User Interface Technology Choice
For Web Deployed J2EE
Applications

An Oracle White Paper
April 2005
User Interface Technology Choice for Web Deployed J2EE Applications

OVERVIEW
This paper discusses some of the factors to consider when making user interface technology decisions for building browser deployed, Java 2 Enterprise Edition (J2EE) applications. The paper is primarily targeted at developers who have or are just embarking upon new application development at this moment in time and have to choose between several alternative user interface technologies, primarily JavaServer Pages, JavaServer Faces and the Oracle ADF UIX framework.

INTRODUCTION
The J2EE platform provides a set of low level routines in the form of the Servlet Application Programming Interfaces (APIs) which allow the programmer to directly construct the output that is streamed to the browser. On top of this basic set of APIs exists a whole hierarchy of convenience layers, which make it simpler to output dynamic content in particular. These layers encompass both the user interface visualization and the Front Controller functionality required for typical multi-page applications.

There are many players in the UI and Controller framework space, some of which serve both functions, some only one. Later on the document we’ll focus on the big fish in this pond, but that’s not to deny the presence of many alternatives, some of which are listed in the glossary.

FAST FORWARD TO THE FUTURE
By this time next year, this paper will be effectively obsolete and the discussion contained within largely irrelevant. The industry wide road map for development of thin client applications is very clear. JavaServer Faces (JSF), a standard defined by JSR-127 and accepted into the J2EE 1.5 standard, is rapidly gaining headway as an effective and rich way to develop applications. Within a year we will see JSF support built into many of the mainstream application servers as a core service, and of course design time support will also be available from the various vendor and open source IDEs. We’ll discuss the benefits of JSF in more detail later on in this paper.
Although JSF is a standard environment, it is not the only possible standards based stack for enterprise development: Servlets and JSP are, and will continue to be, a part of the core J2EE standards and still be applicable to mainstream development.

It is likely that many smaller frameworks such as Tapestry, Webworks, Velocity and many custom JSP tag libraries will fall by the wayside, and the thin client development space will coalesce into two distinct spheres: JSF and JSP. Of these, the latter will still largely depend on external controller functionality such as Apache Struts or Spring MVC.

THE PLAYING FIELD TODAY

So, we have a good idea where the industry will be in a dozen months time or so, but what does this all mean to a developer using Oracle JDeveloper as an IDE and Oracle ADF as a framework today?

Right now, the 10.1.2 release of JDeveloper 10g provides the developer with the choice to use just about any interface technology or framework that they choose. The IDE and the ADF design time integration provides extra support for developing with both JSP and ADF UIX with visual WYSIWYG design time environments and drag and drop data binding to any business service via the ADF Model layer (JSR-227). Developers are also encouraged to adopt a style of development that utilizes a separate controller for handling page flow, and Apache Struts, as the de-facto standard in this space, is provided along with a corresponding design time environment.

Looking then at the two most common options for Web UIs:

JSP in detail

JSP is a widely used and understood technology. The current generation of tools across the industry provide productive design time environments for JSP with good editors and design time rendering. Much functionality within JSP is provided in the form of sets of tag libraries. These extend the syntax of the JSP markup within the page, to represent data in a richer way and provide other value added functions such as conditional logic, on top of the basic HTML capabilities.

Recent advances in the JSP arena, specifically the introduction of the JSTL tag libraries and the JSP 2.0 specification, have improved the JSP working environment further by reducing the need for 3rd party tag libraries, whilst at the same time making the construction of the same simpler if required.

On the negative side, JSP presents a relatively low level interface into the world of web application development. The programmer is beholden to understand much of the Servlet API, and even with the aid of frameworks such as Apache Struts and Oracle ADF they are not totally shielded from this reality. Once understood, however, this closeness to the low level APIs does provide the programmer with an immense amount of power and flexibility if they need it.
ADF UIX
Whereas JSP represents a very markup-centric approach to development, which is very close to the underlying HTML in its capabilities and user interface, ADF UIX takes an alternative approach. UIX stands for User Interface XML, and rather than being a version of the markup to be streamed to the browser, ADF UIX is a more abstract representation of the page as component model defined in XML. At runtime the actual implementing markup required for the browser is generated from this definition. Such a level of abstraction offers several benefits over straight JSP/HTML:

- The page abstraction is not tied to a particular browser or indeed markup language such as HTML
- The model defines a large set of components which themselves can be rich and complex and result in an actual implementation that consists of both markup, images and JavaScript, all handled transparently by the ADF UIX framework rather than the programmer
- The logical page contents as defined by the XML can be manipulated programmatically before rendering. This allows for applications that can be much more dynamic
- The framework handles page layout through formal components and templates making it simple to swap out the look and feel, encompassing not only the colors and fonts but also the gross layout of the page and styles of buttons and tabs

So ADF UIX provides a very rich and flexible UI creation framework, but is not aligned with any industry standards for UI definition. With the 10.1.3 release of JDeveloper, ADF UIX will still be available but most developers will probably want to choose the standards based JSF framework for new developments instead.

