# Java Web Framework Sweet Spots

**Sponsored by**

TheServerSide JAVA SYMPOSIUM

March 23-25, 2006 in Las Vegas

**and**

VIRTUAS

This document was compiled by asking a number of different Java web framework authors the following questions:

1. What is your framework's "sweet spot," and for what type of projects should it strongly be considered?

2. What type of scenarios does it not fit in to? Would you recommend another framework in this scenario? If so, which one?

3. Of the other web frameworks mentioned below, have you tried any of them? If so, which ones, and what did you like about them? What didn't you like?

4. What is the future of your web framework? What's coming that'll make it easier for users to develop with? Do you support Ajax natively? If not, are you planning on adding support?

5. Are there myths about your framework you'd like to challenge? If so, which ones?

6. What do you think of Ruby on Rails?

The frameworks represented in this paper and their authors are as follows (in alphabetical order):

JSF, Jacob Hookom
RIFE, Geert Bevin
Seam, Gavin King
Spring MVC, Rob Harrop
Spring Web Flow, Rob Harrop and Keith Donald
Stripes, Tim Fennell
Struts Action 1, Don Brown
Tapestry, Howard Lewis Ship
Trails, Chris Nelson
WebWork, Patrick Lightbody
Wicket, Eelco Hillenius

## JSF

### 1. What is your framework's "sweet spot," and for what type of projects should it strongly be considered?

For when you want to bring desktop-like functionality to the browser with the reliance of a standard specification and large amounts of third-party features. JSF places heavy emphasis on being able to scale requirements and complexity well. No adaptor code in actions, no model dependencies — just your view. This simplicity makes development highly agile and promotes reuse within your applications. These facilities allow even novice developers to work with a sense of high accomplishment without requiring them to coordinate views and specialized actions, parameter passing, state management, and rendering — with or without tooling.

### 2. What type of scenarios does it not fit in to? Would you recommend another framework in this scenario? If so, which one?

Because JSF takes care of managing the state of the UI for the developer, it adds overhead such that it's probably not ideal for large, read-only web sites. In those cases, it's probably more reasonable to use Struts because of the wide knowledge pool available.

### 3. Of the other web frameworks mentioned below, have you tried any of them? If so, which ones, and what did you like about them? What didn't you like?

#### *Seam*

Pros: Extremely simple and intuitive

Cons: Could be too simple for large projects; doesn't have a good story yet for modulization of application code (uses simple @Name identities and @Role)

#### *Struts*

Pros: Again, large pool of knowledge; can't go wrong

Cons: State/behavior separation in the framework is very old school

#### *WebWork*

Pros: Better than Struts in practice

Cons: Like other action frameworks, complex UIs can be difficult to maintain without lots of code/special cases

#### *Tapestry*

Pros: Huge in innovation; can handle complex UIs

Cons: For a component framework, it still sticks to the page/action paradigm

### 4. What is the future of your web framework? What's coming that'll make it easier for users to develop with? Do you support Ajax natively? If not, are you planning on adding support?

This is kind of an odd question for JSF. JSF is a spec, and based on everything that's been added by other vendors and community developers, there's really no clear answer — what's in store for Seam, Shale, ADF, Tomahawk, Facelets, Sun Creator, JDeveloper, ILOG, Crystal Reports, etc.?

For AJAX, we're planning on pursuing an extension to JSF 1.2 called Avatar, based on various EG members' ideas that will allow any and all parts of JSF to be used in AJAX applications. Because JSF isn't tied to the action/page paradigm, parts of the component tree can be agnostically processed without requiring special cases/development.

The next revision of JSF (2.0) will probably include a case for a "Partial Faces Request" (AJAX). Also, there will probably be more use of annotations to ease the component development aspect of the spec.

### 5. Are there myths about your framework you'd like to challenge? If so, which ones?

There are so many tutorials and books out there on JSF that stick to handling simple Struts-like interactions. While they prove that JSF is very easy to use in building simple applications, they fail to highlight JSF's ability to scale requirements and manage UI interactions that would've been near impossible in other frameworks without lots of specialized code and hand waving in your views and actions.

Also, JSF can handle GET requests just as easily as other frameworks. Because of JSF's managed bean (IoC container), you can do the same kinds of things as you can with WebWork — from parameter assignment to backing beans. An example is linking from page to page where id's are passed — you can go the same route as Struts or WebWork and just render "employee.jsf?id=#{emp.id}" — you don't have to use stateful component communication between pages. For those who want the "action" event, JSF1.2 will fire the created/destroy events as per the common-annotations specification.

### 6. What do you think of Ruby on Rails?

It's just as good as WebWork but finally realized the benefit of convention in frameworks to the nth degree via scaffolding, etc. I think all of the frameworks listed can start taking advantage of convention, but that's all I see coming out of RoR — the recent blogs online about patterns in RoR prove that RoR will eventually go the route of other frameworks as developers expect more from it.

## RIFE

### 1. What is your framework's "sweet spot," and for what type of projects should it strongly be considered?

It's a full-stack framework, which means that you get 90% of what all the others deliver with 10% of the effort. You also get it in a consistent fashion without having to figure out how everything works together. We revisited a lot of the concepts in all the layers and re-designed them with web application development in mind, offering unique features and new approaches such as pure Java web continuations, run-time POJO-driven CRUD generation, scalable component-oriented architecture, state handling without sessions, focus on REST as an application API, bi-directional logicless template engine, and the integration of a content management framework.

In addition, our component approach offers real reusability on every level (AOP, site, sub-site, page, widget, portlet, etc.), not just for widget components.

