An Oracle White Paper
July 2010

Build Rich ADF User Interfaces for Enterprise Java Beans

A Case Study

Introduction

The Enterprise Java Bean (EJB) specification has improved considerably with the release of EJB 3.0 as part of the overall JEE 5 specification. EJB development has become much easier with annotations, resource injection, and the Java Persistence API (JPA). However, the development effort required to build effective JEE user interfaces for EJBs is still considerable. This paper is a case study of how the Oracle Application Development Framework (ADF) provides a declarative approach to address this issue, improving developer productivity significantly. An end-to-end application for managing movie ticketing is described in detailed steps, covering the development of database schema, EJB middle tier, and subsequently a JSF user interface based on ADF Faces.

Building solution using EJB + JPA and ADF Binding

Let us consider a use case scenario to illustrate the feature richness of ADF and see how does ADF binding help to reduce the coding effort while building enterprise solutions using Java EE technology stack. In this demo we shall look at the 'Movie Ticketing Application'. One can perform the following

- Browse through the theaters for a selected city
- Browse through the movies for a selected city
- Choose movie, theatre and book tickets
- Pay for the tickets

We learn to use the following –

Business Service Layer

- Creating a Java EE application with EJB Model project.
- Setup DB connection
- Create
    - Entities,
    - Session beans

View Layer

- Create
    - ADF Datacontrols
    - Taskflows to describe flow of data
    - Page fragments and add them to the same page

The demo application (Movie Ticketing Application) built based on the above use case can be downloaded here.

Set up Application Infrastructure

Setup the schema

The tables used by the Movie ticketing application can be installed into any schema. Run the script to install the tables. The ER diagram below depicts the tables used for building Movie ticketing application and the relation between the entities.

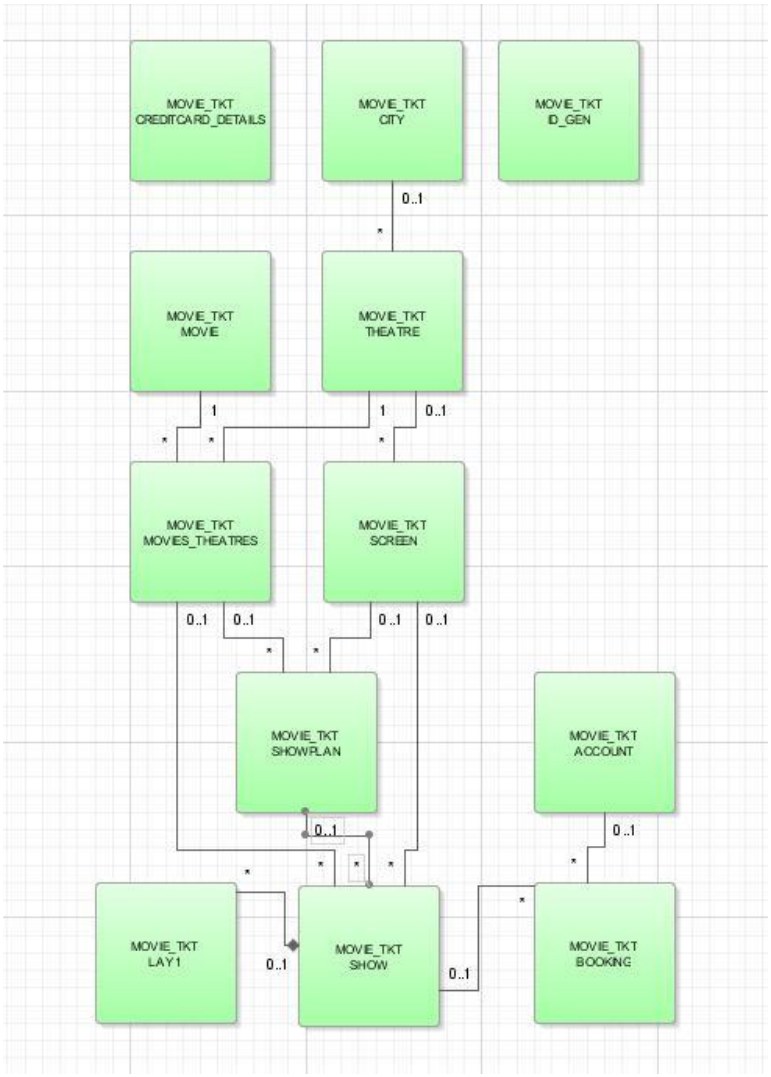


Figure 1: High level class diagram for Movie Ticketing (Demo application)

Create an Application from a Template

Create a new application by choosing Java EE Web Application template.

