ADF Patterns for Forms Modernizations

A NEOS and Vgo Software Whitepaper

Author:

Robert Nocera

Optimizing the Busniness-Technology Evolution

ADF Patterns for Forms Modernizations

Robert Nocera, NEOS & Vgo Software

Introduction

Through our modernization practice we have been exposed to a variety of different Oracle Forms applications, many of which have been around for decades. Many of them have been upgraded to each latest release of Oracle Forms and yet the applications are still starting to show signs of age. A number of these applications were written when the people working on them were still renting movies to watch on VCRs and though most people have by now upgraded to DVD and Blu-Ray players, companies have not been so eager to upgrade their existing Forms applications. The motto is usually “if it ain’t broke, don’t fix it,” and indeed, in a lot of cases, the sentiment makes business sense.

Many of these applications perform core business functionality and echo a business process that was in place decades ago with tweaks and changes to that process that have caused changes to the application. The applications can contain a lot of business logic implementing rules that even the users are no longer familiar with. It is no wonder that a business would rather do as little as possible to disrupt that application and process than modernize the application.

However, this philosophy can only succeed for so long. Eventually the platform that the application runs on expires, the hardware is no longer available and the system must be upgraded. This upgrade can take many forms, sometimes just performing the bare minimum to get the application supported again. Increasingly, however, the need to upgrade these applications is seen as an opportunity to move the application to a platform that is considered more “modern”.

With the advent of the web and the prevalence of web applications within businesses, users have become accustomed to certain UI standards. This familiarity with web applications makes it easier to gain user acceptance for changes in the UI design of the application that may be necessary because of the new framework, it also sometimes is the impetus to modernize an existing application because users are expecting more and more. Of course, the UI changes to the application are probably just the tip of the iceberg. The entire technological foundation of the application would be changing from Oracle Forms to a modern java-oriented ADF 11g framework.

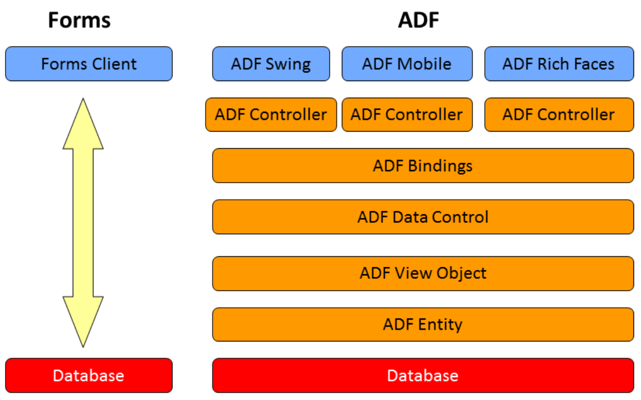


Figure 1 Forms vs. ADF High Level View

This paper will discuss some of those changes and introduce some patterns and best practices that can be used when modernizing an application from Oracle Forms to ADF 11g. Some of those patterns will be UI changes that attempt to limit the changes to the business process while making changes to the screens that make sense to the new framework. Others of those patterns and best practices will be hidden from the user completely and really just a better way to implement the same functionality in the new framework.

Basic Functionality

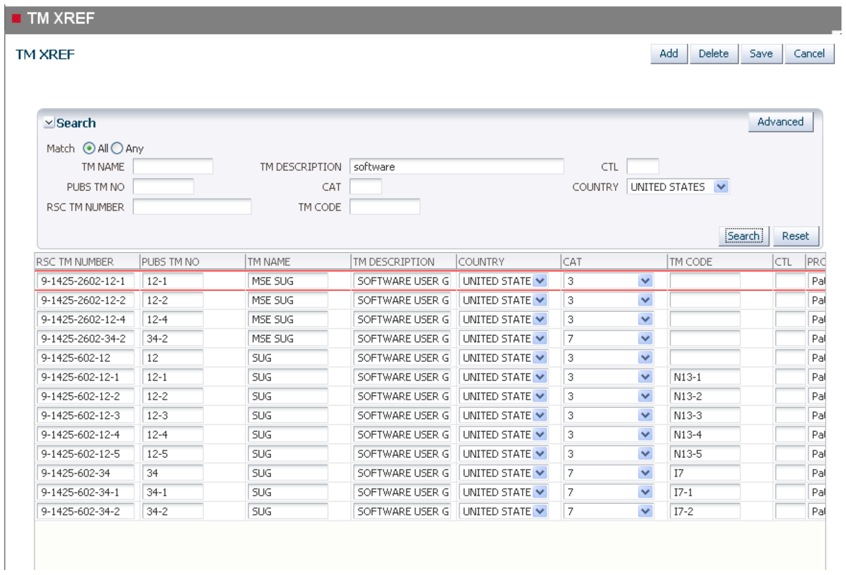
Oracle Forms have a lot of functionality built-in. The framework allows developers to quickly build applications. The design of the framework is also conducive to some patterns that are not as intuitive as they could be for the users of the applications built with it.