I'd say that it's very well suited for public web sites that are developed by a team of people who want to deliver quickly, need to maintain the application over time, and are interested in building up a collection of reusable web components.

## 2. What type of scenarios does it not fit in to? Would you recommend another framework in this scenario? If so, which one?

I'd say in these situations:

- A lot of developers are on the team with knowledge of another framework, and it's not worth training them.
- You want to develop a web application with stateful server-side web components instead of using RIA or Ajax.
- Large vendor support is considered important.

In those situations, I'd recommend JSF.

Also, when you are developing an XML document publishing application, Cocoon would be much better suited.

## 3. Of the other web frameworks mentioned below, have you tried any of them? If so, which ones, and what did you like about them? What didn't you like?

I briefly looked at WebWork, JSF, and Wicket, although probably too little to be able to say anything worthwhile. I'll try anyway though.

I like the simplicity of WebWork, but I don't like its template approach. I also miss encapsulated components.

JSF's tool support is impressive, as is the vendor adoption. I think that their approach to hide everything related to the HTTP from the developer is a mistake, however. The stateful web components that it offers are interesting in some ways, but I think that with RIA and Ajax these components aren't actually needed anymore and alienate the developer too much from the web.

Wicket's strength is its pure Java approach, but that's also its crutch. Configuration and declaration can't be summarized and waived away by stating that "XML sucks." Wicket doesn't provide a general overview of the navigation and state transitions. In such, Wicket builds vertical component trees and totally neglects the horizontal relationship and state transitions. This not only leaves the developer in the cold for these important concepts, but it also makes a Wicket application much more difficult to introspect and read.

## 4. What is the future of your web framework? What's coming that'll make it easier for users to develop with? Do you support Ajax natively? If not, are you planning on adding support?

We're working on adding annotation support for declarations of certain parts in the framework. We will also be adding some features that make the development and use of widget components easier and more intuitive. Continuations will also receive some major improvements to make them usable beyond entrance methods. Last, RIFE/Crud will obtain support for other interface styles, and some extensions to RIFE's site structure will make RIFE/Crud completely capable of integrating into any existing RIFE state transition.

About Ajax: We just integrated DWR in the last release and will probably stick to that for Ajax support. It's very easy to use Dojo, Scriptaculous, or Prototype on the client-side and interface with RIFE in a RESTful manner.

## 5. Are there myths about your framework you'd like to challenge? If so, which ones?

1. RIFE is "XML heavy."

2. RIFE is a "continuations server."

3. RIFE reinvents the wheel and offers nothing new.

4. RIFE's template syntax is horrible and too simplistic/amateurish.

5. RIFE is a request-based framework.

6. RIFE has too many features, requiring you to learn everything at once.

## 6. What do you think of Ruby on Rails?

RoR is one of the best things that could happen to the Java community because at least alternative approaches and meta programming are now getting the credibility they deserve. Technologically though, I think that RoR is nothing special and that all its benefits come from the Ruby language.

I really don't like:

- Its templates.
- Partials (or what RoR sees as being components).
- Scattered handling of URLs.
- The fact that Active Record provides a DSL from the database schema and not from the domain model.
- Lack of l10n and i18n support.
- Manual handling of state transitions.
- Its inability to run on the JVM (so far).
- The fact that scaffolding generates real code.
- The fact that Ruby lacks tools and IDEs.

## Seam

### 1. What is your framework's "sweet spot," and for what type of projects should it strongly be considered?

The sweet spot is applications with rich user interactions: conversational state, relatively complex navigation and/or business process management. Makes it amazingly easy to implement multi-window operations, back button support, and multiple workspaces in a single window, stateful navigation. Integration of business process as a first-class construct is totally unique.

Seam also makes it easy to build data-driven applications that use O/R mapping at the backend.

- *The* solution for people adopting Java EE 5 standards (JSF, EJB3).
- *The* solution for applications that support multitasking (multi-window/multi-workspace).
- *The* solution for BPM-heads.

### 2. What type of scenarios does it not fit in to? Would you recommend another framework in this scenario? If so, which one?

- It's not for splashing data from a table onto a webpage. Use PHP or Ruby on Rails for that.
- It's not (at least not right now) for people who want designers to write HTML with embedded presentation logic. Use Tapestry or Wicket for that. (It is great for people who want to write semantic HTML with CSS.)
- It's not for people using JDK 1.4.
- It's not for people who are totally happy with Struts.

### 3. Of the other web frameworks mentioned below, have you tried any of them? If so, which ones, and what did you like about them? What didn't you like?

#### *JSF*

Love the eventing/interaction model. Love the use of EL to bind model to view. Don't like the XML (why no annotations?). The managed beans component model is simplistic (although the idea to use POJOs is great). Creating new custom controls is overly complex. JSP sucks, so use Facelets.

#### *Tapestry*

Very nice. The form validation stuff is *killer*! Templating approach was very innovative in its time. The non-POJOness of the component model is a showstopper for me.

#### *RIFE*

Just Plain Weird, but *very* innovative and interesting. I want to learn more. Developers need to stop trying to reinvent *everything* though. I doubt this will ever be mainstream but maybe it can seed the community with new ideas.

### *Struts*

Innovative in its time but has failed to reinvent itself. It was always way, way too overly complex to bind model data to the view. The course-grained event model was OK at the time, but fine-grained events are a better approach today.

### *WebWork*

Struts++. Missed its time. XWork was some great thinking at the time, but it is dated now.

## 4. What is the future of your web framework? What's coming that'll make it easier for users to develop with? Do you support Ajax natively? If not, are you planning on adding support?

