From Struts to JavaServer Faces

Evolving Your Web Applications to Support the New Standard

Kito D. Mann, editor-in-chief, JSF Central
May, 2005

Professional Tools for Eclipse

Table of Contents

Executive Summary

Last year, the Java Community Process (JCP) completed the initial version of JavaServer Faces (JSF), the new standard framework for web application development. JSF offers a user interface (UI) component model, an event-driven programming model, and several other features that ease the process of developing web applications.

With the advent of JSF, organizations that have standardized on other web frameworks, such as Struts, are evaluating their options for future development. First, they are interested in the key benefits that moving to JSF will provide, including the true value of JSF tools. Secondly, they are examining the possibilities for leveraging their existing code base and skill sets. Finally, they are interested in best practices for building new applications with JSF. This white paper addresses these questions with a thorough discussion of key JSF features and an overview of how your organization can begin migrating to JSF.

The Evolution of Java Web Application Frameworks

In the past few years, the process of Java web development has been aided by a healthy abundance of web development frameworks. These frameworks have built upon the foundation laid by servlets and Java Server Pages (JSP) to provide robust facilities for form handling, layer separation via the Model-2 design pattern, navigation handling, templating, internationalization, and a host of other features.

Most of these frameworks grew out of real-world development experiences, filling in the areas where standard Java technologies – servlets and JSP – lacked functionality. Over time, Apache Struts emerged as the framework of choice, largely because it was early to market and provided all of the core features necessary at the time. Struts has also maintained its popularity partly because of a strong commitment to end-users, a vibrant community, and a solid understanding of the importance of backwards compatibility. Struts is also the only framework whose popularity has generated a significant market for development tools and books.

Even though Struts is the most popular Java web development framework, many don't consider it the best choice on the market. As a result, the landscape has been flooded by alternative options such as WebWork, Tapestry, Echo, and several others. Even more comprehensive frameworks like Spring offer basic web development services such as Spring MVC.

The multitude of choices for Java web development has led to *framework paralysis* – choosing the correct framework has become a complicated, time-consuming task. While the competitive aspects of choice in the Java ecosystem can improve individual products, the impact becomes negative when there are too many choices.

JavaServer Faces – The New Standard

As competition between the Java and Microsoft .NET communities heats up, the appeal of the Microsoft ASP.NET's single-solution approach becomes obvious. From a management perspective, having a clear path is not only appealing, but more cost-effective in terms of employee resources. The simplicity of a single choice, coupled with ASP.NET Web Forms' powerful component model and vibrant component marketplace, makes it a compelling alternative to the multitude of choices in the Java landscape.

Component models are not foreign to Java – web frameworks such as Tapestry and Echo have had components for some time – but leading frameworks such as Struts do not offer these features.

With the advent of ASP.NET and the continual crowding of the Java web framework marketplace, it has become clear that Java requires a standard web framework with a powerful component model. Such a framework could leverage the best features the Java ecosystem can offer, while simplifying the selection process for development shops and simultaneously opening up a third-party component market. This is the goal of JavaServer Faces (JSF) technology.

JSF offers several features that establish it as the foundation for the next generation of Java web development:

- Robust tool support
- A solid component framework
- A set of standard UI components
- Form validation
- Extensible support for multiple client devices
- User interface event-oriented development
- Internationalization and localization support
- Extensible type conversion
- Navigation handling
- Declarative association of the UI with application code
- An extremely extensible architecture

While many of these features have been present in other Java web frameworks, JSF has the particular distinction of heavy industry support and the backing of the Java Community Process (JCP). Within the first year, JSF has already generated the market for third-party tools and books that Struts required years to create.

In addition, JSF's features are geared towards ease of use. With support for UI components and a simplified event-oriented programming model, developing web applications is significantly easier and faster than with alternative frameworks such as Struts.

Let's examine a few of these features in more detail.

User interface event-oriented development

Traditional web application development involves thinking in terms of HTTP requests and responses, and requires a high degree of familiarity with the nuances of the protocol. This contrasts with traditional thick client application development promoted by rapid application development (RAD) tools like Microsoft Visual Basic and Borland Delphi as well as Swing. These tools and APIs work in terms of user events like clicking on a button or changing the value of a text control. These events are then associated with application code via event handlers.

This event-oriented approach is a more natural and productive method of developing applications, and lets developers concentrate on responding to the user's actions as opposed to pulling values from an HTTP request.

Although a few Java web frameworks have promoted this event-oriented approach, JSF is the first to push it into the mainstream. In JSF, all application code responds to user interface events so that developers no longer need to concern themselves with the nuances of HTTP. In addition, JSF can automatically associate form values with application objects, eliminating the need for handling this process manually for every request.

