Case Study: Redeveloping an Oracle Forms application using Oracle JDeveloper and Oracle ADF

An Oracle White Paper
August 2007

Case Study: Redeveloping an Oracle Forms application using Oracle JDeveloper and Oracle ADF

Redeveloping an Oracle Forms application using Oracle JDeveloper and Oracle ADF

INTRODUCTION

Oracle retains a range of different tools for developing enterprise business applications. For nearly two decades, Oracle Forms has been key to application developers building on the Oracle platform. However, with the emergence of the Enterprise Java platform (Java EE), both Oracle's traditional developer base and those new to the Oracle platform, have another choice.

The Oracle tools statement of direction is available at http:/./otn.oracle.com/products/forms

While Oracle remains committed to Oracle Forms, and many customers run their business on Oracle Forms applications, there are those who seek to exploit the features of the Java platform. For those coming from a Forms or PL/SQL background this paper presents a study of a redevelopment of a Forms application on the Java platform using Oracle JDeveloper and Oracle Application Development Framework (ADF). The paper seeks not to necessarily promote redevelopment, but to use the exercise of redevelopment to show how the concepts used in a Forms application can be mapped to the Java platform using Oracle ADF.

Technology Assumptions

Details of JDeveloper and Oracle ADF, including a glossary of terms, can be found at http:/./otn.oracle.com/formsdesignerj2ee

While this paper assumes knowledge of Oracle Forms, the focus is on JDeveloper and Oracle ADF. The paper assumes a familiarity with JDeveloper and Oracle ADF but does not assume more than a high level understanding of terms and concepts.

Introducing the Forms Application

The Oracle Forms Summit Application can be downloaded from http://www.oracle.com/technology/products/forms/files/summit.zip

The Summit Forms demo application will be used as the application for the redevelopment case study. As a "typical" Forms application, it allows the browsing and editing of customers' orders in a fictional sporting goods shop. This involves the navigation and update of database data, calling stored procedures, displaying summary and calculated fields, images and popup dialogs.

Figure 1 – The Summit Application

REDEVELOPING SUMMIT APPLICATION

The goal of redeveloping this application is to show the mapping of concepts, which are at the heart of Oracle Forms, can be mapped to the Java EE world using Oracle ADF. Thus, the resulting application should try to mimic as closely as possible the functionality of the original application.

Architecture Decisions

In redeveloping the Summit application using JDeveloper and Oracle ADF, two main choices have to be made at the outset relating to:

- the look and feel of the application
- the business model implementation

User Interface Implementation

For a discussion on UI technology choices see http://www.oracle.com/technology/pub/articles/nimphius-mills-swing-jsf.html

There are essentially two choices for the application user interface (UI): a Java Swing UI or and Java Server Faces (JSF) UI. Both have their strengths but with the goal to redevelop the application so that it closely mimics the look of the original Forms application, a Swing UI is the more obvious choice.

Business Services Implementation

Oracle ADF offers a number of technology choices for building the business services element of your application. The choice can be influenced by a number of factors, but given the Forms and PL/SQL bias of the source application, and the assumption that the redevelopment will be done by developers from a Forms background, ADF Business Components (ADF BC) gives the closest mapping of Forms to Java EE business services.

Building the business model

For further details on building a master/detail Swing application go to: http://www.oracle.com/technology/obe/obe1013jdev/10131/adf%20swing/master_detail_page_adfswing_bc.htm .

Just like Forms, JDeveloper and ADF Business Components provide powerful wizards, dialogs and properties for declaratively developing business services based on database tables. JDeveloper also provides a number of facilities to aid development not available in Oracle Forms.

Visualizing the data

Given that much of the functionality of the application is based around database tables, the first task is to use JDeveloper to create a database diagram of the schema.

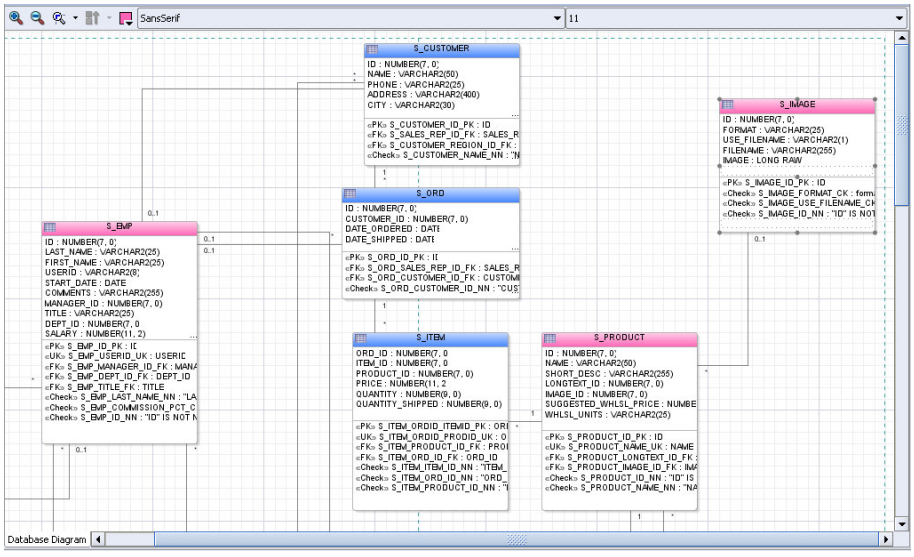

Figure 2 – The Summit Application Schema

This can be achieved by simply dragging database tables onto an empty database diagram in JDeveloper. This helps visualize the database tables and their

relationships as shown in figure 2. By studying the database diagram and the original forms, it can be concluded that the bulk of the data manipulation is around the S_Customer, S_Ord and S_Items tables (colored blue). S_Emp, S_Product, and S_Image are used as lookups (colored pink).

Building Entity Objects

You may not need entity objects if you are only viewing read only data, but as a "first cut", creating a entity for each table is a reasonable assumption.

