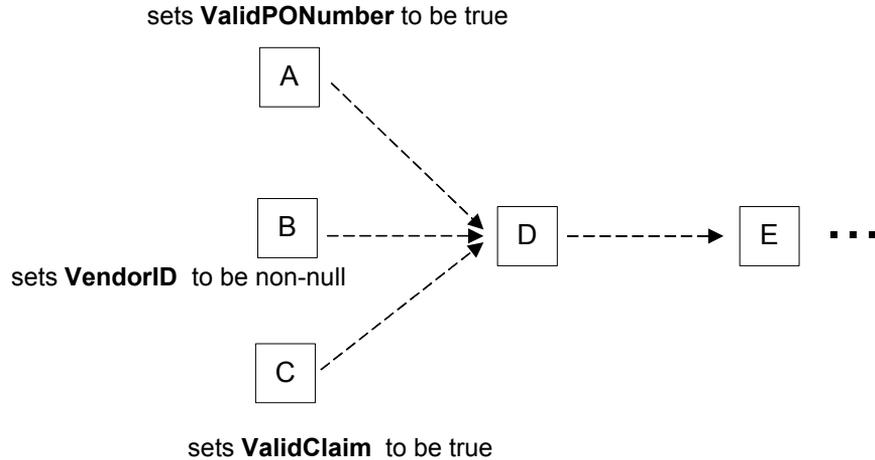
You need to write the start progression rule for `Node D` where you only enter `Node D` once Nodes `A`, `B` and `C` have all completed.

The progression rule is as follows:

```
(    this.data.ValidPONumber.changed()
  || this.data.VendorID.changed()
  || this.data.ValidClaim.changed()
)
&&
  (    this.data.VendorID       != null
   && this.data.ValidPONumber == true
   && this.data.ValidClaim     == true
  )
```

where you test that one or more of the attributes has been changed by this event, and if so, that the values for the three attributes are all set to the values you require.

6 Creating A Business Flow

This chapter takes you through a slightly more real life/complex scenario where you define a business flow for an Insurance firm.

As you work through this chapter there are labs that get you defining the `Insurance Claim` flow. In between the labs there are explanations taking your through the possible discussions/decisions that might be made when building a real-life flow.

Initial Investigations

Let's consider the kind of initial investigation work that goes into defining your flow.

Initial Flow Definition

The first task is always to gain an understanding of the underlying business process and then to sketch out the business flow that you want to monitor.

This initial flow should be a high-level abstraction of the underlying business process. Keeping it simple.

So the initial proposal for your Insurance flow is as follows:

Figure 13 Stage 1 Insurance Flow

Claim Received   **Initial Review**   **Settle Claim**
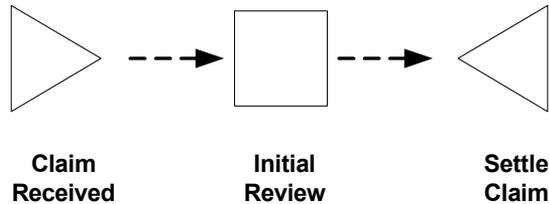
where:

- **Claim Received**

  When a claim is received, the details are entered into the claims handling system. The claim is not officially recognized until the claims handling system has assigned a claims number. A claims number is not assigned until the input data has passed a number of data validation tests, for example, validating that the customer has a policy that is in date.

- **Initial Review**

  When the claims system has acknowledged receipt of the claim, it is assigned to a claims handler, an individual who oversees the processing of the claim. Initial Review is a review by the claims handler to evaluate the nature of the claim.

- **Settle Claim**

  When a claim has been accepted, a settlement letter is drafted and the payment details are transmitted to the settlement system. A record of the letter and the claim details are archived.

Initial Data Definition

You need to start defining the kind of data that your flow is to use/maintain.

You should start this by identifying how it is that you know when you are in a particular node. That is, you need to understand what activities start each node and what activities signal that a node is complete.

These might include:

- When a claim number is assigned to a claim, the claim is officially recognized as started and the `Claim Received` node is started

- When a claim has been assigned to a claims handler, the `Initial Review` node is started

- When the claim has been accepted, the `Settle Claim` node is started

- When a customer letter is drafted, the `Settle Claim` node is completed

In addition to needing data to allow you to know when nodes start and complete, you also need to define the data that you would like to maintain for monitoring purposes. Ask yourself the question:

"What statistics do I want to get from this flow?"

