6
Portlets

In the previous chapter, we looked at how to build a page containing information on our example company's requirements, and the offers made by its suppliers. Now, we are interested in adding a chart to that page that graphically shows the status of requirements pending and closed, as a percentage. However, this graph will not only be required on this page but in many others. Therefore, it is necessary build this chart as a reusable component. In this kind of scenario, the technology of portlets can help us.

A portlet is a component of software that can be part of a page. In consequence, a page can be enriched by many portlets allowing us to grab the contents of various sources of data. This chapter will help us to understand the concept associated with this technology and how it can apply to our projects with Oracle WebCenter.

During this chapter, you will learn the following:

- JSF specification concepts
- The types of portlets you can build with WebCenter
- Developing a portlet using ADF
- Integrating portlets with custom Applications

Portlets, JSR-168 specification

Specification JSR-168, which defines the Java technologies, gives us a precise definition of Java portlets:

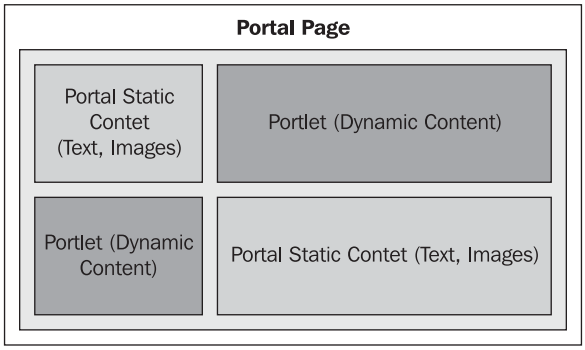
> *Portlets are web components – like Servlets – specifically designed to be aggregated in the context of a composite page. Usually, many Portlets are invoked to in the single request of a Portal page. Each Portlet produces a fragment of markup that is combined with the markup of other Portlets, all within the Portal page markup.*

You can see more detail of this specification on the following page:
`http://jcp.org/en/jsr/detail?id=168`

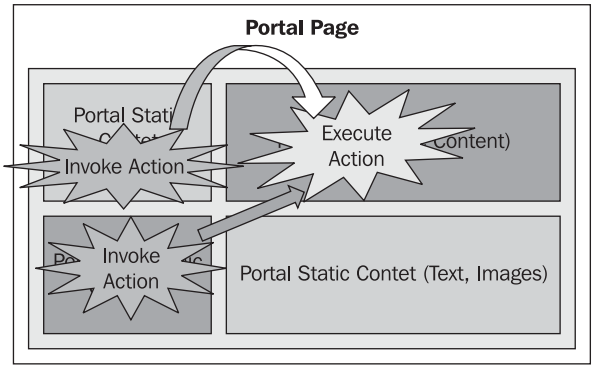
While the definition makes a comparison with servlets, it is important to note that the portlets cannot be accessed directly through a URL; instead, it is necessary to use a page-like container of portlets.

Consequently, we might consider portlets as tiny web applications that return dynamic content (HTML, WML) into a region of a Portal page.

Graphically, we could view a page with portlets as follows:



Additionally, we must emphasize that the portlets are not isolated from the rest of the components in the pages, but can also share information and respond to events that occur in other components or portlets.



WSRP specification

The WSRP specification allows exposing portlets as Web services. For this purpose, clients access portlets through an interface (`*.wsdl`) and get graphic content associated. Optionally, the portlet might be able to interact directly with the user through events ocurring on them.

This way of invoking offers the following advantages:

- The portals that share a portlet centralize their support in a single point.
- The portlet integration with the portal is simple and requires no programming.
- The use of portlets, hosted on different sites, helps to reduce the load on servers.

WebCenter portlets

Portlets can be built in different ways, and the applications developed with Oracle WebCenter can consume any of these types of portlets.

- **JSF Portlets**: This type of portlet is based on a JSF application, which is used to create a portlet using a JSF Portlet Bridge.

- **Web Clipping**: Using this tool, we can build portlets declaratively using only a browser. These portlets show content from other sites.
- **OmniPortlet**: These portlets can retrieve information from different types of data sources (XML, CSV, database, and so on) to expose different ways of presenting things, such as tables, forms, charts, and so on.
- **Content Presenter**: This allows you to drop content from UCM on the page and display this content in any way you like or using a template.
- **Ensemble**: This is a way to "mashup" or produce portlets or "pagelets" of information that can be displayed on the page.
- **Programmatic Portlets**: Obviously, in addition to the previous technologies that facilitate the construction of portlets, it is also possible to build in a programmatic way. When we build in this way, we reach a high degree of personalization and control. However, we need specialized Java knowledge in order to program in this way.

As we can see, there are several ways in which we can build a portlet; however, in order to use the rich components that the ADF Faces framework offers, we will focus on JSF Portlets.

Developing a portlet using ADF

The portlet that we will build will have a chart, which shows the status of the company's requests. To do this, we must create a model layer that represents our business logic and exposes this information in a page.

Therefore, we are going to do the following steps:

1. Create an ADF application.
2. Develop business components.
3. Create a chart page.
4. Generate a portlet using the page.
5. Deploy the portlet.

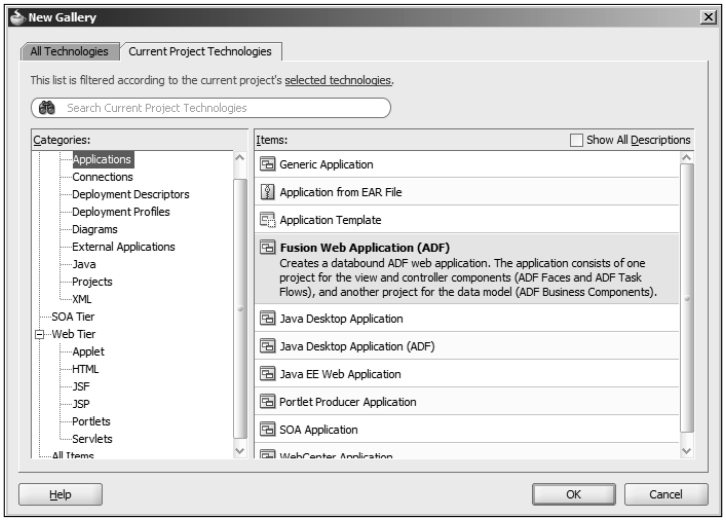
In this example, we use a page for the construction of a portlet; however, ADF also offers the ability to create portlets based on a flow of pages through the use of ADF TaskFlows. You can find more information on the following link:

`http://download.oracle.com/docs/cd/E15523_01/web.1111/b31974/taskflows.htm#BABDJEDD`

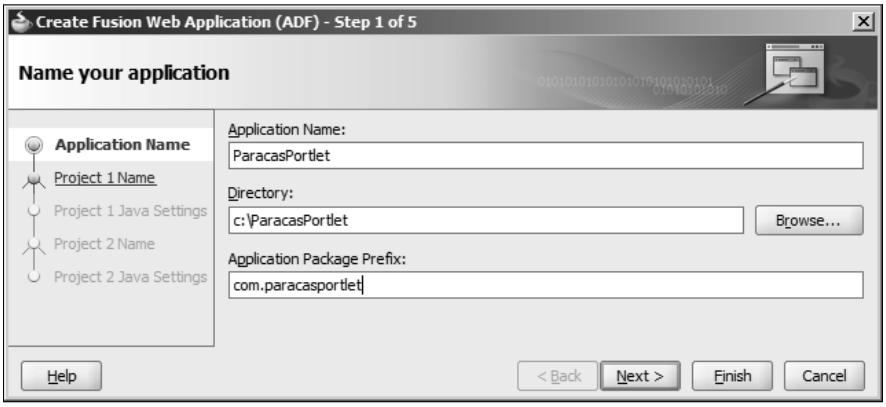
Creating an ADF application

To create the application, do the following steps:

1. Go to JDeveloper.
2. In the menu, choose the option **File | New** to start the wizard for creating applications. In the window displayed, choose the **Application** category and choose the **Fusion Web Application ADF** option and press the **OK** button.