Proceed to next step by clicking 'Next' button on the wizard

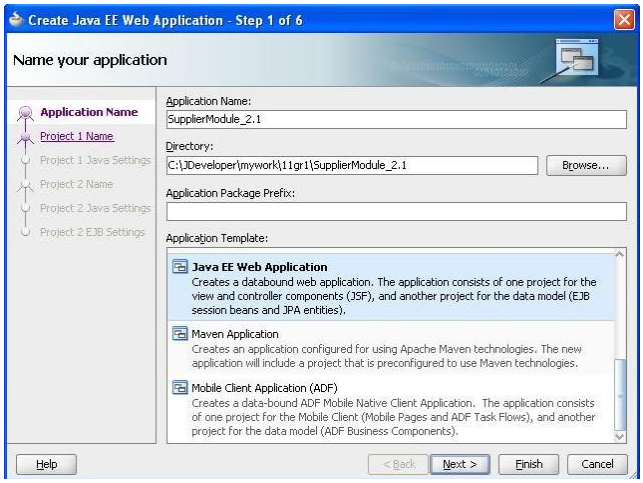


Figure 2: Wizard to create new application based on Java EE template

To the Project that defines the View Layer for the application, add ADF Pageflow libraries and ADF Faces to develop Web application and ADF Swing to develop Swing application. Please see the picture below

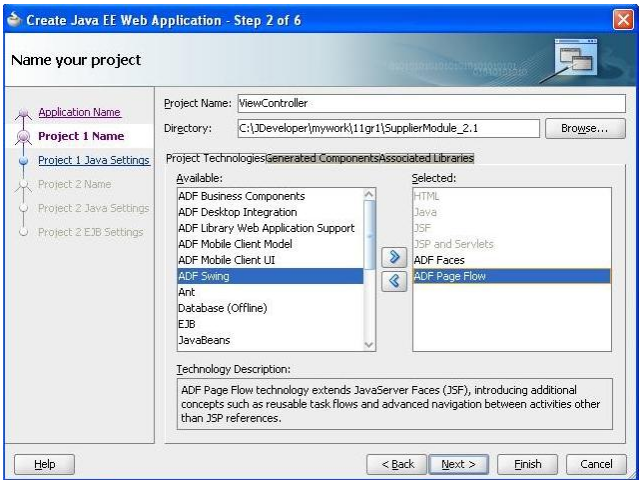


Figure 3: Wizard to create new application based on Java EE template

Click Finish button to save the settings

Set Up the Connection

Go to the Application Resources pane

Right Click and select the wizard to create a new Database Connection for the application. Connection must be based on the schema where the tables have been created.