You might consider the following:

- How many claims are processed

- How long it takes to accept a claim

- The value of claims accepted

More Detail Required?

At this stage you could go ahead with the current simple flow definition, however, in understanding the business process you start to realize that settling a claim is not the only outcome of the Initial Review node.

You then decide that you do wish to monitor these additional outcomes and be able to answer questions such as:

- How many claims are rejected
- How long is it taking to resolve a disputed claim
- How long is it taking to obtain a medical or technical report
- When does the number of outstanding settlements exceeds a predefined threshold

You decide that you need to expand the flow definition to include more detail...

Stage Two Investigations

To monitor these additional outcomes of the `Initial Review` node, you need to introduce more nodes in the flow.

You decide that the flow now look as follows.

Figure 14 Stage 2 Insurance Flow
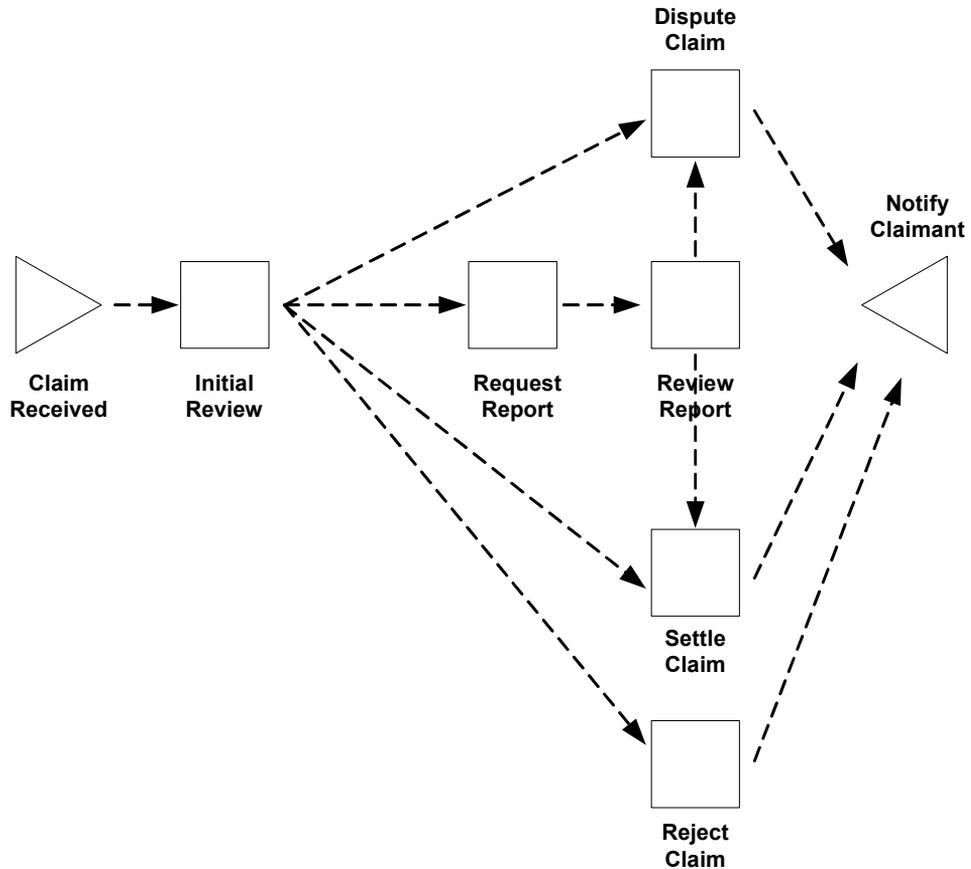
where:

- **Claim Received**

  A flow instance is initiated when the OVBPI Business Impact Engine receives an event from the claims handling system signaling that a claim number/identifier has been assigned.

- **Initial Review**

  When the claims handler has been assigned by the claims handling system, the claim moves to the `Initial Review` node, so the start condition is the assignment of a claim handler identifier to the claim record.

  The review can complete with four possible outcomes:

  — the claim is rejected; this may be due to a number of technical reasons.

  — the insurance company disputes the claim; and further information is required.

  — the claim requires further investigation by a specialist (e.g. a doctor or mechanic).