3. Next, enter the following properties for creating the application:
   - Name: **ParacasPortlet**
   - Directory: **c:\ParacasPortlet**
   - Application Package Prefix : **com.paracasportlet**

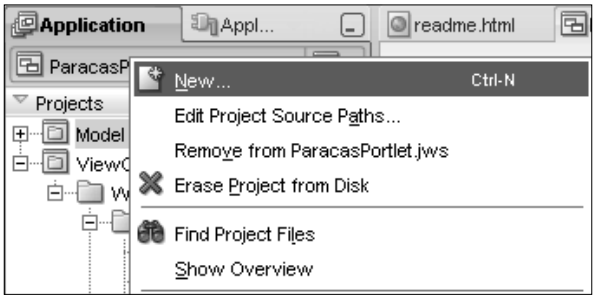


4. Click **Finish** to create the application.

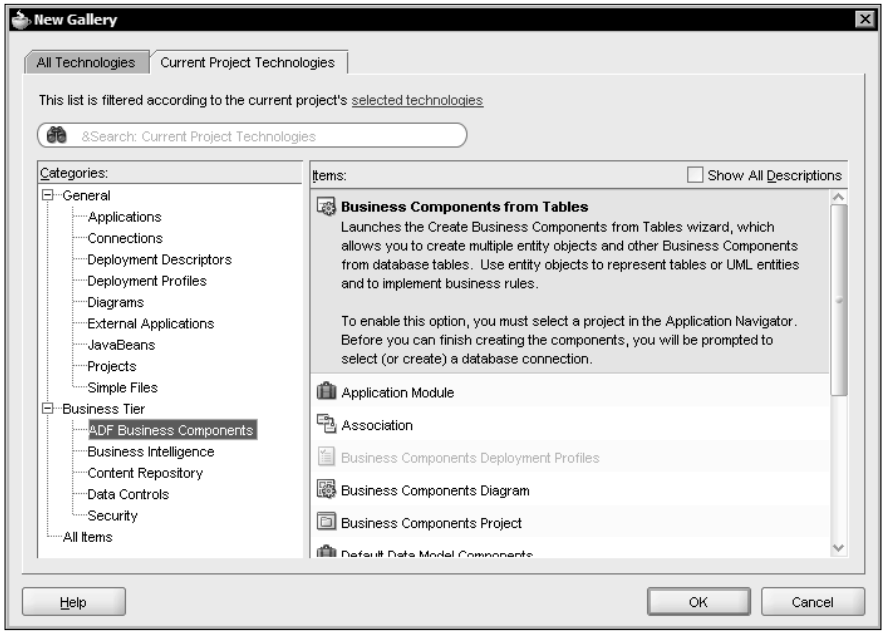
Developing business components

Before starting this activity, make sure you have created a connection to the database. To recollect this process, see in *Chapter 3*, *Setting up the Development Environment*.

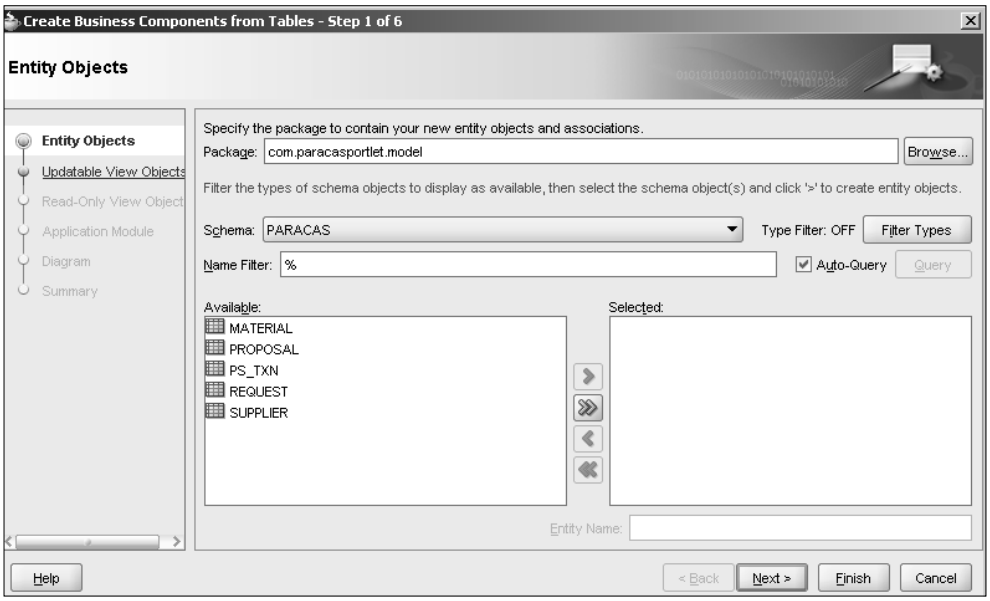
1. In the project Palette, right-click on **Project Model**, and choose **New**.



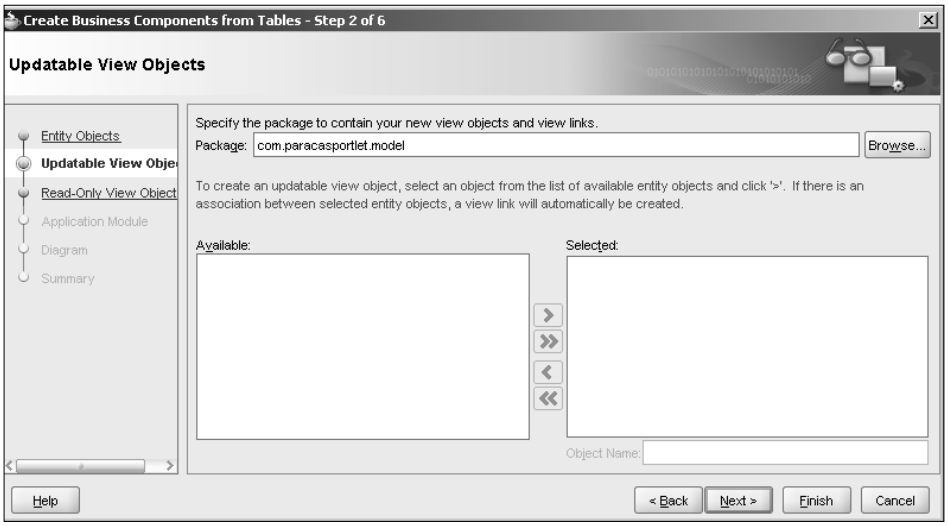
2. On the next page, select the category **Business Tier | ADF Business Components** and choose **Business Components from Tables**. Next, press the **OK** button.



