Forms to Fusion

A discussion of Oracle Forms positioning for the Future State

By: Ernst Renner, CEO, Vgo Software
www.vgosoftware.com

INTRODUCTION

This document reviews the current Oracle Forms landscape and options Forms users have in regard to their existing investment. This document will review when to upgrade or convert, what role ADF plays in Forms applications and what technologies will come into play in an SOA-environment. Finally, this document will review what you need to know about your current environment when you decide when and where to go with it in the future.

The term "Fusion Middleware" surfaced at Oracle Open World around 2004 or so. When it came out, there was a vague definition that it was the conglomeration of all of Oracle's technologies and everything it will buy, forever. Over the years, the term "Fusion" has become more concrete with the addition of mature (or maturing) technologies and a clear vision of how those technologies tie together. You still can't "buy Fusion", but you certainly can buy the components that make up the Fusion Middleware stack. Those technologies include SOA Suite, EDA Suite (Event Driven Architecture for complex event management), BPEL (for business process orchestration) and Business Intelligence among others.

For the remainder of this paper, we will be discussing legacy Oracle Forms applications and how they can be brought forward into a Java architecture and how that new position relates to SOA and Oracle Fusion Middleware. This will cover a broad spectrum of topics, most of which warrant their own individual whitepapers or even books, in some cases.

The point of this document is to have the reader think *beyond* their individual Forms application and consider their position within a service oriented architecture.

This document is geared toward management-level people and is not intended for deep development knowledge or tips.

FORMS FUTURE

Oracle Forms, as any technology, has been evolving and previous versions are being retired. To date the calendar can be viewed as:

- 6.0.8. *sustaining* support ended in January 2008
- 9.0.2 *sustaining* support ends in July 2008
- 9.0.4 extended support ends in December 2010
- 10gR2 extended support ends December 2011
- 10gR3 extended support ends December 2011

But a larger question emerges: *will Oracle Forms go away all together?*

This is an interesting question. According to product development, Oracle Forms will be around forever. However, with the emergence of JDeveloper and ADF, as well as the evolution of Fusion Middleware, it is also equally possible to see that JDeveloper and ADF will take the place of Oracle Forms. In fact, most organizations are developing new applications in Java or ADF and very few are developing applications with Developer Suite or Forms anymore.

Oracle, correctly, has extended support for its Forms products and this level of support is very important to its customers. The Oracle Forms development community is alive and strong, but there is a growing community of JDeveloper and ADF resources to support the "next wave" of Oracle development.

Business View & Positioning

An interesting perspective is to analyze Oracle's business strategy since 2000; heavy acquisition of complimentary solutions *and* technologies. Prior to 2000, Oracle had conducted acquisitions that would augment or defend its core technology namely based around the database. In recent years, massive acquisitions of Siebel, PeopleSoft, JD Edwards and Retek (a few among many others) have created a formidable business solution architecture that effectively competes with other global solution providers such as SAP and Microsoft.

With these acquisitions, one can see that:

A) They need to be integrated to be most effective to Oracle's customers
B) Integrating the applications will also provide opportunity to offer more solutions (i.e. greater sales opportunities)

The maturation of technologies in the broader technology realm, namely Java, web services and SOA, also provided a great platform to offer integration of solutions and positioning for greater adoption and adaptation as all of these technologies continue to evolve. With these industry moves and Oracle's business acquisitions, one can see that someone had considered Fusion as a key ingredient in continued business growth at Oracle. With Fusion as a platform for not only acquisitions, but as a product integration strategy, Oracle has done a splendid job of positioning.

ADF Glue

The "glue" of Fusion is the SOA Suite and with that, ADF and JDeveloper, at a base level. As this position has been adopted, existing Oracle Applications, written in Forms technology, have to be reconsidered. Oracle is actively moving its Applications to be "Fusion Ready". Meaning, they are moving them to ADF. When Oracle Applications move to ADF, so will its clients and also those who have modified/augmented the Application stack with custom Forms.

If Oracle's *business* strategy counts on Fusion and therefore ADF, it will inevitably require that all Oracle Forms customers eventually become ADF-compliant in nature (opinion of the author). There are a number of options for getting to this point of which only ADF conversion is discussed later in this document. If all were discussed, this would turn into a book.