The next task is to create a first cut data model based on the *S_Customer*, *S_Ord* and *S_Items* tables. An entity object is created for each of the above three tables, selecting all the available attributes. This results in three entity objects *SCustomer*, *SOrd* and *SItems*.

It is also worth creating the entity objects for the tables that will be used for lookups: *S_Emp*, *S_Product*, and *S_Image*.

Figure 3 shows the entity objects and their associations.

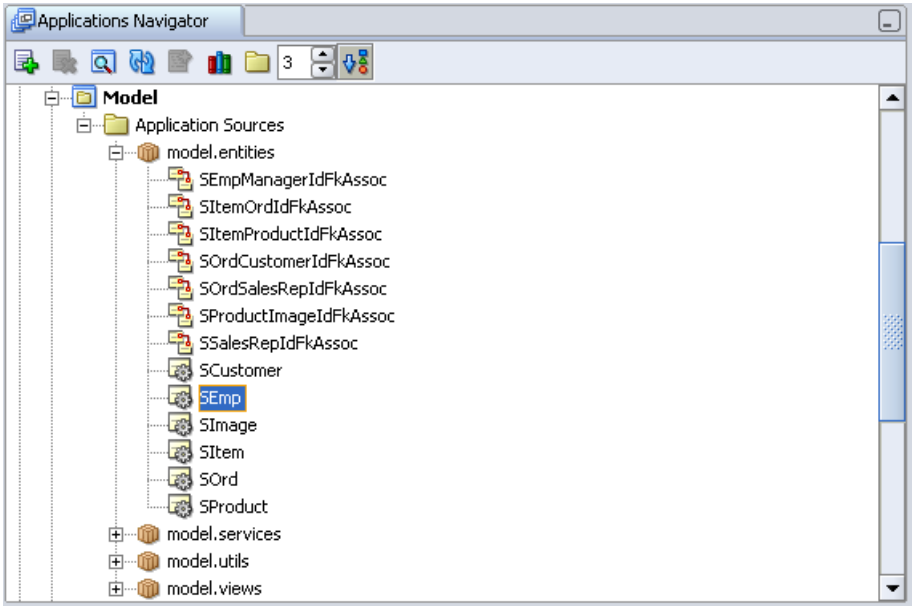

Figure 3 – Entity Objects for the Summit tables

Building View Objects

The entity objects give the physical representation of a row from the underlying database table, but you need a particular view of that data for the Summit application. Using the Forms object navigator as shown in figure 4, you can see the items in the upper most frame on the canvas, relate to items in the *S_ORD* block.

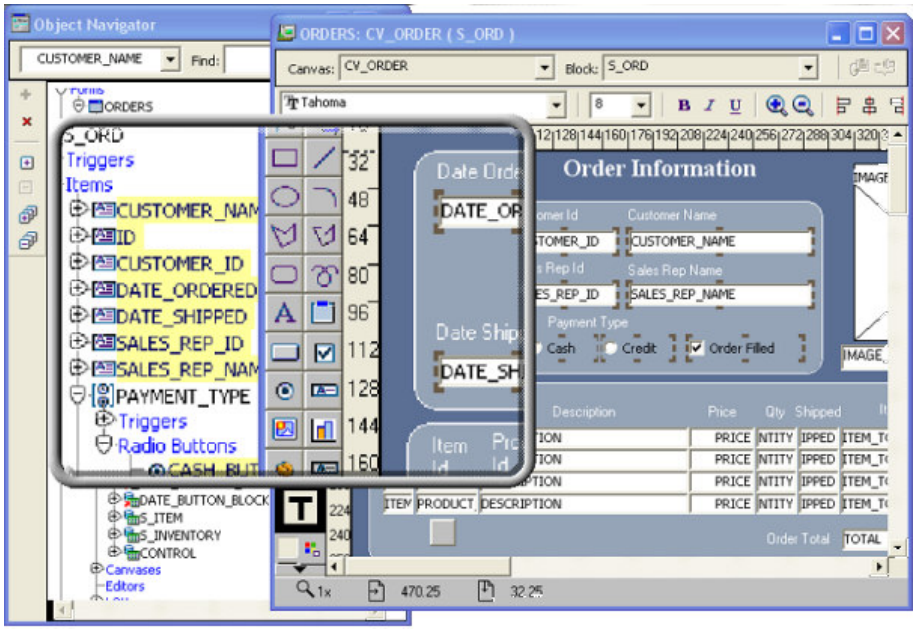

Figure 4 – Forms blocks and items

Some of those fields (e.g. *DATE_ORDERED*, *CUSTOMER_ID*, *DATE_SHIPPED*) are database fields based on columns in the *S_Ord* table. Some fields (e.g. *CUSTOMER_NAME* and *SALES_REP_NAME*) have values that are derived from other tables via a lookup.

Creating the corresponding view object is simple in JDeveloper. The view object wizard allows the specification of the underlying entity objects and the attributes of those entities that make up the view.