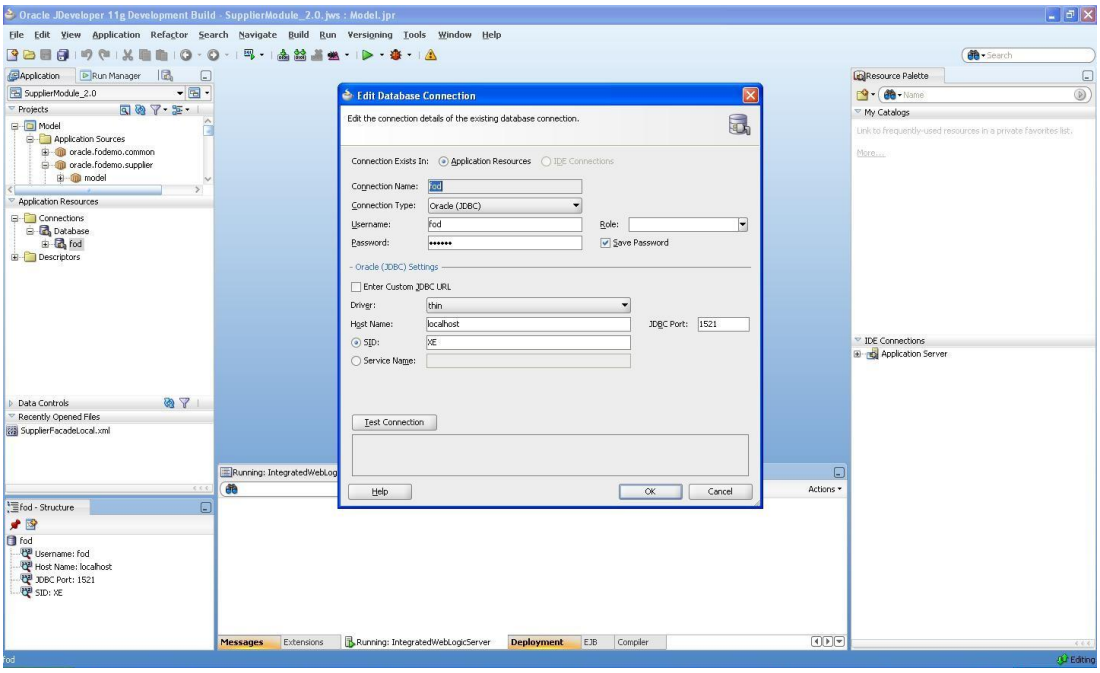


Figure 4: Connection Wizard

Create JPA entities from existing tables

The underlying assumption here is that schema for this application is already defined and tables are in place by having run the script. We are trying to create JPA entities through reverse engineering.

Select the model Project, then right click and select EJB displayed under Business tier. From tree view displayed on right hand side, select Entities From Tables and click OK. Follow the wizard and generate entities by selecting the following tables –

- Account
- Booking
- City
- Movie
- Screen
- Movies_Theatres
- Show
- Showplan
- Theatre

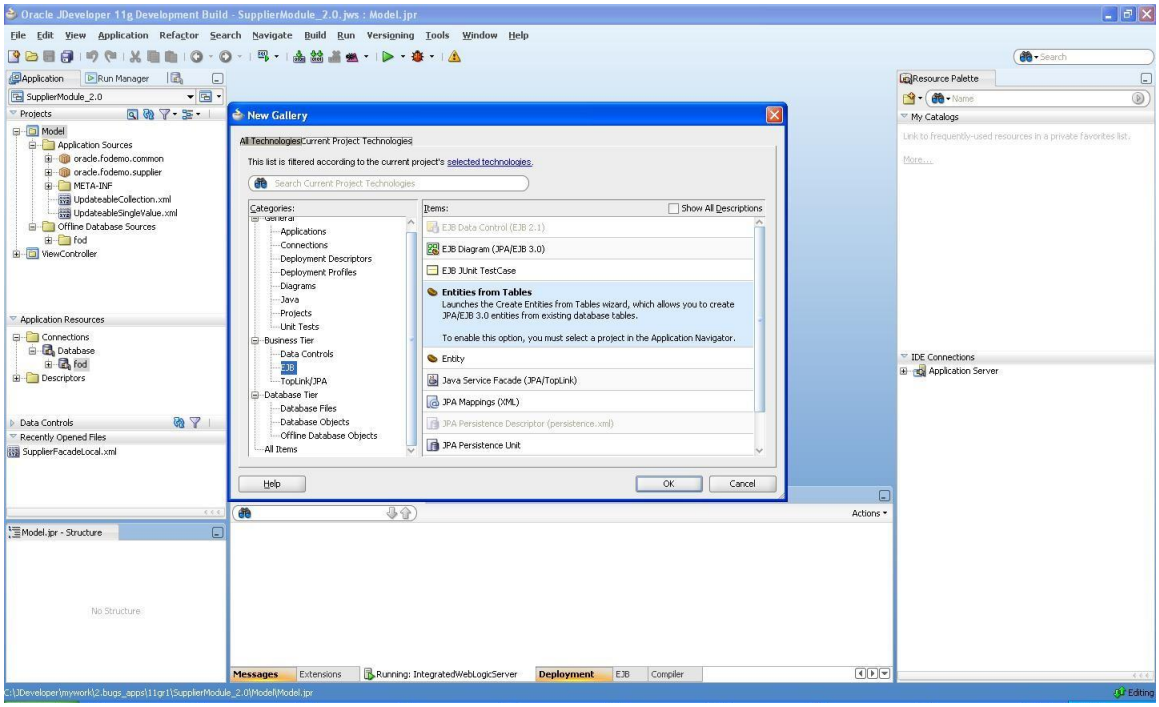


Figure 6: Wizard to create JPA entities from tables

Once the entities are generated add them to the EJB diagram and compare them to the entities shown in *Figure 5: Entity data model for the demo application* and ensure that the business service layer looks as shown below.

