How Will You Build Your Next System?

Dr. Paul Dorsey, Dulcian, Inc.

There are many hotly contested debates within the Information Technology (IT) world. IT Professionals have strong opinions concerning everything from the basic process that should be employed to build systems to the correct products to use to help build systems. Over time, one would expect that the industry would converge on the best solution. The philosophy of science indicates that studies would be done to help us to determine the best way to build systems. But this doesn't seem to be the case with information systems development.

In the past, there seemed to be convergence on workable (if not ideal) solutions. In the 1970s and 1980s, we built systems based on IMS or IDMS data structures using COBOL or FORTRAN. The decisions to make were few since there was not much to choose from. You worked with whatever the platform supported (remember punch cards?). Each system was built according to the philosophy of the development team. Frequently each developer had his/her own development philosophy and built whichever way he/she thought best.

In the 1990s, we built systems based on relational databases. Within the Oracle environment, we eventually converged on using Oracle Designer and Developer to build client/server-based systems. Although there were always a few competing development environments, the different styles of development were relatively well partitioned and there was general agreement about system development best practices. This was a huge improvement over the past. Many (if not most) teams actually had a coherent development architecture.

In these "good old days," an Oracle database system developer with reasonable knowledge of SQL, PL/SQL and some common sense could be very productive in a short amount of time.

With the popularity of Object-Oriented (OO) technology and web-based development at the turn of the century, we moved into a world with more questions than answers. The open source revolution has plunged us into chaos where no two development organizations build their systems in the same way. For the last decade or so, we have been told by the IT industry, that the next new thing will solve all of our problems. Yet projects still consistently fail. This paper discusses the current system development environment and will hopefully provide some answers to the question: How will you build your next system?

What Are the Options?

So, your organization is ready to build a system to work with an Oracle database....Which technologies will you use? What skills does your development team have? There are now so many...

Oracle Forms

Oracle Forms 4.5 was released in 1993 for Client/Server systems. It was the first "GUI" version of Forms that worked. It included a built-in IDE. Over the next decade, huge numbers of Forms applications were built and deployed in all types of organizations worldwide. Many sites deployed thousands of Forms applications internally. There was a great deal of support for the product since Oracle Applications (Financials) was built in Forms.

Oracle Forms 6*i* was released in 1997 to work with the Oracle8 database and help with the shift to the web. After that, the Forms release numbers were synchronized with the database releases. In 2000, Forms 9i was the first version with no client/server support. Forms Server (Web Interface) was the only runtime option. There were many changes at the server level in an attempt to improve communication between the server and the selected browser.

Forms 10g was released to work with the Oracle 10g database in 2004, and 11i in 2007 included some minor modifications. However, overall, there have been minimal new features added to Forms since version 4.5.

Oracle Application Express (APEX)

Although Oracle Forms was widely used by many organizations in a myriad of business arenas, the learning curve and depth of knowledge required was significant. Internally, many at Oracle were using something called WebDB (later HTML DB) and finally renamed Application Express (APEX) which was released in 2008. APEX is largely perceived as being an easy-to-learn, easy-to-use, productive, capable environment. However, not all users are satisfied; the learning curve is significantly longer than advertised; and not are all projects successful. APEX's flexibility of supporting custom JavaScript and HTML templates is an important extension to the tool but has also provided the tool with a great propensity for abuse.

APEX has some advantages. It is free and is included in the database. It was built by the database group and not the application development team. It is therefore possible to use a "thick database" approach with APEX. APEX has become very popular for getting applications up and running quickly as long as an organization is willing to stay within the limits of the functionality that it provides. It has gained ground much more quickly than JDeveloper/ADF which has a very steep learning curve.

The core problem is that APEX has multiple ways of implementing business rules in the product. Development can be done in a number of ways:

1) In the IDE
2) Using the APIs
3) In JavaScript
4) In HTML templates

This has added to the confusion and complexity of building systems even with a more "user-friendly" tool like APEX.

Java EE

Java EE (stands for Enterprise Edition as opposed to Standard Edition) is Oracle's Java computing platform. It includes a runtime environment and an API to build and run enterprise software. Java EE encompasses network and web services, and other large, complex applications. Java EE is based on modular components built in Java and using the XML language for configuration.

The umbrella term "Java EE" may encompass thousands of "products." If you look at ten different Java EE projects, you are likely to encounter ten completely different architectures. Figure 1 shows an example of a typical Java EE architecture.

The environment is evolving so quickly that last year's best practices may be obsolete this year. In addition, many Java developers are database-hostile which often results in applications that must be rewritten year after year as the technologies change.

