An Introduction to ADF Best Practices

An Oracle White Paper
July 2006

An Introduction to ADF Best Practices

INTRODUCTION

Oracle is leading the way in opening up the enterprise Java platform to both the 4GL and Java development audience through the productive and declarative environment of Oracle JDeveloper and Oracle ADF (Application Development Framework). Declarative business logic, visual page flow modeling and WYSIWYG user interface development free the developer from the chore and drudgery of the low level "application plumbing" and allow valuable time and resources to be focused on the specifics of the application.

Yet, with any development project, careful consideration must be given to the best way to build applications in the development environment in which you are working. This paper introduces some of the best practices for building applications with JDeveloper and Oracle ADF; allowing you to learn from the experiences gained from Oracle's own Fusion development teams.

APPLICATION DEVELOPMENT BEST PRACTICES

Further reading on ADF for 4GL developers can be found at http://download.oracle.com/docs/pdf/B25947_01.pdf

Any paper on application development best practices could potentially be a lengthy and in-depth checklist of best practices and potential pitfalls; all of which depend on the specifics of the application. Instead, this paper gives an introduction to some of the areas where you should consider the impact of your decisions, and presents options you may wish to consider in order to realize the best chance of success.

Technology choices

Given you are reading an Oracle white paper on JDeveloper and ADF; we will make the assumption that the decision to use JDeveloper and ADF has already been made for you. However, with the mantra of "*Productivity with choice*" there are still a number of decisions to be made before a single line of code can be written.

Business Services technology choices

In some ways choosing the Business Service provider is fairly simple. For anything but the simplest of cases, the jobs of data retrieval and caching are just too complex to re-invent. It is therefore difficult to justify not using a framework of some sort to help you. The business services choice within ADF represents literally hundreds of man-years effort, proven through countless successful application deployments.

There are three factors that will help you to decide which business service to choose.

What is your background and experience?

What is the background and development experience of your team? If they come from a relational database or 4GL tools background (like Oracle Forms or PeopleTools) and are used to developing from the database up then ADF Business Components (ADF BC) makes the most sense.

There is also the option of EJB 3.0 and JPA (Java Persistance API) for performing query, update, insert and delete operations on lightweight objects but that is currently outside the scope of this paper,

For a development team coming from more traditional object oriented world, or one where you have much less control over the database schema, and possibly develop from the object layer back to the database then TopLink is a much more comfortable match.

How do you define your service?

The next question to ask is how do you prefer to define your service – if you want to do as much as possible declaratively through meta-data then again ADF Business Components fits the bill. Amongst other features, ADF Business Components provides rich declarative validation features and uses familiar SQL statements to define data, “where” and “order by” clauses

Using the databinding framework of the ADF Model you can achieve more declarative validation with Toplink.

TopLink, on the other hand, demands more coding, although a proportion can be done using declarative mapping, expression builder, SQL or EJB QL. However you won’t get any of the UI hints or declarative validation that ADF Business Components gives you out of the box.

What is your deployment model?

The third consideration is how you will deploy your application. Be wary of any pre-EJB3 distributed entity beans for reasons of complexity and performance. If you require this kind of distributed architecture then both ADF Business Components and TopLink can be deployed as n-tier solution session beans with the UI generation layer being separate from the business tier.

Summary

So the choice of business services technology can be made, by and large, based on the criteria above. ADF Business Components provides the largest layer of abstraction and is probably most suitable for “traditional” Oracle developers. For those who consider themselves "pure" Java developers it is worth noting that EJB 3 very much follows the current TopLink pattern and so can be thought of as a standardization of that.

Finally, it is worth noting some of the business service technologies to avoid.

Rather than using web services for "day to day" business services they become valuable objects for integrating platform independent, loosely coupled pieces of business functionality.

Generally speaking it is best to avoid web services. This is not to say that they are not useable within a business service, but having a web service as the primary form of access is probably not going to perform as well as you would like. More typically you would expose your business services directly to your UI and also expose separately as web service, if required.

You should probably also avoid using raw JDBC calls – it is more programming intensive and you end up having to implement your own caching, pooling and error handling – which the above frameworks already provide for you.