Impacts

When considering adopting ADF and SOA into an organization, one must consider the learning curve of adopting new technologies. In this case, it is a significant change in thought for developers. Typically, Forms developers have developed sophisticated client/server code where re-use was somewhat limited to the use of libraries or "cut and paste" as a pattern. Code has been duplicated within Forms and within Forms applications. Some forward thinking organizations in the late 1990's abstracted their database's by creating a CRUD (create read update delete) PL/SQL layer and moving substantial (in processing cost) code from their Forms to a PL/SQL business rule layer that calls the PL/SQL CRUD layer.

However, in the world of Java, ADF and SOA, different architectural choices abound. The "leap" to these design choices can be considerable as it requires a change in mindset – instead of what's good for your Form, you have to consider what's good for your application. Re-use, consolidation, object design, process execution and transaction state all have to be reconsidered when designing enterprise-class applications. Customers should consider training and forward development using JDeveloper and ADF in the near to mid-term to get their developers to begin the transition to the web world.

From a resource perspective, it is becoming increasingly difficult to find highly-talented Oracle Forms development resources. Even off-shore staffing houses are having a hard time retaining Forms developers. In five more years, it will be extremely difficult to find people to support a decidedly "legacy" environment such as Oracle Forms.

OPTIONS – UPGRADE, EVOLVE or RE-WRITE

Considering where Oracle is going and the maturation of technologies such as ADF, existing Oracle Forms users have several options that will provide support or position strategically for increased business value. This section outlines these options and considerations in making the decision.

Option 1: Upgrade to Web Forms

If a Forms customer is in an early or de-supported version of Forms, the easiest and least risky option is simply to upgrade Forms. Oracle provides several methods for customers to upgrade Forms to newer versions. These methods will be discussed later in this document. The following Pro's and Con's can be noted for this option.

Pro's:

- ☑ Relatively inexpensive
- ☑ Safe – not very risky
- ☑ Retain applications look and feel within the constraints of the new version (i.e. moving from v3 to v10gR2 there will be definite changes in look and feel)

Con's:

- Lifecycle and support (i.e. "sustained support" for 6i and 9i has an end date; will the other releases?)
- True interoperability with other applications that are not in Forms, but new to the enterprise
- Resource availability
- Validity as a modern architecture

Upgrade When…

Based on the Pro's and Con's of this option, it is pretty clear when someone should consider upgrading. If it is required that the application maintain support from Oracle, retain it's look and feel (for the most part) and there is no budget for re-architecting the application, then it is easy for a Forms customer to make this their choice.

Option 2: Re-engineer to ADF

Conversion to ADF is a powerful alternative to upgrading Forms. The main issue is how to best accomplish this. Oracle does not provide a conversion alternative as it does provide an upgrade solution. Keys to a successful conversion are discussed below. Options for users include purchasing a re-engineering product, such as Vgo's Evo product or conducting a conversion with a service provider (Vgo is 1 of 2 companies in the world who do this kind of work as well as providing a product).

There are several distinct Pro's and Con's for this option:

Pro's:

- ☑ Convert your business logic
- ☑ Gain efficiencies in creating (minimally) the application architecture for your application
- ☑ Create a true Java/JEE/ADF application
- ☑ Much faster than a re-write
- ☑ Clean up and consolidate objects in Forms app's
- ☑ Can incorporate process re-design
- ☑ Can support Forms/ADF co-existence points

Con's:

- Still manual re-work which may be considerable based on the structure/quality of the old Forms
- More expensive than an upgrade
- Requires Java/ADF programming skills

- Beware of “blind” conversions – simple generation of 1:1 Forms code, no understanding of existing application or “black box” implementations
- Watch out for un-maintainable Java apps and bad application architecture (don’t convert to a Java representation of your Forms application)

Re-engineer When…

The main choice of re-engineering is centered on the vision of the overall architecture: positioning to adopt SOA or clear, industry-standard, integration of applications. As Oracle points out, there are some opportunities to integrate Forms into a larger context with SOA positioning (wrapping PL/SQL in Java). However, there are limitations when considering non-Oracle technologies. These limitations can be overcome when including SOA Suite components like BPEL and Business Rules repository.