  — the claim can be settled.

  Any of these four outcomes are considered a legitimate conclusion to this step and any one of them could be the completion condition for the `Initial Review` node.

- **Reject Claim**

  The start condition for this node is that the insurance company has rejected the claim following an initial review. Once a claim has been rejected, a rejection letter is drafted and a record of the letter and the claim details are archived. The completion condition is when the draft letter is registered with the settlement system.

- **Dispute Claim**

  The start condition for this node is that the insurance company is disputing the claim. Once a claim is in dispute, a dispute letter is drafted and a record of the letter and the claim details are archived. The completion condition is when the draft letter is registered with the settlement system.

- **Settle Claim**

    The start condition for this node is that the insurance company is settling the claim. Once a claim has been accepted, a settlement letter is drafted and the payment details are transmitted to the settlement system. A record of the letter and the claim details including the settlement are archived. The completion condition is when the draft letter is registered with the settlement system.

- **Request Report**

    The start condition for this node is that the insurance company is requesting a report for the claim. Once a report has been requested, a requisition letter is drafted and transmitted (faxed) to the appropriate specialist. The completion condition is when the draft letter is registered with the settlement system.

- **Review Report**

    The start condition for this node is when the claims handler receives the report back from the specialist. Note that the flow does not include the precise details of how the report is received or when reminders need to be sent. The acquisition of reports could be a flow in its own right; here you are only interested in monitoring the high level flow.

    The start condition is when the claim handler receives the report. The possible outcomes of the review are that the claim is either disputed or settled. The completion condition is either when a settlement letter is drafted and sent to the settlement system, or when a dispute letter is drafted.

- **Notify Claimant**

    This step enables you to measure how long it takes for customers (claimants) to be notified of the results of a claim. This node starts when the drafted claims letter is registered, and completes when the claims letter is actually sent (passed to the paper mail system).

    The completion of this step results in the completion of the flow.

Now you have identified the basic start/completion conditions for each node, you need to assess what other data is required in order to answer the question:

"What statistics/monitoring does the customer want to get from this flow?"

Answers to questions such as:

- What are the most common paths though the flow
- Where are the hold ups in the flow
- What is the value of claims outstanding
- Details of the backlog of claims that are currently outstanding

The data definition side of things is considered in more detail later in this chapter, but for the moment let's go ahead and create the flow definition using the OVBPI Modeler.

Lab - Defining the Basic Flow Diagram

In this lab you defines the flow diagram.

The Flow Diagram

- Run the OVBPI Modeler

- Create a new flow, called: `Insurance Claim`

- Draw the flow so that it matches the flow diagram shown in Figure 14 on page 207

  This is the Stage 2 Insurance flow diagram - the one that includes nodes for all the different outcomes such as `Dispute Claim`, `Reject Claim`, etc.

A Sequenced Arc

You want this flow to send an alert if the `Review Report` node is started before the `Request Report` node (because this should never happen!), so:

- Set the arc that connects `Request Report` to `Review Report` and mark it such that its sequence is checked

---

Well done! You have reached the end of the lab.

Data Requirements

Developing the flow and associated data is an iterative process and at some point you need to check that the flow can be implemented, and that the data on which you plan to report can be obtained from the underlying application systems. You can check this later with the person who is integrating the business flow into the operation infrastructure.

If you are unsure about what data to include in the flow, think about what your measures are. Keep asking yourself the questions:

"What statistics/monitoring does the customer want to get from this flow?"

"What statistics/monitoring do I want to get from this flow?"

Is it the value of an order, the cumulative number of orders, or is it queue sizes to indicate orders building up? The business metrics that you want to measure for impact dictate the data that you need in your business flow.

Be aware that if you define too much data, the performance and manageability of the flow are impacted, so you need to make sure that you include only the data that you need; you do not need to include all the data that is available. Too much data also creates the integrator problems when trying to obtain the data from the lower-level application systems; it also makes the flow too heavyweight and affects performance. Keep in mind that you are monitoring and not trying to control the business flow.

Data Normalization

The data you need to define within the OVBPI Modeler actually comes from underlying application systems through data events. That is, the data is already out there in the various application system. You do not need to redefine all of that data within the OVBPI Modeler - just the data that you would like to monitor.