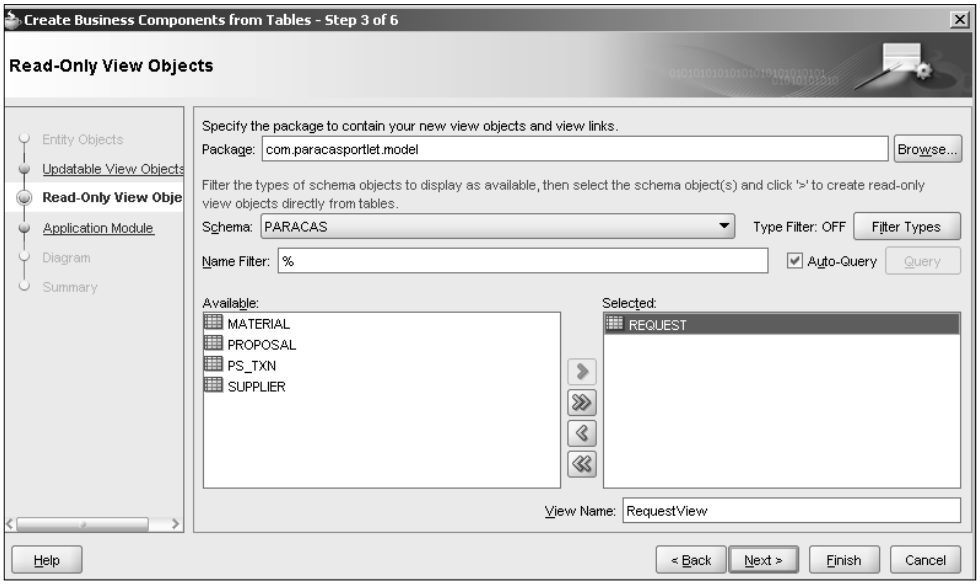
3. In the following page, you configure the connection to the database. At this point, select the connection **db_paracas** and press the **OK** button.
4. In order to build a page with a chart, we need to create a read-only view. For this reason, don't change anything, just press the **Next** button.



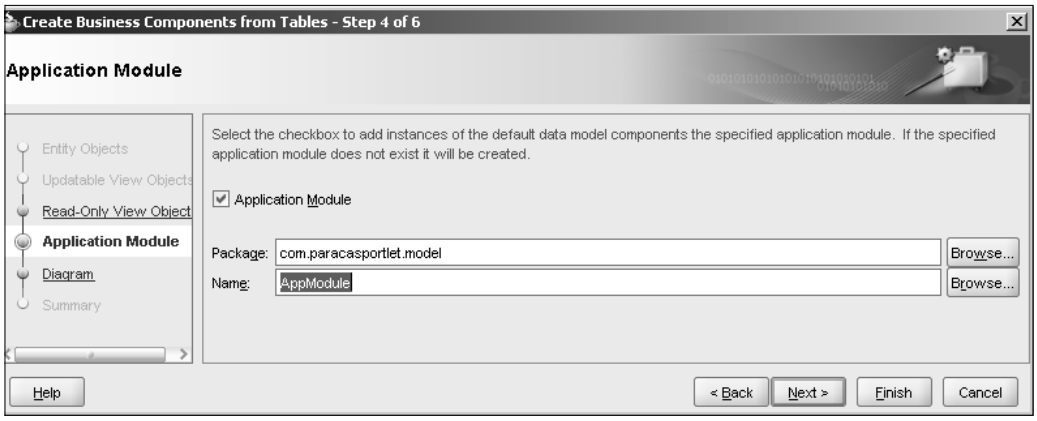
5. In this next step, we can create updateable views. But, we don't need this type of component. So, don't change anything. Click the **Next** button.



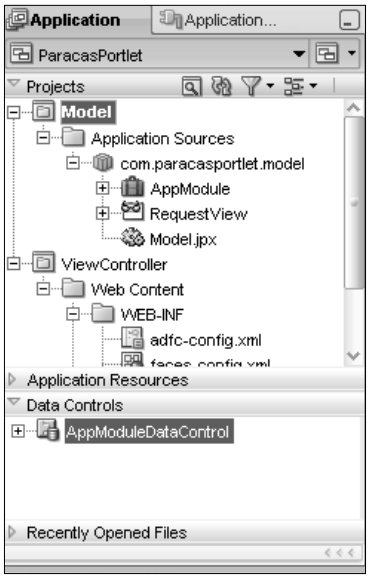
6. Now, we need to allow the creation of read-only views. We will use this kind of component in our page; therefore select the table **REQUEST**, as shown next and press **Next**.



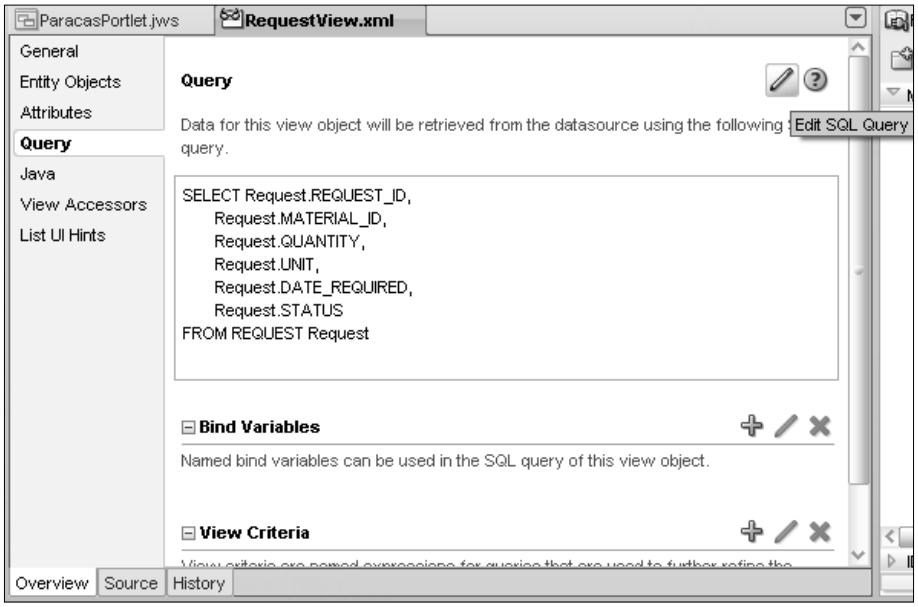
7. Our next step will allow the creation of an application module. This component is necessary to display the read-only view in the whole application. Keep this screen with the suggested values and click the **Finish** button.