A solid component framework

Events in JSF are generated by UI components. A UI component can be something as simple as a text box, or more complicated like a data grid (a la Swing). This allows developers to build UIs by assembling a collection of components and then associating them with object properties and event handlers. Gone are the days when a developer had to spend a disproportionate amount of time developing basic UI elements and writing large amounts of JavaScript code which was irrelevant for the applications' business domain.

While the JSF specification contains a basic set of UI components such as text fields, list boxes, check boxes, panels, and so on, the real power of JSF is its ability to support third-party components. There are already several components on the market (some commercial and some open source), which provide pre-built, drop-down menus, tree controls, enhanced data grids, and so on. Often these components make use of JavaScript and DHTML when appropriate, which creates a more client-friendly UI without the additional custom programming that is often necessary.

As the market matures, Java web development will concentrate more on assembling these components than working with HTTP. This reduces the overall costs of development by limiting the amount of work required to develop robust UIs, allowing developers to focus on core business logic. In addition, since components are developed by organizations with a vested interest in improving their products, the capabilities that they provide often exceed the capabilities provided by in-house development teams. The end result is a more powerful, attractive, and functional user interface developed with fewer man-hours.

Support for multiple client devices

While the HTML browser is still the dominant client delivery mechanism, any framework which desires relevance in the long-term must also support alternative choices. JSF has the ability to dynamically display different types of markup to different client devices. As a matter of fact, UI components can be decoupled from the way they are displayed. This allows developers to switch to entirely new devices with minimal coding effort. Currently, JSF components can be displayed as WML, SVG, XML, and even displayed to telnet devices. JSF components can also expose rich DHTML interfaces and Asynchronous JavaScript and XML (AJAX) functionality to enable a richer client experience.

Robust tool support

Since JSF was designed with tool support in mind, it has garnered heavy support for tool vendors. Those who have worked with powerful Struts development tools such as M7's NitroX know that solid development tools can greatly enhance the productivity of a development team.

JSF's tool support enables the kind of IDE integration that is the domain of Java tools with visual Swing editors, or visual-oriented IDEs such as Microsoft Visual Studio.NET and Borland Delphi. This includes support for a component palette, property editors, and the ability to drag and drop components onto a design surface. JSF tools also support visual editing of navigation rules – the path from one page to the next. In addition, tools like M7's NitroX for JSF exhibit a key understanding of the different artifacts involved in a JSF project, automating the synchronization between configuration files, JSPs, and application code.

Figures 1 through 4 show how tools can bring a powerful approach to rapid application development of Java web applications.