You may only need to bring in key values (such as `ClaimID`, `ClaimantID`) because when you come to write statistical reports, you can use these IDs to access the underlying databases directly to retrieve the necessary additional information. That is, there is no need to hold the full personal details of each claimant within the insurance flow - just the `ClaimantID` is enough.

Where this can get tricky is when your underlying business process actually moves across disparate application systems. For example:

The business operates across all of these...
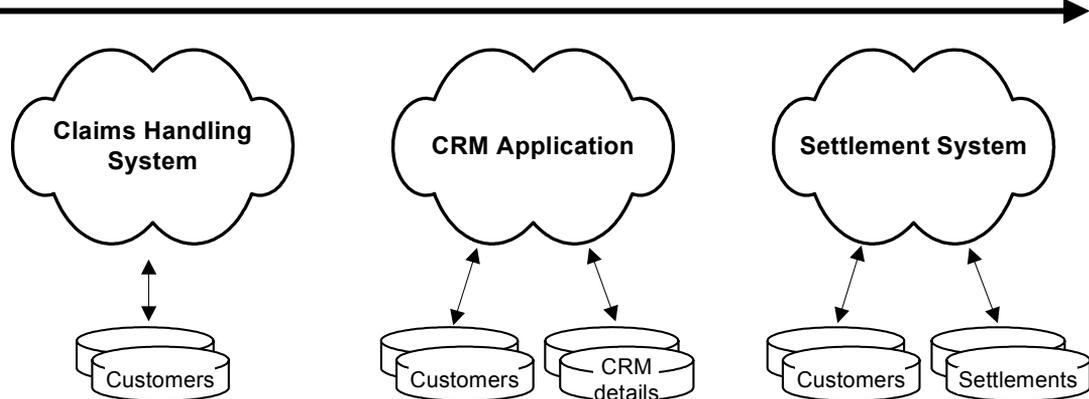
All these applications, potentially, have a different view of the Customer, Claim and other data, depending on the function of the application within the business. And typically the unique ID (for example, the `ClaimID`) is not the same between each system.

To be able to model a data definition that can track a claim as it passes across these different application systems you require one or more of the following:

- You could have your applications people set up a common "overall" id for your claim that is valid across all applications. This is unlikely to happen...but could be possible

- The applications people could provide a mechanism for sending an event into the OVBPI Business Impact Engine that maps between the various key values

  That is, the application systems generate events something like this:

  — When a new claim is raised in the Claims Handling System, the ClaimID is sent in as (for example) the value `123`

— When ClaimID 123 moves across to the CRM Application, an event is generated saying that ClaimID `123` is now mapped to CRM-ClaimID `ABC-45H`

— When CRM-ClaimID `ABC-45H` moves across to the Settlement System, an event is generated saying that CRM-ClaimID `ABC-45H` maps to SS-ClaimID `20040319-Bent45`

• The applications people could set up a system (database, application, etc.) that keeps track of all the different IDs across the three systems and allows cross reference lookups

This allows events to contain all the key values as they are available. That is, each event looks up in this cross-reference data table and adds on the current known values of all three keys.

When you define the data definition within the OVBPI Modeler you define all three IDs as being unique. That is, the claim can be uniquely identified by any one of these keys.

For situations where the data is across disparate systems, data normalization is required as you need some way to know that the same claim is known by different IDs within each of the systems. Whilst this can be time consuming to set up, at least it is only required for the actual key ID fields. There is no need to normalize all your data across all your applications - just the key ID fields such that a claim can be tracked. (This topic is also covered in

For the lab work in this chapter, you can assume that the ClaimID is unique across all underlying applications.

Let's now consider the data properties that you might maintain for your `Insurance Claim` flow:

Data Used To Progress The Flow

The first set of data that you need to define is the data used to progress the flow, that is, the data that provides the start and completion conditions for each node. You did this when you were gathering information to draw out the flow diagram.

Additional Data Properties

In addition to the data required to progress the flow, you define the data that you need, and ensure that the required business impact and alert information is available, for example, amount($) of the claim, etc..

Here are the properties for the claims data definition to be used with the `Insurance Claim` flow.

Claim ID

The claim ID is communicated in an event from the claims handling system. The event is produced when a recently entered claim has passed the data validation tests.

Claim Handler ID

When a claims handler is assigned to a claim, the claims system generates an event to give you this claims handler ID.