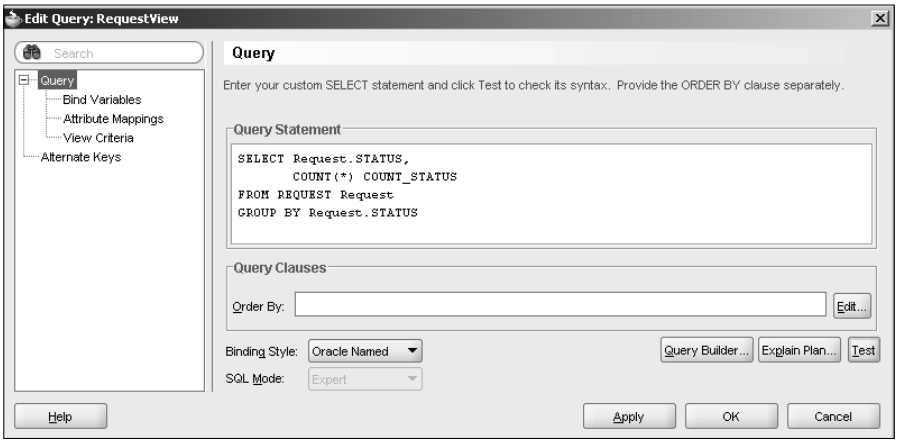
8. Check the Application Navigator. You must have your components arranged in the same way as shown in the following screenshot:



9. Our query must determine the number of requests for status. Therefore, it will be necessary to make some changes in the created component. To start, double-click on the view **RequestView**, select the **Query** category, and click on the **Edit SQL Query** option as shown in the following screenshot:

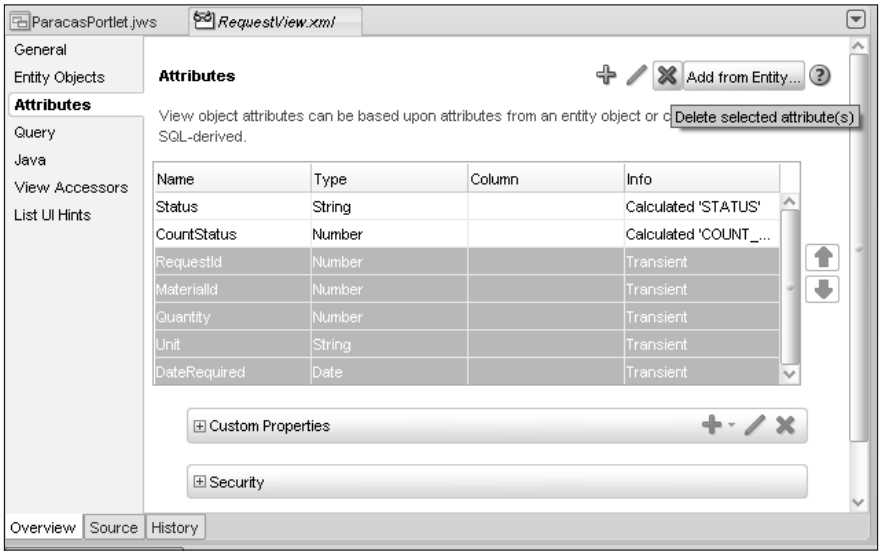


10. In the window shown, modify the SQL as shown next and click the **OK** button.

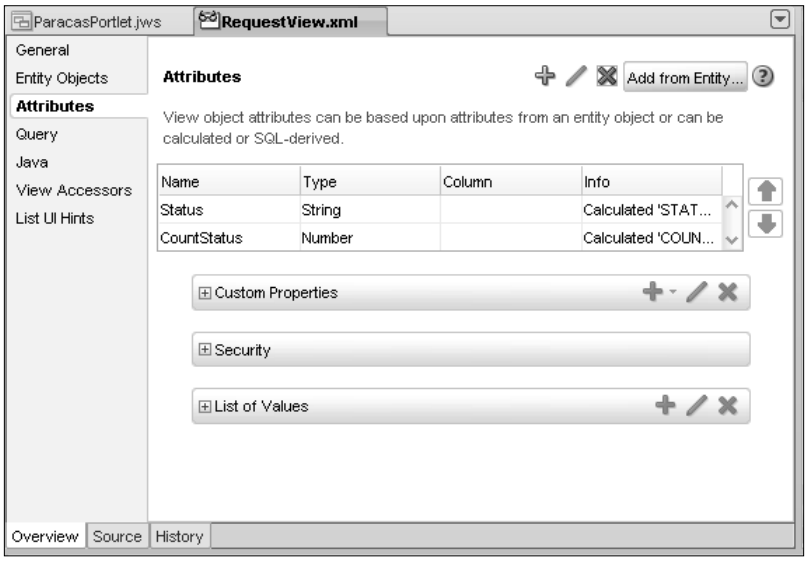



```
SELECT Request.STATUS,
       COUNT(*) COUNT_STATUS
FROM REQUEST Request
GROUP BY Request.STATUS
```

11. We only use the attributes **Status** and **CountStatus**. For this reason, choose the **Attributes** category, select the attributes that are not used, and press **Delete selected attribute(s)** as shown in the following screenshot:



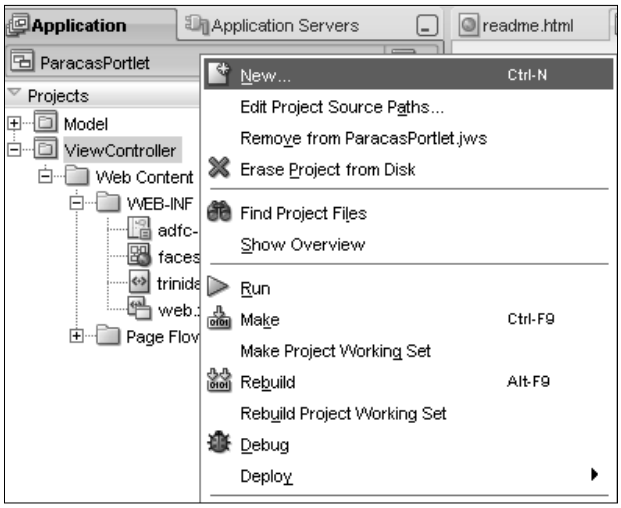
12. Save all changes and verify that the view is similar to that shown next:



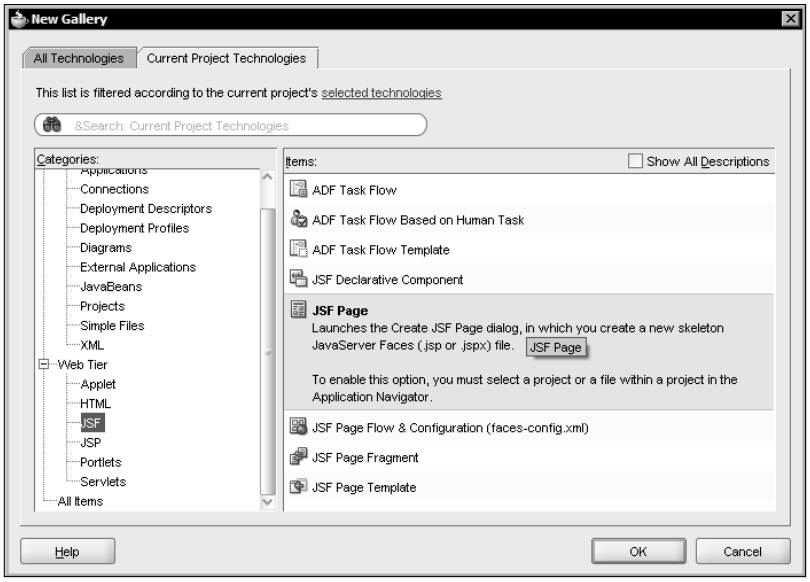
Creating a chart page

Once we have completed the stage of building the model, we will create the page that hosts our chart.