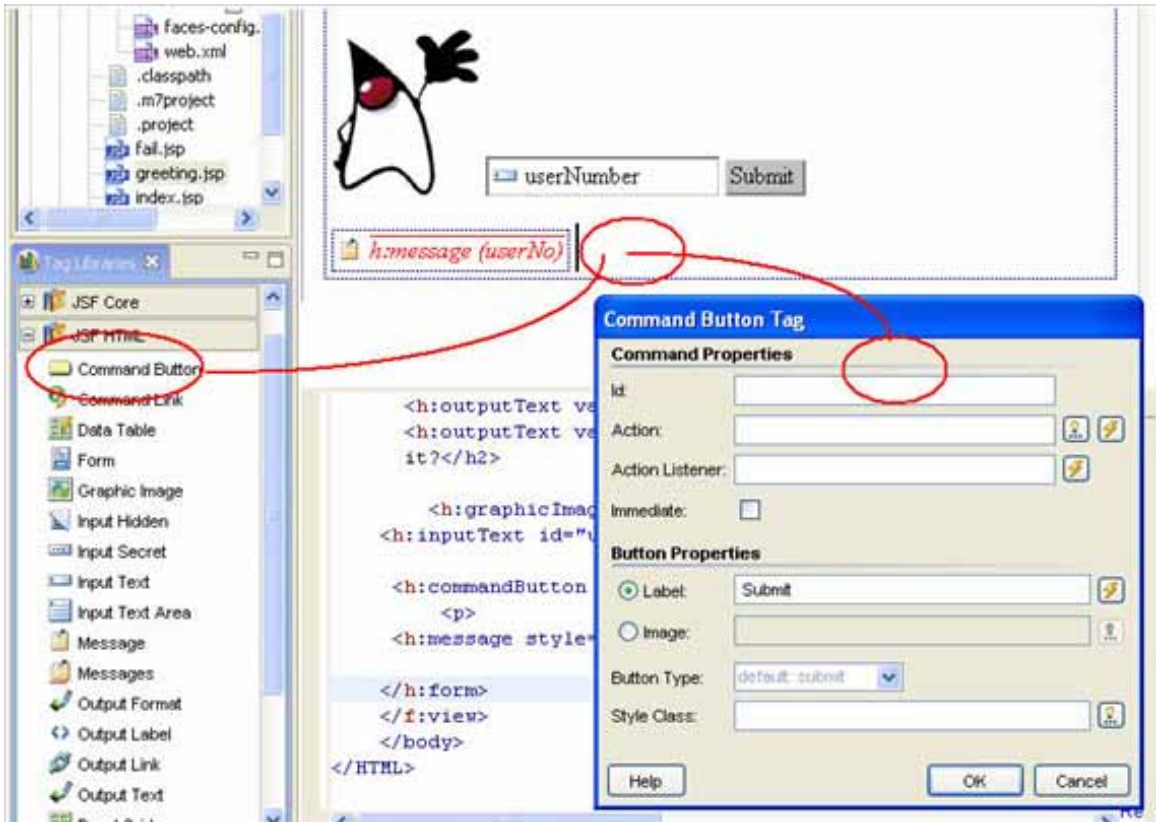

Figure 1. M7 NitroX JSF drag & drop development

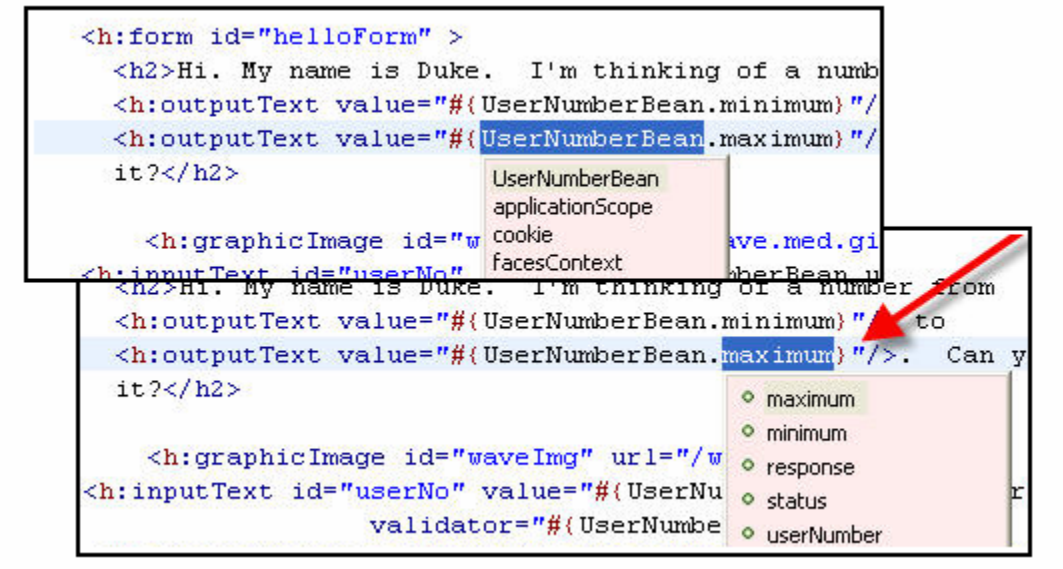

Figure 2. M7 NitroX code completion for JSF related artifacts

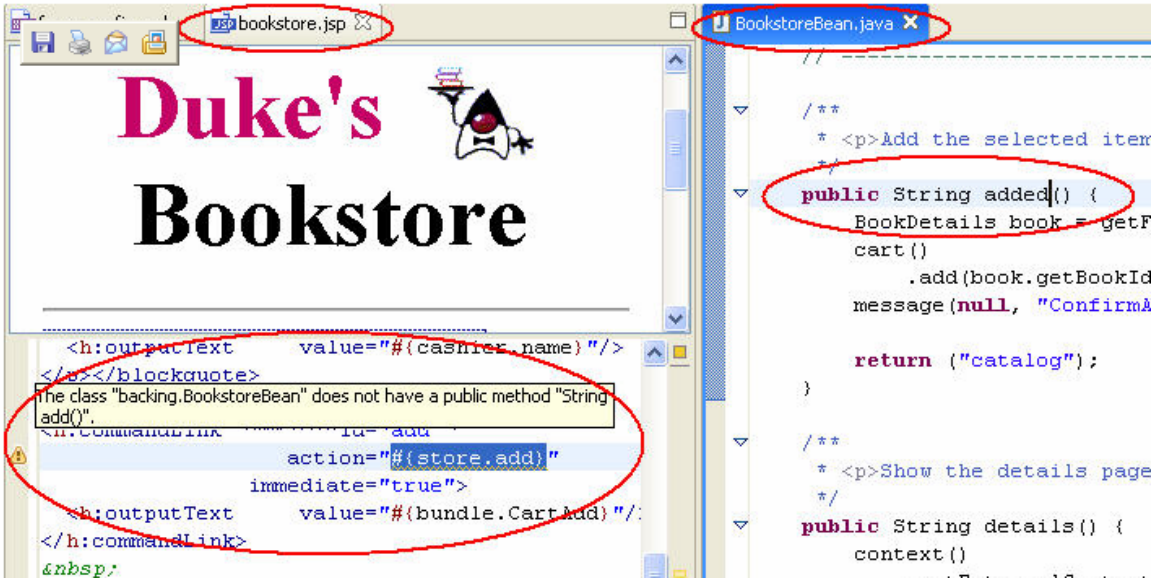

Figure 3. M7 NitroX validation & error checking between JSF related artifacts, JSP & Java files

