

CHAPTER 2

An Introduction to Oracle JDeveloper and Oracle ADF

10 Quick Start Guide to Oracle Fusion Development



As outlined in the previous chapter, Oracle chose Java EE, SOA, and Web 2.0 as the technology pillars for the Fusion initiative. Each of these pillars includes a veritable alphabet soup of technologies and standards: JPA, EJB, JDBC, BPEL, DOM, SOAP, WSDL, XML, and HTML, to name only a few. And herein lies the fundamental problem: to use these technologies, developers feel they have to pretty much know them inside out, which, arguably, is beyond the abilities of all but the most elite developers.

To many developers who are used to a declarative or fourth-generation programming language (4GL) approach to development, including Oracle E-Business Suite developers, the Fusion initiative might seem like a step back into the Dark Ages. Rather than focusing development effort on writing code to solve a business problem, the developer was now going to be confronted with a Pandora's box of technologies that had to be understood, integrated, and maintained. Even using external solutions from third parties only added to the integration headache by introducing a mishmash of code and licenses.

So, how is Oracle able to harness the power of these technologies in a way that abstracts away the complexities of the technologies, enabling developers to focus on the application-specific challenges? The answer: Oracle JDeveloper and Oracle Application Development Framework (ADF).

Oracle JDeveloper

Whether you are writing a computer program, fixing a leaking radiator, or changing the spark plugs in your car, having the right tool makes the job so much easier. Oracle JDeveloper is Oracle's tool for Fusion development. JDeveloper provides an integrated development environment (IDE) that aids the developer in the full life cycle of building Fusion applications right across the various technology pillars.

J Is for Java

As you might have expected, the *J* in JDeveloper stands for Java. Typically, Java development involves the creation and management of hundreds of files, and JDeveloper provides an environment that aids the developer in writing, building, and running Java programs. In addition to code-editing features like formatting and refactoring, JDeveloper provides other, more advanced features such as code insight, syntax checking, code templates, file comparison, pop-up documentation, debuggers, code profilers, and a host of other features to support the Java coder.

But JDeveloper Is Not Only for Java

On the other hand, JDeveloper and Fusion development is not all about Java. You might be just as likely to build a Web 2.0 user interface or hook together some web service calls; your development environment is still JDeveloper.

For Web 2.0, JDeveloper provides page designers, simplified property inspectors, and palettes of UI components to take the building of UI pages into the realms of drag-and-drop and WYSIWYG. Development of SOA services is driven through wizards and visual designers, so you don't have to manually write configuration files one line at a time. In each of these cases, JDeveloper provides tooling that aids in the generation and manipulation of the source files of that technology in a much

Chapter 2: An Introduction to Oracle JDeveloper and Oracle ADF 11

more productive way than having to roll up your sleeves and hand edit the source directly (although you can if you feel comfortable doing so).

However, the real power of JDeveloper comes when exploiting the power of a framework that protects the developer from the complexities of all the underlying technologies and provides common building blocks and features to both simplify and accelerate development.

Oracle ADF

As outlined in the preceding section, the complexities of the technologies underpinning Fusion presented a considerable challenge to developer productivity; thus, a framework of common services and features was required. Oracle ADF was built specifically to address this requirement by bringing together a number of subframeworks and standards to provide an end-to-end, metadata-driven framework for building online transactional processing (OLTP) business applications. While Oracle ADF and JDeveloper are two separate products that can be used independently, their full power is realized by using them together. Oracle ADF provides the building blocks and runtime features that are easily created and managed through JDeveloper.

But before embarking on a more detailed description of Oracle ADF, it's worth establishing, what is a framework?

The Role of a Framework

For many developers, it may be a new term in their vocabulary, even though they might have been using a framework without knowing it. Essentially, the role of a framework is to provide a flexible abstraction of common, but generic, features over a more complex underlying layer.

Consider this example; if you are planning a new kitchen for your house, you might decide that you are going to handcraft each cabinet and countertop to your precise requirements. That's fine if you are a skilled carpenter with the tools, knowledge, and time necessary to accomplish the task. Alternatively, with the framework approach, you could go to a kitchen supply company that sells cabinet carcasses and countertops that you cut to size and then customize with your own door fronts, handles, and surface materials. You are still building your kitchen to your specific requirements, but with a lot less blood, sweat, and tears by using a generic framework for cabinets and countertops that you fit to your needs.