Claim State

The claims state attribute is required by the business flow to be one of Initiated, Rejected, Disputed, Settled, or Report Requested. These states are not directly represented in the claims handling system. The claims handling system does maintain a state, but it is much more detailed in order to represent the various conditions of each state.

The state field in the underlying claims handling system can contain the following values:

- `R101` - rejected for form errors
- `R102` - rejected for lapsed policy
- `R103` - rejected for unknown identity

- `D104` - disputed, not covered by policy
- `D105` - disputed, claim is covered elsewhere
- `D106` - disputed, Terms and Conditions not met.
- `S107` - settled, payment to follow
- `S108` - settled, payment withheld
- `S109` - settled, no payment due
- `E110` - medical report requested
- `E111` - mechanical report requested
- `E112` - loss adjusters report required
- `null` - initial state

But the states required for the business flow can be mapped from these more detailed claims handling system states.

Claim Value

Quite simply, the dollar ($) value of the actual claim.

Claim Letter Status

The claim letter status attribute is required to show the status of the claim letter that is managed by the insurance company.

The possible value for the claim letter status attribute are as follows:

- `Drafted` - Letter drafted, waiting to be sent
- `Sent` - Letter sent to client
- `Archived` - Copy of letter archived after being sent to client

Claim Letter Type

The claim letter type attribute is required to show the types of letter sent by the insurance company.

The possible values for claim letter type are as follows:

- `Settlement` - Letter to client states that the claim is being settled
- `Dispute` - Letter to client states that the claim is being disputed
- `Rejection` - Letter to client states that the claim has been rejected

Claim Report Status

The claim report status attribute is required to show the status of the reports being requested by the insurance company as part of dealing with the claims.

The possible values for report status are as follows:

- `Requested` - Report has been requested
- `Received` - Report has been received back from medical, mechanical or loss adjustor
- `Queried` - Questions have been sent back to medical, mechanical or loss adjustor about the original report
- `Closed` - Report accepted and closed

Claim Report Type

The claim report type attribute is required to show the types of report being requested by the insurance company as part of dealing with the claims.

In the example, the report types are not used as the flow does not analyze that level of detail. However, it is likely that the flow may be expanded in the future to account for the report types, so they have been included for completeness.

The possible values for report type are as follows:

- `Medical` - Report required from medical team
- `Mechanical` - Report required from mechanical team
- `Loss Adjustor` - Report required from loss adjustor

Lab - Defining The Data/Progression Rules

In this lab you define the data for your `Insurance Claim` flow, relate the data to the flow, and then configure the progression rules for each node.

Data Definition

From within the OVBPI Modeler:

- Create a new data definition, called: `Claims/Data`

- Now define the data properties as follows:

Name	Type	Constraints	Unique
Claim_ID	Integer		Yes
Claim_Handler_ID	Integer		
Claim_State	String	30	
Claim_Value	Currency	USD	
Claim_Letter_Status	String	15	
Claim_Letter_Type	String	15	
Claim_Report_Status	String	15	
Claim_Report_Type	String	15	

Relate The Data To The Flow

- Go ahead and relate this `Claims/Data` definition to the `Insurance Claim` flow

- Select `Claim_ID` as the Identity

- Select `Claim_Value` as the Weight

Progression Rules

Configure the following progression rules:

Claim Received

Start when:

Claim_ID is assigned a value (in other words, when Claim_ID goes from null to something).

Complete when:

Claim_Handler_ID is assigned a value.

Initial Review

Start when:

Claim_Handler_ID is assigned a value.

Complete when:

Claim_State is assigned a value.

Request Report

Start when:

Claim_Report_Status is assigned the value Requested.

Complete when:

Claim_Report_Status goes from Requested to any other value.

Dispute Claim

Start when:

Claim_State changes to a value that starts with the character D

Enter this using the advanced style of progression. The rule looks as follows:

```
this.data.Claim_State.after().starts("D")
```

Complete when:

The value of `Claim_State` currently starts with the character `D` and the `Claim_Letter_Status` becomes `Drafted` and the `Claim_Letter_Type` becomes `Dispute`.