In a lot of cases, the same screen is used in an Oracle Forms application to perform a search and edit data. The types of layouts on these screens range from simple table-based layouts to more complex master-detail layouts. The user will normally select "Enter Query" from the menu, enter some search criteria, then select "Execute Query" from the menu to get the results of the query. Those results can then be edited from that same screen. A user can also add a new record and use the same screen to populate the data. This type of functionality can be divided into two categories; table-based, and form-based.

Table-based Search/Edit

A table-based search and edit pattern is a screen which contains a single table. The user can search on data by entering search criteria in the first row and searching on that. The results are displayed in the same table and can be edited by the user.
For this simple category of screen, typically used to maintain administration data such as lists of countries or product types, the UI pattern typically used in an ADF application is very similar to the

pattern used in the Form application. The main difference is that there is a separate component used to perform the search.



Match [✓ All] Any

TM NAME		TM DESCRIPTION	software	CTL	
PUBS TM NO		CAT		COUNTRY	UNITED STATES
RSC TM NUMBER		TM CODE			

RSC TM NUMBER	PUBS TM NO	TM NAME	TM DESCRIPTION	COUNTRY	CAT	TM CODE	CTL	PRC
9-1425-2602-12-1	12-1	MSE SUG	SOFTWARE USER G	UNITED STATE	3			Pal
9-1425-2602-12-2	12-2	MSE SUG	SOFTWARE USER G	UNITED STATE	3			Pal
9-1425-2602-12-4	12-4	MSE SUG	SOFTWARE USER G	UNITED STATE	3			Pal
9-1425-2602-34-2	34-2	MSE SUG	SOFTWARE USER G	UNITED STATE	7			Pal
9-1425-602-12	12	SUG	SOFTWARE USER G	UNITED STATE	3			Pal
9-1425-602-12-1	12-1	SUG	SOFTWARE USER G	UNITED STATE	3	N13-1		Pal
9-1425-602-12-2	12-2	SUG	SOFTWARE USER G	UNITED STATE	3	N13-2		Pal
9-1425-602-12-3	12-3	SUG	SOFTWARE USER G	UNITED STATE	3	N13-3		Pal
9-1425-602-12-4	12-4	SUG	SOFTWARE USER G	UNITED STATE	3	N13-4		Pal
9-1425-602-12-5	12-5	SUG	SOFTWARE USER G	UNITED STATE	3	N13-5		Pal
9-1425-602-34	34	SUG	SOFTWARE USER G	UNITED STATE	7	I7		Pal
9-1425-602-34-1	34-1	SUG	SOFTWARE USER G	UNITED STATE	7	I7-1		Pal
9-1425-602-34-2	34-2	SUG	SOFTWARE USER G	UNITED STATE	7	I7-2		Pal

Figure 2 ADF Table Based Search and Edit

If you are familiar with the ADF 11g framework you know that the simplest way to provide the type of CRUD capabilities in these types of forms is to create an Entity based on the table being edited. A View Object would then be created for that Entity and exposed to the user in the page. To provide the search capability to the user, the developer would create a set of View Criteria. This View Criteria is then used to create an ADF Query Panel that the user of the application will use to search the data.

That the ADF framework provides such a component is a very good thing, it can reduce the complexity of what the developer needs to do greatly. Of course, great power comes with great restrictions. The down-side is that developers do not have complete control over how the query component is rendered. Developers do have, indirectly, the ability to influence the rendering of the query component. By creating a View Criteria, ordering the source attributes in the View Objects, and modifying some of the attributes on the component itself, the query panel's display will be modified.

In order to take advantage of the benefits of the ADF Query Panel, it must be included on the page and this in itself is a difference from a typical Form application that in some respects influences the Search and Edit patterns that we use.

The simple table-based pattern keeps the UI very similar to the Form screen except for the addition of the ADF Query Panel control and buttons for adding and deleting records. One common pattern that we use is to remove the header menu that provides a lot of the functionality in a Forms application and use buttons instead.

In the simple layout, the buttons used include Add, Delete, Save and Cancel. In this case, the page is popped up from a menu, in cases where it is not, a back or return button may be necessary to allow the user to navigate away from the page.