User Interface technology choices

When it comes to making a choice of user interface technology the choice can be broadly categorized as:

- Web
- Desktop

For a desktop UI the choice is simply ADF Swing. For a web UI the choices are principally Struts/Java Server Pages (JSP) and Java Server Faces (JSF). While Struts/JSP are widespread and successfully adopted technologies, JSF is Oracle's recommendation for new web applications.

Based on JEE standards, the componentized approach of JSF offers a much simpler programming and event model, and in effect, "future proofs" the application developer by encapsulating component implementation details.

In addition, Oracle's library of JSF components, ADF Faces, is the UI technology for Oracle's Fusion technology and is leading the way in providing Rich Internet Applications (RIA).

General Best Practice Principles

Probably the first, and arguably the most obvious, best practice to adopt is the organization of your application.

Application Templates

The application is the top-level container for you application source code. To help you, JDeveloper provides a feature called Application Template (figure 1). Application and project templates can assist you in organizing your projects and standardizing on that organization. When you combine templates with technology scoping, you can streamline the way you design and develop your applications.

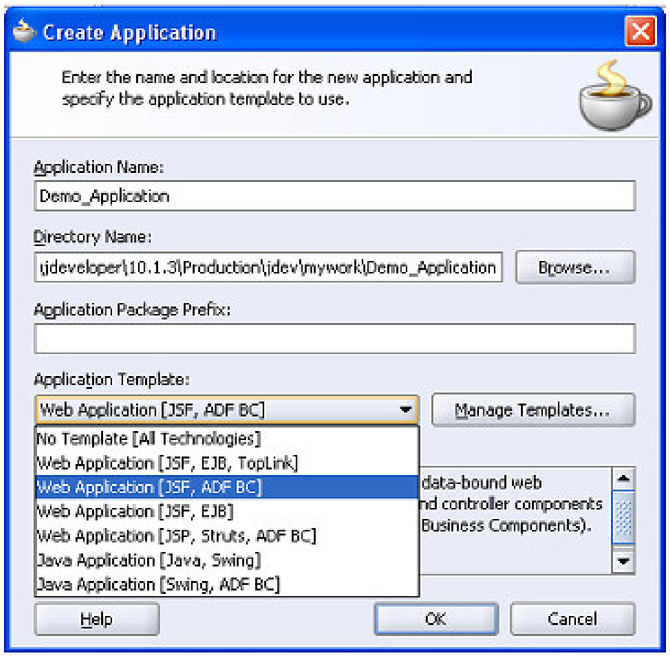

Create Application
Enter the name and location for the new application and specify the application template to use.
Application Name:
Demo_Application
Directory Name:
\jdeveloper\10.1.3\Production\jdev\mywork\Demo_Application
Browse...
Application Package Prefix:
Application Template:
Web Application [JSF, ADF BC]
Manage Templates...
No Template [All Technologies]
Web Application [JSF, EJB, TopLink]
Web Application [JSF, ADF BC]
Web Application [JSF, EJB]
Web Application [JSP, Struts, ADF BC]
Java Application [Java, Swing]
Java Application [Swing, ADF BC]
data-bound web
nd controller components
Business Components).
Help
OK
Cancel

Figure 1 – Application Templates in JDeveloper

This is a list of pre-defined (and also user definable) application technology choices. By choosing, for example, Web Application [JSF, ADF BC], JDeveloper will not only simplify the technology choices within the IDE to those chosen, but also partition the application in defined folders and use defined project names and packages. Using application templates as a best practice streamlines the technology options presented to the developer and also helps promote defined structure and naming standards.

Projects

While the default partitioning is a helpful starting point, you should not feel forced into keeping that structure. A project is a logical container for a set of files that define a JDeveloper program or portion of a program and as you add more source files to your application you should consider creating further projects to logically partition your code (figure 2).