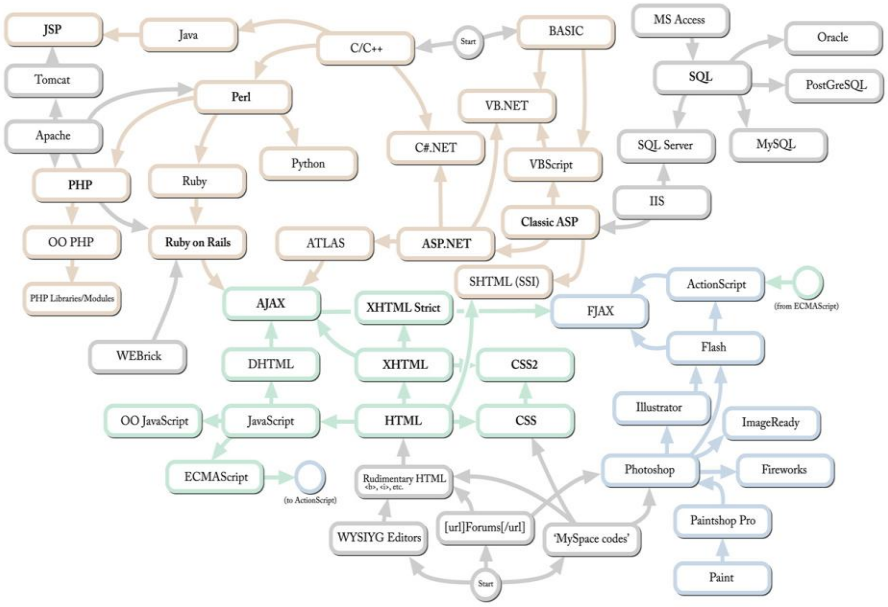

Figure 1: Java EE Architecture Example

ORACLE'S APPLICATION DEVELOPMENT FRAMEWORK (ADF)

Oracle 's Application Development Framework (ADF) attempted to provide an effective means for creating JavaEE-based applications to work with an Oracle database. Although it is better than anything else in the JavaEE space, it has lots of moving parts, and is still evolving. Projects built with ADF involve high risk and are very complex. Even with this Oracle framework any ten projects may have eight different architectures.

The JDeveloper Version 11i of ADF is frequently described as being as easy as Forms. Although ADF has made great advances over the years, it is still a very complex environment that requires skilled developers and, when not used correctly, is very difficult to be successful with.

Even with all the advances, it is still a complex JavaEE framework. For example, here is the code required to get the value from an on-screen component. This would have been a simple ":BLOCK.ITEM" reference in Forms.

```
<af:selectOneChoice

binding="#{ProjectSearchBean.cityNr}"
</af:selectOneChoice>
public class ProjectSearchBean extends ....
{
private Number cityOid = new Number(0);
...
...
public void setCityOid(Number cityOid)
  {
    this.cityOid = cityOid;
  }
  public Number getCityOid()
  {
    return cityOid;
  }
...
}
```

The underlying architecture of ADF is very complex. Figure 2 shows a sample ADF application life cycle diagram.

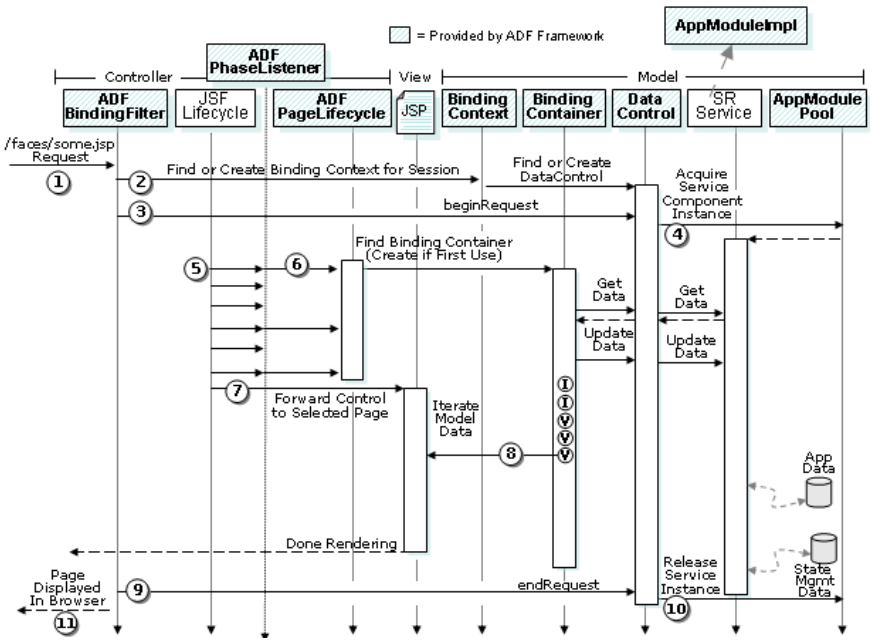

Figure 2: ADF Application Life Cycle

.NET

Microsoft's answer to the system architecture problem is .Net. So far, it seems that .Net is not as hard to learn as JavaEE. However, it does not have as large a following as JavaEE and its developers tend to be database-clueless (and intend to stay that way).