Now consider a typical Oracle example of building an application based on some database tables. Anyone building this kind of application will be solving the same old problems of having to connect to the database, retrieve and cache records, manage database transactions, implement validation, and display results on the end user's screen. Doesn't it make sense to have all those features prebuilt for you, so that you just have to "cut them to size" to fit your particular need? That's the beauty of a framework: it provides the lower-level features and common building blocks that every application needs, but in a way that can be customized for a specific application. This allows the developer to focus on solving the business problems, and leave the technology problems to the very clever people who build the framework.

Model-View-Controller

Oracle ADF is based around the architectural principle of Model-View-Controller (MVC). What that means is that the framework separates the implementation of the business services (Model) from the implementation of the user interface (View), with the Controller managing application flow.

12 Quick Start Guide to Oracle Fusion Development

Having distinct and separate layers has a number of advantages, such as promoting reuse and testing independence.

Model

The role of the Model is to implement the business service or data model. In most cases (and the primary use case for this book) a business service will involve the querying and manipulation of data from a relational database. This requires the Model layer to provide a number of functions.

Object-Relational Mapping Given that a Fusion application is built in an object-oriented Java environment and that the data is sitting in a relational database, a mapping must occur between these two different worlds. Often called O/R mapping, the Model is responsible for mapping a Customer Java object that lives in the application to the underlying Customer table in the relational database.

Default Operations With a business service based on a database table, the chances are you will want to perform actions on that data, such as query, delete, insert, next record, previous record, commit, and rollback actions. It is therefore reasonable to expect that the Model will provide common functions to manipulate the data represented in the business service.

Validation For a business service, you probably want to validate the data to ensure that it meets specific business and data requirements. For example, you need to enforce business rules such as not allowing a customer record to be created without first specifying the customer name, or not allowing a customer's credit limit to be more than \$4000.

View

The role of the View is to expose the business service to the end user, typically through a browser. Some of the high-level features of the View layer are described next.

UI Component Set The principal goal of the View is to provide a set of rich UI components. These components range from the familiar buttons, check boxes, and text fields, to richer components like tree hierarchies and data tables, and also layout components to help position content on the page.

Adaptive Rendering A UI component should be able to render itself correctly regardless of the target device. For example, it should render correctly whether it's on one of several different browsers or on a mobile device like a cell phone.

Programming Model The View should be responsible for providing infrastructure for allowing UI-specific code to be associated with a page.

Look and Feel The capability to define corporate templates and application skins is a core feature of the View. For example, you can build an application for different customers that enables each customer to choose their own look and feel.

Controller

The final building block of MVC is the Controller. The Controller is responsible for event handling and determining page navigation. Some broad requirements of the Controller are described next.

Chapter 2: An Introduction to Oracle JDeveloper and Oracle ADF **13**

Task Flow For an application that contains many pages, you would want to be able to define “*if I am on this page, then when this action happens, I want to navigate to this new page.*” The Controller allows you to define this application page flow. However, you may also want to implement a case such as “*if I am on this page, then when this action happens, I want to perform some particular task first, and then navigate to the new page.*” This extended use case, called task flow, goes beyond page flow to include the calling of business actions.

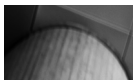
Reuse Given that a typical application could involve hundreds of pages, the Controller should allow subsets of application flow to be built and then reused within the main application flow whenever needed. For example, a login task flow might involve a specific flow of pages that can be reused throughout the application.

Metadata-Driven Framework

A common misconception of Oracle ADF is that the functionality is implemented by thousands of lines of Java code automatically generated as a result of stepping through the wizards in JDeveloper. This is definitely not the case. It is true that Oracle ADF itself is implemented in Java; however, the way the framework is configured for a specific application’s use case is largely driven by metadata. That means that an application-specific framework use case, such as defining that the application should navigate from page A to page B, is driven by metadata defined in XML, not Java code. At runtime, the generic framework classes read the XML to implement the specific navigation from page A to page B.