When re-engineering, a similar methodology of project execution can be implemented as re-writing, except that some aspects are “lighter” in process rigor, such as extracting, examining and applying “requirements” from existing legacy components. The method of re-engineering is a topic that warrants its own paper and is not discussed in great detail here, but suffice it to say that it is a big undertaking and under-estimating the size, scope and complexity is common in almost all clients we have spoken to.

Option 3: Re-writing from Scratch

Re-writing a Forms application isn’t as simple as opening JDeveloper and mapping your business components and generating a new application. This seems to be, for some reason, how simplistic a number of folks present it. When re-writing from scratch, a customer should be contemplating their business process from scratch as well. Seeing that, it entails all the rigor of new application development: collecting business requirements, determining functional flows, creating business rules, creating design specifications (for infrastructure and database as well as the application itself), testing and QA methodology and criteria, user acceptance criteria, GUI specifications and usability, testing at all levels, security implications, end user training, documentation, rollout and a myriad of detailed components mixed in there.

Pro’s:

- ☑ End result is exactly the application required
- ☑ Function and technology should be a match
- ☑ Architectural design is optimized based on current technology

Con’s:

- Longest and most costly option
- High risk to business
- No re-use of any existing components is likely
- Low business tolerance; “Why are we doing this again when we already have a good system?”

Re-write When…

The most compelling option in completely re-writing an application isn’t based on technology; it’s based on *business needs*. Alone, business need doesn’t seem touched upon in the countless blog’s and papers of the technorati out there.

If your business has evolved in its processing or has major process changes in store that increase value, you should re-write. If your business partner is happy conducting business the way it always has, the other options may suit schedule, direction and budget in a better way.

Even in a re-write there may not be a need to throw everything out and start over. It is unlikely that a new application will create entirely new rules if the process is still the same. For example, a Forms application that processes new business for an annuities business is still going to process the same work, but perhaps in a different manner. Therefore, code segments that validate a contract id or ensure that the product chosen with the contract is a valid

combination in a given state, are still likely to be valid business rules. We therefore recommend doing an assessment of the existing Forms application and extracting out rules that can be re-used in a new application. Vgo Software has a product called Evo ART, an application portfolio assessment tool that provides deep technical insight into what is in each Form. Additionally, Evo ART will store this data into a repository which then can be mined and transported in to the Fusion Business Rules repository. Otherwise, open each Form and go through them manually to extract business rules and collect the rules (though this will be tedious and Forms developers may decide to jump ship).

The remainder of this paper discusses Upgrades and Re-engineering as options and where to consider SOA adoption with these two approaches. Discussing a rewrite using Jdeveloper and ADF cannot be contained in the scope of this document.

Upgrade Options

There are multiple options to upgrade Forms, once you've decided you wish to upgrade.
Two are outlined briefly here and the others (using the JDAPI and third party tools) are not covered in this paper.

When upgrading from client/server Forms to web forms, you must consider any and all of the changes that are required. The change from 6i to web forms is most significant as there may be client-side operations that must change in order to work through the 10gRx applet architecture. Following is a brief list of things to look out for and resolve *before* going through the upgrade:

- HOST, ORA_FFI, User Exits, Java Importer
  - In a web world you have to accommodate these differently via JavaBeans, PJCs, etc.
- Local client operations will change to operate on the Application Server – READ_IMAGE_FILE, TEXT_IO, etc.
- Other points to consider: tabbed canvases (might re-do to stacked canvases, settings), SYNCHRONIZE (might not need it), TIMERS, some mouse events don't work (_mouse_over, mouse_enter, mouse_leave) and Icons (need to be converted to JPG or GIF)

Evo ART has reports that generate exactly where these "critical built ins" reside, so you can address them proactively ensuring a higher degree of automated success.

Forms Builder

In Forms Builder, you can open each Form and re-compile in the newest version of Developer. The only issue is when upgrading from a very old version, such as v3, where you need to upgrade in a step-wise manner; from 3 -> 4.5 -> 6i -> 10g -> 11. The nice aspect of this approach is you get instant feedback. But the obvious down-side is that it is very slow.