THE SOFTWARE DEVELOPER'S MANIFESTO

Why do we accept "new" environments that are worse than our old environments?

I would like to set out some basic principles of software development that should appear in any acceptable development environment. Since all of these ideas were part of Oracle Forms 20 years ago, I find it hard to believe that we can't do as well on the web as we could in a client/server environment 20 years ago.

1) Using features from Oracle Forms that I should still be able to use without them being any harder.

   Forms had lots of nice capabilities that seem be absent or very difficult to implement in modern web environments. A few key features come to mind:

   - Pixel-level positioning of components
   - Complex panel layouts (nesting, multiple panels in a window)
   - Support for "Post" and "Commit" – transaction consistency

2) Using only PL/SQL

   Why do I need to learn 5 different programming languages? Shouldn't I be able to do everything in a single language? Since I need PL/SQL for the DBMS anyway, why not use the same language for my UI development?

3) Low complexity

   It should not take a year to learn a development environment. Both ADF and APEX users report that it takes a full year to become proficient in the tools. There should not be 10 different technologies in a development environment. A single developer should be able to build an application. It should not take a group of several experts in different technologies. It should not take 2 days and digging through a handful of books to figure out how to make a new feature work.

4) High performance without high effort

   I should not have to be careful to ensure that my application will scale. I should only have to worry about obvious things like poor SQL and too many round trips between parts of the system (like between the application server and the database).

5) Productivity

   I should be able to build a screen in about a day, just like we did in Forms.

6) Cross-platform support

   With Java EE, we have the classic "write once - debug everywhere" problem. Every version of every browser behaves differently. Building for mobile devices is its own sub-specialty requiring totally different technologies.

WHAT DO I USE?

For many years, I built systems in Oracle Forms. In the last few years, I have tried to make ADF work. I looked carefully at APEX but it was not able to do many of the things required in the systems I was building. After a number of frustrating years, I decided to write my own framework. It is "Forms-like" and uses a "thick database" approach.

Given the endlessly changing system front-end options, it is logical to focus on the database since the back-end database platform rarely changes. This has been a best practice for years. It is critical to build as much business logic as possible into the database. In this way, it is possible to keep the logic close to the data. Using this approach, create a view for each screen. Also, it is important to limit the "architecture" choices to selecting the front-end for UI deployment.

WHAT SHOULD YOU DO?

"The average lifespan of a deployed application is 5 years".....NOT! This unfortunately seems to be the case in most organizations. The environment is changing so rapidly, so what can be done?

1. Select an architecture and go.
2. Minimize the number of moving parts
3. Use a "thick database" approach

Even with these strategies, you can almost invariably count on a long ramp-up time for developers to acquire the necessary skills to build a system that can evolve effectively along with the technology. When you are finished building the system, the world will still have moved on.

Conclusions

The inevitable conclusion to be reached is that there is no clear answer or even a good answer. The application development world is in a continual state of flux and the risk of failure is higher than ever. Thick database is your only defense.

About the Author

Dr. Paul Dorsey is the founder and president of Dulcian, Inc. an Oracle consulting firm specializing in business rules and web-based application development. He is the chief architect of Dulcian's Business Rules Information Manager (BRIM®) tool. Paul is the co-author of seven Oracle Press books on Designer, Database Design, Developer, and JDeveloper, which have been translated into nine languages as well as the Wiley Press book *PL/SQL for Dummies*. Paul is an Oracle ACE. He is President Emeritus of NYOUG. In 2003, Dr. Dorsey was honored by ODTUG as volunteer of the year, in 2001 by IOUG as volunteer of the year and by Oracle as one of the six initial honorary Oracle 9*i* Certified Masters. Dr. Dorsey's submission of a Survey Generator built to collect data for The Preeclampsia Foundation was the winner of the 2007 Oracle Fusion Middleware Developer Challenge and Oracle selected him as the 2007 PL/SQL Developer of the Year. Paul can be contacted at paul_dorsey@dulcian.com

AUTHOR'S NOTE: This paper originated from a joint keynote presentation given at NYOUG's Spring General Meeting in 2011 with Michael Olin, President and CEO of Systematic Solutions, Inc. Many thanks to Michael for his valuable contributions to this paper and corresponding presentation.