1. In Application Navigator, right-click on the project **ViewController**, then choose **New**.

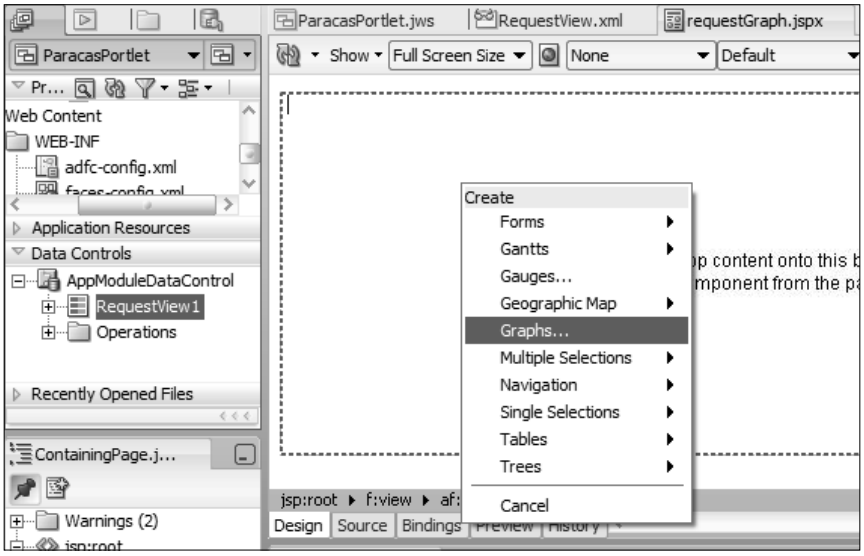


2. In the next window, select the category **Web Tier | JSF**, select the option **JSF Page,** and click the **OK** button.

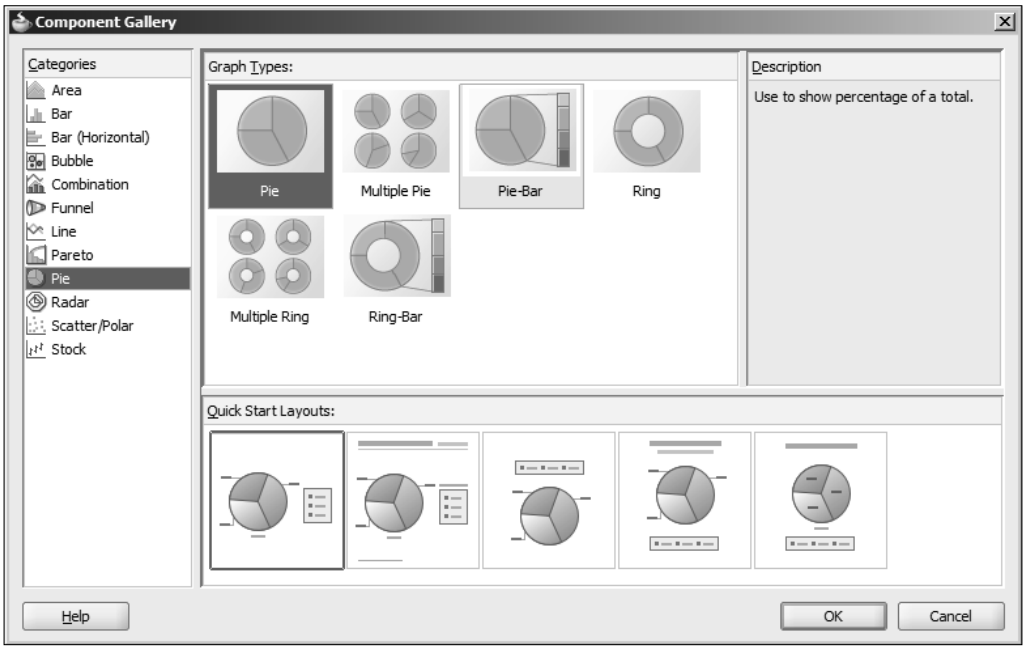


3. Then, we define the name of the page: `requestGraph.jspx` and press the **OK** button.

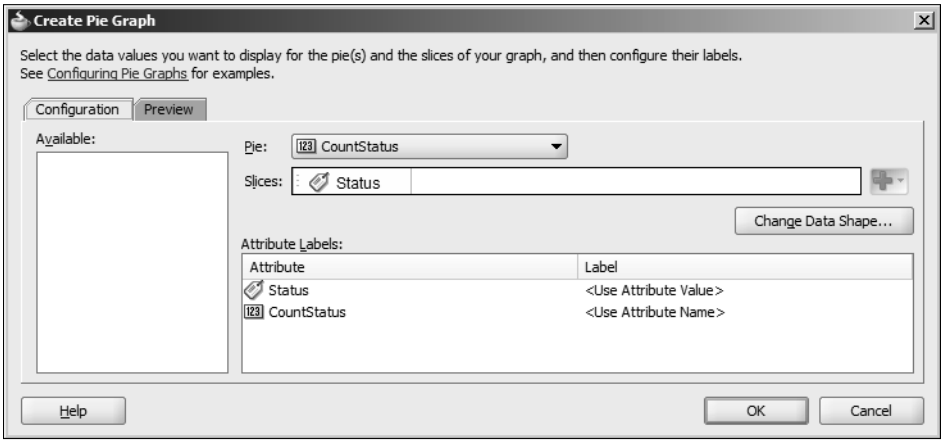
4. Now, locate the Data Controls palette and drag the view **RequestView1** to the page. Then choose, from the context menu, the option **Graphs** in the same way as shown in the following screenshot:



