
The Geronimo renegade: Facing JSF

What open source development is really like

Skill Level: Intermediate

[Nicholas Chase \(ibmquestions@nicholaschase.com\)](mailto:ibmquestions@nicholaschase.com)

Freelance Writer

06 Feb 2007

The last time I wrote a "Geronimo renegade" column, I was wondering what the big deal was about Spring, and when I followed up, I joked that now I needed someone to tell me why JavaServer Faces (JSF) was so hot. Wouldn't you know it that the next thing that crossed my desk was a request to do a piece on the intersection between Apache Geronimo and Apache MyFaces, focusing on their implementation of JSF? Well, it turns out to have been a good thing. I struck up a conversation with Tim McConnell at IBM®, who's in the process of integrating Geronimo and MyFaces, and I learned a lot more than I expected about how specifications like this actually get implemented.

What is Apache MyFaces?

The first question I needed to know was why Apache Geronimo was even concerned with Apache MyFaces in the first place. Tim explained that it was part of the migration from Java™ 2 Platform, Enterprise Edition (J2EE) 1.4 to Java Platform, Enterprise Edition (Java EE) 5. "Geronimo is already a J2EE 1.4-compliant application server, but as part of the 1.5 specification, JSF needs to be supported, and the way we're doing that is with Apache MyFaces."

Already I was intrigued. Here's a technology considered so important that it has actually been added to the Java EE specification. But what is it? "The idea behind JSF," Tim told me, "is to make it easier for Web application development. It's modeled after the Model-View-Controller (MVC) paradigm. It's actually almost an implementation of that. That's really the idea behind it. A lot of people who haven't done Web development are still familiar with MVC, so they can pick up and learn JSF quickly and more efficiently and develop their Web applications."

That sounded reasonable, but there had to be more. So I sat down to research

exactly what JSF was. What I found is that I've been causing myself considerable pain in avoiding JSF.

If you've been doing Java Web development, you already know about JavaServer Pages (JSP). These are pages in which you can embed Java code directly within HTML and have it be executed accordingly. But more important than that, JSP enables you to create custom tags that represent Java code. Not only is this more efficient from a server performance standpoint, but it's much more maintainable and gives non-technical people the ability to edit pages.

JSF takes this process a couple of steps further. It looks at the common things that people need to program into Web pages -- forms, for the most part -- and includes pre-made components -- those custom tags I was talking about -- to help you do it. (MyFaces also includes hundreds of additional components to add functionality.) For example, consider the JSF form shown in Listing 1.

Listing 1. The JSP

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<html>
  <head>
    <title>JSF Sample Page</title>
  </head>
  <body>
    <f:view>
      <h1>
        <h:outputText value="Login"/>
      </h1>
      <h:form id="loginForm">
        <h:outputText value="Username: "/>
        <h:inputText value="#{userBean.username}" />
        <br />
        <h:outputText value="Password: "/>
        <h:inputText value="#{userBean.password}" />

        <h:commandButton action="login" value="Log In" />
        <h:commandButton action="register" value="Register a new account" />
      </h:form>
    </f:view>
  </body>
</html>
```

Notice that it's a mix of straight HTML along with tags from two tag libraries, each designated as a URI (which represents the namespace) aliased to a prefix. These tags have certain behaviors, which in this case is to output specific HTML. But notice that there's something a little different in the `inputText` elements. Here it's referencing a specific Java bean, so there might be a class like that shown in Listing 2.

Listing 2. The Java bean

```
package com.backstop.renegade;

public class UserBean {

    String username;
    String password;

    public String getUsername() {
```

```

        return this.username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
    public String getPassword() {
        return this.Password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
}
...
}

```

This bean represents the data structure you're writing a form for. To tie them together, you have the faces-config.xml file (see Listing 3).

Listing 3. Open source code for the faces-config.xml file

```

<?xml version="1.0"?>
<!DOCTYPE faces-config PUBLIC
    "-//Sun Microsystems, Inc.//DTD JavaServer Faces Config 1.1//EN"
    "http://java.sun.com/dtd/web-facesconfig_1_1.dtd">
<faces-config>
  <managed-bean>
    <managed-bean-name>userBean</managed-bean-name>
    <managed-bean-class>com.backstop.renegade.UserBean</managed-bean-class>
    <managed-bean-scope>request</managed-bean-scope>
  </managed-bean>
</faces-config>

```

You'll have to check the [Resources](#) for a link to more information on how to set this up, but for now, notice that you link the object name used in the form (`userBean`) to the class (`com.backstop.renegade.UserBean`) and specify the scope to refer to a single Web request.

But JSF has one more trick up its sleeve. You still need to submit the form, but to where? On a traditional HTML page, having different actions for each button would be cumbersome to say the least. JSF lets you take care of the problem by using *navigation rules* in the faces-config.xml file (see Listing 4).

Listing 4. Open source code for adding navigation rules to faces-config.xml