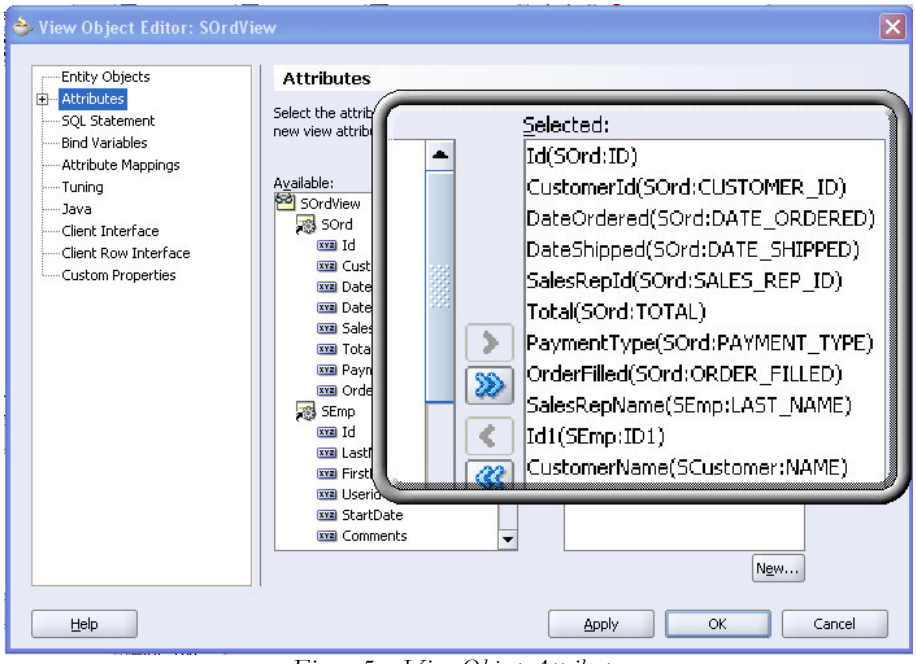

Figure 5 – View Object Attributes

Figure 5 shows attributes such as *DateShipped* and *PaymentType* coming from the *SOrd* entity object, and also *CustomerName* and *SalesRepName* coming from the *SCustomer* and *SEmp* entity objects. Oracle ADF provides the functionality that automatically looks up these values based on the primary/foreign key relationship, without any additional coding.

Post-Query Lookups

In the example above, the view object is using a primary key to fetch lookup information from another table. This is a common feature of a Forms application that is facilitated through the use of the *Post-Query* trigger to populate non-base table fields (e.g. the *CUSTOMER_NAME* or *SALES_REP_NAME*). In the original Forms application, the developer would have had to code a *Post-Query* trigger to populate the lookup information as shown in figure 6.

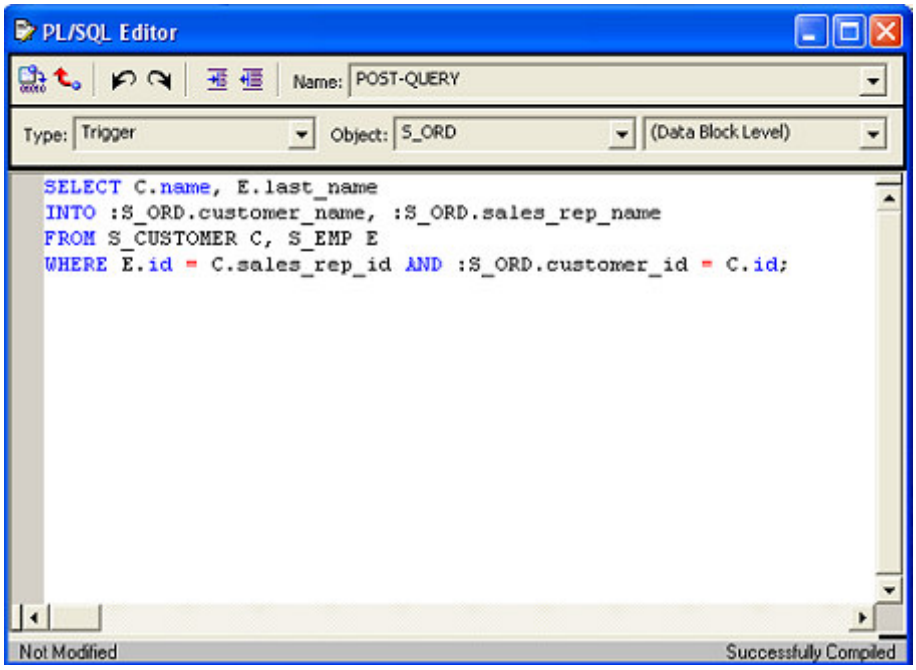

Figure 6 – Oracle Forms Post-Query trigger

However, as shown in Figure 5, using JDeveloper and Oracle ADF, this code is not required. The developer simply specifies the entities and attributes required, and the correct SQL statement is created automatically. Thus, in Oracle Forms where you wrote a *Post-Query* trigger to implement a look up, you can use the declarative features of Oracle ADF.

Figure 7 shows how this functionality is automatically implemented by means of the SQL statement in the view object.

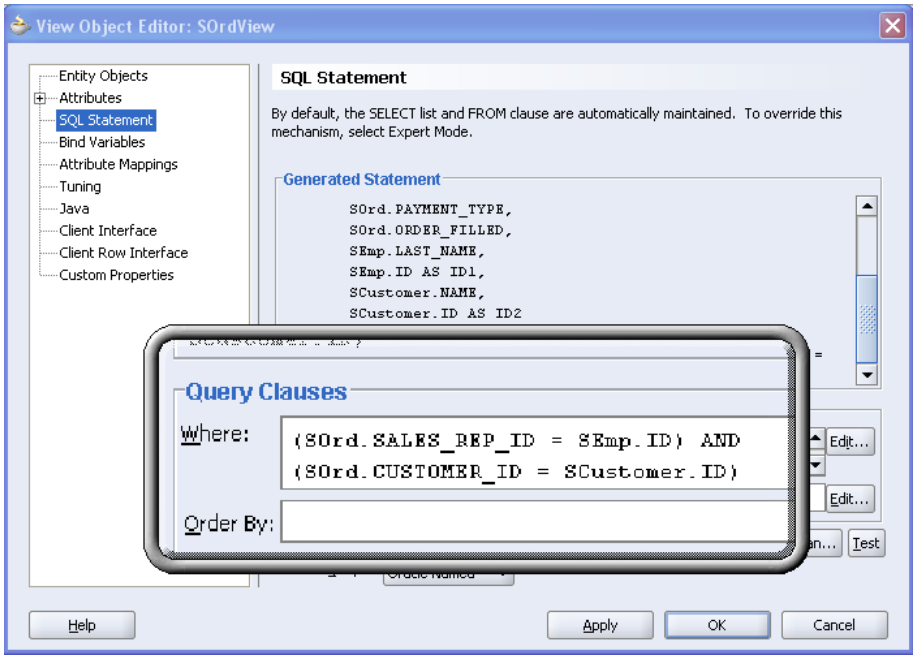

Figure 7 – View object SQL implements the lookup

Refining the business model

After building a first cut of the data model to manage the manipulation of data, you can start refining the business model.