Batch Compiler

A batch compiler is provided with the Oracle developer installs since v6 (or even before; I'm not totally sure). The batch compiler can be executed with DOS or shell scripting and passed a series of flags. Pointing the batch compiler at the folder where the Forms are and compiling with the upgrade flag to "Y", it will attempt to create FMX's for all the FMB's that it can successfully compile. An. ERR file is created to hold errors that are created during the upgrade process. Manually review the .ERR file and then, most likely, adapt your original Forms to accommodate the error. Each version has its own distinct list of issues.
The batch compiler option is nice in that it is the fastest of the Oracle provided options. Using Evo ART to foresee issues and then resolve them, combined with the batch compiler, is a solid performing option.

RE-ENGINEERING CONSIDERATIONS

Re-engineering and modernizing your Oracle Forms application is a major undertaking that requires planning and design considerations. It is not for the faint of heart and requires organizational readiness in its adoption and in its undertaking.

Organizational Readiness & Preparation

Prior to re-engineering your application, you still must consider the following items:

- What business functions or processes can, or should be, changed in the move to a true web architecture?
  - i.e. define the business value
- Staff resourcing: do I have ADF knowledge in-house? If not, will my partner help me position this to be maintainable by my staff?
- If there are changes to the application, will the business accommodate them?
  - Our experience is that the business will accommodate them if you agree to add sought after changes in the application.
- Testing and quality assurance methodologies *must be* a major focus in the planning of a re-engineering project. More provided later in this document.
- Proper project planning and management that include all phases of a lifecycle, but abridged to consider the notion of re-engineering.

Role of Requirements Gathering & Testing

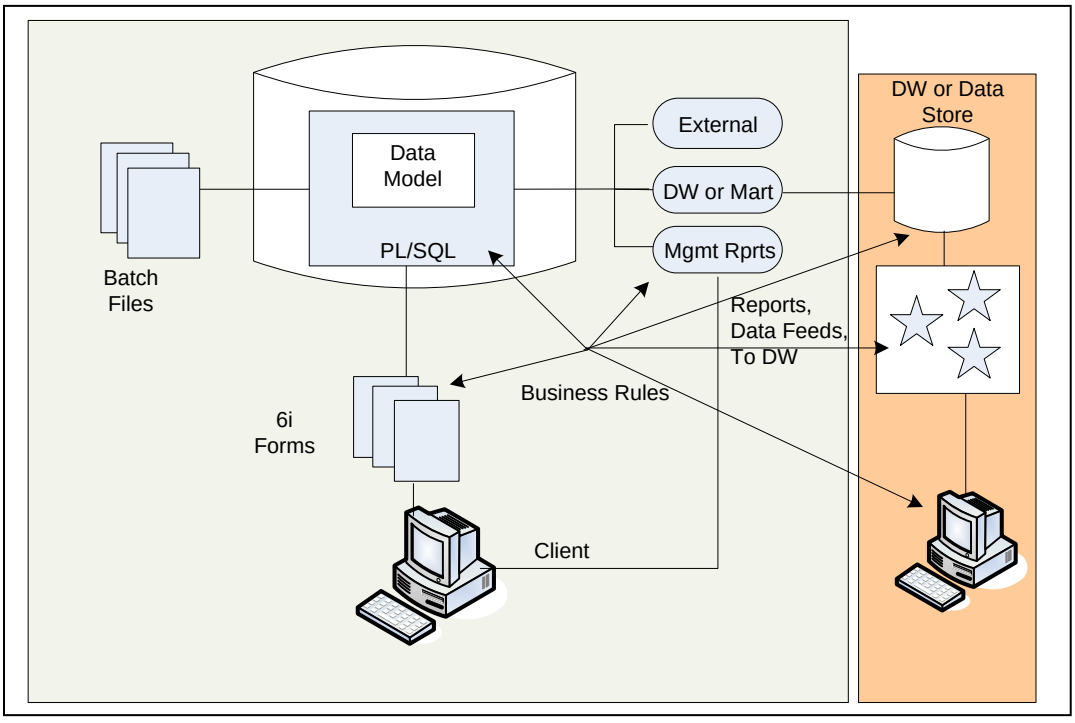
In most of our experience, customers believe that the simple existence of a "working" Forms application is enough of a living requirements and testing document as anything. This is not the case. Forms are very complex applications that have many "in grown" pieces of functionality that do not translate to web architectures. Knowing ahead of time where functional changes will be initiated will help in communicating the need for new functional requirements from the business or extra attention in testing.