In order to alleviate any unnecessary button clicks, the ADF Panel Query is provided fully disclosed. The user can then search and see the results on the same screen. Edits to the resulting data can be performed in the table itself and records can be deleted from the returned results. When the user wants to add a record, they simply click the add button which will insert an empty row into the table and then the user can enter data directly into that row.

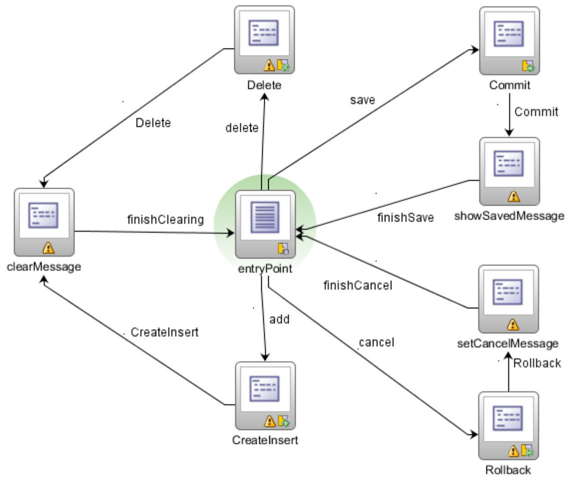


Figure 3 Task Flow for Table-based Search and Edit

The Task Flow for this type of search and edit functionality explicitly shows the messages being set and cleared to indicate to the user that an action has been successfully performed.

Users of the original application will normally react well to a change such as this. Though the UI has changed from the original form, it provides them some additional functionality that they did not have before via the ADF Query Panel. If the Oracle Metadata Repository was also used, the users could even save their favorite queries using this panel and have them available whenever they return to the screen.

The resulting screen is also fairly intuitive. It is fairly obvious to the user how to perform the functions that they need to perform on this screen and so even though it is different, it is easy for them to use.

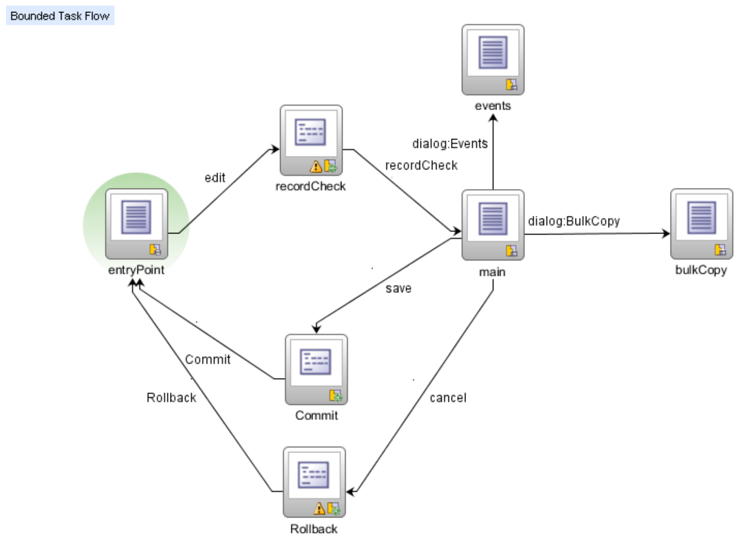


Figure 4 Example Task Flow for Separate Search and Edit

Another example of a search and edit form is the more complex search and edit screen. This is a screen where the record is displayed to the user, not in a table, but in a Form layout that allows the components to take up more real-estate on the screen. In a lot of these cases, Master-Detail blocks are displayed in the Form that allow the user to not only add, edit and delete master records but add, edit and delete detail records as well.

In these situations a pattern similar to the simple table-based search and edit is followed except that the search and edit functionality is split up into two separate screens. It would be possible to provide an ADF Query Panel at the top of the page and show the results one at a time similar to the original form, but providing a separate search page with a table for results allows the users to search and select the exact row they want to edit without having to scroll one by one through the list.

The edit page will normally contain just Save and Cancel buttons but if users want to be able to scroll through and edit the entire result set without navigating back to the list page, it is easy enough to add Previous and Next buttons as well similar to the existing Forms screen.

Tabs vs. Scrollbar

One problem that Forms developers faced was the amount of screen real-estate allowed for their application. If a Form screen had many fields and many detail tables to show, there may not have been enough screen space to show it all. A solution that is very common among many applications of that era was to provide a set of tabs to separate the data. Often different detail tables would be on different tabs.

Today, in modern web applications, users are accustomed to scrolling up and down a page to see an entire set of information. Many of these older Forms screens with lots of tabs can be handled by removing the tabs and listing all of the information on one page. This allows the user to scroll up and down and see all they want to see.