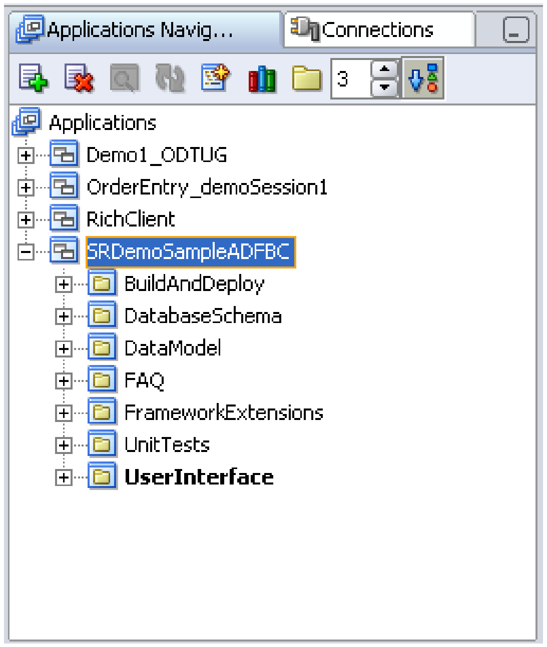


Figure 2 - Partitioning using projects

For example, the ADF BC SRDemo sample uses the Web Application [JSF, ADF BC] template but the application source has been partitioned into further projects, for example:

- DatabaseSchema for containing the database diagrams and, possibly, SQL scripts for the schema
- UnitTests for containing the unit tests for exercising the application
- FAQ is a project containing components to implement the Frequently Asked Questions functionality.

As a best practice, the use of projects helps promote application partitioning and reuse.

Packages

The Java package is the next granular level of code partitioning. It can be difficult to exactly map out the package structures in advance so don't be afraid to let the partitioning of code into packages evolve as you develop. JDeveloper provides numerous refactoring tools to allow you to refactor code between packages.

Coding Practices

JDeveloper provides a number of aids to ensure consistent coding practices. You can define the code style used in the code editor and also auto format the code to your specific style.

You can also run an audit on your code. Auditing is the static analysis of code for adherence to rules and metrics that define programming standards.

The use of these features will help promote readable and maintainable code.

Business Services

The business service is the heart of your application. Some best practices to bear in mind:

Think Service Interface

When developing your business services, regardless of the technology used, try to think "service interface". That is, only exposed the views of data and methods for manipulating that data that you need to (figure 3).

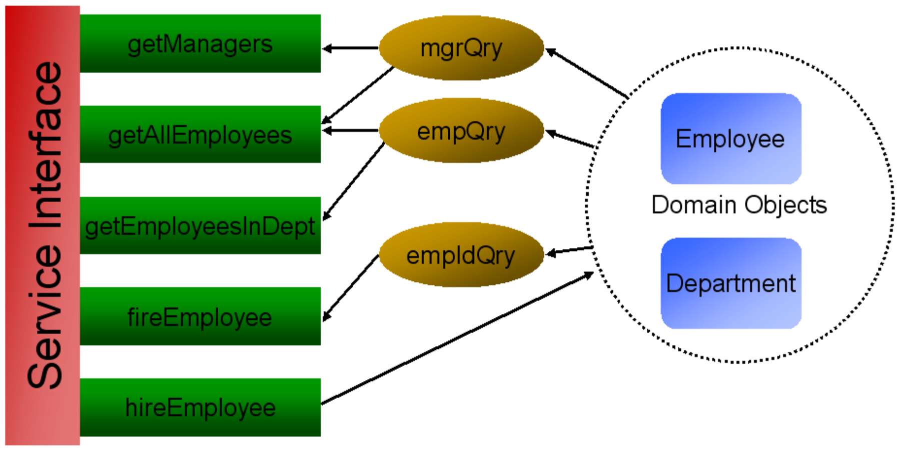


Figure 3 – Develop a service interface

Your service interface is therefore the gateway to your application – controlling the scope and transaction of your application – for consumption through a UI or web service.

All access to the business services should be made through this service interface. Where possible, avoid any "back door" coding that bypasses the service interface. If you find this is something you are doing, consider whether that functionality could be exposed through the service interface.