WHAT DOES JSF BRING TO THE PARTY?
JSF is in many ways similar to ADF UIX, and this is no great surprise given that Oracle was heavily involved in the definition of the JSF standard. Like UIX, JSF provides the developer with a more abstract description of a user interface based around a component based layout model.

JSF also provides a listener based approach to event handling in the page. This is much more natural for both Java Swing programmers and users of 4GL tools such as Oracle Forms. Code is simply associated with an action event, and the programmer is shielded from all of the low level tasks such as inspecting the Servlet request to detect what events to handle. In fact, JSF is designed in such a way as to insulate the programmer from the whole Servlet API and the loose coupling provided by the JSF abstraction means that JSF is not confined to serving up applications based around HTML markup through a web browser. JSF as a development approach can equally be used to create applications for running on portable devices, phones and even telnet style terminals.
Part of the JSF standard also defines exactly how components for use with the framework should be created. This opens up the market to an influx of components all of which are cross compatible, both with JSF runtime implementations and design time environments. Oracle for instance has already released early access versions of its JSF ADF Faces component suite comprising of over 100 components, many of which have evolved from the UIX framework to embrace the new component standard.

Looking even further forward, JSF has one very important strength. With the component based model that JSF uses, it becomes possible for the framework to adapt to newer user interface technologies as they evolve without changing the fundamental page or programming model. There is already a move within the industry towards very rich dynamic html clients using DHTML and XMLHttpRequest. JavaServer Faces is ideally placed to embrace that technology, giving rise to interactive desktop applications running inside a browser but without the need for applets or downloads.

**IS THAT ALL?**

So far this paper has really only concentrated on UI technology that is directly supported by the JDeveloper IDE. However, the application development space has many niches. Some of the alternative UI creation frameworks such as Apache Velocity and Tapestry are playing firmly in the same space as JSP and JSF and we're discussed how they may have a harder time competing with these standards in the future. But other frameworks exist that operate outside of the conventional Servlet model, particularly those which provide additional “rich client” functionality within the browser. Although these frameworks may still utilize pieces of a J2EE infrastructure, such interactions are totally obscured behind proprietary APIs provided by the frameworks themselves. Examples of such frameworks are:

- Oracle Forms
- Macromedia Flex
- Laszlo

Each of these frameworks use the concept of some kind of screen rendering engine which is either implemented as a Java Applet, in the case of Forms, or using the Macromedia Flash plug-in in the case of Laszlo and Flex. They all also have a Server component with which the client rendering engine communicates at runtime to handle data interaction. And, finally they all expose a programming API to the developer in the form of a simple scripting language, such as PL/SQL or JavaScript.

There is no doubt that such rich client frameworks can be used very successfully to create highly interactive and functional applications. The reliance a particular vendors proprietary server solution and the requirement for a client plug-in for the browser may reduce the attractiveness of such solutions in some situations, but for
many functional application requirements these products provide the only viable solution today.

DECIDING ON A USER INTERFACE TECHNOLOGY TODAY

Now we’ve met the main protagonists in the discussion, let’s return to the question: What technology should I be using today?

These scenarios are based around the following facts and assumptions:

- We’re concentrating on a thin client (no plug-ins) application
- Both JSF and the JSP + Struts development approaches will continue to be mainstream
- The JSP + Struts combination will continue to be supported with ADF as long as industry acceptance is widespread
- Full ADF and JSF support will be combined in the JDeveloper 10.1.3 release due this summer
- JSF will become the preferred UI development platform for use with the Oracle ADF framework

Let’s look at a series of common scenarios.

Scenario 1: I don’t use ADF Model data binding

In this case, if you want to start to use JSF then you can do so today, either with JDeveloper 10.1.2 or with the JDeveloper 10.1.3 J2EE developer preview, which provides visual editor support for the JSF Reference implementation and the ADF JSF component set.