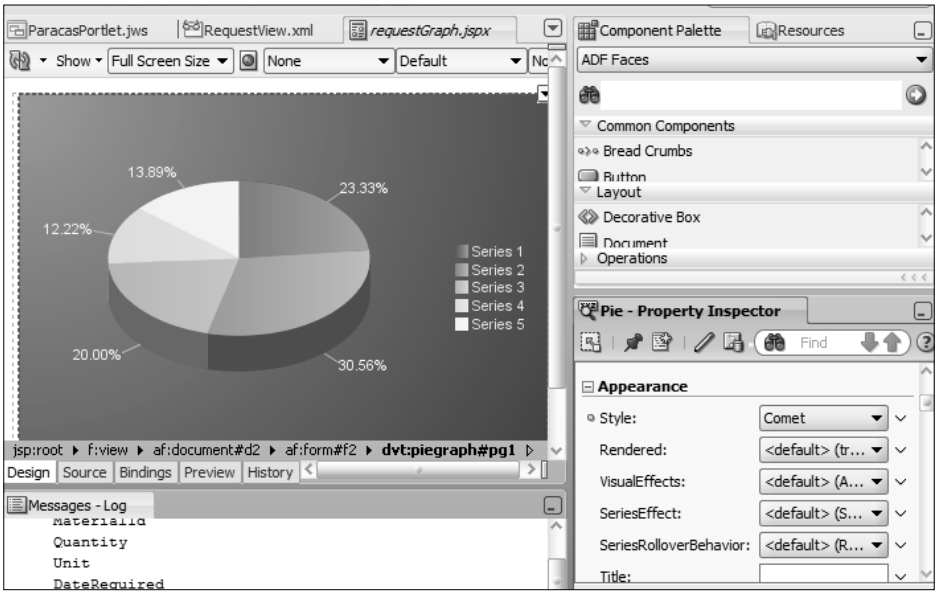
5. Next, choose the **Pie** category and Pie type item, as shown next:



6. After the last step, a screen will appear that indicates the attributes used to populate the graph. Choose **CountStatus** in the Pie item and **Status** in the Slice. Press **OK**.



7. Finally, change the graphic appearance. To do this, once you have created the component, go to the Property Inspector, select the **Appearance** region and change the value of property **Style** to **Comet** and the **3D Effect** to **true** as shown in the following screenshot:

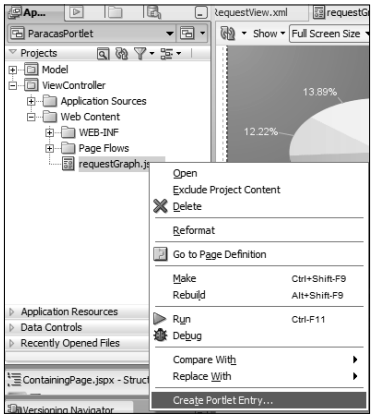


8. Save all the changes.

Creating a Portlet using the page

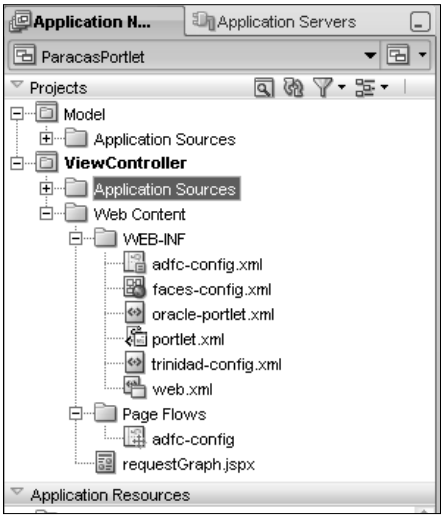
Now that we have our page, we will use it.

1. Go to Application Navigator and right-click on the page `requestGraph.jspx` and select **Create Portlet Entry** as shown in the following screenshot:



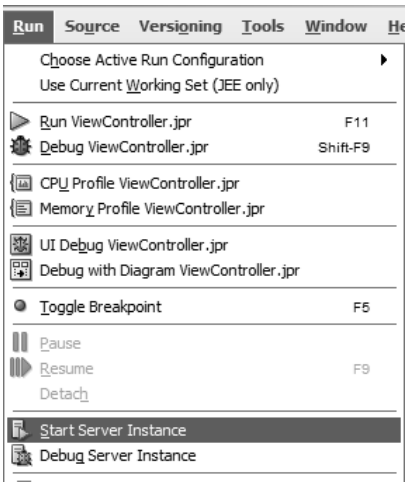
2. In the next window, change the names and descriptions of the portlet as shown next and click the **OK** button.
    - Portlet Name: **ParacasPortlet**
    - Display Name: **ParacasPortlet**
    - Portlet Title : **Requests by Status**
    - Short Title : **Requests by Status**
    - Description: **Requests by Status**

3. This last step will be sufficient to create the portlet based on the data page. Click **OK**, then check that the files `portlet.xml` and `oracle-portlet.xml` are created as shown in the following screenshot:



Portlet deployment

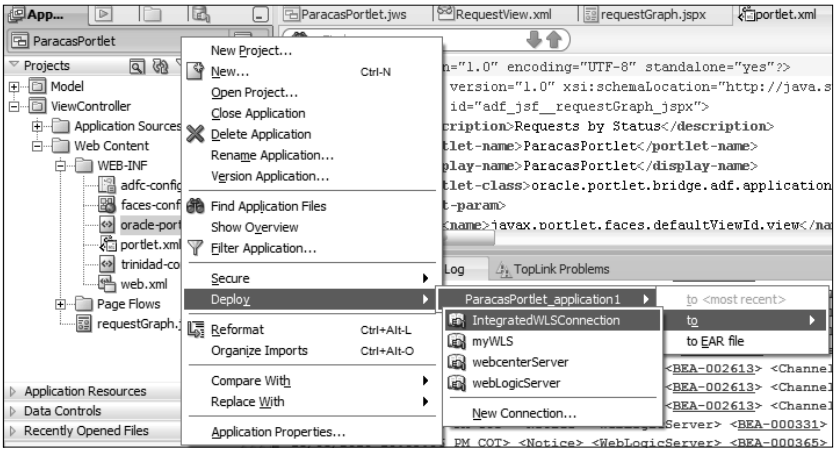
1. Once we have built our Portlet, it is necessary to include it in our Portal. To do this, we must obtain this portlet from a repository. To start with, we must ensure that the server is active. In order to start up this server, choose the menu option **Run | Start Server Instance**.