ARCHITECTURAL CONSIDERATIONS

This section outlines typical Forms application architecture and how that aligns with Oracle SOA Suite components.

Business Rules in a Single Application

A typical Oracle Forms architecture is presented in the figure below.



This diagram outlines not only the Forms application itself, but an entire architecture: transactional database to which the Forms are attached, data feeds into/out of the transactional database that present data through Forms to downstream systems such as reporting.

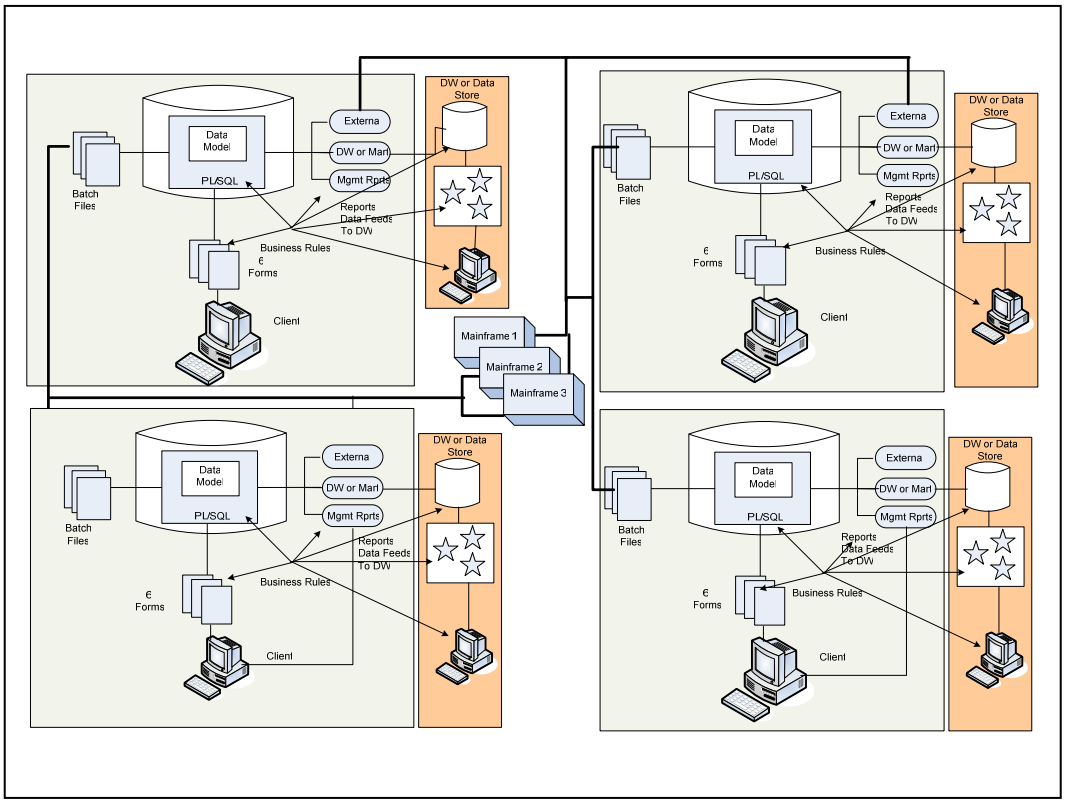
The main point of the diagram is the location of business rules. There are not only rules in the Forms (program units, validations, etc.) but also in the database, in the programs that create or read in external data and in the downstream systems (in this case, a data warehousing system with associated data marts that are accessed in canned and ad-hoc manners). These business rules are very likely to be redundant and duplicated multiple times just in the Forms application (think of all those pre-form validation triggers that validate a contract status or similar). If these are not consolidated and cleaned up in the re-engineering process, they will be made redundant in the resultant Java or ADF application. Remember earlier when we mentioned a "blind conversion"; a blind conversion simply attempts to translate exactly what was in each Form to Java. Imagine the mess it makes when you have at least 3 times the source files in a Java app then you had in 1 Form. The application is simply un-maintainable and not functionally scalable. So, attempt to consolidate, re-use and properly design an object architecture using your existing code as a baseline.