Business Objects vs. Access Object

Again, regardless of the technology used, consider that your business objects (like an Employee or an Order) may have many different *views* or access to that *entity* (in ADF Business Components parlance). It is less common to have a 1-1 mapping and so you would probably consider multiple views on the same entity, with each view being particular to your application needs.

Keep Entities Clean

Regardless of the technologies you use for implementing your business entities, you should try and keep the entities as clean as possible: meaning they should model closely the business object and not include extraneous information pertaining to objects or attributes outside the business object. It could be that this information is best placed in a view rather than in the entity.

User Interface and Controller

Whether you have separate teams developing the business services and user interface, or a single team, it is good practice to ensure a clean separation of the two implementations. Apart from being good modularization, a cleaner separation ensures easier reuse by different consumers. For example, some business services may be exposed through web services or by different UI implementations (e.g. the same business service accessed via a Swing desktop UI and also a mobile device).

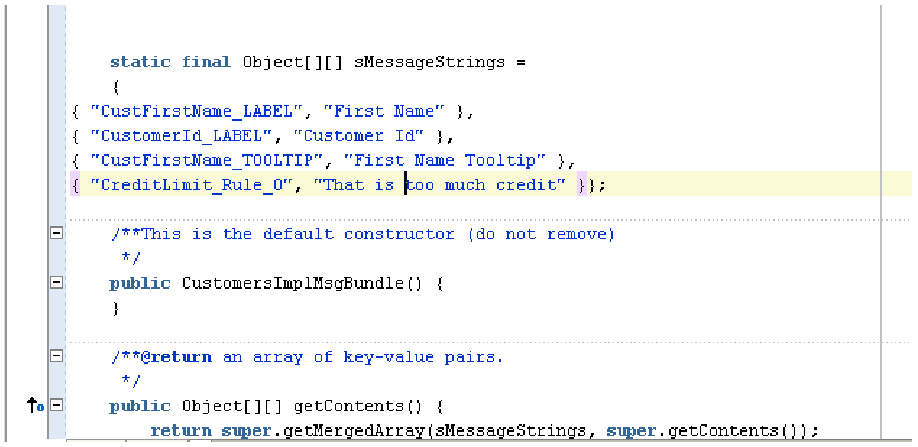
Using application templates, as outlined earlier, helps promote this best practice.

Strings

Your application will no doubt contain many labels and error messages. While it may be simple to hard code these values, it means that maintenance, reuse and internationalization of your application becomes much more difficult.

Resource bundles

Resource bundles are a collection of messages/strings that can be referenced by a token or index. The advantages being you reference a particular string in your application independent to the string itself. ADF Business Components automatically promotes this best practice by using resource bundles for error messages, labels and tooltips in the business services.



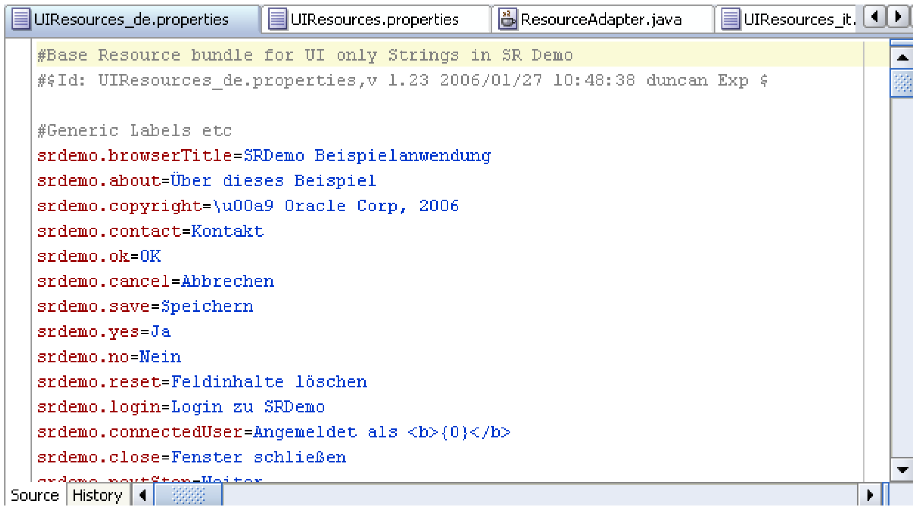
```
    static final Object[][] sMessageStrings =
    {
{ "CustFirstName_LABEL", "First Name" },
{ "CustomerId_LABEL", "Customer Id" },
{ "CustFirstName_TOOLTIP", "First Name Tooltip" },
{ "CreditLimit_Rule_0", "That is too much credit" }};

    /**This is the default constructor (do not remove)
     */
    public CustomersImplMsgBundle() {
    }

    /**@return an array of key-value pairs.
     */
    public Object[][] getContents() {
        return super.getMergedArray(sMessageStrings, super.getContents());
```