This is not to say that there is no place for tabs at all. Tabs can be very useful in partitioning the information shown to a user. In fact in at least one example we added tabs to a page that did not have them originally in order to provide more screen space for the detailed data.

LOVs vs. Dropdowns

Forms applications use a lot of Lists of Values (LOVs). Newer applications have a mix of LOVs and dropdowns. A lot of times, when an application used a list of values, after the user selected a value from the list, a description field would be populated.

ADF 11g provides an easy way to produce this same behavior but you can also just as easily display the description instead of the key when a value is selected or show the key and the description when a value is selected. The ADF 11g LOV components also provide some more enhanced search functionality.

For short lists of values, consider replacing with single choice selects, for longer lists use popup LOVs to postpone the initial query that retrieves the list until it is actually needed.

Post-Query vs. View

Post Query Triggers in Oracle Forms applications can be used for many things and depending on what function the trigger was performing in the Form, implementing the same functionality in ADF 11g can be quite similar or very different.

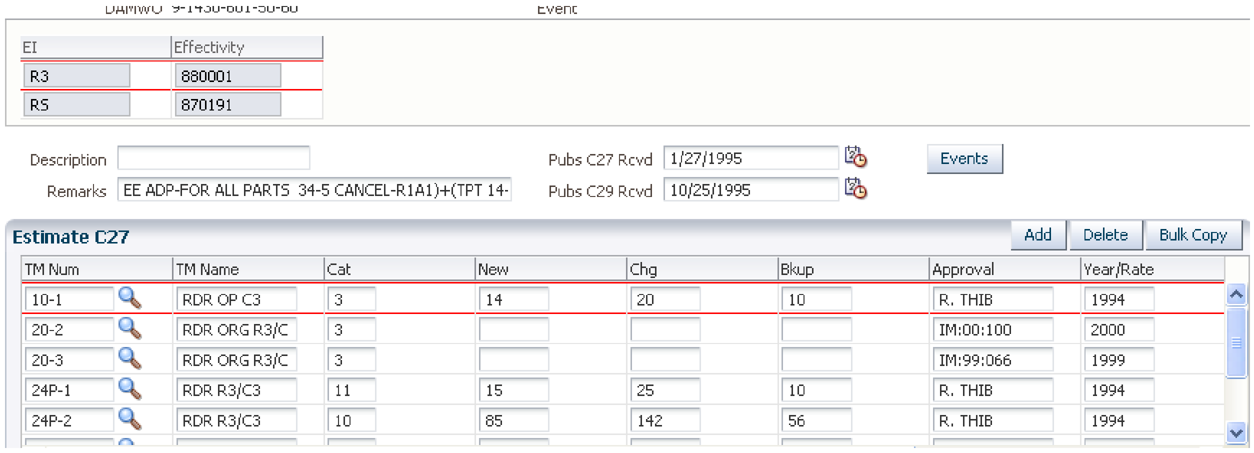
Oracle Forms allows you to make an awful lot of database calls in a very short period of time having hardly any impact at all on the performance of the application. This makes, for example, populating a description field via Post-Query Trigger an acceptable option. This option is not as acceptable for applications built in other frameworks, including ADF. Instead, in ADF, when you create a LOV and add a transient description field from another Entity, as any number of blog posts and development guides will tell you to do, ADF will create a join in the query for the View that contains the description field so that the field will be populated at the same time the rest of the data is retrieved from the database. This eliminates the need for a separate call and even a separate call per row of results returned in some cases.

This is a very common and simple example but at the same time it illuminates a point that should be recognized. The performance of your application is going to be greatly affected by how many database calls are made. Watch out for situations where you might be executing a query per row of a result set and try to reduce that access as much as possible.

Keystrokes and Menu Items vs. Buttons

In many older Forms applications the users are used to a lot of keystroke commands to execute certain functions. You can mimic some of this behavior with shortcuts on buttons but no matter what you do, some of the functionality is almost certainly going to be replaced by buttons.

Forms is also very good at keeping track of context, knowing where the cursor is in the UI and executing the right event depending on the field that has focus. For example, if there is a master record and several detail tables, clicking add will add a record to the right record depending upon what block had focus when the button was pressed.