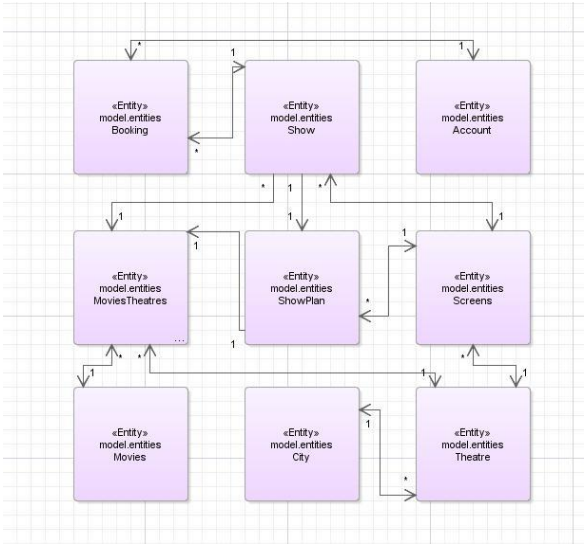


Figure 5: Entity data model for the demo application

Create Session Façade for the model

Right Click on the model project, select 'New' to see the New Gallery.

In the tree view, select EJB from Business Tier.

Select Session Bean from the right pane. Finish the Wizard

Create the following session beans (code can be found in the sample application) –

- BookingOperations,
- CityOperations,
- MovieOperations,
- ScreenOperations,
- ShowOperations
- TheatreOperations

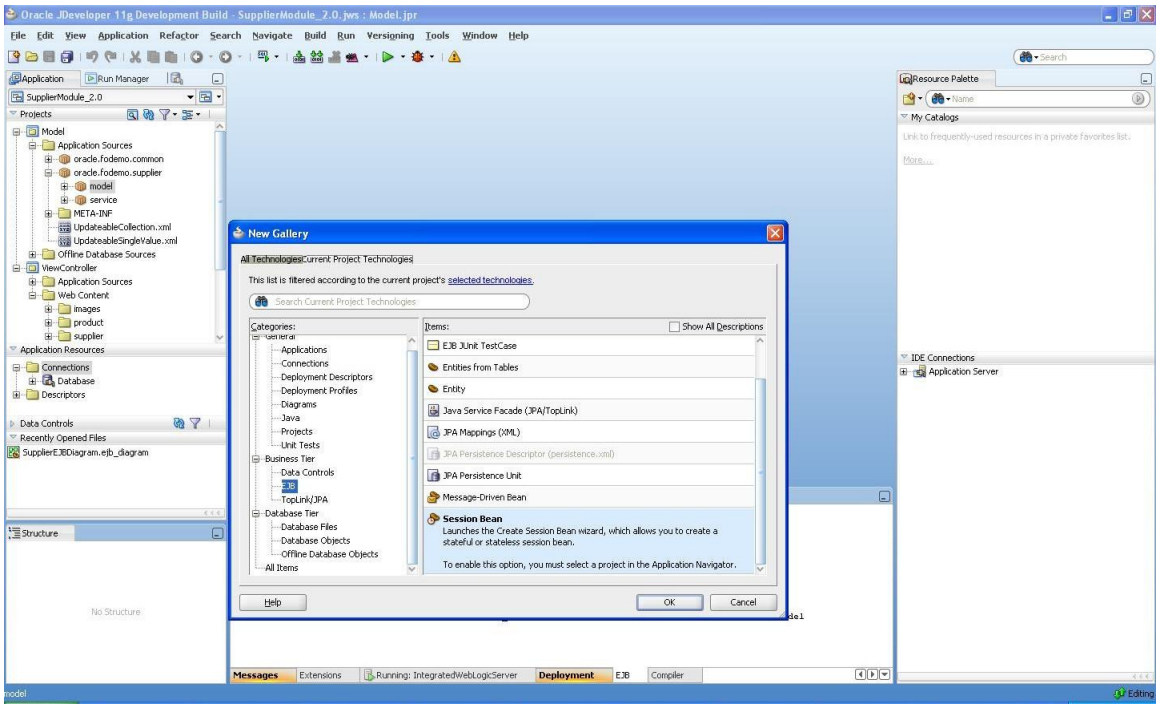


Figure 7: Wizard to create Session Facade

Create Data Control for Session EJB

Right Click on the Session Bean created in the above step and select Create Data Control option.