2. The following message at the Console confirms that the server is initialized:

```
DefaultServer started.
```

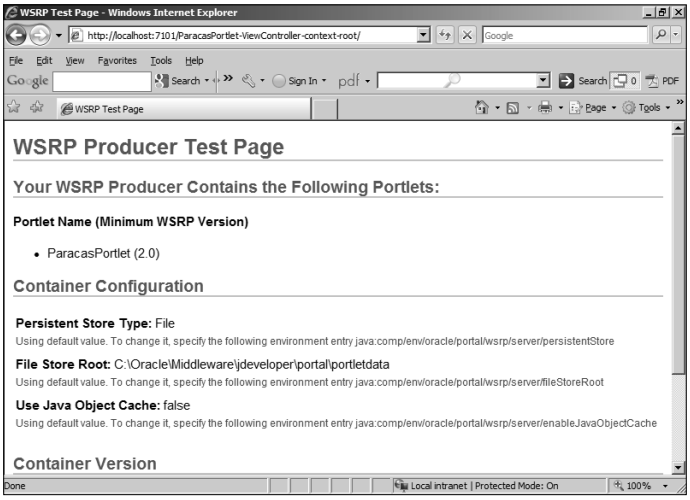
3. Right-click on the **ParacasPortlet** application and choose **Deploy | ParacasPortlet_application1 | to | IntegratedWLSConnection**, in the same way as shown next:



4. In the next window, click the button **OK** to complete the process.

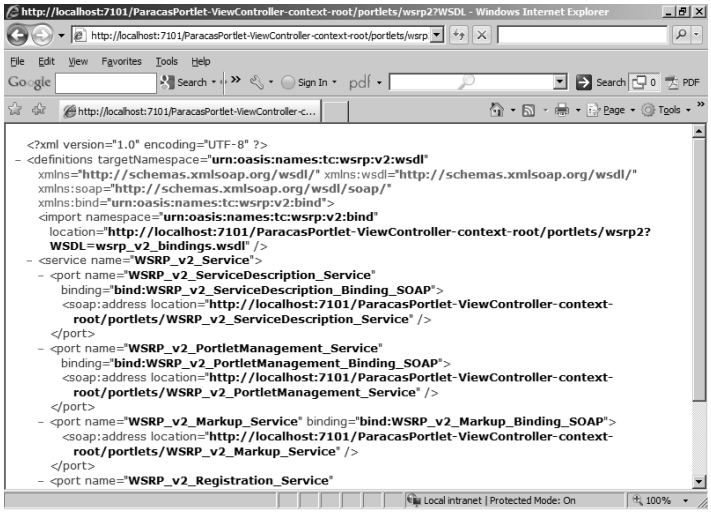
5. Finally, in order to check the deployment of portlet, go to the following page:

   `http://localhost:7101/ParacasPortlet-ViewController-context-root/`



This page shows the deployed portlet's information on the server and, amongst other things, provides links to the interface's WSRP. These interfaces are part of a specification-oriented communication between portlets. We'll use this interface to consume the portlet from other sites. Therefore, click on the link WSRP v2 WSDL and see this URL:

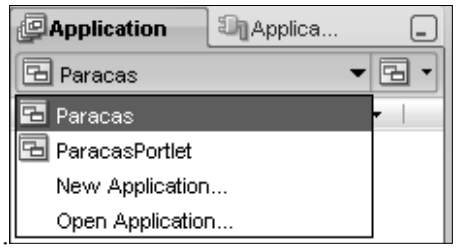
`http://localhost:7101/ParacasPortlet-ViewController-context-root/portlets/wsrp2?WSDL`



Consuming a portlet

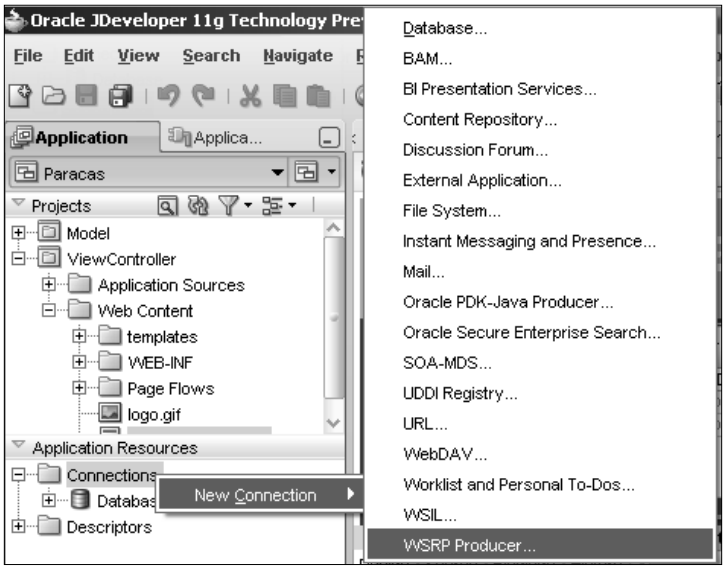
After the construction and deployment of the portlet, it's time for it to be re-used by another application. For this, it is necessary to create a reference to this portlet in the application that needs to use it. Let's see the steps for this integration:

1. In the **Application Navigator**, change the current application ParacasPortlet to the Paracas Application, which we were building in *Chapter 5*, *Design and Personalization*.



Alternatively, you can use the `ParacasEndOfChapter5.zip` file that is included in the code bundle of the book.

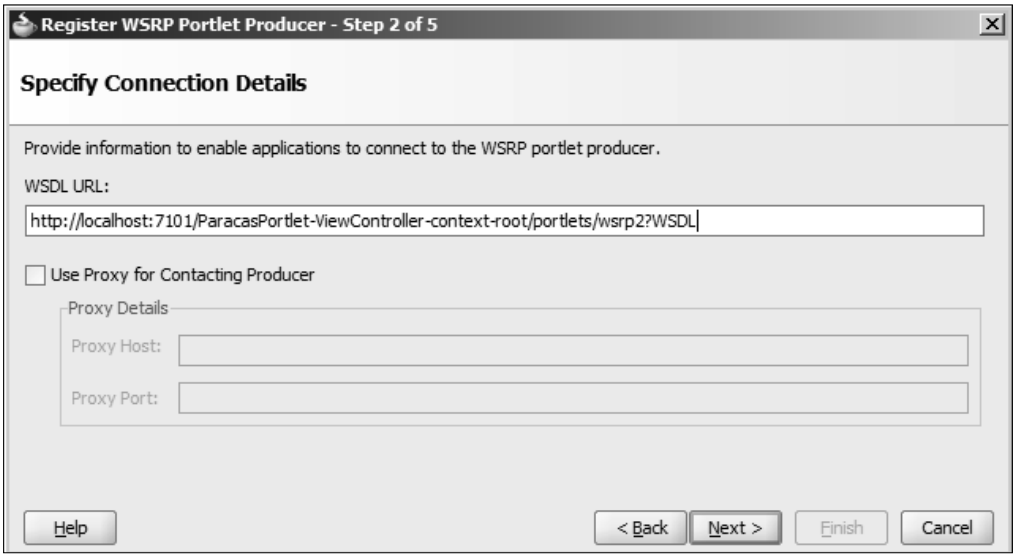
2. In the Application Navigator, locate the section Application Resources. Then, right-click on **Connections** and choose the **New Connection | WSRP Producer**.