Adding a calculated attribute

In the Summit order form, the line total for an item (*QUANTITY_SHIPPED* multiplied by *ITEM_PRICE*) is calculated in the *ITEM_TOTAL* attribute of the *S_ITEM* block.

In ADF Business Components, the same functionality can be achieved by adding a transient attribute to the *SItemView* view object. This attribute is defined from a SQL query that multiplies the *Price* and *QuantityShipped* attributes.

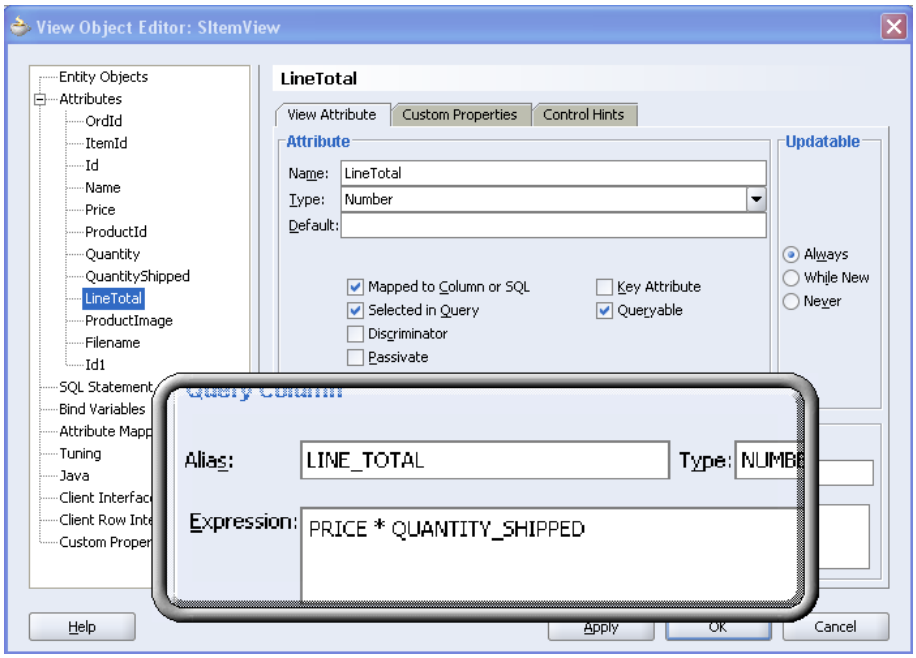

Figure 8 – A transient attribute implements the line total

This means, that when the *SitemView* view object is populated, the *LineTotal* attribute will show the calculated value. However, this attribute will not be updated if the value of either *Price* or *QuantityShipped* is changed. For this to take effect the *LineTotal* attribute has to be updated. To do so, code is added to the *setPrice* and *setQuantityShipped* methods to null the *LineTotal* attribute forcing it to be recalculated.

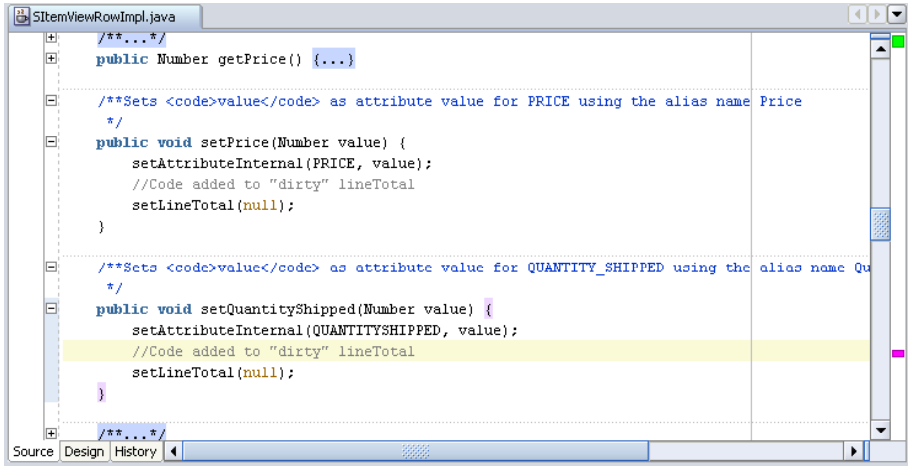

Figure 9 – Updating the line total

Displaying an image from file

In the Summit application, the image of the currently selected item is read from the file system. A database procedure is passed the *PRODUCT_ID* and returns a filename for display. A call is then made to READ_IMAGE_FILE to read the derived filename from disk. This code is executed in a *When-New-Record-Instance* trigger at the block level on the *S_ITEM* block as shown in figure 10.

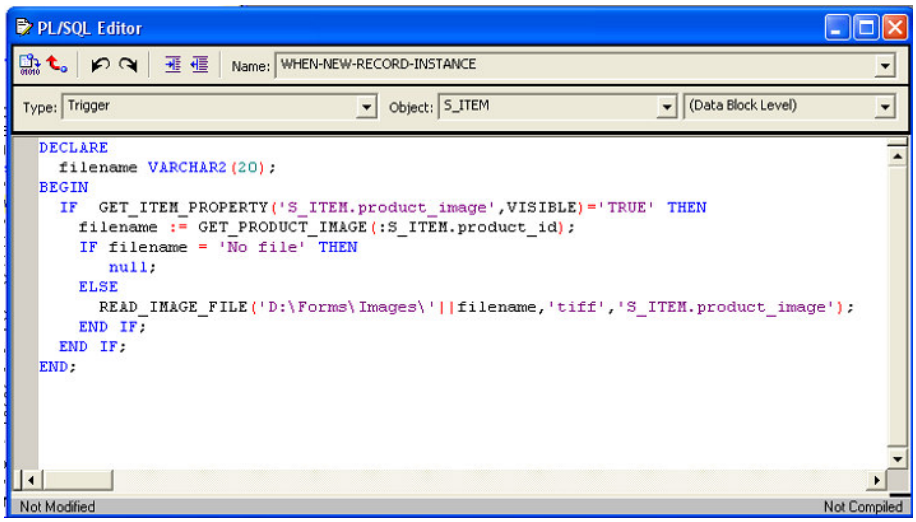

Figure 10 – Reading and image from file

Using ADF Business Components, the same functionality is facilitated though a transient attribute named *ProductImage* of type BlobDomain. ADF Business Components automatically exposes a method *getProductImage* that the framework calls when the *ProductImage* attribute is populated. To read the filename and the image from a disk, see figure 11, code is added to this method to call the stored procedure, and then uses the result of the procedure call to create a filename, which is then read from the file system.