### *Seam 1.0*

- Portal support.
- The Seam Remoting Framework (Ajax).
- Some really cool stuff for templated messages.

### *Future*

- Integration with the ESB.
- Integration with scheduling/calendaring and asynchronicity.

## 5. Are there myths about your framework you'd like to challenge? If so, which ones?

1. JSF can't handle GET requests.

2. JSF can't do redirect-after-post.

3. JSF doesn't work with REST.

None of these statements are true.

## *6. What do you think of Ruby on Rails?*

It's a great alternative to PHP.

It could compete with Java in the low end if it had a decent persistence layer.

## Spring MVC and Spring Web Flow

### 1. What is your framework's "sweet spot," and for what type of projects should it strongly be considered?

### *Spring MVC*

#### Solid, reliable and flexible

Spring MVC aims to be a solid platform upon which developers can build without worrying about on-going change. The number of features in Spring MVC to support i18n, file upload, exception handling, and so on provides users with the confidence that they can choose Spring MVC and know it will not let them down when the time comes to add more complicated features to their applications

#### Best Practice

Encapsulates best practices and provides an ideal base on which to build applications whilst not having to worry about doing things "the right way."

#### Spring Integration

Builds on the core Spring Framework — doesn't just stop at the web tier.

#### Support

The Spring community is large and active, providing developers with the means to obtain the support they need. Organizations can buy commercial services from companies like Interface21 to reduce the risk of adopting any new product.

#### Struts +1

Struts developers migrating to Spring MVC become more productive without having to learn a new paradigm.

#### Project types

Spring MVC is a foundational framework. It integrates a number of different technologies and as a result is applicable to a wide range of project types. It should be considered a strategic base platform for web application development.

### *Spring Web Flow*

#### Embeddable task-execution engine

Developers using Spring Web Flow use it to orchestrate stateful tasks with a linear progression.

#### Abstraction

Spring Web Flow abstracts a lot of common issues away from the user, such as conversation state management, flow enforcement, and back button handling. This level of abstraction allows developers to focus on the task at hand and not on the underlying plumbing.

### Project types

Spring Web Flow is a specialized product focused on solving one problem really well. It should be considered for the parts of a site that execute controlled navigations to complete business tasks. SWF plugs in to Spring MVC, Struts, and JSF as a complement (an embeddable task-execution engine addressing the "C" in MVC).

## 2. What type of scenarios does it not fit in to? Would you recommend another framework in this scenario? If so, which one?

### *Spring MVC*

Spring MVC addresses request-driven processing at the HTTP level. It does not aim to provide a component model in which individual components embedded on pages are responsible for their own content rendering and event callbacks. We recommend JSF or Tapestry for scenarios in which component-driven UI development adds value.

### *Spring Web Flow*

Spring Web Flow is designed to solve the problem of enforcing linear page flow. It should not be used for "free browsing" scenarios in which there is no task for the user to complete. As an embeddable task-execution engine, Spring Web Flow can be embedded within both request-driven (Struts and Spring MVC) and component-driven (JSF) environments.

## 3. Of the other web frameworks mentioned below, have you tried any of them? If so, which ones, and what did you like about them? What didn't you like?

Yes. In addition to providing Spring MVC, the Spring Framework ships integration support for Struts and JSF. We recognize Struts as the "old guard" in this space and provide integration to make it easier to use Struts and Spring together. We are hopeful about the potential of JSF as a standard now that the community is investing more in building upon it (for example, Facelets and Spring Web Flow).

## 4. What is the future of your web framework? What's coming that'll make it easier for users to develop with? Do you support Ajax natively? If not, are you planning on adding support?

### *Spring MVC*

- View simplification: JSP form tags.
- Rich configuration schema for all core MVC infrastructural elements (controllers, handler mappings, view resolvers, etc).
- Convention-based defaults for controllers, model object names, and view resolvers. Both Rails and Stripes are influences.

- Improved data binding and validation (ability to bind directly to fields and to generic data structures such as maps).
- Portlet support.

Beyond 2.x, we will continue to monitor user feedback and develop along the paths that our users request.

### Regarding Ajax

Spring MVC has Ajax support via the DWR library. We are constantly assessing what our users want when it comes to Ajax in Spring MVC, and we will ensure that their needs are met.

## *Spring Web Flow*

- Rich configuration schema for configuring the flow-execution engine.
- Convention-based defaults for automatically exposing bean method return values to view templates. For example, if SWF invokes the method "Collection getPeople(SearchCriteria)" on the "Phonebook" service-layer facade, the returned collection will be exposed to the view with the name "people".
- Transparent variable resolution support for searching multiple namespaces to resolve a flow-variable value. For example, instead of referencing a qualified method binding expression: method="placeOrder(${flowScope.order}))", developers can simply write: method="placeOrder(${order}))".
- View template support, including automatic bookmark-friendly flow URL generation and a <spring:webflow> tag library.
- Data mapping enhancements to further abstract a flow definition from its execution environment; for example, support for a declarative mapping of flow input attributes from the request path.
- Transparent conversational component lifecycle management with *the option* of full control over conversational state when needed (via a memento-based or an annotation-based strategy).
- Monitoring in-progress conversations via JMX.
- Conversation history subsystem (to support breadcrumbs, automatic restart of expired conversations).
- Packaged Hibernate and Acegi integration (long session per flow and flow-level security, respectively).
- Additional integration with other frameworks.
- Additional samples demonstrating JSF integration in a component-rich environment.
- Patterns for implementing long-lived flows (that involve a durable store like a DB).
- Flow inheritance.
- Support for metadata-driven continuation invalidation to prevent backtracking once a certain step of a flow is complete. For example, after a flow exits a "non-repeatable" view-state, going back to any previous view state will be disallowed.
- Support for applying flexible flow input-mapping rules. For example, a REST-style mapper might map the URL http:/localhost/phonebook/matt to a new execution of the "phonebook" search flow, mapping the "name" input attribute from the request path. This input attribute would be consumed consistently by the flow execution regardless of

whether it was launched by a browser via a top-level URL or spawned internally as a subflow from another flow.