```

<?xml version="1.0"?>
<!DOCTYPE faces-config PUBLIC
    "-//Sun Microsystems, Inc.//DTD JavaServer Faces Config 1.1//EN"
    "http://java.sun.com/dtd/web-facesconfig_1_1.dtd">
<faces-config>

  <navigation-rule>

    <from-view-id>/pages/login.jsp</from-view-id>

    <navigation-case>
      <from-outcome>login</from-outcome>
      <to-view-id>/pages/processlogin.jsp</to-view-id>
    </navigation-case>
    <navigation-case>
      <from-outcome>register</from-outcome>
      <to-view-id>/pages/registration.jsp</to-view-id>
    </navigation-case>

  </navigation-rule>

```

```
<managed-bean>
  <managed-bean-name>personBean</managed-bean-name>
  <managed-bean-class>jsfks.PersonBean</managed-bean-class>
  <managed-bean-scope>request</managed-bean-scope>
</managed-bean>
</faces-config>
```

This way, you can specify where the server should go from your login page, based on the `from-outcome`, as determined by the action you specified on your buttons. Of course, there are other ways to trigger movement, but that's well beyond the scope of this article.

The point is that JSF simplifies Web development quite a bit. And that's how it got on Tim McConnell's to-do list.

Moving to Java EE (formerly known as J2EE 1.5)

"There was no JSF specification in the J2EE 1.4 requirements, so this is new," Tim explained. "Because this is open source, it's not a concrete plan, but we hope that Apache Geronimo is [Java EE] 5-compliant as of next year, and that would include the JSF 1.2 spec. We're striving to do that in a much more efficient way than we got to J2EE 1.4."

I'm always interested in the inner workings of a project like Geronimo, so I asked Tim, "What's different this time?"

"I think there's more involvement from IBM now," he answered, "more of a push to have it compliant. There are a lot of enterprise customers who won't use it unless it is compliant." He pointed out that there are a number of new specifications in Java EE 5. "So there's a lot of work to do."

Something he'd said earlier intrigued me. "You say it's almost an implementation of the Model-View-Controller pattern. Why *almost*?"

"It's different because it's Web development," he explained, "and there are little caveats and sophistications that have been added to the specification. For example, what's interesting about one thing that they specify is that they have the notion of events that are triggered when a form -- or even a form element on a Web screen -- is changed, so that the implementer can register to listen to events that are effectively changes to that field. So a programmer can register to listen to events on that field so he doesn't have to check to see if it's changed, because he'll get a notification if it's changed. It's actually pretty slick. To me that's fairly sophisticated in that a lot of the work to expose those events is done by the implementation and not the programmer; the programmer just has to register an interest in that change. The idea is to make development easier."

Tim went on to explain what that means for implementing JSF in Geronimo. "There are some things that are incumbent upon the server in the specifications, such as the APIs that have to be supported. They have to come from the server, not the Web application. For example, in MyFaces 1.1, the programmer -- the person developing the Web application -- just has to include the JAR file, so for 1.2, we need to override

that to make sure that it's coming from the [Java EE] 5 stack. The idea is that when it's compliant with [Java EE] 5, the programmer won't have to do that, because it'll be included with Geronimo; they won't have to provide the JAR, just the application itself. There's code that we have to develop to adhere to the specs, and there's code that we have to develop to enable these applications that are using 1.2. Also, the spec gives you a way to determine if the application is using JSF, and then you have to do things of that nature."

Implementing a specification

All of that got me thinking: What is it like to actually implement some of these specifications? I mean, it's nice when you have it and it just works, but how does that happen? It's much like the day my wife made mayonnaise. These things don't magically appear, fully complete; somebody has to build them. (Or get together the eggs and lemon juice and make them, but you get the idea.)

"Right now I am implementing the JSR 88 spec, which is the deployment spec," Tim told me. "That one is not so bad because it's very well written and easy to understand. Not that JSF is hard to understand, but the specification is more about what the exposed API should look like. It's like any other specification. Some are better than others."

I asked Tim if he could provide me with a little bit more detail, and as he pointed out package definitions and descriptions, I began to realize that these specifications are not so different from the specifications we developers deal with every day. The JSR-252 specification, which defines JSF 1.2, defines the packages involved and provides an XML schema for documents that must be supported. What each component must do is defined in the text of the specification.

So implementing a JSR specification is much like implementing any other specification, except that things are essentially written in stone, because changing them involves going through the Java Community Process, so it's certainly more complicated than just going to your boss and hashing it out.

Developing for open source versus non-open source

Tim calls working on Apache Geronimo his "dream job," so I asked him what was so special about working on open source.

"The thing about doing open source development is that your contribution is actually the code that you develop. You don't get to become a committer on Geronimo or any other open source that I know of without demonstrating your level of proficiency with Java coding or coding in general, without contributing code to it. So in a sense, it's really 'code speaks the loudest.' To me there's good and bad about that, but it's just the way that open source has evolved."