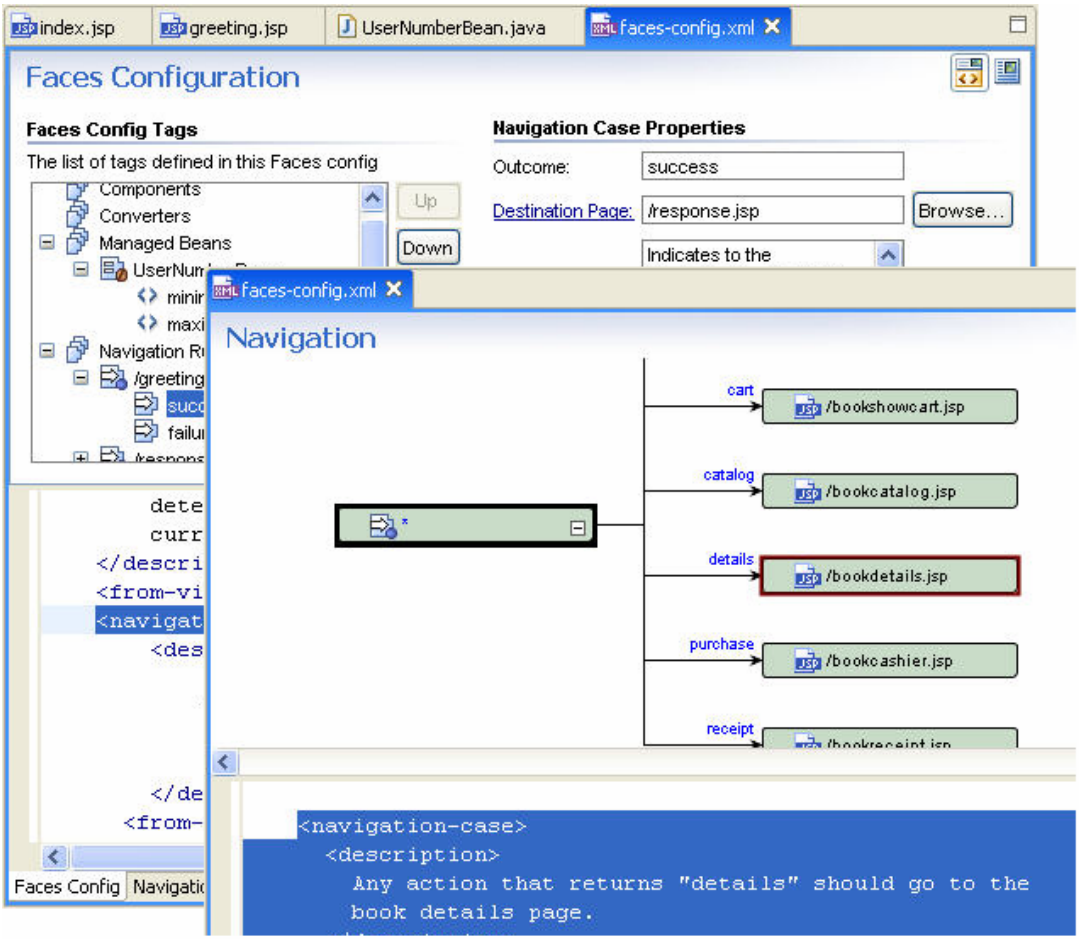

Figure 4. M7 NitroX JSF faces-config.xml editors

These advanced features offer a dramatic value proposition when compared with developing web applications without tool support, which is often the case with other Java web frameworks.

An extensible architecture

While the goal of JSF is to provide a single standard for Java web development, the members of the JSF expert group are not oblivious to the current state of the Java web development landscape. As such, they placed an extraordinary emphasis on providing extension points throughout JSF's architecture. These extension points allow for integration with existing frameworks, enabling development organizations to continue using their framework of choice in conjunction with JSF. This protects existing investments while providing a migration path for the future. Moreover, this open architecture allows other frameworks to support one of JSF's key benefits – UI components. Already, both Struts and Spring have provided integration libraries that utilize these extension points.