EI	Effectivity
R3	880001
RS	870191

Description		Pubs C27 Rcvd	1/27/1995
Remarks	EE ADP-FOR ALL PARTS 34-5 CANCEL-R1A1)+(TPT 14-	Pubs C29 Rcvd	10/25/1995

TM Num	TM Name	Cat	New	Chg	Bkup	Approval	Year/Rate
10-1	RDR OP C3	3	14	20	10	R. THIB	1994
20-2	RDR ORG R3/C	3				IM:00:100	2000
20-3	RDR ORG R3/C	3				IM:99:066	1999
24P-1	RDR R3/C3	11	15	25	10	R. THIB	1994
24P-2	RDR R3/C3	10	85	142	56	R. THIB	1994

Figure 5 Use of Buttons on Detail Table

While this can be achieved in ADF 11g, it makes the application more complex, most of the time unnecessarily complex, and it also has the effect of not being intuitive for new users. Instead, put your detail tables inside a panel header group and add a toolbar with add/delete buttons to the header's toolbar facet. This puts more buttons on the page, but it makes their use very intuitive.

More Advanced Post-Query Triggers

More complex post-query triggers will require some forethought as to where that functionality belongs in an ADF application as there are many choices.

For complex functionality that is affecting multiple views it is best to create a method in the Application Module's implementation class and expose it so that it can be run declaratively in the Task Flow. As mentioned before, other Post-Query trigger events may be better implemented by creating a more complex query in the View Object itself that takes into account the logic in the Post-Query trigger.

Bounded Task Flow per “Form”

Existing Forms lend themselves well to being mapped to Bounded Task Flows in ADF. Use the “Use Existing Transaction If Possible” selection so that if you nest your task flows the transaction will be handled by the parent.

Each Form will normally encapsulate a particular process in a system. If the original application was designed correctly the mapping will be very close to 1 Form to 1 Task Flow. If the original Forms application is too complex and contains too many screens it would make sense to break it into multiple Task Flows that would then be nested inside a parent Task Flow.

Summary

Any project that is undertaken to modernize a legacy Oracle Forms application and implement it in ADF should consider that some aspects of the usability of the application will change. Most of these changes will result in an application that has a modern look and feel and the user base should be accepting of those changes.

Though the underlying architecture of the application will change completely, and though ADF is a very different platform than Oracle Forms, it is still possible to map functionality of an Oracle Forms application into functionality in an ADF application. A large part of these mappings are straightforward, such as mapping Oracle Forms’ Blocks to ADF’s View Objects. There will, however, be functionality that must be implemented in an ADF application that will not map directly from a Forms application and that functionality will be where the team working on the project will spend most of its time.

Taking time upfront to map out how an application will move from Forms to ADF and informing the user base of any potential impacts to the work they perform will greatly reduce the time spent developing the application in ADF and getting it rolled out to the user base.

About the Author

Robert Nocera is the CTO and co-founder of NEOS and Vgo Software. He is tasked with driving technology direction for the company’s modernization solutions but has also developed a focused understanding of Oracle Fusion Middleware and ADF v11 in particular. Robert has spoken at ODTUG (2008), UKOUG (2010) and a number of regional OUGs (NYOUG, NEOUG, and more). He also has spoken on ADF and ADF security at the 2009 OOW "unconference." Robert is deeply skilled in all Java related technologies/languages. He has over twenty years of IT experience and is the author of the www.java-hair.com blog.

About NEOS and Vgo Software

NEOS is a management consulting and technology services company, offering business system modernization and enterprise data services capabilities. NEOS partners with its clients to best position their business by leveraging our collective experience, capabilities and proprietary products, across all industries and business functions.

NEOS was founded in Connecticut in 2000 and has expanded its business throughout the US, into Europe, the Middle East and Japan. NEOS clients range from large, mid-cap companies to the Global 1000 with specific industry expertise in financial services and insurance. Vgo Software, founded in 2005, was nominated as one of Connecticut's "Fast Track" companies and has enjoyed positive growth since its beginnings.

Vgo Software provides modernization solutions and professional services that support legacy application strategies, assessments, conversions, and development. Vgo utilizes proven proprietary methodologies and software products including Evo, (Oracle Forms® to Java); ART™, (Application Portfolio Assessment for Oracle Forms® and PowerBuilder Applications); and Evolutions™ (Application Conversion Methodology). Vgo is one of only two certified third-party solutions conducting Oracle Forms® to Java modernization.

Vgo Software was created based on intellectual property developed through many professional services engagements conducted by its parent company, **NEOS LLC**. Our products and offerings are based on real-world experience from hands-on projects delivered to clients.

Vgo Software's product lines will continue to expand in functionality as well as grow in scale. It is our intent to continue to provide superior, thoughtful and innovative automation solutions to customers world-wide.

Contact

Please contact **NEOS and Vgo Software at +1-860-519-5601** for more information or visit us at www.neosllc.com and www.vgosoftware.com.