Driving the Framework

Of course, you might think that XML is just as difficult to maintain as machine-generated Java code; and this is where JDeveloper exploits the full power of Oracle ADF. JDeveloper allows the manipulation of the metadata in a much more intuitive and simplified way. In the case of page navigation, the developer is presented with a page flow visual editor that shows the pages of the applications and how they flow together. As a developer, you are working with a visual representation that closely matches the job at hand. The fact that changing the visual page flow is actually changing an XML file is, generally speaking, something you don’t see or need to see.



NOTE

While JDeveloper presents the editing of the framework metadata through visual diagrams, dialog boxes, and property inspectors, there is nothing to stop you from directly editing the source files as well. Whether you’re editing the XML file directly or through one of the more declarative means, the same underlying file is being updated.

Metadata Advantages

There are a number of advantages to this metadata-driven approach to the framework beyond the fact that you do not have to maintain masses of generated Java code. First, it helps you to keep a clean separation between the code you write to address an application-specific business problem and the implementation of a framework feature. The metadata implements the directives for driving the framework features, while your application-specific code can reside in its own separate Java classes.

14 Quick Start Guide to Oracle Fusion Development

The second advantage is found at runtime. Because the metadata is read at runtime, and is not compiled at design time, you could conceivably swap in and out different metadata for different deployments without having to recompile all your Java code. Why would you want to do that? Well, you might have one set of framework validation rules that is used for some customer deployments but a different set of validation rules that is used for other customers. If these validation rules were in code, this swapping in and out would be more difficult to achieve.



NOTE

The management of metadata to allow the customization and personalization of an application is handled by a framework feature called Metadata Services (MDS). Although MDS is beyond the scope of this book, it is worth pointing out that this feature exists and is a core element of the Fusion stack for applying customizations to Fusion applications.

The Building Blocks of Oracle ADF

Now that you understand the role of a framework and the high-level architecture of Oracle ADF, the next step is to look at the specific building blocks of Oracle ADF.

If you have had the opportunity to use JDeveloper already, you may have noticed the splash screen “Productivity with choice.” This mantra refers directly to Oracle ADF. Oracle ADF offers a number of different solutions for each of the layers in the MVC architecture. As you can see in Figure 2-1, the View layer offers a choice of technologies for a desktop UI or a browser-based UI. So, how do you choose which technology to use? The choice of technology stripe through Oracle ADF could be a chapter in itself, but given that the purpose of this book is a quick start guide to Fusion, we’ll focus on the technology stripe used to build the Oracle Fusion Applications. This technology stripe is highlighted in Figure 2-1.

ADF Business Components

ADF Business Components (ADFbc) is the framework used in Oracle Fusion Applications for building database-centric business services. It provides a highly declarative way of developing Java objects on top of relational database tables. In addition to providing the core feature of the Model layer, described earlier, it provides value-added features such as the ability to declare calculated attributes, lists of values, query by example, and predefined filter criteria.

So, if you want to build an application based on the Customers and Orders tables in the database, ADF Business Components allows you to rapidly build a business service based on that data model. It will automatically implement database constraints for any master/detail relationships, and provide methods for manipulating the data model such as commit, rollback, and delete.

ADF Model

The ADF Model (ADFm) is the “glue” between the business services and the View layer. With an architecture like MVC, there has to be a way of binding a UI widget to an underlying piece of data. In an ideal world, the assembly of a web page shouldn’t really require the UI developer to understand whether a Customer record is coming from a database table or a web service, and so it’s the responsibility of ADFm to abstract the implementation of the business service from the View layer. You can think of ADF Model as a façade that exposes the attributes and actions of a business service to the View layer in a consistent way.