Migrating to JavaServer Faces

As momentum behind JSF continues to grow, development managers face key issues about how to move forward with existing and new projects. They must determine situations under which migration is necessary or cost-effective, and which migration strategy is the best choice.

When to migrate

In the past, most development teams chose Struts because of its strong industry backing and its strong basic capabilities. Use of JSF, however, is rapidly growing due to leaps in developer productivity. Organizations that wish to actively develop existing applications (whether or not they use Struts) should definitely consider migrating to JSF.

Migrating to JSF enables teams to take advantage of the framework's key benefits, and to leverage the growing market of third-party components and tools. For applications that are in maintenance mode and do not foresee any major changes, migration is not necessary, especially if the maintenance staff is already well versed in Struts.

Migration barriers

There are few obstacles to beginning work with JSF. There are several books on the market[i], on-line documentation[ii], third-party components[iii], on-line resources such as JSF Central[iv], and development tools such as M7 NitroX[v]. There are also two implementations – Sun's reference implementation (RI)[vi] and Apache MyFaces[vii].

Currently, both JSF implementations use JavaServer Pages (JSP) as their templating mechanism for declaring pages. JSF has the ability to support other templating mechanisms such as pure XML or Java code, but that support has not yet been exploited in shipping products. Consequently, migrating to JSF at this time requires using JSP.

The most important barrier is the skill set of the development team. For teams that have spent the majority of their development careers building web applications and using frameworks such as Struts, the component and event-oriented model for JSF development can prove difficult to understand initially. Developers usually benefit from a quality book, and are often more productive once they fully master the concepts. Many of the skills developed using other frameworks are essential for developing JSF applications as well.

Developers that have been exposed to web frameworks such as Tapestry or Echo, Java Swing designers, or tools such as Microsoft Visual Studio.NET, Microsoft Visual Basic, Borland Delphi, or Sybase PowerBuilder usually grasp component-oriented methodologies quicker.

Migration strategies

JSF's extensible architecture allows for a variety of approaches for working with existing applications. The most conservative approach is to use JSF components without other framework features. Alternatively, it is possible to migrate incrementally or perform a complete migration.

Components only

Supporting JSF components in existing non-JSF applications allows you to take advantage of the growing UI component marketplace while minimizing the cost of rewriting existing elements of your applications. This is especially beneficial when you have a large, complex application and limited time.

Using JSF components with an existing application requires integrating JSF with the application's existing framework. For example, Struts has a library called Struts-Faces[viii], which integrates JSF and Struts (there is also a Spring integration library[ix] available).

Struts-Faces enables you to build UIs using JSF components while using your existing Struts actions. This is achieved with specialized JSF event listeners that execute the Struts request processing chain. The Struts actions and forwards work as usual, and are unaware of JSF. This is possible because both Struts and JSF are implemented as servlets that execute within the same web application (see figure 5). Since they are running within the same application, they can also share variables and application logic.

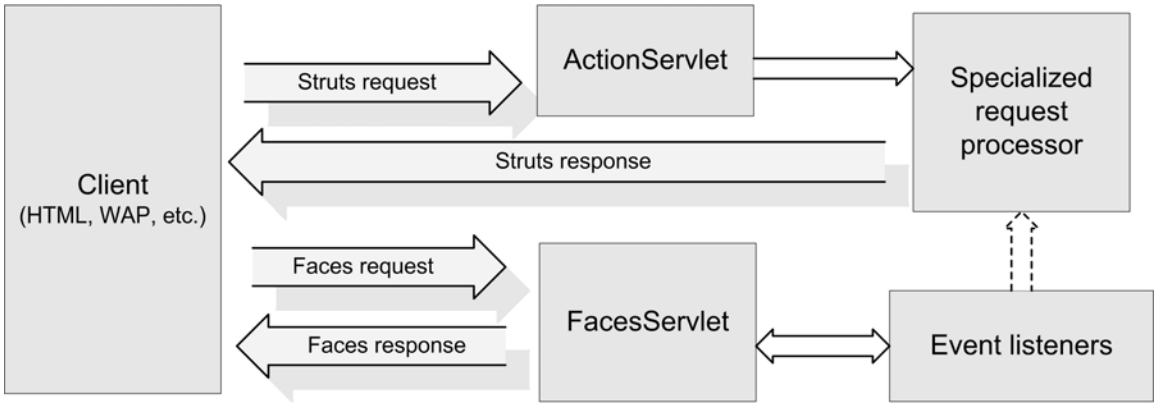

Figure 5. When using JSF components only, all events are forwarded to Struts. Both the JSF and Struts servlets execute within the same web application.