#### Regarding Ajax

Spring Web Flow ships a sample application that demonstrates an Ajaxian-integration strategy using the Prototype JS library.

### 5. Are there myths about your framework you'd like to challenge? If so, which ones?

Spring MVC is difficult to configure. With Spring 2.0, this is largely solved with the customized, schema-based configuration.

### 6. What do you think of Ruby on Rails?

### *Spring MVC*

RoR is a very interesting framework, and it is certainly naive to discount it out of hand. As with any new technology, you have to take the initial hype with a pinch of salt and focus on the core technology instead. We have looked at RoR closely and — where needed — we have embraced the "RoR-way." My biggest gripe is that some of the RoR features sound great, but the execution isn't always exactly as it should be. A case in point is rendering the plural form of variable names in conventions. We added conventions to Spring MVC, and I thought that the method of rendering the plural form was great. I wrote the code for it and then started to ask around for feedback. As it turns out, users seem to find it very messy and non-deterministic. When I thought about it, this actually makes sense. English has some ridiculous rules for plurals, making some of the conversions strange (person -> people), especially if English is not your native language. In the end, we opted for adding "List" to the end of the singular so that "person" becomes "personList". The end result is the same — you have a set of conventions for model parameter naming. Ours is a little less "cool," but it certainly reaches a wider audience.

## Stripes

### 1. What is your framework's "sweet spot," and for what type of projects should it strongly be considered?

Stripes' sweet spot is probably very similar to that of Spring MVC, WebWork, etc. While it has some innovative functionality, one of Stripes' biggest goals is to take the functionality of a high-quality action-oriented framework and implement it in a way that is much simpler to ramp up on — and much more productive in day-to-day development.

That said, one area where Stripes really shines is in applications with lots of complex data interactions. Its type conversion, binding, and validation are very powerful and make it easy to manage large, complex forms and map them directly to domain objects, etc.

## 2. What type of scenarios does it not fit in to? Would you recommend another framework in this scenario? If so, which one?

One area I can think of (and this is generally true for action-oriented frameworks) is for user interfaces that are non-AJAX driven and involve several unrelated or loosely related stateful components on the same page.

As an example, imagine a Google homepage where you could pop open multiple search panels and search and page through results etc. in each panel. Each time you hit next on one panel, the whole page would be refreshed, the panel you clicked on updated, and all the other panels left as they were.

Managing a UI like this in a way that is multi-window safe can be very difficult. This is the problem that component-oriented solutions like Wicket and JSF were designed to solve. My view is that for 90%+ of web applications, component-oriented frameworks are overkill, and that for another 9%, AJAX is more a more effective way to manage a stateful UI as described above.

If I were in a situation in which I needed to develop such an application for a client and the client didn't want to use AJAX, I would probably look at Wicket. I've looked at Wicket before (although not used it in anger) and am impressed with the quality and focus on simplicity in the project.

## 3. Of the other web frameworks mentioned below, have you tried any of them? If so, which ones, and what did you like about them? What didn't you like?

Of the frameworks on the list, I've used Struts for developing commercial applications (not my choice), and I've tried out WebWork and Spring MVC. Personally, I found very little to like about Struts. I spent a lot more time working around issues with Struts than I gained in productivity from using Struts.

WebWork appears to be much higher quality, but its documentation is pretty sparse, making it hard to ramp up quickly or to find out how to use advanced features.

My main problems with the existing frameworks that I've tried are three-fold:

- The frameworks expose too much complexity to the developer, especially for simple applications. A developer should have to know a minimum about a framework to get started. Existing HTML/JSP and Java knowledge should be enough for the most part.
- Frameworks rarely do the little things that together can make developers' lives much easier, like providing really good error and warning messages. Another example: How many web frameworks make you specify a single format for parsing dates? Does Excel force you to pick one format for entering all dates? Would it be tolerable if it did?
- Documentation is usually pretty poor. In fairness, a lot of people working on OSS are doing it for free on their own time, and their passion is software development, not documentation. But a good framework with poor documentation is much more difficult to use.

## 4. What is the future of your web framework? What's coming that'll make it easier for users to develop with? Do you support Ajax natively? If not, are you planning on adding support?

An under-the-covers kind of change that is coming in Stripes 1.3 is the introduction of an Interceptor framework. The idea is that you can easily intercept any request at well-defined lifecycle stages without having to break out an AOP tool. The Spring integration is being migrated to an Interceptor (after ActionBeans are instantiated, intercepted, and wired in any dependencies using Spring). It should enable all sorts of useful plug-ins and extensions. Security checking based on ActionBean classes and events is another obvious choice.

There will also be some enhancements to the validation system. One makes it easy to have multiple validation methods per ActionBean and decide when each should run. Another allows developers to enable/disable individual required field checks based on the event being accessed within an ActionBean.