Figure 4 – Resource Bundles

Figure 4 shows a resource bundle used by ADF Business Components to populate attribute strings.

UI Properties Files

A similar concept to the resource bundle is the properties file. This is simply a file of tokens and matching strings that can be read, or bound to.



```
#Base Resource bundle for UI only Strings in SR Demo
#$Id: UIResources_de.properties,v 1.23 2006/01/27 10:48:38 duncan Exp $

#Generic Labels etc
srdemo.browserTitle=SRDemo Beispielanwendung
srdemo.about=Über dieses Beispiel
srdemo.copyright=\u00a9 Oracle Corp, 2006
srdemo.contact=Kontakt
srdemo.ok=OK
srdemo.cancel=Abbrechen
srdemo.save=Speichern
srdemo.yes=Ja
srdemo.no=Nein
srdemo.reset=Feldinhalte löschen
srdemo.login=Login zu SRDemo
srdemo.connectedUser=Angemeldet als <b>{0}</b>
srdemo.close=Fenster schließen
```

Figure 5 – Resource Properties

Figure 5 shows an example of a resource properties file in German. This illustrates the benefit of message bundles and resource properties files: that the same token can point to different translations of the same string and , depending on the NLS settings, will display the correct string for the correct language. Note the use of the Unicode escape sequence \u00a9 to represent the copyright symbol © in the `srdemo.copyright` resource. Symbols and characters outside of the normal ASCII set should be represented as Unicode escape sequences in this way rather than as HTML markup values (e.g. &copy;) or by direct encoding in the properties file with a specific font which way not be portable.

As a closing point on strings, there is a decision on whether strings should be defined in the model or the user interface. As is often the case with this type of question, there is no hard and fast rule. The fact that ADF Business Components lets you define strings at the business service level, you may feel encouraged to define your strings and error messages here. That decision could be strengthened by the fact that the error message strings would be defined close to the point from where they are called. In addition, any UI implemented on top of the business service would automatically be exposed to these business service strings.

However, you may feel more comfortable having all strings, since they will be displayed to the user, defined in the user interface layer.

Both arguments are equally valid and in reality you will probably end up using a mix of the two.

Team Development

It is expected that your development teams will use some form of source control to manage the team development and that you have experienced and knowledgeable administrators. JDeveloper provides integration points with many of the popular source control solutions such as CVS.

Committing changes

When committing any changes back to you source repository there are a number of simple rules to bear in mind. Merging differences in XML files can prove more challenging the greater the differences between the files: therefore, commit little and often.

When you commit your changes use the comment features to document the changes and it is generally good practice for each developer have their own identity.

When these changes are committed it may prove useful to communicate the changes to the development team. As such, you may consider using RSS feed or email to communicate this information.

Finally, be careful of automatically committing changes; it could be that you are committing files that you are not aware changed and do not want to be updated: for example JDeveloper may refresh a project or configuration file.

Key file owners

When developing an application with ADF you may find a number of key files central to the development of your application: for example the `faces-config.xml`, `struts-config.xml` or the application module `Impl` file. Because these files are central to the application and the impact of changes is far reaching, you may consider nominating an owner of the file.