For applications which do not use Struts or Spring, you must either use a similar third-party integration library, or develop such a library yourself. The rules are the same, however. The migration process consists of replacing proprietary UI custom tags with JSF component tags, perhaps creating some specialized custom tags that integrate with the framework, and developing JSF event listeners that delegate control to your existing application logic. The general strategy is depicted in figure 6.

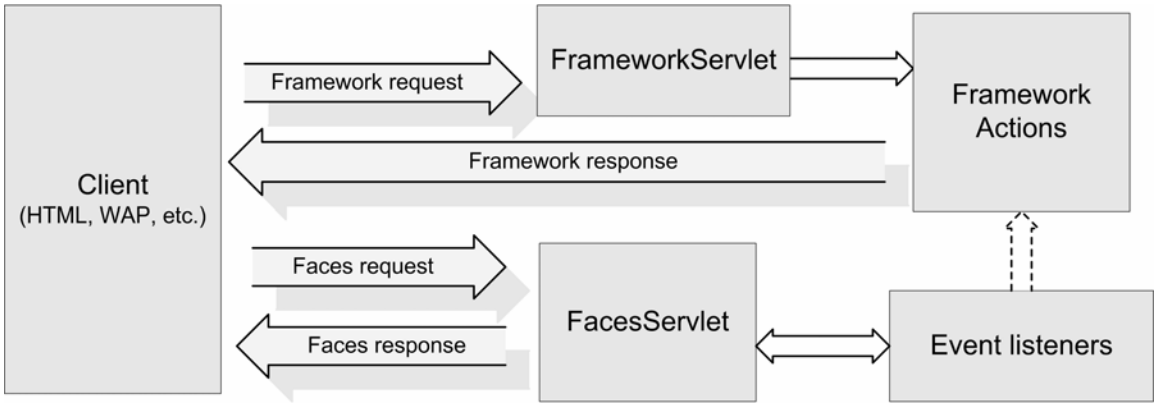

Figure 6. For non-Struts applications, the process is the same: JSF event listeners delegate control to existing application logic.

Incremental migration

When the eventual goal is to migrate to JSF completely, you can do so incrementally. This allows you to begin taking advantage of JSF's benefits quickly, with minimal impact on your existing infrastructure. Incremental migration is especially useful when you need to continually roll out new features while upgrading to JSF at the same time. It is also particularly well suited for agile development approaches, because each release can include migration of a specific portion of the application.

Incremental migration can be achieved using integration libraries much like component-only migration. As a matter of fact, component-only migration can be the first step towards incremental migration. Once your application works using JSF components instead of proprietary custom tags, the next step is to begin moving application logic to JSF backing beans. Backing beans in JSF are a combination of Struts actions and action forms (similar to WebWork actions), but do not require a base class.

For well-segmented applications with business logic in separate domain-specific objects, moving the application logic is a fairly trivial task. The task is trivial because the code in the web layer is basically responsible for executing business logic classes, evaluating the result, and directing the user to the proper response page. Migration simply involves moving this code to JSF backing beans.

For situations where business logic is dispersed throughout web-layer actions, the process can be more complicated. In these cases, migration is an opportunity for refactoring the code to be more loosely coupled.

Incremental migration can be performed by converting portions of application logic, after migration of all pages to JSF components, or by migrating related pages and application logic simultaneously. This is possible because all of the code is running in the same web application, so even if part of the application uses JSF and part uses another framework, the same objects can be shared (see figure 7).

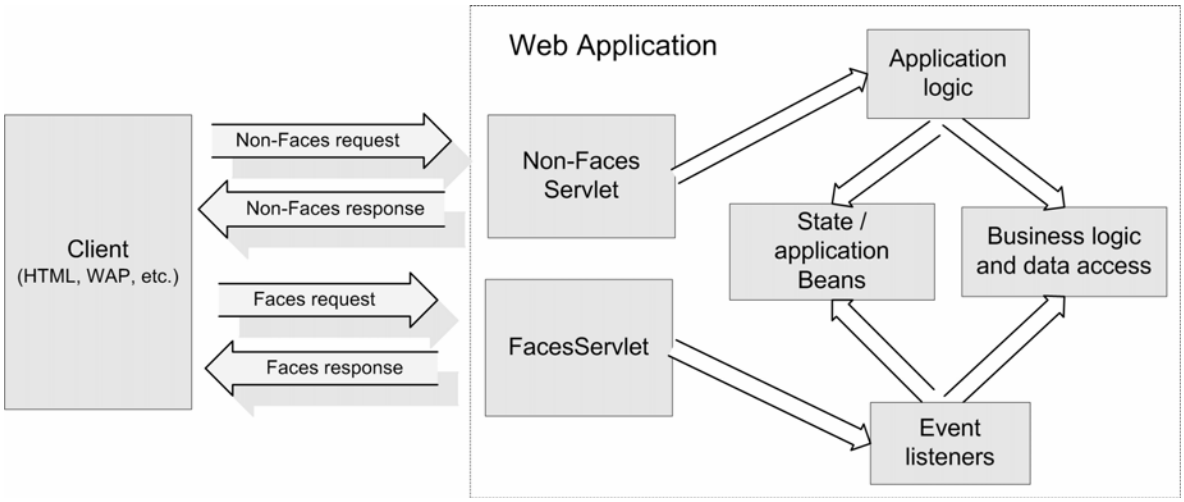

Figure 7. The key to integrating with JSF is understanding that you can have multiple servlets, and that application logic in a JSF environment and a non-JSF environment can access the same objects, as long as they're in the same web application.