Tim is a software engineer for IBM proper -- as opposed to solely working on Geronimo -- so I asked him how open source differed from commercial product


development. "In open source [development]," he told me, "once you contribute code, it's reviewed by the community, so everyone has a chance to look at it, review it, and give you feedback. That's the process. Product development at IBM is a lot more stringent. You have a development methodology that's followed, and then you have distinct phases: analysis, plan, design, implementation, unit test, function and system test.

"You do that in open source [development]," he continued, "but it's a little more concise. You don't have really distinct phases. In Geronimo, it's all community driven, but in product development, your work is typically dictated based on the architecture of the product. In open source, the community agrees it will do this one thing, and let's say this one thing is [Java EE] 5 compliance. The thing that's neat about open source is that anybody can work on anything they want to. Like, you say 'I'll work on JSF, and I'll work on the JSR 88 deployment spec.' So you publish that, and you get feedback from the community. They say they're fine with it, and you say, 'This is how I plan to do this,' and the community reviews and either agrees or doesn't do anything, which is a kind of implicit agreement. There's more freedom, because you can work on just about anything once you show you can do it and you become a committer. So as a developer, you do your unit testing, obviously, and the review process is part of the committing of the code, because once you commit, everyone can see what you did and review it and comment on it. So there's no distinct schedule, as in, in two weeks we'll be in this phase. Particularly for IBM product development, it's a very rigid methodology, which is good, because in my experience you get good quality software. The quality of the open source software is very good too, it's just a different methodology."

But Tim had one more distinction between open source and commercial product development: "Open source is a lot more fun, too; it really is."

Share this...

 [Digg this story](#)

 [Post to del.icio.us](#)

[Slashdot it!](#)

Summary

Tim had given me a lot to think about. First, I had a new tool in my arsenal: JSF in general and Apache MyFaces in particular. I've been using JSP for years; this seems like a great way to simplify application development. Just the navigation rules can simplify development, so when you add in all of the other features I didn't have room to touch on in this article, you have a real winner here!

Tim had also given me some insight into the real world of contributing to an open source project, demystifying the process of implementing a specification and working with the community.

Resources

Learn

- Read "[UI development with JavaServer Faces](#)" (developerWorks, September 2003) for an overview of JSF, including the basics for developing Web applications using the technology.
- Get involved in the [Geronimo project](#).
- [Join the Geronimo mailing list](#).
- Check out "[Applying the Apache License, Version 2.0](#)" for guidance, both inside and outside the Apache projects, in understanding what's needed to apply the Apache License, Version 2.0.
- Check out the developerWorks [Apache Geronimo project area](#) for articles, tutorials, and other resources to help you get started developing with Geronimo today.
- Find helpful resources for beginners and experienced users at the [Get started now with Apache Geronimo](#) section of developerWorks.
- Check out the [IBM Support for Apache Geronimo](#) offering, which lets you develop Geronimo applications backed by world-class IBM support.
- Visit the [developerWorks Open source zone](#) for extensive how-to information, tools, and project updates to help you develop with open source technologies and use them with IBM's products.
- Browse all the [Apache articles](#) and [free Apache tutorials](#) available in the developerWorks Open source zone.
- Browse for books on these and other technical topics at the [Safari bookstore](#).
- Stay current with [developerWorks technical events and webcasts](#).
- Get an [RSS feed for this series](#). (Find out more about [RSS](#).)

Get products and technologies

- Download [Apache Geronimo](#), including Little G.
- Download [Apache MyFaces](#).
- Innovate your next open source development project with [IBM trial software](#), available for download or on DVD.
- Download your free copy of [IBM WebSphere® Application Server Community Edition V1.0](#) -- a lightweight J2EE application server built on Apache Geronimo open source technology that is designed to help you accelerate your development and deployment efforts.

Discuss

- [Participate in the discussion forum for this content](#).

- Stay up to date on Geronimo developments at the [Apache Geronimo blog](#).
- Get involved in the developerWorks community by participating in [developerWorks blogs](#).

About the author

Nicholas Chase

Nicholas Chase has been involved in Web-site development for companies such as Lucent Technologies, Sun Microsystems, Oracle, and the Tampa Bay Buccaneers. He has been a high school physics teacher, a low-level radioactive waste facility manager, an online science fiction magazine editor, a multimedia engineer, an Oracle instructor, and the chief technology officer of an interactive communications company. He is the author of several books, including *XML Primer Plus* (Sams).

Trademarks

IBM, the IBM logo, and WebSphere are a registered trademarks of IBM in the United States, other countries or both.

Java is a trademark of Sun Microsystems in the United States, other countries, or both.