To cope with the `Claim_Letter_Status` and `Claim_Letter_Type` properties being set within different events you would code this as follows:

```
this.data.Claim_State.starts("D")
&&
   (
      (   this.data.Claim_Letter_Status.changed()
       ||
          this.data.Claim_Letter_Type.changed()
      )
    &&
      (   this.data.Claim_Letter_Status == "Drafted"
       &&
          this.data.Claim_Letter_Type   == "Dispute"
      )
   )
```

If you are sure that these two properties are both be updated within the one event then you can simplify this rule to become:

```
this.data.Claim_State.starts("D")
&&
   (   this.data.Claim_Letter_Status.after() == "Drafted"
    &&
       this.data.Claim_Letter_Type.after() == "Dispute"
   )
```

For this lab assume that these properties are updated by the **one** event.

If you did code up the rule that coped with these properties possibly being set by different events, it would also work if they were updated by just a single event - it copes with either situation.

Review Report

Start when:

`Claim_Report_Status` changes to the value `Received`

Complete when:

The value of `Claim_State` changes from an `E` coding to either a `D` or an `S` coding.

Your rule is:

```
 this.data.Claim_State.before().starts("E")
&&
  (
     this.data.Claim_State.after().starts("D")
   ||
     this.data.Claim_State.after().starts("S")
  )
```

Settle Claim

Start when:

`Claim_State` changes to a value that starts with the character `S`

Enter this using the advanced style of progression. The rule looks as follows:

```
     this.data.Claim_State.after().starts("S")
```

Complete when:

The value of `Claim_State` currently starts with the character `S` and the `Claim_Letter_Status` becomes `Drafted` and the `Claim_Letter_Type` becomes `Settlement`.

Assuming that the `Claim_Letter_Status` and `Claim_Letter_Type` properties are both updated within the one event, your rule is:

```
this.data.Claim_State.starts("S")
&&
(   this.data.Claim_Letter_Status.after() == "Drafted"
 &&
     this.data.Claim_Letter_Type.after() == "Settlement"
)
```

Reject Claim

Start when:

Claim_State changes to a value that starts with the character R

Enter this using the advanced style of progression. The rule looks as follows:

```
this.data.Claim_State.after().starts("R")
```

Complete when:

The value of Claim_State currently starts with the character R and the Claim_Letter_Status becomes Drafted and the Claim_Letter_Type becomes Rejection.

Assuming that the Claim_Letter_Status and Claim_Letter_Type properties are both updated within the one event then your rule is:

```
this.data.Claim_State.starts("R")
&&
(    this.data.Claim_Letter_Status.after() == "Drafted"
 &&
     this.data.Claim_Letter_Type.after() == "Rejection"
)
```

Notify Claimant

Start when:

Claim_Letter_Status changes to the value Drafted

Complete when:

Claim_Letter_Status changes to the value Sent

Well done! You have reached the end of the lab.

Data Events

You have identified the data that your flow needs in order to progress. The next step is to identify the events that carry this data.

Full details of how to build adapters and how to set up the Business Events Handler are described in the *OVBPI Integration Training Guide - Business Events*.

But for now, you need to define the events that you expect to receive from the underlying application systems.

Lab - Defining Events/Subscriptions

In this lab you define the events that will drive your `Insurance Claim` flow and configure your data definition to subscribe to these events.

Event Definitions

From within the OVBPI Modeler, go ahead and define the following data events:

- `Claims/New`

  Properties:

  ```
  Claim_ID
  Claim_Value
  ```

- `Claims/HandlerAssigned`

  Properties:

  ```
  Claim_ID
  Claim_Handler_ID
  ```

- `Claims/Reviewed`

  Properties:

  ```
  Claim_ID
  Claim_State
  Claim_Report_Status
  ```

- Claims/ReportStatus

  Properties

  ```
  Claim_ID
  Claim_Report_Status
  Claim_Report_Type
  Claim_State
  ```

- Claims/LetterStatus

  Properties

  ```
  Claim_ID
  Claim_Letter_Status
  Claim_Letter_Type
  ```

Event Subscriptions

Go ahead and subscribe your `Claims/Data` definition to these events as follows:

- `Claims/New` subscription

  — It is a specific subscription matching on `Claim_ID`

  — The subscription assigns:

  ```
  this.Claim_ID = event.Claim_ID;
  this.Claim_Value = event.Claim_Value;
  ```

- `Claims/HandlerAssigned` subscription

  — It is a specific subscription matching on `Claim_ID`

  — The subscription assigns:

  ```
  this.Claim_ID = event.Claim_ID;
  this.Claim_Handler_ID = event.Claim_Handler_ID;
  ```