Select Local interface option and click OK to finish the job.

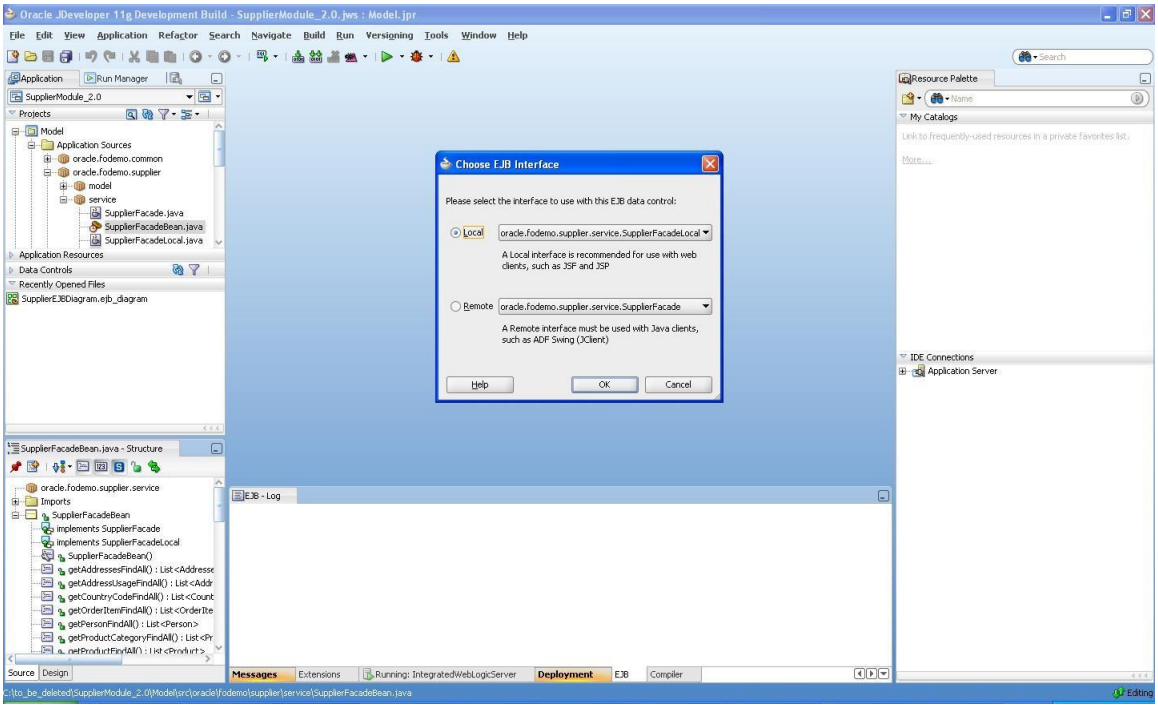


Figure 8: Wizard to create Session Facade

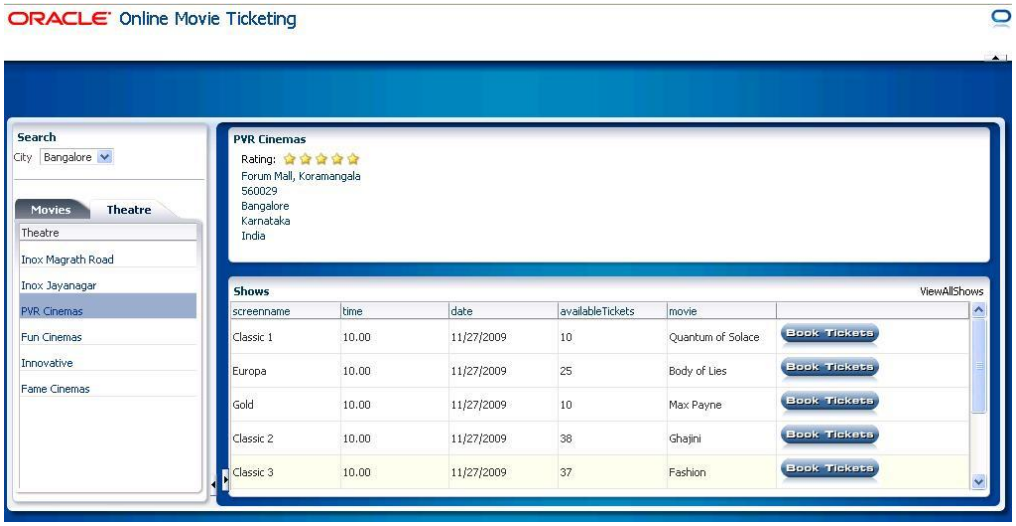
The basic infrastructure for building the application is in place now. We have not done any coding so far, JDeveloper has generated significant amounts of code behind the scenes! JDeveloper generates methods for standard operations like persist, merge, constructor etc.

Build Application Visually

Let us build the UI for the business services that we created in the previous section. To understand the power of ADF Binding