Now, consider *outside* of the Forms application. What is redundant in the database? Does any of my Forms code reside in the database as PL/SQL? Does any of the PL/SQL in the database really belong in the Model layer of the ADF application? Which of these rules can be re-used with my inbound data feed systems? In working through a proper re-engineering effort these questions and processes, and many more, should be executed in order to maximize the value of the project.

If someone chooses to simply upgrade an application, then the location of application-specific business rules may not be important. However, when re-engineering or contemplating a re-write, the location of business rules is very important to gain the most value out of the exercise.

Business Rules in the Enterprise

Considering the abundance of rules in one application, what about *enterprise* rules and their impact? In the following diagram, the single Forms application is duplicated into what may be a very common picture of an enterprise application architecture within a large company.



This diagram assumes interaction between application silos as well as with legacy mainframe applications. How many of the business rules are now redundant? How many external, perhaps purchased, data feeds are redundant? If components of the enterprise are consolidated, what can be ultimately removed and how much of that "sunsetting" results in bottom-line savings to the enterprise?

It's these questions that should be asked when re-engineering applications, whether they are Forms based or not. The SOA movement, embodied by the Oracle SOA Suite, presents itself as the best architectural choice for resolving these issues.

MAPPING FORMS TO FUSION

This paper is supposed to be about Forms and so we'll stick to that for this section. This section looks at what are equivalent, or optional, mappings into the Fusion stack. The area of Forms mapping to ADF v11 is addressed here as well.

Forms to ADF v11

Vgo Software has some "early adopter" experience with v11 of ADF prior to it becoming general release. In early efforts, our developers have found v11 to be the best release of ADF to come out ever. Finally, approaches to managing intensive or long running transactions can be more easily supported using ADF "taskflow's" and security, though having some (as of this writing) missing pieces, is much better than it has been in the past.

The chart below reflects some of the components listed in the previous section and can relate to either the single silo or to the entire enterprise application. The mappings and decisions may be the same, though broader and more intense in an enterprise setting.

Legacy Environment	Fusion or ADF Mapping	Complexity	When?
Forms LOV's Query-based blocks Tables (DB) Transactions Windows & Canvases	ADF v11 TR3 Read-only View Objects View Objects Entity Objects Task Flows JSPX pages & PanelGroups, resp.	Mid/High	Specific to app
DB PL/SQL	Leave in DB Convert to ADF (Entity Object) BPEL Business Rules BAM	Simple Medium/Low Medium/Low	Specific to app Better performance, broader use External use
File processing	BPEL ADF Business Rules BAM	High Medium Low Med/High	External parties involved Re-use in silo Cataloging Enterprise scale

Is this the "end all be all" of mapping? Admittedly, no. However, it's intended to give the reader some ideas of where individual components could be moved to adopt an SOA architecture using Oracle's SOA Suite.

Closing

In closing, Oracle Form's users definitely have a sound, stable and supported environment until at least 2011. It stands to reason that Oracle itself will adopt ADF in its pursuit of integrating its business using Fusion as the driving force behind its business vision. When that time comes and is "burned in" through products and support, it is this writers' opinion that Forms will eventually be completely de-supported and removed from circulation.

From our experience with v10.1.2 of ADF and ADF v11, there are tremendous advancements that are critical in the transition of a Forms architecture to ADF. V11 is clearly what Forms users should be aiming for; attempting v10.1.2 ADF conversion is not advisable with v11 ahead in so many critical areas.

But the main point of this paper is to consider a broader application context than simply a Form. Consider the technical architecture, the business needs and the broader enterprise direction in any move you make with Forms. As Oracle has done in considering its business direction (acquiring solution-based companies), consider your business drivers. Will your company be acquiring other businesses? Will you be getting rid of legacy applications for commercial ones? Will you be developing new applications using different technologies? If the answer to any of these is "yes", then the adoption of the components in SOA Suite is also a building block to being able to adapt to this changing business landscape. The rules in your Forms application and, indeed, throughout your enterprise are important pieces to be leveraged going forward.

Look for updates or specific items of interest on our blog at www.java-hair.com. Our co-founder and CTO, Robert Nocera, provides information on ADF, Java in general and other topics, not all technology related.

The author of this document is Ernst Renner, CEO of Vgo Software, Inc. Mr. Renner has worked with Oracle technologies since 1992, working together with Mr. Nocera developing v3 Oracle Form's applications and now working together in delivering enterprise applications all over the world.