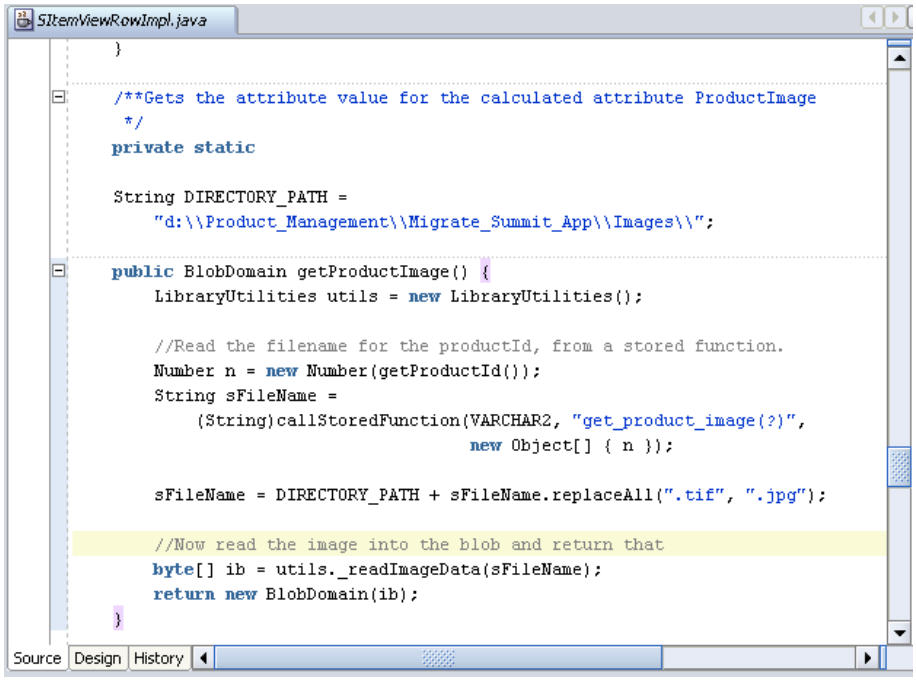

Figure 11 – Adding custom code to getProductImage

Adding validation rules

After building the basic data model for the application, the next step is to look at data validation.

Ensuring shipped date is after ordered date

The Summit application uses a *When-Validate-Record* trigger to ensure the date the order is shipped is after the order date. Figure 12 shows how ADF Business Components provides a simple declarative mechanism that includes a method validator at the entity object level. Thus, when the *SOrd* entity is validated, *validateOrderDates* will be called to confirm the shipped date is after the order date.

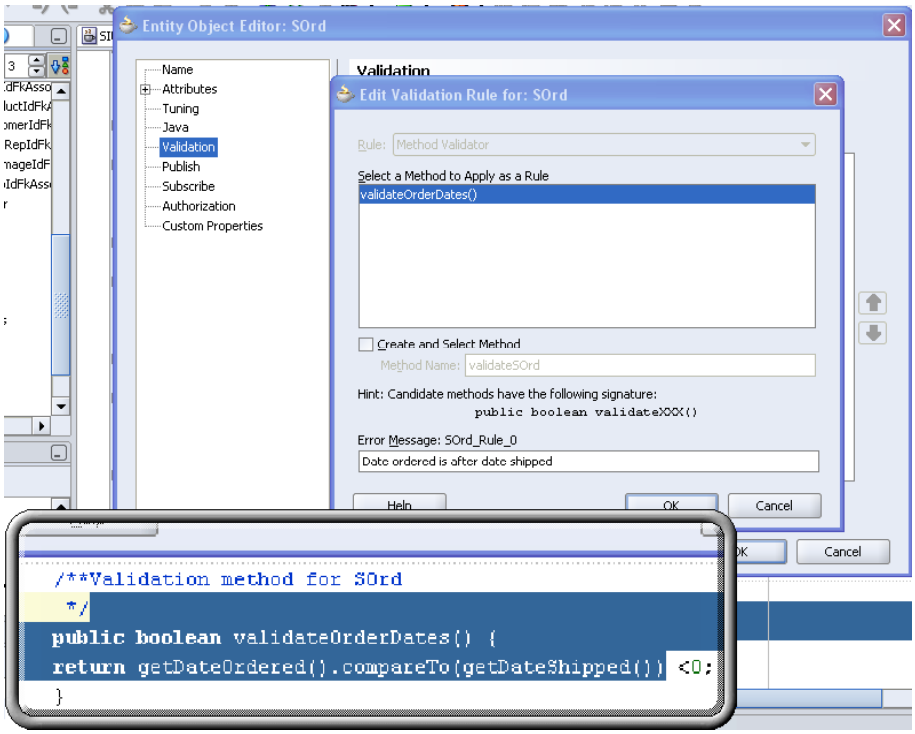

Figure 12 – Validating the order date

Changing Product ID

In the Summit application, when displaying the list of order items, the user can change the *PRODUCT_ID* of any order item in the list. In doing so, information for that line of data must be updated. So, changing *PRODUCT_ID* will change the *DESCRIPTION*, the *PRICE* of the item (which is now set to *SUGGESTED_WLSLS_PRICE* for the new product id) and of course the line total will change as well. There is also a check that the value entered into *PRODUCT_ID* exists in the *S_Products* table.