and taskflows lets consider a use case from the demo application and look at the implementation in detail.

Use Case: Search shows for a Theatre and display the result on a table

- The screenshot below shows the 'Search by Theatre' at run time



ShowByTheatre

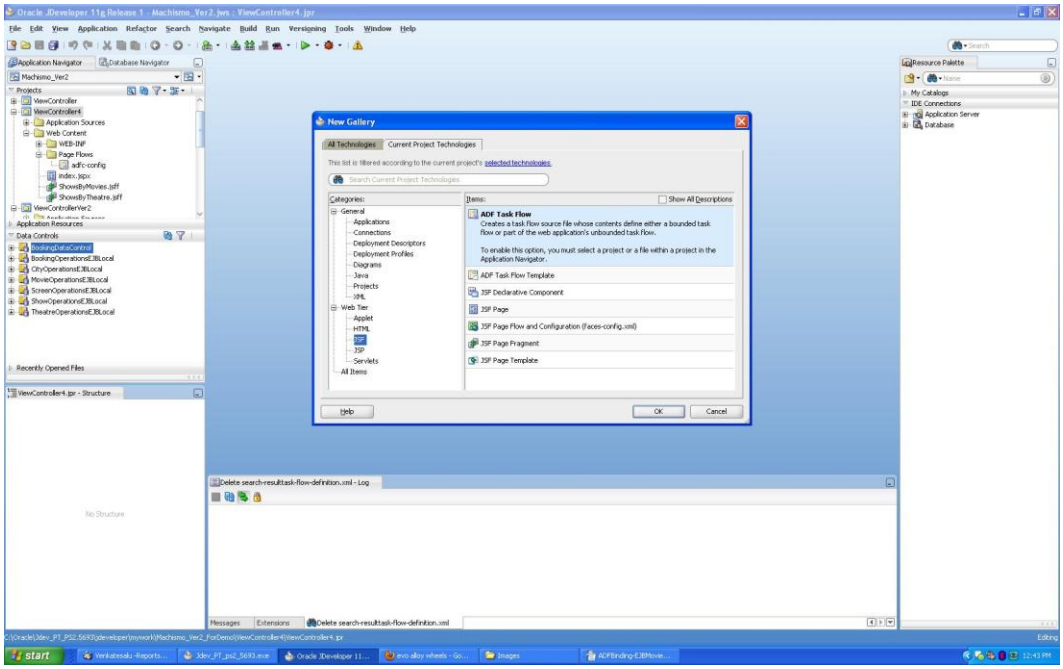
- This usecase implements the following –
    - Based on value selected in 'City' list, refresh the 'Theatre' tab
    - Based on value selected in 'Theatre' table, refresh the 'Shows' table
    - Clicking 'Book Tickets' confirms the purchase.

The usecase makes extensive use of the ADF task flows and Regions.