Check-out files

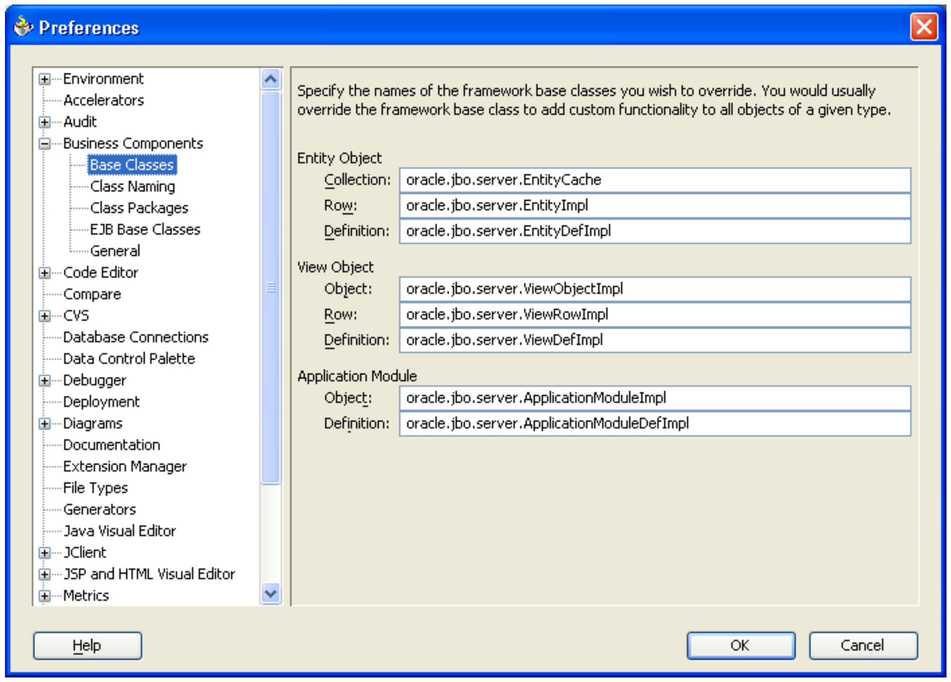
It is good practice to regularly do a clean check-out from the repository into a new directory. Continually updating your working copy may hide problems such as incomplete commits.

ADF Business Components

Assuming you have chosen ADF Business Components to implement your business services, there are a number of simple steps that can ease your development experience.

General Principles

When creating business component objects, instead of using the default base class (as shown in figure 6), you can define that JDeveloper uses your own base class (which extends the default base class).



Specify the names of the framework base classes you wish to override. You would usually override the framework base class to add custom functionality to all objects of a given type.

Entity Object

Collection:	oracle.jbo.server.EntityCache
Row:	oracle.jbo.server.EntityImpl
Definition:	oracle.jbo.server.EntityDefImpl

View Object

Object:	oracle.jbo.server.ViewObjectImpl
Row:	oracle.jbo.server.ViewRowImpl
Definition:	oracle.jbo.server.ViewDefImpl

Application Module

Object:	oracle.jbo.server.ApplicationModuleImpl
Definition:	oracle.jbo.server.ApplicationModuleDefImpl

Figure 6 – Override the default base classes

By doing this you are providing a layer of abstraction on top of the default base class meaning that if the default behavior of the base class does not suit your needs (or changes) then the behavior can be implemented in your own base class.

You should also ensure that you adopt a naming convention for your classes and packages and ensure you partition the code appropriately into the packages. As already mentioned previously, use the refactoring features of JDeveloper to repartition your code as necessary.

View Objects

ADF Business Components view objects represent a specific view, or shape, of the application data. As such, when designing view objects think in terms of the screen designs or lists of values where you want the data used.

One to many

As you build your view object you will be aware that there is rarely a one-to-one mapping of entity and view objects. So don't be afraid to use the view objects to pull information from various entities (e.g. lookups) – ADF Business Components takes care of the "hard work" in implementing functionality like joins.

Read only

You should also consider using the read-only view objects for reference data (e.g. static lists of data or data changes that are not persisted to the database). This may help performance in your application as read-only view objects bypass the entity cache.

Define at design time

You should also try to define your view objects at design time rather than runtime. View objects created a runtime will incur an overhead in performance as the various structures are created and the query is processed each time, since it is regarded as a dynamic query. You also lose much of the simplified tuning features available through the IDE. In most cases you can define your view objects at design time and "shape" at runtime using bind variables and the runtime API.