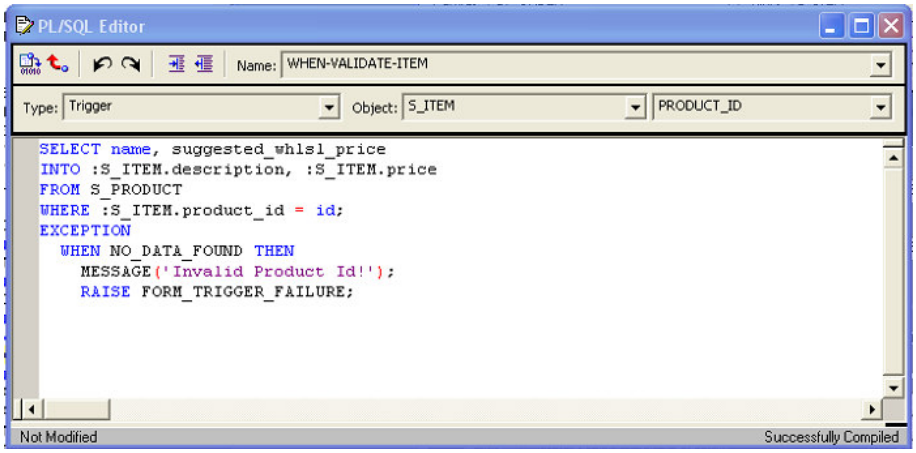

Figure 13 – Ensuring the product id is valid

In Forms, this is achieved by a *When-Validate-Item* trigger on *Product_Id*, which will select the product description and wholesale price from the *S_Product* table, as shown in figure 13. If no record is returned, an error is raised indicating an invalid *Product_Id*.

ADF Business Components provides two simple hooks to provide the same functionality with minimal coding.

List Validation

ADF Business Components provides a list validator on an attribute that allows an attribute to be validated against a static, or SQL generated, list. Figure 14 shows how *ProductId* can be validated against a list of data from a database table.

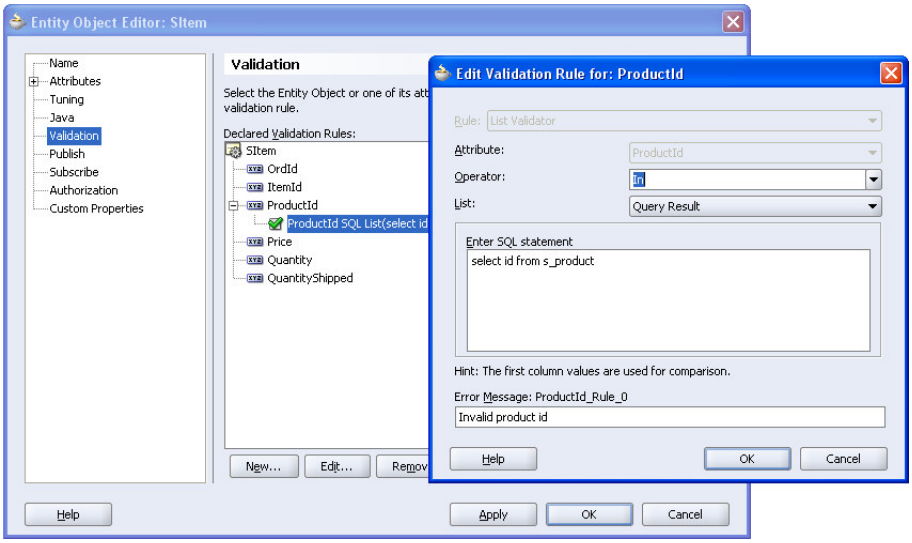

Figure 14 – Validating against a list of product ids

Augmenting Set Methods

When *ProductId* is changed, the price must be updated to the wholesale price for the new *ProductId*. This involves performing a lookup using the product id to find the wholesale price. Adding *SuggestedWlslPrice* as an attribute of *SItemView* view object easily facilitates the look up. ADF Business Component will automatically coordinate the *SuggestedWlslPrice* with *ProductId*.

ADF Business Components also creates get and set methods for manipulating each of the attributes in a view object. So, to update *Price* with the *SuggestedWlslPrice* when the *ProductId* is changed a call is made to *setPrice* in the *setProductId* method as shown in figure 15.

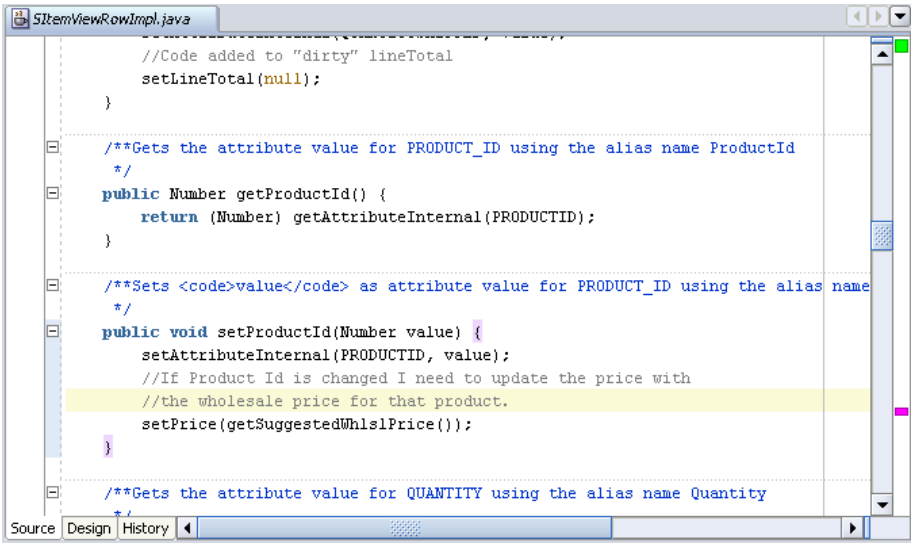

Figure 15 – Changing the price for a new product id.