- `Claims/Reviewed` subscription

  — It is a specific subscription matching on `Claim_ID`

  — The subscription assigns:

  ```
  this.Claim_ID = event.Claim_ID;
  this.Claim_State = event.Claim_State;
  this.Claim_Report_Status = event.Claim_Report_Status;
  ```

- `Claims/ReportStatus` subscription

  — It is a specific subscription matching on `Claim_ID`

  — The subscription assigns:

  ```
  this.Claim_ID = event.Claim_ID;
  this.Claim_Report_Status = event.Claim_Report_Status;
  this.Claim_Report_Type = event.Claim_Report_Type;
  this.Claim_State = event.Claim_State;
  ```

- `Claims/LetterStatus` subscription

  — It is a specific subscription matching on `Claim_ID`

  — The subscription assigns:

  ```
  this.Claim_ID = event.Claim_ID;
  this.Claim_Letter_Status = event.Claim_Letter_Status;
  this.Claim_Letter_Type = event.Claim_Letter_Type;
  ```

Well done! You have reached the end of the lab.

Services

As you define your business flow and data, you need to identify the services that you utilize and what information you want from them. A single node might represent a whole phase of a process and can therefore depend on many services. For example, the `Settle Claim` node is dependent on the `Claims Handling System` and the `Settlement System`.

The following diagram shows the links between the underlying systems and the nodes in the flow:

Figure 15 Insurance Flow With IT Services
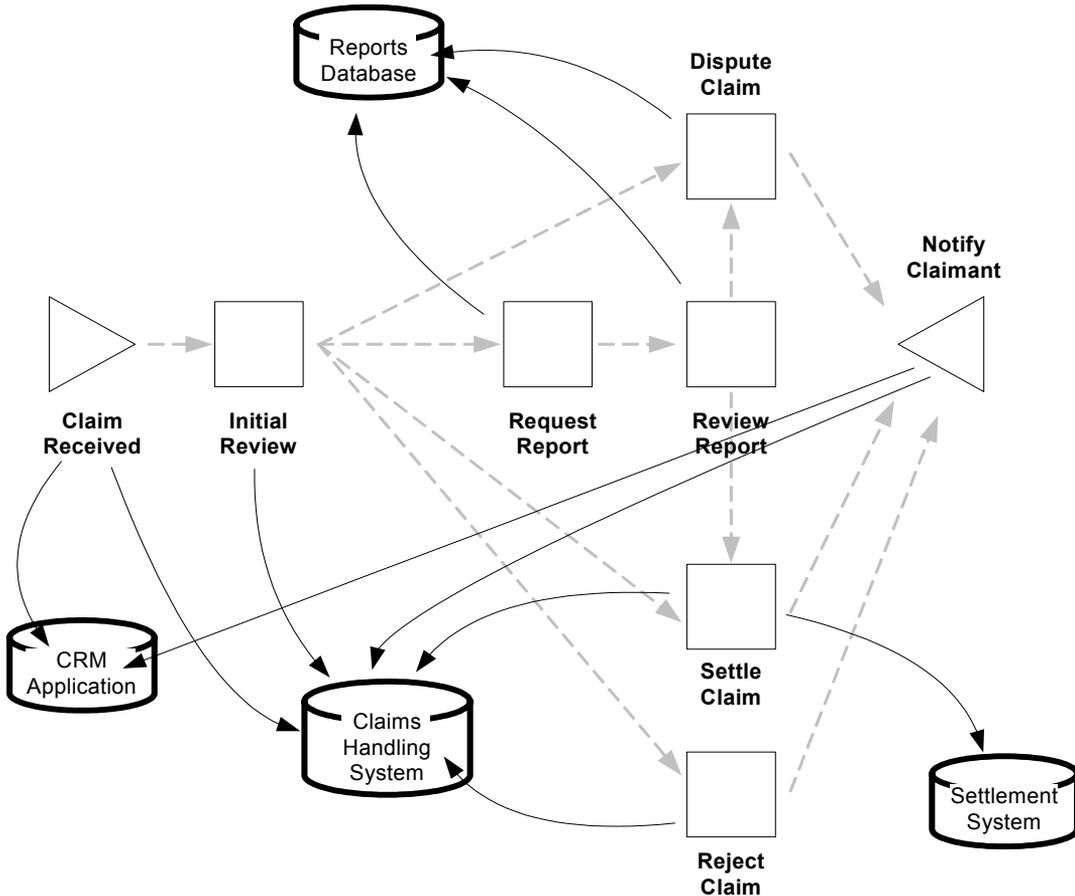
Note that this is not how the flow is shown through the OVBPI Modeler, and not all the service links are shown on Figure 15 on page 226.