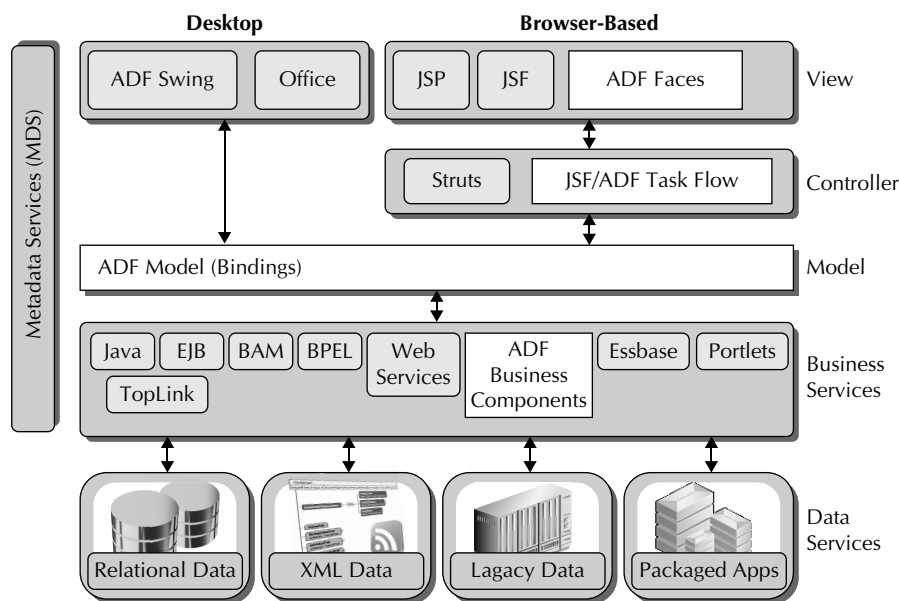


FIGURE 2-1. *The Oracle ADF technologies and Fusion technology stripe through Oracle ADF*

ADF Model is currently under proposal to become a Java standard and is subject to Java Specification Request 227 (JSR-227).

ADF Faces

ADF Faces (full title: Oracle ADF Faces Rich Client) is the ADF View (ADFv) technology for building a rich browser-based interface. ADF Faces is a set of components and features written to the Java Server Faces (JSF) standard for component-based web UIs.

In the early days of web applications, development of the front end was very much a code-focused effort. Developers had to write code, called servlets, that would dynamically generate the HTML displayed by the browser. The next evolution in web frameworks was the introduction of tag libraries, the idea being that the developer had a library of tags for rendering content, like data tables, and each tag referenced a servlet that would in turn generate the dynamic markup for that component. However, the developer building the UI still had the same problem in that designing the layout of a page with tags or servlets bore little resemblance to how the page would actually look at runtime.

With JSF, the web UI developer is able to work with WYSIWYG UI components. JSF defines a standard for a componentized framework, and vendors build their own library of UI components based on that standard. Each component exposes properties such as background color, font, height, and width. Components can also have defined behavior; for example, a button can be clicked or a text field can gain focus. At runtime, it is the responsibility of the component to generate the correct markup to be displayed in the end-user device.

There are two major benefits of this componentized view of UI widgets. First, the UI developer is working with a set of components that represent how they will be seen by the end

16 Quick Start Guide to Oracle Fusion Development

user at runtime. Second, the UI developer isn't burdened by having to understand how that component will render itself; that is the responsibility of the developer who built the component.



NOTE

For the purposes of this book, you can expect that components generate HTML and JavaScript as the markup that will be displayed by a browser. However, one of the powerful features of JSF is that a component can be configured to generate different markup depending on the device on which it will be rendered; for example, if you are accessing the application from a cell phone, the component would generate different markup specific to that device. A component might even generate different markup depending on whether the browser is Internet Explorer or Firefox.

ADF Controller

The JSF standard includes a specification for a controller that handles the flow of application pages. However, for Fusion Applications, it was evident that this specification had its limitations and thus a solution was required that went beyond the JSF specification. ADF Controller (ADF_C) was developed as a new feature in Oracle ADF 11g and is an extension of the standard JSF controller that includes features to allow application flow to be modularized and easily reused. It also extends the concept of application flow beyond just pages to also include features like calling code as part of the flow.

Summary

In this chapter you have learned that:

- Oracle JDeveloper is the development environment for building Fusion applications across different technology pillars.
- Frameworks provide common features and building blocks that aid application development.
- Oracle ADF is based on an MVC architecture.
- The main building blocks of Oracle ADF for Fusion development are ADF Business Components, ADF Model, ADF Controller, and ADF Faces.

So, armed with a solid understanding of the Fusion message, tools, and the technologies behind Oracle ADF, the next step is to get hands on with JDeveloper and get comfortable with the key features you need for building a Fusion application.