Parameterizing

Realistically you will probably want to filter the data displayed to the user depending on a number of factors. For this kind of behavior, consider using Oracle style named bind variables in the view object instances:

```
Where ID= :custId and Name like :CustName
```

As well the performance benefits of not having to reparse the SQL statement, ADF Business Components allows you to define control hints for the bind variables, test them through the ADF Business Components application module tester, and also allows the use of *executeWithParams*. This appears in the data control palette as an operation allowing each bind parameter to be dragged onto a page and to act like a "search form".

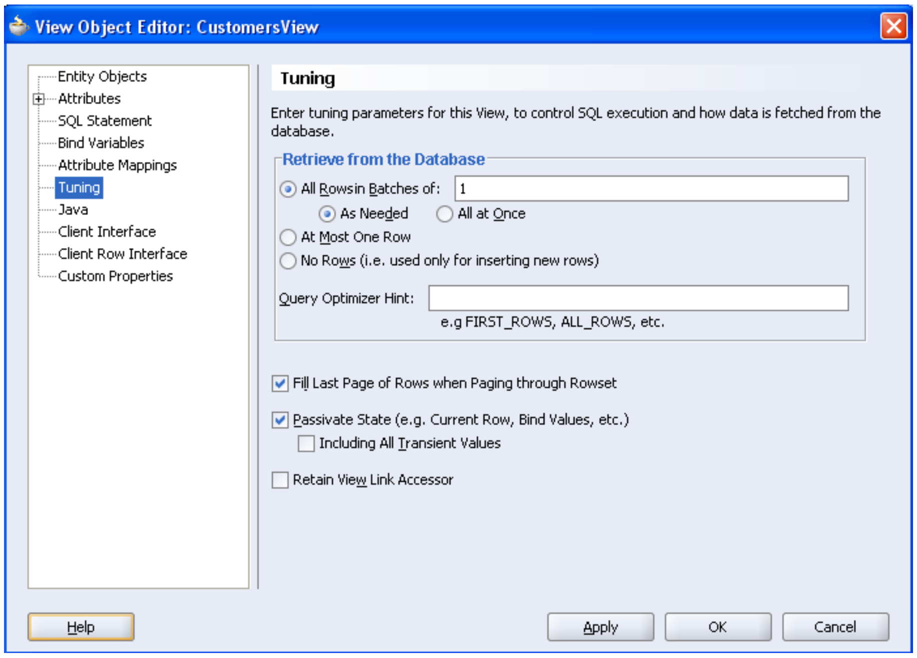
Performance

The subject of performance is a paper all in itself; however, there are a number of basic points to be aware of with respect to ADF Business Components, and specifically view objects

As shown in figure 7, for each view object, you can define how many, and what manner, rows will be fetched from the database. You can also define optimizer hints and for the SQL query you can display the query plan the database will use when executing the query.

Further reading on view object performance can be found at http://www.oracle.com/technology/oramag/oracle/06-may/o36frame.html http://www.oracle.com/technology/products/jdev/tips/muench/voperftips/index.html

Of course, there is no hard on fast rule on what the settings should be for each view object; that is of course specific to the characteristics of the view object. However, be aware that if you only view a small set of data through the view object, or whether you expect the user to frequently page through the full range of data, the performance of the application may be altered using the settings on this page.


View Object Editor: CustomersView

- Entity Objects
- Attributes
- SQL Statement
- Bind Variables
- Attribute Mappings
- Tuning
- Java
- Client Interface
- Client Row Interface
- Custom Properties

Tuning

Enter tuning parameters for this View, to control SQL execution and how data is fetched from the database.

Retrieve from the Database

[✓ All Rowsin Batches of:] 1
[✓ As Needed] All at Once
At Most One Row
No Rows (i.e. used only for inserting new rows)

Query Optimizer Hint:
e.g FIRST_ROWS, ALL_ROWS, etc.