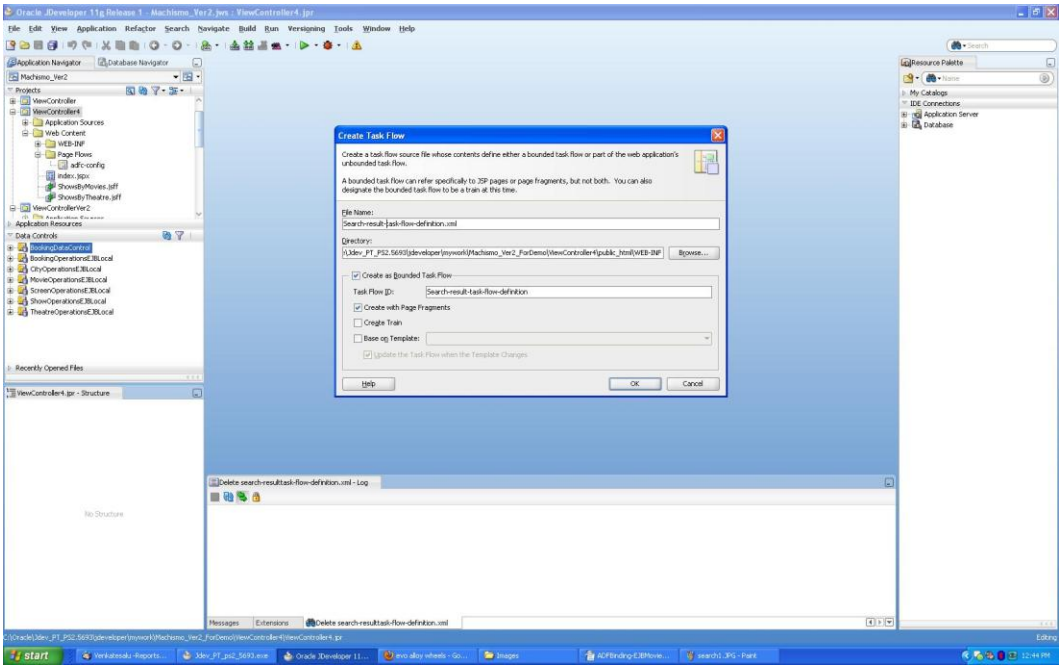
Taskflows provide a modular approach for defining control flow in an application. A single large JSF page flow can be broken into a collection of reusable task flows.

Create ADF Task Flow

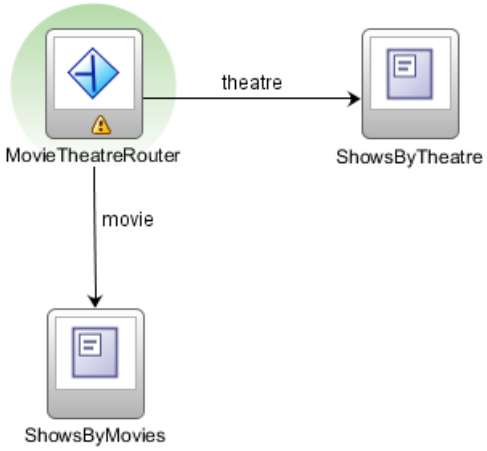
- Create a bounded task flow definition from the New Gallery.



- Create the taskflow with page fragments



- From the component palette, select and insert two View Activities named *showsByMovies* and *showsByTheatre*
- Double click the views to create page fragments - *showsByMovies.jsff* and *showsByTheatre.jsff*.
- The taskflow has router as the default activity. The router routes to the appropriate search page based on the 'searchBy' parameter, which is passed as a parameter to the taskflow. Create a router from the component palette and name it as MovieTheaterRouter (In Property Inspector palette set *Default Outcome*: movie)
- From the component palette, draw a flow case from MovieTheaterRouter to showsByMovie with the property *From Outcome* set to - movie
- From the component palette, draw a flow case from MovieTheaterRouter-> showsByTheatre with the property *From Outcome* set to theatre
- Select the MovieTheaterRouter view, in Property Inspector *Default Outcome* should be : movie



- Add new cases to the Router, In expression click on the Expression builder and build the below expression

Expression	Outcome
#{pageFlowScope.searchType=='movie'}	movie
#{pageFlowScope.searchType=='theatre'}	theatre

- The left pane displays the search by movies; theatres result in panel-tabbed layout.
- Drag and drop the Search result taskflow as Dynamic region on right pane of the main page.
- The task flow accepts parameters required for both Search by movies and Search by theatre pages.
- Set the refresh property of the region to 'ifNeeded' so that the region is refreshed when the parameter changes else the region will be refreshed on page refresh.
- Set Partial trigger of tables in tab to City drop down box and Movie date and the cache property to false so that the change in parameter will automatically trigger the iterator and is reflected in the table.
- Create an action listener associated to the managed bean and associate it to the command link in the search results table in the left. The action listener sets the parameter required for the taskflow in page flow scope and the refresh type is set appropriately.