Stripes has some AJAX support, and this is definitely a direction for future expansion. Unlike most other frameworks, Stripes' ActionBeans don't return symbolic results; rather, they return objects that are executed. This makes it easy to build results that stream data back to the client, as is usually done in the AJAX model (stream back XML or JSON to be used in client-side processing). Stripes has an interesting JavaScriptResolution that can take any Java object and "serialize" it to a set of JavaScript statements that reconstruct the object-web's state in JavaScript on the client. It even manages circular object webs!

I'm currently looking for a good JavaScript/AJAX framework to integrate with and recommend for use with Stripes. My problem is that — like with a lot of open source — I'm still trying to find one with good documentation, and I don't want to set users up with something that is difficult to understand and poorly documented.

## 5. Are there myths about your framework you'd like to challenge? If so, which ones?

1. Stripes uses annotations instead of XML, so it's not really removing configuration; it's just moving it around.

There are two parts to this. In more recent versions of Stripes, there is support for generating the URLs and event names for action classes based on class names. Stripes uses a fairly sensible default pattern, and it's easy to change it to use a different one. So, really, you only need to annotate classes and events when you want to step outside a predictable scheme. And because the class names don't show up elsewhere (in configuration files, for example), it's often just as easy to change the class name.

Also, which of the following is more repetitive and error prone:

```
   @Validate(required=true, minvalue=1, maxvalue=1024)
   private int myField;
```

or

```
   <field property="myField" depends="required,integer,intRange">
      <arg position="0" key="myField"/>
      <arg position="1" name="intRange" key="${var:min}"
resource="false"/>
      <arg position="2" name="intRange" key="${var:max}"
```

```
resource="false"/>
      <var><var-name>min</var-name><var-value>18</var-value></var>
      <var><var-name>max</var-name><var-value>65</var-value></var>
   </field>
```

2. Annotations mean you can have only one set of configurations. First, I have to believe that 90%+ of actions have a single set of configuration! We're talking about classes that are built to service a specific web UI. However, Stripes has two ways out of this "one configuration only" myth.

Stripes looks for annotations starting at a class' most distant parent and comes down the hierarchy. Annotations on child classes override those on parent classes. Want to use the same ActionBean with two sets of validations? Subclass it and specify different validations.

Another change that's coming in the 1.3 release allows validations to be selectively applied based on the UI event being serviced. It's backward-compatible, which means that if you don't need this capability, you don't have to be aware of it. If you need it, it'll be there.

### 6. What do you think of Ruby on Rails?

I think there are a lot of good ideas in RoR that the Java community can learn from, and there are some things it can learn from us. Stripes has been compared to the front half of RoR for its focus on minimal configuration and sensible defaults and generally just for doing things that the developer would want and expect. However, when I look at RHTML, it reminds me of the bad old days of huge scriptlet blocks in JSPs.

Along similar lines, I tend to think that one of the reasons to take to ActiveRecord is that it's a fairly simple concept that is well implemented. It's much easier to understand and use ActiveRecord effectively than it is to understand and use Hibernate or a more complete O/R mapping tool. If folks in the Java space spent as much time designing easy-to-comprehend-and-use frameworks as the Ruby community does, RoR wouldn't be making such big waves.

## Struts Action 1

### 1. What is your framework's "sweet spot," and for what type of projects should it strongly be considered?

Documentation, broad user base, books, and support. If you want a framework that is very well known and easy to hire for and train on, then Struts is good. Although other projects, I'd say, are technically superior in different ways, the gap is smaller than folks tend to think. Really, the web tier is pretty easy and straightforward.

### 2. What type of scenarios does it not fit in to? Would you recommend another framework in this scenario? If so, which one?

If you need portlets or complex pages with lots of things going on, Struts either wouldn't work, or it would be too tedious. A component framework would work better in the latter case. I'd probably pick a better supported, more well-known framework like JSF or possibly Tapestry/Wicket.

### 3. Of the other web frameworks mentioned below, have you tried any of them? If so, which ones, and what did you like about them? What didn't you like?

I've tried most and find them good in their own way.

### 4. What is the future of your web framework? What's coming that'll make it easier for users to develop with? Do you support Ajax natively? If not, are you planning on adding support?

Struts Action 2, based on WebWork 2, will open up new possibilities. It already supports Ajax components well, but we are looking to add easier development, JSF support, better continuation support, and scripting language support.

### 5. Are there myths about your framework you'd like to challenge? If so, which ones?

Struts is so outdated or "uncool" that it isn't worth using. Really, you can do anything you want with Struts, either out of the box or through extensions. I think you'll be much more limited by your development team than your web framework.

### 6. What do you think of Ruby on Rails?

Good example of a project focusing on developer productivity without resorting to D&D tools. I hope to bring many of its lessons to Action 2.

## Tapestry

### 1. What is your framework's "sweet spot," and for what type of projects should it strongly be considered?

I think Tapestry's real strengths come through on medium- to large-sized projects (although you can have fun even on a single-page application). Those are the projects where you'll get leverage by being able to easily create new components. Tapestry so doesn't care about where the data comes from, and a lot of the "project type" is based on that aspect (i.e., CRUD vs. RSS feed vs. etc.). I feel Tapestry's really easy IoC integration (with HiveMind and/or Spring) is a huge bonus for teams that want to produce strong, testable code.

### 2. What type of scenarios does it not fit in to? Would you recommend another framework in this scenario? If so, which one?

I'm not aware of another Java framework that has significant advantages over Tapestry. There's the spreading shadow of RoR, but not having used it myself, it's hard to say what a real project in RoR would be like (vs. one in Tapestry). I'm not well versed in RIFE, which seems to have a lot going for it, especially an ActiveRecord-like access layer.