[✓ Fill Last Page of Rows when Paging through Rowset]
[✓ Passivate State (e.g. Current Row, Bind Values, etc.)]
Including All Transient Values
Retain View Link Accessor

. *Figure 7 – View Object tuning*

Application Modules

The application module is the container of views for a particular transaction. Generally speaking you would ensure that you only include views and relationships that are required for your transaction; therefore be careful when using some of the default wizards in JDeveloper , which may include combinations of views, some of which are not required for your purposes.

Configuration

By right clicking on an application module and bringing up the context menu you can view the configuration for an application module. When developing, it is useful to set the `jbo.ampool.doampooling` parameter to false. This means that as you exit out your application while testing, re-runs of the application won't be locked by the application you just exited.

Also, for the purposes of testing, you may want to ensure that `jbo.dofailover` is set to false thus disabling application failover. This will allow the application to run without having to save failover information.

Finally, if your application uses the same database user to access application data (as many do) ensure that you are not running with dynamic JDBC credentials. Use the configuration `jbo.ampool.dynamicjdbccredentials` set to false.

Testing

ADF Business Components provides a built in tester to facilitate the exercising of the business services without having to build a user interface. The facilities provided by the tester are comprehensive including the testing of bind variables. You should therefore ensure you thoroughly test the business services outside the confines of the user interface.

There are examples of Junit test suites in both SRDemos available from otn.oracle.com/products/jdev

You may also wish to consider a testing framework such a Junit, which allows you to define a collection of regressions tests.

Data Binding

When a UI control is placed on a page to display some business service data, a binding is automatically created to facilitate the joining of the physical UI component and the data.

Superfluous bindings

Be aware that data controls are being created without your explicit instructions and you should remove any bindings that are not required. For example, if you display data as an ADF Faces table, a binding will be created for each column in the table. If you subsequently delete a column the binding may still remain. If it is not required, remove the binding.

Referencing bindings

You can use JSF Expression Language (EL) to reference the bindings on a page. As a general best practice, you should only be referencing bindings on the current page (using the `bindings` EL notation). Using the `data` EL notation you can access bindings on other pages but there is no guarantee that those bindings have yet been instantiated or have not been cleaned up by the framework. Instead, if you need to access data across different pages you should use a managed bean (as discussed later).

Java Server Faces

When it comes to building the web pages of your application using JSF, the first rule to ensure you follow is to not mix JSF and other markup like HTML. There has been some debate on the merits/issues of mixing the two, and while this may change with future version of JSF our current recommendation is not to mix JSF components with other markup such as raw HTML tables for layout. Given there is an extensive range of JSF components, many of which provide layout behavior; there should be no reason to need to mix the two.

As a broad definition, a Managed Bean is a Java class available at a given scope in the JSF world, callable via JSF Expression Langauge from any number of JSF pages or other JSF managed beans.

On the other hand, a backing bean is a specialized managed bean created and managed by JDeveloper and is tied to one specific JSF page. "Accessor" methods for each UI component can be automatically exposed in the backing bean.

Backing Beans

If a web page requires code to process information on the page, that code should be contained in a backing bean for the page. The backing bean is a one-stop-shop for the code associated with a web page and generally speaking it is good practice, if you require code to support a page, to have one backing bean per page.

JDeveloper can create the backing bean for you automatically, and in doing so can create accessors for all the control items on the web page. When do you this try to ensure a clear naming convention for the UI items and the backing bean accessors and use packages to logically partition the code. You should also get in the habit of keeping the backing bean as "clean" as possible by removing any accessor code not required for your application. It is also generally a best practice not to use the backing beans for persisting information across pages (use managed beans instead) and therefore recommended that backing beans be placed in request scope.

Managed Beans

If you require state information to be held for the user interface then this should be done through managed beans. You can then use the managed beans to access the state rather than through the session/request directly. If you access directly you lose the layer of abstraction that is hiding the implementation details of JSF (which could change).

CONCLUSIONS

Oracle JDeveloper and Oracle ADF provide a powerful and productive environment for building enterprise Java applications. These guidelines and experiences are a starting point for your own in-house developed best practice, and should be seen as the foundations for a successful development experience.

White Paper Title
July 2006
Author: Grant Ronald
Contributing Authors: Duncan Mills

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