Lab - Defining IT Services Dependencies

Creating The Services

The `Insurance Claim` flow is dependent upon the following four IT services:

- CRM Application
- Claims Handling System
- Reports Database
- Settlement System

You need to first create these services within your service management/ monitoring system - whether this is OVO, OVIS, or whatever.

- For this lab, go ahead and define these four services within the OVSN Simulator

Linking/Synchronizing The Services

- Within the OVBPI Modeler, link/synchronize to the above four IT services

Adding The Service Dependencies

- Within the OVBPI Modeler, edit the `Insurance Claim` flow, go through each node and assign the following service dependencies:

```
Node Name                    Service
---------                    -------
Claim Received               CRM Application
                             Claims Handling System

Initial Review               Claims Handling System

Request Report               Reports Database
```

```
Dispute Claim              Reports Database

Review Report              Reports Database

Settle Claim               Claims Handling System
                           Settlement System

Reject Claim               Claims Handling System

Notify Claimant            CRM Application
                           Claims Handling System
```

- Save the flow

Well done! You have reached the end of the lab.

Lab - Deploy/Test the Insurance Claim Flow

Now that you have defined your `Insurance Claim` flow, let's test that it works as you expect:

- Go ahead and deploy the `Insurance Claim` flow

- Using the OVBPI Business Process Dashboard you should now be able to see your deployed flow definition

- Using the Generic Event Injector (contributed utility) inject the following events and watch the progression in the OVBPI Business Process Dashboard:

   — **Claims/New**

   ```
   Claim_ID:      10
   Claim_Value:   22000
   ```

   (Make sure you enter them in the right order!!!)

   A new flow instance should appear in the OVBPI Dashboard

   — **Claims/HandlerAssigned**

   ```
   Claim_ID:          10
   Claim_Handler_ID:  999
   ```

   (Make sure you enter them in the right order!!!)

   The flow instance should now move to the `Initial Review` node.

   — **Claims/Reviewed**

   ```
   Claim_ID:             10
   Claim_State:          S107
   Claim_Report_Status:  <leave empty>
   ```

   This should progress the flow to the `Settle Claim` node.

   — **Claims/LetterStatus**

   ```
   Claim_ID:             10
   Claim_Letter_Status:  Drafted
   Claim_Letter_Type:    Settlement
   ```

   This should progress the flow to the `Notify Claimant` node.

— **Claims/LetterStatus**

```
Claim_ID:             10
Claim_Letter_Status:  Sent
Claim_Letter_Type:    Settlement
```

This should complete the flow.

- Let's try sending another claim through the system. This time going through the Request Report node...

    — **Claims/New**

    ```
    Claim_ID:     11
    Claim_Value:  23000
    ```

    — **Claims/HandlerAssigned**

    ```
    Claim_ID:          11
    Claim_Handler_ID:  888
    ```

    — **Claims/Reviewed**

    ```
    Claim_ID:             11
    Claim_State:          E110
    Claim_Report_Status:  Requested
    ```

    — **Claims/ReportStatus**

    ```
    Claim_ID:             11
    Claim_State:          E110
    Claim_Report_Status:  Received
    Claim_Report_Type:    Medical
    ```

    — **Claims/ReportStatus**

    ```
    Claim_ID:             11
    Claim_State:          D104
    Claim_Report_Status:  Closed
    Claim_Report_Type:    Medical
    ```

    — **Claims/LetterStatus**

    ```
    Claim_ID:             11
    Claim_Letter_Status:  Drafted
    Claim_Letter_Type:    Dispute
    ```

— **Claims/LetterStatus**

```
Claim_ID:            11
Claim_Letter_Status: Sent
Claim_Letter_Type:   Dispute
```

• Try taking down some of the services and see the impact reflected within the OVBPI Business Process Dashboard

---

Well done! You have reached the end of the lab.