Building the User Interface

Having built and tested the business model, the next step is to create the UI.

Layout Manager

Essentially a Forms application uses absolute positioning of UI components. This, initially, at least, makes the layout of components simple in that they stay exactly where they are positioned. The downside is that you physically have to line up the items and that on window resize, they do not automatically reposition to fill the new screen area.

Swing, on the other hand, provides a range of powerful layout managers. While powerful, they can seem unintuitive to those new to Swing. However, the JGoodies FormLayout manager integrated with JDeveloper gives a powerful layout manager based on the concept of a grid in which components can be lined up and resizing information allocated to rows or components.

And in the same way that a Forms application may typically use a number of canvases within the same Form, for a Swing application you may typically use a number of different panels to aid layout and manageability.

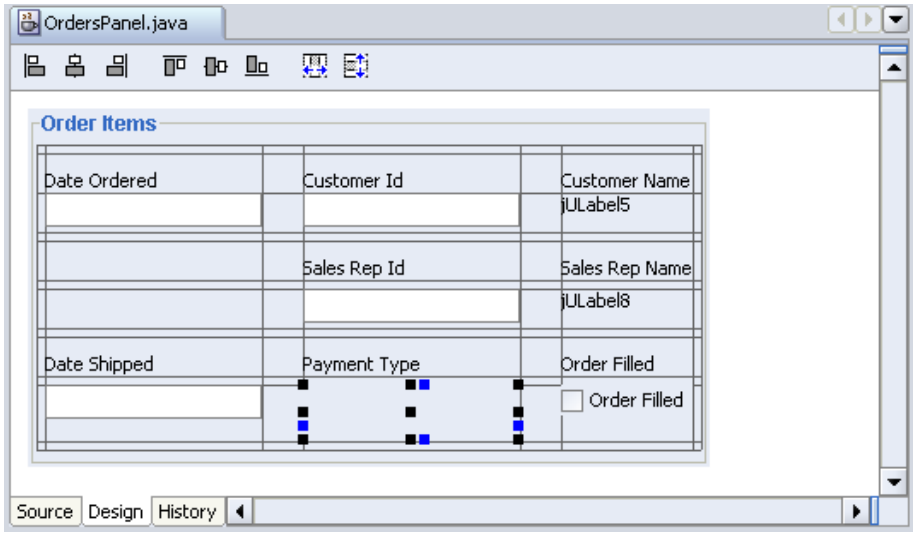

Figure 16 – JGoodies layout manager

Figure 16 shows a grid of six rows and three columns, with space columns, used to layout the order information.

A UI component can be created by dragging an element from the data model onto the Swing panel and Oracle ADF automatically creates the binding from the data model to the UI control.

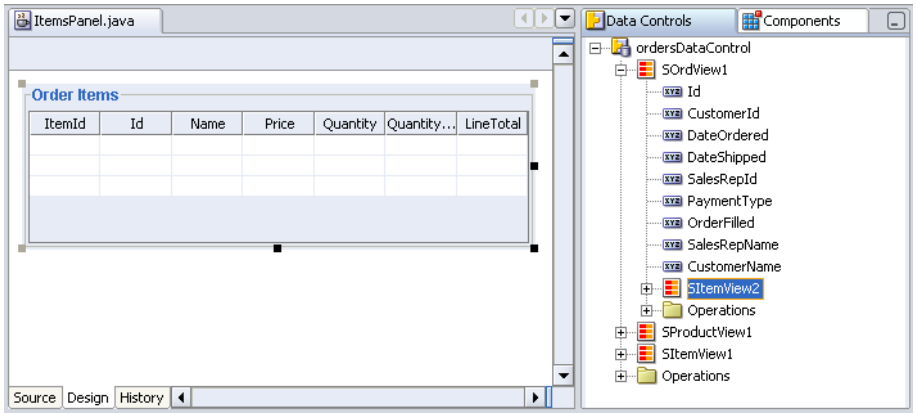

Figure 17 – Dragging from the data control palette

Figure 17 shows the result of *SItemView2* dragged onto a panel as a table.

ADF Swing Components

In addition to standard Swing controls that work with ADF Swing models, there is a set of ADF Swing-specific controls. These additional controls support data bound behavior not provided by standard Swing controls.

JUNavigationBar

JUNavigationBar is a menu bar that automatically supports navigation; find mode, insert and delete of the records.

JUImageControl

JUImageControl supports the display of BLOB and interMedia data.