If you have a compelling need for a very specific URL format, you will be fighting Tapestry over that, and you may not win.

## 3. Of the other web frameworks mentioned below, have you tried any of them? If so, which ones, and what did you like about them? What didn't you like?

Haven't done too much work outside Tapestry in the last two years. I'd love to learn some others, especially RoR, but time is SO limited.

## 4. What is the future of your web framework? What's coming that'll make it easier for users to develop with? Do you support Ajax natively? If not, are you planning on adding support?

Tapestry 4.0 has very good Ajax support via the Tacos library. Tapestry 4.1 will make a number of internal API changes to make that support even better.

Tapestry 5 will be significantly advanced over Tapestry 4 or anything else; planned features are

- No more abstract classes.
- No inheritance imposition.
- Annotations directly on fields rather than on (abstract) methods.
- No XML, just templates and annotations.
- Smart class loader; pick up changes to classes automatically and efficiently.
- Minimal APIs, beyond annotations.
- Aspect-oriented construction of components, using mix-ins.

## 5. Are there myths about your framework you'd like to challenge? If so, which ones?

1. That it's not testable. The tools for testing existed for 3.0 and are part of the framework in 4.0.

2. That it's a one-man show; there are lots of active Tapestry committers and a very active community.

3. That Tapestry has a steep learning curve (it has a steep UN-learning curve).

4. That it's all about prettier templates; it's really all about state (allowing objects to store state instead of being multithreaded singletons and to manage transient and persistent state during and between requests).

### 6. What do you think of Ruby on Rails?

Very influential. It's hard to separate the hype-driven reaction and hype-averse counter-reaction. I do think the templates are unnecessarily ugly. I've heard many things, pro and con. I will say that even my basic understanding has influenced some ideas in Tapestry 4 (that will take greater root in Tapestry 5), such as abandoning fixed APIs and having the framework adapt to the user code (in Tapestry terms, the flexibility of listener methods in Tapestry 4 vs. Tapestry 3).

RoR represents a limitation on choice:. You do it the RoR way, or you are in for a big struggle. However, within the confines of the RoR naming and coding conventions, it looks like you get a lot for your buck. That's a very compelling value statement, and it's surprisingly similar to the Microsoft philosophy (use our tools or else). The Java philosophy can be expressed as "we don't know how you will precisely use our tool, so we'll support any possibility, but in a vague way that requires you to understand our tool to a high degree." Not just Tapestry, but JEE, Spring — pretty much the entire stack. So, I think it's very important for the Java world to shift at least part of the way in the direction of providing solutions, not just tools.

## Trails

### 1. What is your framework's "sweet spot," and for what type of projects should it strongly be considered?

The sweet spot for Trails is probably applications in which a web interface to a persistent domain model is needed. Trails is about quickly giving you an interface to your domain model with no additional code required other than the POJOs themselves. Trails also lets you customize the appearance and behavior of the interface and includes validation, internationalization, and security. It does all this without generating any java source code at all, so people who hate code generators (like me) will be comfortable with Trails.

### 2. What type of scenarios does it not fit in to? Would you recommend another framework in this scenario? If so, which one?

Trails in some ways is about "good enough" software. It lets you very quickly produce an application and put it in front of a customer and say, "Is this good enough?" This requires a definite change in process, and it is certainly not the right approach for all situations. For applications in which a very customized UI is required, Trails will be less of an advantage. There still may be pieces of Trails that can be leveraged, however. In my current gig, we are using a mixed approach. We have some admin screens that just need to be "good enough" and some other screens where a more customized approach is used. We're using Tapestry, Hibernate, and Spring to build those screens, which are the core pieces that Trails builds on.

There are also applications that are mainly reporting and aren't as interactive, and they wouldn't really benefit from Trails. I would be inclined to look at the Eclipse BIRT project for those.

### 3. Of the other web frameworks mentioned below, have you tried any of them? If so, which ones, and what did you like about them? What didn't you like?

I've used Struts a lot. It was a great framework for its time. The main issue I had with it was that it didn't really do type conversion in a way that let us bind input elements directly to our domain model.

I investigated JSF some, and it made a lot of improvements over Struts, but building custom components with it seemed quite difficult. Tapestry makes building components incredibly easy, especially building high-level components that are comprised of other components such as Trails uses.

### 4. What is the future of your web framework? What's coming that'll make it easier for users to develop with? Do you support Ajax natively? If not, are you planning on adding support?

Well, one of the nice things about Trails is that we are "standing on the shoulders of giants," so to speak. Tapestry is making great strides in integrating Ajax functionality, and incorporating that work into Trails is not difficult. We need to create more examples that show how to do this. Work is also being done to make Trails easier to "drop into" an existing project.

Other future features for Trails include instance-based security (currently, we have role-based only) and method invocation.

### 5. Are there myths about your framework you'd like to challenge? If so, which ones?

That it is a port of Ruby on Rails. I'm sure my own choice of name is what started this myth. Trails is inspired by Rails, but it is not at all a port.

### 6. What do you think of Ruby on Rails?

I think there is a tremendous amount we can learn from Rails, and it does a huge service to the developer community by raising the bar for how easy web development can and should be.