Scenario 2: I am already well underway with a project that uses JSP and Struts with ADF

This will be a fully supported scenario in future releases and is already a proven approach to application development. There is no reason to change the technology stack in use.

Scenario 3: I am already well underway with a project that uses ADF UIX and Struts with ADF

As with the JSP case, this configuration will be supported in the 10.1.3 release and so there is no need to revise the technology choice at this time. However, it would not do any harm to consider a future move to upgrade from ADF UIX to JSF, bearing in mind that migration utilities and documentation will be available to aid in this process.

Scenario 4: I am about to start a long-term project which will take many months to complete

In this case, you may have the opportunity to wait for the 10.1.3 release for the creation of the user interface portions of the project. You can start to create the
screen UI today using the 10.1.3 developer preview and ADF Faces, and then retrofit the data binding to the model layer once 10.1.3 is in production.

If you have a prototyping phase ahead, you could consider using JDeveloper 10.1.2 and ADF UIX as a reasonable approximation of the kind of interface that will be achievable with ADF and JSF, whilst still having the productivity benefits of the simple data binding provided by ADF.

If you are risk adverse and don’t want to use the first release of a new technology combination such as ADF and JSF then you should use the existing ADF + JSP + Struts technology stack on the grounds that this is stable and proven.

**Scenario 5: I have a short-term project that needs to be started and completed within the next few months**

As in both scenarios 2 and 3 either choosing JSP or ADF UIX are viable strategies. However, much of the decision-making will have to rest on the skills that are available to you and the kind of user interface that you need to achieve.

Although ADF UIX has a much richer user interface out of the box, it has a somewhat steeper learning curve for the more advanced features so unless you already have ADF UIX skills, or you are using a value added framework such as JHeadStart you should probably adopt a JSP and Struts based solution.

**CONCLUSION**

JavaServer Faces is clearly the technology of choice for future thin client application development, however, given the size of the existing JSP code base it is likely that the industry will largely polarize around JSF on the one hand and JSP and Struts on the other. As such a technology decision based on either of these two paths is likely to be a sound one.
<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADF</td>
<td>The Oracle Application Development framework. A modular development framework for both Web and desktop client J2EE development encompassing several user interface and business service provider technologies (EJB, WebServices, Oracle Toplink, Business Components) all bound through a common declarative data binding layer the ADF Model.</td>
</tr>
<tr>
<td>Apache Struts</td>
<td>Struts is primarily a page flow controller with some custom tag library functionality included as a convenience to the developer. Developed as open source under the umbrella of the Apache Foundation, Struts is, in industry terms, mature and very widely used, and in and of itself is consumed into several higher level frameworks, namely Oracle ADF and BEA Weblogic / Apache Beehive</td>
</tr>
<tr>
<td>Flex</td>
<td>A rich client framework from Macromedia which uses XML and JavaScript to define the application, uses a custom server to run it, and renders the UI in the browser using the Flash plug-in.</td>
</tr>
<tr>
<td>JSR</td>
<td>Java Specification Request. An enhancement to the Java Language or APIs carried out under the auspices of the Java Community Process, where vendors and other interested parties agree on requirements and specifications. Information about JSRs can be found at <a href="http://www.jcp.org">www.jcp.org</a></td>
</tr>
<tr>
<td>Laszlo</td>
<td>An open source rich client framework based around XML and JavaScript which uses a custom server and renders the UI in the browser using the Macromedia Flash plug-in.</td>
</tr>
<tr>
<td>Oracle Forms</td>
<td>A rich client framework based around SQL and PL/SQL for building transaction applications that are tightly coupled to a database. The framework uses a custom server and renders the UI in the browser using a Java Applet.</td>
</tr>
<tr>
<td>Spring</td>
<td>Spring is an open source Inversion of Control container which works on the principle of decoupling resource management (for instance creating a database connection) from the use of that resource. Spring covers many areas of J2EE application development, including this space which is addressed by Spring MVC and Spring WebFlow</td>
</tr>
<tr>
<td>Tapestry</td>
<td>An open source framework, again hosted by Apache, which covers both page UI definition as well as controller functionality. Tapestry presents a component based approach</td>
</tr>
<tr>
<td>Framework</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Velocity</td>
<td>A page templating language used both as an alternative to JSP for web page layout, but also as general templating language for other uses such as within IDEs.</td>
</tr>
<tr>
<td>WebWork</td>
<td>A combined UI and Controller framework mainly positioned as being better and more flexible than Struts. However, despite its technical benefits it does not have the broad popularity of Struts.</td>
</tr>
</tbody>
</table>