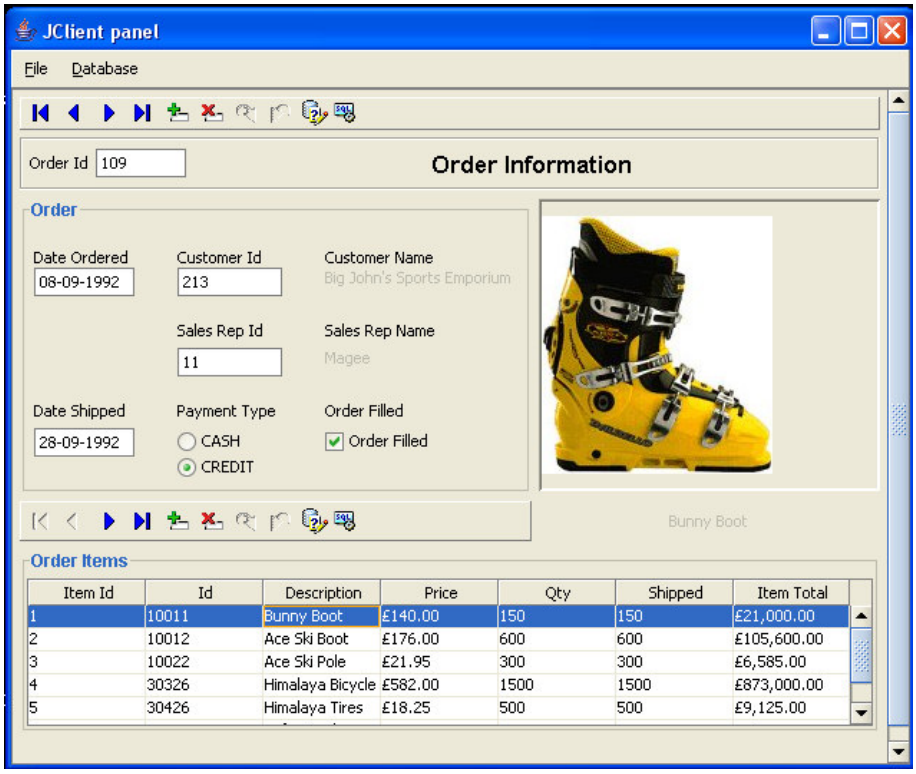

Figure 18 – JGoodies layout manager

Figure 18 shows the *JUImageControl* and two instances of *JUNavigationBar*, one that is attached to the Order data and one to Order Items data.

Adding Push Buttons

The Summit application also includes a button bar for displaying pop up dialogs and implementing application function. Each button implements a *When-Button-Pressed* trigger that in turn calls Forms built-ins to perform DML operations or display various styles of pop up screens.

JDeveloper and Oracle ADF provide similar functionality.

DML Operations

ADF Business Components automatically creates operators on your data model. For example, next record, previous record, delete and commit. By simply dragging the operation onto the required panel, a button is automatically created and the appropriate action is hooked up to the button without having to write any code as show in figure 19.

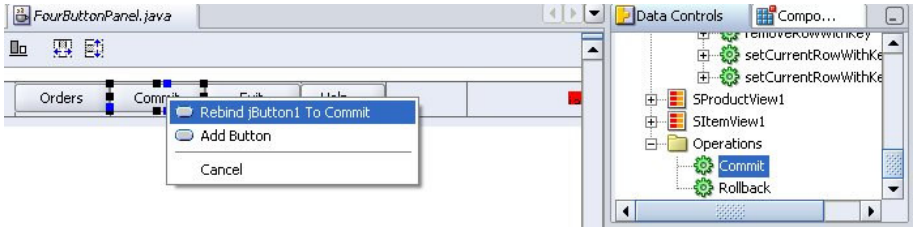

Figure 19 – Assigning DML operations to a button

Launching a dialog

Any panel or dialog previously created can also be easily assigned to a button. As shown in figure 20, simply dragging the dialog onto a panel allows the dialog to be launched on the press of the resulting button.

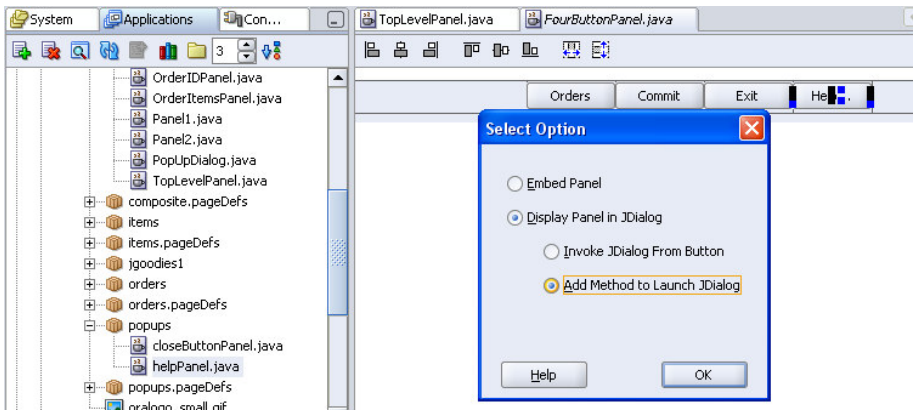

Figure 20 – Launching a dialog from a button

REVIEWING THE COMPLETED APPLICATION

As stated earlier, the goal of this paper is not to promote the redevelopment of Forms application but instead show that the concepts core to a Forms application have direct mappings to those presented by JDeveloper and Oracle ADF.

Conclusions

From redeveloping this application the following broad conclusions can be made.

Database access

As expected, the core feature of any Forms application is the reading and writing of database tables. Oracle ADF provides an almost identical mapping allowing the developer to map and manipulate database tables without resorting to code. Often coded features in Forms, such as look up, are available declaratively in Oracle ADF.

Validation

While Oracle ADF also provides the same declarative validation as Forms, it further extends the validation to provide a much richer set of options. When validation needs to branch out into code, Oracle ADF also provides a simple mapping of the Forms validation triggers.

UI

The UI development, by the very nature of Swing, is richer and therefore has a steeper learning curve than Forms. However, the *JGoodies* layout manager provides an intuitive layout and should be familiar to anyone who has used an HTML table. The assigning of actions to buttons was similar to Oracle Forms. Although most UI manipulation was done through the WYSIWYG editor, the nature of a Swing UI means the page itself is code and so would increase the code count if a comparison were made between the Forms version and the Java version of the application.

Image manipulation

The only area that required more coding that in the Forms application was reading the image file from file system. However, once the code for reading from the file system had been written, it could be easily reused from a library.

Summary

For more information on building Java applications for Forms developers go to: http://otn.oracle.com/formsdesignerj2ee

At the start of this project, the Summit application was chosen, not because it was built as an application that could be rewritten easily, but because it was a typical Forms application. The redevelopment shows that most of the concepts that are core to a Forms application can be done in a similarly codeless and declarative manner. In some instances common Forms coding practices, like *Post-Query* lookups, can be implemented without code using Oracle ADF.

It can also be seen, that Oracle ADF provides an abstracted layer that affords developers productivity gains in line with, and in many cases, beyond, those previously available in Oracle's classical tools suite.

Case Study: Redeveloping an Oracle Forms application using Oracle JDeveloper and Oracle ADF
Augsut 2007
Author: Grant Ronald
Contributing Authors:

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
oracle.com