Figure 8 compares a related set of Struts artifacts with their JSF equivalents. As the figure shows, there are three layers that must be migrated to JSF: configuration, JSP pages, and application logic. In many cases, these artifacts map particularly well. For example, a Struts *forward* is roughly equivalent to a JSF *navigation rule*; both are handled through configuration files. Also, most Struts tags have JSF equivalents.

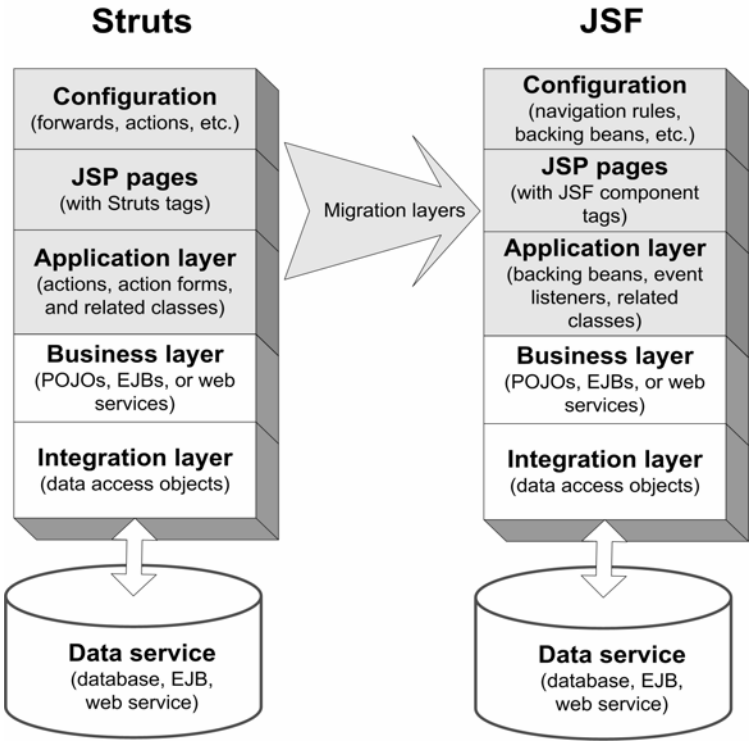

Figure 8. Migrating to JSF requires moving configuration elements, updating JSPs, and refactoring or re-writing application layer code.

Full migration

Rather than migrate an application incrementally, you can migrate the entire application at once. This approach is somewhat more cohesive, because the application is never in an inconsistent state – it uses either one framework or the other at all times. Full migration is more time consuming, but it is a very logical approach when there is a need to rewrite or substantially update the application, and you have the luxury of more time before an initial deployment. In addition, it provides an opportunity to take advantage of some of JSF's unique features, as opposed to simply porting the application.

The process of full migration is conceptually similar to the process of incremental migration, as shown in figure 8 – the configuration, page, and application layers must be converted. The key difference is that no integration library is necessary, which means that the entire application must be migrated before all of its functionality will be available. Since no integration library is necessary, this option is especially well suited for home-grown frameworks and commercial or open-source frameworks for which no library currently exists.

Building New JavaServer Faces Applications

JSF's extensive industry backing, powerful architecture, component model, and growing third-party component market make it a compelling choice for Java web development. Those beginning new web development projects should use JSF, in order to take advantage of its benefits and ensure long-term community and industry support.

The heavy support of tools vendors can not be understated, because they can dramatically increase productivity, which may directly impact on the overall cost of a project. JSF tools come in a variety of flavors, from basic plug-ins, to added support in full-fledged IDEs, and powerful, dedicated plug-ins like M7 NitroX.

Vibrant community support is available through the Apache MyFaces project, which is growing in leaps and bounds, as well as community projects at java.net[x], which provide transparent access to both the development of the RI, as well as the development of the specification itself.

JSF is a compelling choice not just for plain vanilla HTML applications, but also for powerful AJAX applications, and other clients such as WAP devices or even telnet devices. Support for all of these clients is possible through its component technology and powerful rendering architecture.