3. In the next window, rename the registration of the portlet to **WsrpPortletParacas** and click **Next**.

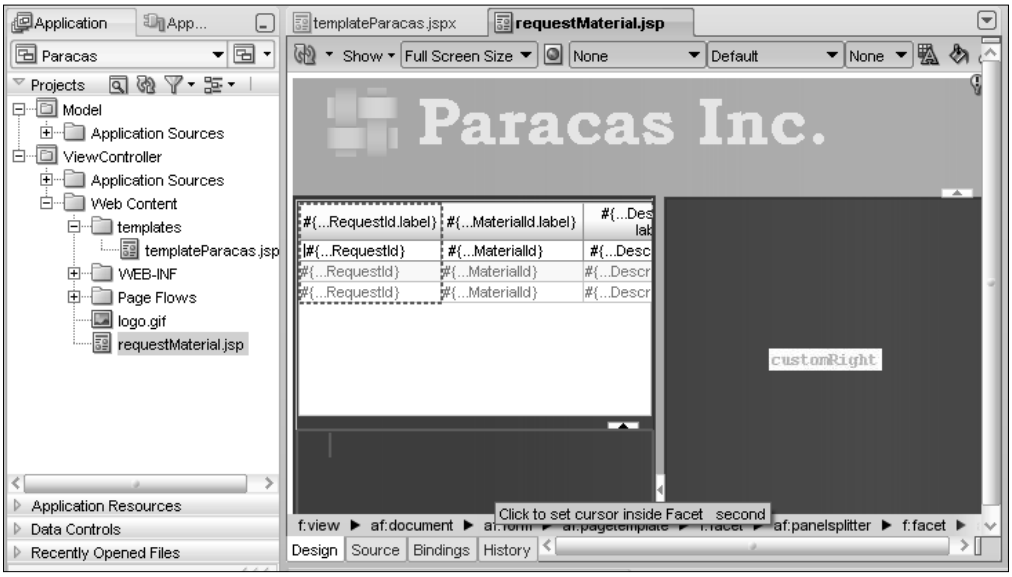
4. In the URL for the endpoint, type the URL associated with the previously deployed portlet and press **Next**.

   `http://localhost:7101/ParacasPortlet-ViewController-context-root/portlets/wsrp2?WSDL`

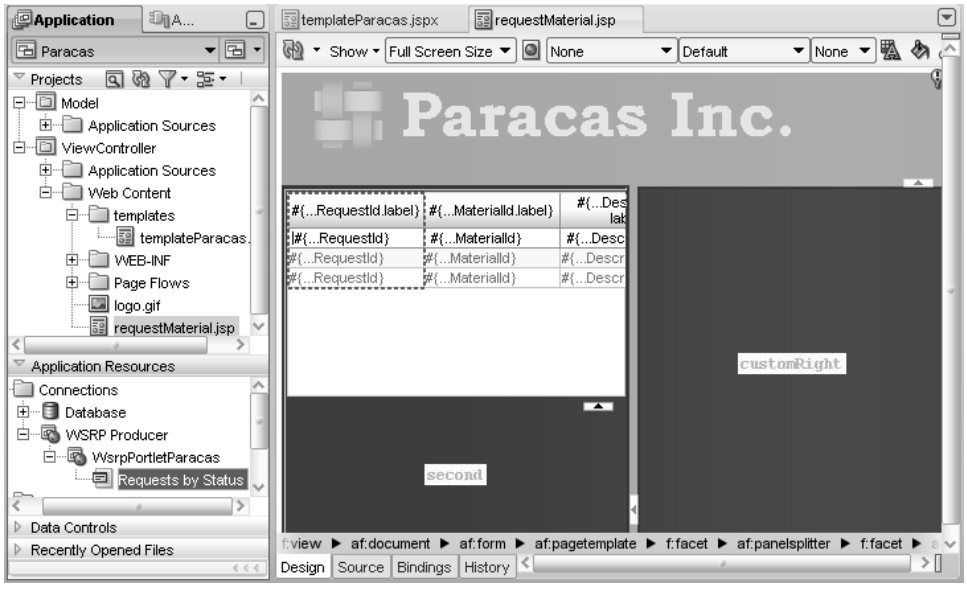


5. Now, click the **Finish** button to complete the establishment of the connection to the portlet.

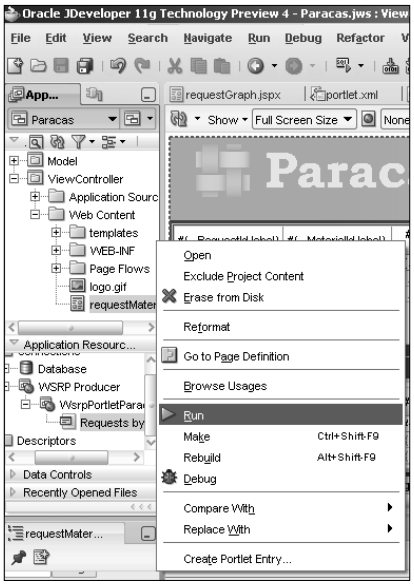
6. Let's go back to the Application Navigator and double-click on the page **requestMaterial** to show it in the editor.



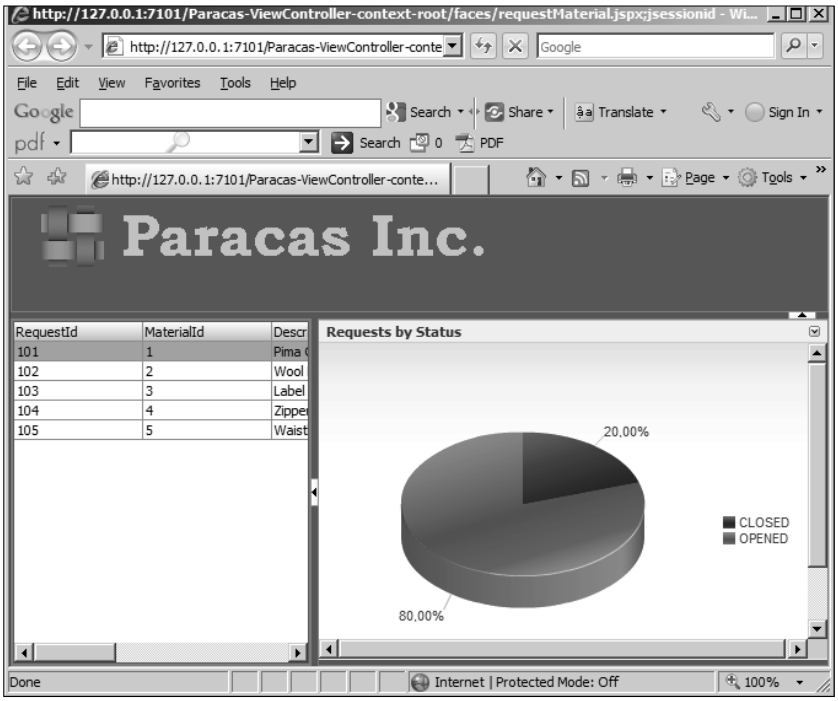
7. Subsequently, drag the Portlet reference **Requests by Status** to the **customRight** region in the page, as shown next:



8. Right-click on the page and select **Run** in the Application Navigator.



9. Finally, the page will show the following result:



Summary

This chapter has shown us the power of portlets, a mechanism that can be reused between pages. In this case, we have taken advantage of the JSF and wizards offered by JDeveloper to quickly build graphical components.

In this chapter, we discovered the features offered by portlets as a technique of component reuse. Also, we applied this technology to build a portlet using JDeveloper. Next, we hosted the said component in a repository offered by Oracle WebCenter. Finally, we consumed this portlet through our Paracas portal, which we built earlier in this book.