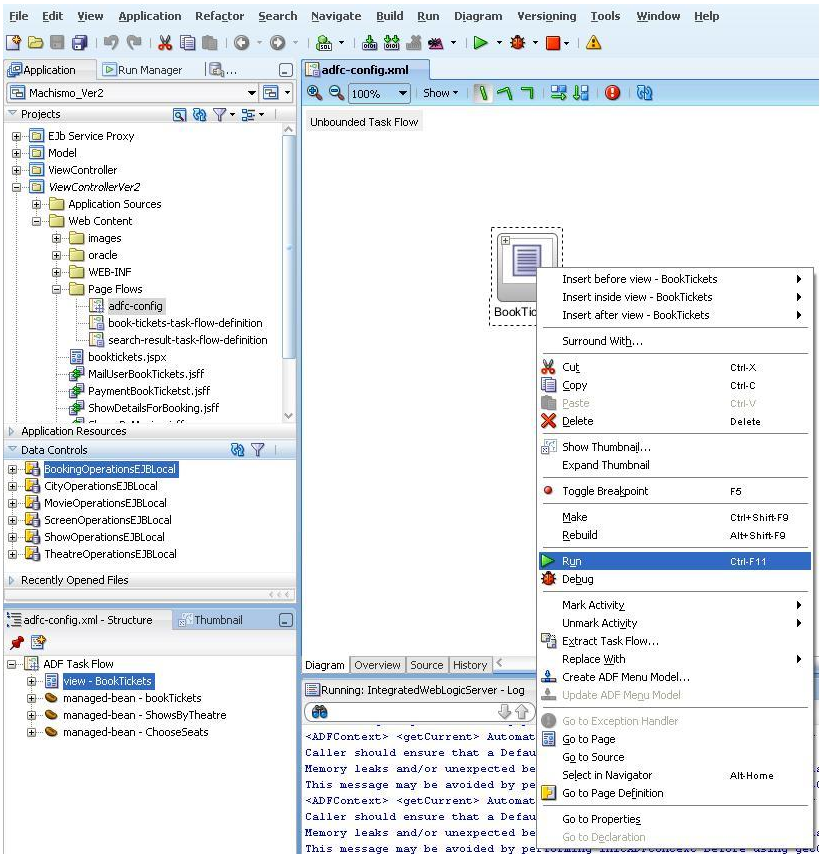
Input Parameter Definitions

Name *	Class	Value
searchType	java.lang.String	#{pageFlowScope.searchType}
theatreId	java.lang.Integer	#{pageFlowScope.theatreId}
movieId	java.lang.Integer	#{pageFlowScope.movieId}
showDate	java.util.Date	#{pageFlowScope.showDate}
cityId	java.lang.Integer	#{pageFlowScope.cityId}
refreshTaskFlow	java.lang.String	#{pageFlowScope.refreshTaskFlow}

The above use case is intended to give you a feel of the ADF Binding layer and task flow. To gain a deeper insight in the application, please download the Movie Ticketing application at the link provided at the start of the article.

Running the Application

Now that you have built your application, you need to test it. JDeveloper makes it easy to test JSF pages through a built-in application server. The server is automatically launched when you test a page from within JDeveloper.To run the application, return to the page flow diagram. Right click the **BookTickets** page icon and select **Run** from the context menu.



Conclusion

This article has described the steps to develop a JSF user interface for an EJB application based on Oracle's ADF Faces framework, showing how the ADF approach enables JEE user interfaces to be developed in a declarative and productive manner.

REFERENCES

- Fusion Developer's Guide for Oracle ADF
- Build a Web Application with JDeveloper 11g Using EJB, JPA, and JavaServer Faces
- The Java EE 5 Tutorial

ORACLE

Build Rich ADF User Interfaces for Enterprise Java Beans - A Case Study
July 2010
Author: Praveen Thulasiraman

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