Even though JSF has proven itself as a powerful framework for building applications, due to its youth, it may not have every feature available in more established frameworks. However, its extensibility means that one can develop additional functionality on top of the JSF core. This approach makes it easy for development teams to extend JSF with custom functionality.

The best example of this approach today is the Struts Shale project[xi], which adds an extensible request processing approach using the "chain of responsibility" pattern, additional support for developing dialogs, automatic creation of backing beans, and even support for non-JSP display technologies. In addition, component sets like Oracle ADF Faces[xii] provide additional functionality above and beyond the standard JSF runtime. By leveraging these third-party extensions as well as custom enhancements, development teams can maximize their productivity while building upon a solid foundation simultaneously.

Conclusion

Over the years, the Java web development landscape has become fragmented, due to lack of a standard web application framework. Apache Struts has emerged as the *de-facto* standard, but a steady stream of competitors continue to proliferate, attempting to provide additional features which Struts lacks. This proliferation has created framework paralysis – difficulty in deciding which framework is the best choice for a given project.

Framework paralysis has a negative impact on productivity, and decreases the overall value proposition of Java development. JavaServer Faces (JSF) is the standard best-of-breed framework that is intended to ease the process of web application development in Java. It offers a powerful UI component framework, a basic set of UI components, an extensible architecture, and several other features that position it as the new choice for web application development.

Teams that are currently using older frameworks such as Struts have several options for migrating to JSF. The simplest of these options is to utilize JSF components while maintaining application logic within the existing framework. This approach makes it easy to take advantage of the growing market of third-party JSF UI components while having a minimal impact on the application's overall architecture.

Migration can also be performed in an incremental fashion, converting only pieces of the application to JSF at a time. This approach makes sense when the entire application must be ported eventually, but new features must be rolled out quickly. Incremental migration is also especially well suited for agile development shops.

For situations where more time is available, or when a major update or rewrite is necessary, full migration is a viable option.

New projects should consider using JSF because of the market for powerful tools like M7 NitroX, components, and growing open source community around products like Apache MyFaces and Apache Struts Shale.

Kito D. Mann is editor-in-chief of JSF Central (http://www.jsfcentral.com) and the author of JavaServer Faces in Action (Manning). He is also a member of the JSF 1.2 and JSP 2.1 expert groups and Principal Consultant at Virtua, Inc., specializing in enterprise application architecture, development, training, mentoring, and JSF product strategy. He holds a BA in Computer Science from Johns Hopkins University.

References

[i] For a list of JSF books, see http://www.jsfcentral.com/reading.

[ii] Sun Microsystems. The J2EE 1.4 Tutorial, chapters 17-21. http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html

[iii] For a list of third-party JSF components, see http://www.jsfcentral.com/products.

[iv] Virtua. JSF Central JavaServer Faces community and FAQ. http://www.jsfcentral.com.

[v] M7. NitroX for JSF Eclipse-based web IDE for open source and standard platforms, http://www.m7.com/product.do.

[vi] Sun Microsystems. JSF reference implementation. http://java.sun.com/j2ee/javaserverfaces.

[vii] Apache. MyFaces open source JSF implementation. http://myfaces.apache.com.

[viii] Apache, Struts-Faces integration library. http://cvs.apache.org/builds/jakarta-struts/nightly/struts-faces/.

[ix] Mindmatters. JSF-Spring integration library. http://jsf-spring.sourceforge.net.

[x] Sun Microsystems. Project home for the JSF Reference Implementation. https://javaserverfaces.dev.java.net.

[xixi] Apache. Struts Shale next generation Struts subproject. http://wiki.apache.org/struts/StrutsShale.

[xii] Oracle. ADF Faces components. http://www.oracle.com/technology/products/jdev/htdocs/partners/addins/exchange/jsf/index.html.

JSF Central (http://www.jsfcentral.com) is the leading community for managers, architects, and developers interested in using JavaServer Faces technology. The site features an extensive listing of news, articles, blog entries, products, and other resources. For more information, contact info@jsfcentral.com.

JSF Central is a service of Virtua, Inc., an enterprise consulting, training, and web publishing company. For more information, contact info@virtua.com.

Professional Tools for Eclipse

M7 Corporation, Inc, the leader in web application tools based on open source and open standard technologies, is exclusively focused on improving the development of web applications for serious developers who want to avoid platform lock-in. M7 draws on its long history of experience in understanding the complexity of web application development. Drawing on this depth of understanding, M7 develops innovative products, such as NitroX, which fuel an incredibly high level of productivity in creating, editing, or debugging any part of a complex web application.

Based in Cupertino, CA, M7 is privately held and venture backed by Highland Capital Partners, Redpoint Ventures, CIR Ventures and Genevest. For additional information, visit the company's web site at www.m7.com.