## WebWork

### 1. What is your framework's "sweet spot," and for what type of projects should it strongly be considered?

For general teams, WebWork usually fits in best with small teams that are willing to get their hands dirty and learn a lot about the open source tools they use. WebWork is not meant for the "armchair programmers" who prefer drag-and-drop development. WebWork rewards users who take the time to learn the framework. For example, those who know the ins and outs (and read the reference docs, which are quite good) could easily make an ActionMapper and a Configuration implementation that provide for great use of "convention" in place of configuration. My most recent example of this makes URLs like "/project/123/suite/456/test/create" map to "com.autoriginate.actions.project.suite.test.CreateAction" and automatically pass in the parameters projectId=123 and suiteId=456. Results are assumed to be "[action].jsp" or "[action]-[result].jsp", or can be defined using annotations.

The point is, WebWork is best utilized when used as the basis for a framework within your application. It works great out of the box too, but that's not the best sweet spot currently.

Beyond the generalities, WebWork is an absolute must for anyone who is developing a shipping product that will utilize plug-ins and extensions. Think JIRA, Confluence, or Jive Forums. Because of WebWork's broad support for template languages (Velocity and FM), including complete tag support, writing modules that can be plugged in is much easier. A single jar can include the actions and view files. The end result is very nice: In Confluence, you can upload a jar through the admin UI, and the plug-in is deployed dynamically. Again, this was possible because Atlassian made the commitment to have a solid understanding of the framework.

## 2. What type of scenarios does it not fit in to? Would you recommend another framework in this scenario? If so, which one?

Like any action framework, WebWork is poor at handling state and wizards. If you're writing lots of long-running wizards, perhaps JSF is the way to go. I'd also say that until the documentation improves (the reference docs are top-notch, but the tutorials, cookbooks, and FAQs are very poor), novice users might want to stay away unless they have a WebWork champion leading the team.

For a really simple application, I'd recommend just plain JSP or perhaps Spring MVC if the users are comfortable with Spring. But generally I'd say that when it comes to action frameworks, WebWork is the best choice.

## 3. Of the other web frameworks mentioned below, have you tried any of them? If so, which ones, and what did you like about them? What didn't you like?

I've tried out RIFE, Spring MVC, Struts, Spring Web Flow, and just a tiny bit of Tapestry. I found the *concepts* of RIFE are nice, but the implementation is completely annoying. Not surprising considering that Geert still uses "m_" as his field prefixes. It just doesn't "feel" like Java. The templates irritate me to no end also.

Spring MVC is OK, but again, it's just a very simple action framework implementation. WebWork's data binding (type conversion in the WebWork world) is nicer and almost always automatic (as opposed to writing your own property editors). The support for alternative view technologies in Spring is too limited, and the UI tags that WebWork provides just don't exist in Spring. You can write your own tags pretty easily using JSP 2.0, but without support for other languages, it becomes a non-starter.

Spring Web Flow: The XML drives me nuts. It's overkill, and despite Keith's claims, it doesn't "support any web framework." It is a web framework that sidesteps 99% of

WebWork, Spring MVC, Struts, or anything else. That's not to say it's a totally bad idea, but we should be honest that it is effectively a complete framework.

Tapestry is neat, but the HTML attributes (jwcid) are pretty annoying. I understand the idea, and in practice it sometimes turns out quite nicely (Shale has something similar). But in reality, I've found that "HTML/CSS developers" and "app developers" rarely are separated like Tapestry likes to pretend they are. In fact, with AJAX gaining momentum, I would argue that more and more "application logic" is being embedded in the UI; therefore, the Tapestry model makes even less sense.

## 4. What is the future of your web framework? What's coming that'll make it easier for users to develop with? Do you support Ajax natively? If not, are you planning on adding support?

In Struts Action 2.0, expect to see:

- Better docs.
- Support for standards (if we can, we'd like to drop OGNL in favor of JSTL, but we'll only do so if we don't lose functionality).
- Enhanced AJAX support in the form of more widgets: auto complete, etc.
- Address configuration hell; perhaps offer various options like the one outlined above.

## 5. Are there myths about your framework you'd like to challenge? If so, which ones?

1) It requires a lot of configuration: I just showed above that you can make the conventions even nicer than Rails (Rails conventions don't allow for nested controllers).

2) Docs are bad: This is only partly true. The reference docs are great, and getting started is really as simple as downloading the distribution and copying the "starter" webapp (it's a skeleton app to get you started).

## 6. What do you think of Ruby on Rails?

The most important thing I can say about this question is that Ruby on Rails can't really be compared to most of the frameworks, except for **possibly** RIFE and maybe Seam (don't know enough about Seam). Comparing Ruby on Rails to WebWork is like comparing Hibernate to JDBC. One is a full stack, and the other is just part of the stack.

The second most important thing I can say is: DHH is a jerk, but a smart jerk. He's effectively turned a non-issue into a debate, all to make a name of Rails. You mentioned this in your blog recently re: marketing.

OK, with that out of the way, here are some random thoughts:

- I've used Rails to build a small app. I threw it away and rewrote it in Java in less time. I would say that Rails does let you get 80% of your app done rather quickly; the last 20% still takes much longer than the first 80% took. This is classic application development. The scaffolding stuff is very cool, especially how the ActiveRecord changes the classes at runtime. But in the end, you end up having to write custom UIs, custom validation rules, custom security mappings, etc. Just because you can have a CRUD-based app "working" in 30 minutes doesn't mean that you will continue at that pace.

- The web framework part of Rails is horrible compared with WebWork. It really isn't even a competition. It is much closer to Spring MVC in terms of simplicity/offerings.
- The integrated stack is *amazing*. They did a great job here, and there is room for Java to offer something similar. WebWork could easily be the web stack, but the persistence solutions aren't very promising right now. The biggest issue is that while WebWork and SiteMesh, for example, support configuration reloading and even dynamic class reloading, Spring, iBatis, and Hibernate do not. They need to step up to the plate and at a minimum support configuration reloading before a good stack similar to Rails can be offered. Similarly, Hibernate and iBatis offer poor hooks into the guts of their framework like WebWork does. With a single class, I was able to get rid of the requirement for xwork.xml in WebWork. That cannot be said for the persistence libraries. Once they get their act together, perhaps a complete stack can be pushed out that does all the things Rails does.
- Claims about "5X faster" or "10X faster" are stupid. Solid engineers know that when developing a product, the biggest area where time is spent is *not* in writing code, but in thinking about the code. That is, defining requirements, getting feedback, identifying a market, designing the database tables, mapping out a user interface. No matter what language you write in, you need to *think* for a long time. No framework or language is going to speed that stuff up.
- Finally, more than anything, Rails offers this: A well-written stack in a scripting language. Most other frameworks in Python, Perl, and especially PHP haven't been done well. I think a lot of that has to do with the fact that "PHP monkeys" typically weren't high-caliber engineers. By providing a good framework in a scripting language, people can write well-designed apps and see the results right away (because of the nature of the scripting language). More than "convention over configuration" or even the integrated stack, what Java needs is the ability to see changes reflected right away. A lot of open source leaders in Java honestly think that "touch web.xml" is an acceptable way to reload. This mentality has to stop. We need to not only support configuration reloading and class reloading, we also as a community need to pressure Sun to introduce *full* HotSwap support in their next JVM. Alternatively, more complex application models, such as OSGi, may need to become more embraced by the community. I'd rather prefer we didn't have to go that route.

## Wicket

### 1. What is your framework's "sweet spot," and for what type of projects should it strongly be considered?

There are 5 sweet spots in my opinion:

1. Wicket is code centric. This attracts people who regard programming as an art rather than just a vehicle to get things done. With Wicket, you are in control of how components are created and how they cooperate, and you will be able to directly use object-oriented programming in your view layer — not just some half-baked variant of it.
2. Components are truly self contained. They are easy to create (just extend the proper class) and use (just have them in your class path). They won't be in the way of other components, and easy reusability will save your project from copy 'n paste code. I would

argue that Wicket is #1 when it comes to how easy it is to create custom, reusable components.

3. Separation of concerns is clear and enforced. You use HTML (or whatever markup you use) for presentation, components for any dynamic aspects of that presentation, and models for any business logic.
4. Clean templates. Not everyone is convinced that having this is important, and sometimes page scripting — like you can do with JSP, PHP, etc. — actually works faster. But having clean templates certainly makes your templates easier to read and maintain, and if you have a separate team of HTML/CSS guys, they'll love it. And maybe it is just me, but I hate looking at HTML pages with a lot of distracting crap in them.
5. It scales well for development — just like OO does. Your code's complexity will grow linearly at most as your UI becomes more complex. It is easy to increase the number of people on your team without them getting in each other's way, and it is a viable option to hire HTML/CSS wizards to mock up your pages and let your programmers fill in the dynamic parts. And overall, using Wicket gives you very maintainable code, which will pay back on larger or longer-running projects.

Wicket is very well suited for intranet/extranet applications, where the UI is relatively complex and where you want to make the best use of your developer resources. It is a pleasure to program with, and it will help your junior programmers develop their object-oriented skills much better than most of Wicket's competitors.

## 2. What type of scenarios does it not fit in to? Would you recommend another framework in this scenario? If so, which one?

Wicket may be less suited for projects where the UI is simple and largely read-only. Page scripting might work better there, and for these projects you might consider not using any framework at all besides JSP and leave the door open for JSF when you need it.

Also, if you are looking for the ultimate in scalability, you might want to look at a framework that is focused on that and isn't state-oriented by default. From version 1.2 on, Wicket has the concept of stateless pages, so the same kind of scalability can be met as in any other framework, but it is not Wicket's recommended way of using the framework because you are partially back to thinking in request parameters again.

## 3. Of the other web frameworks mentioned below, have you tried any of them? If so, which ones, and what did you like about them? What didn't you like?

### *JSF*

Months before getting involved with Wicket, I tried to get people together to implement a non-JSP view layer. I didn't succeed in that, and I decided not to do it by myself because I did not like JSF enough to put the extra effort in it.

Pros:

- Component oriented.

- Huge industry backing.
- Tool support.
- Some of the extensions are making it much nicer to work with.
- It makes a scenario possible where you start with plain JSP and then scale up to JSF when you need it.

Cons:

- It misses the elegance that some other APIs have. JSF comes across as being very procedural, very "do this in order to get that," and it just lacks a general philosophy that makes your jaw drop.
- JSP-centric. Sure, there are ways around it, but in general the preferred template mechanism is JSP, and when you create custom components, you have to know about JSP's details.
- Lots of configuration and in general ugly to code with. I'm sure it does the job at the end of the day, but not in a way that lets me enjoy my work and be proud of it.
- Various implementations. I don't think it is an advantage at all that a framework (or specification) has multiple implementations. I prefer one implementation of something that is good and complete, and my experience with multiple implementation frameworks is that they can give you very nasty surprises once you do switch.
- Everything about it reminds me of EJB1/2, including the big marketing push, the framework being tool-vendor centric, and the way its proponents aggressively defend its shortcomings.

### *RIFE*

I haven't used RIFE, but I like the fact that it takes a different approach. I think Geert is a smart programmer with excellent marketing skills, so I definitely think RIFE is a grower. I have looked at the examples and tutorials, and it's not a framework I would love to work with, but I probably wouldn't hate it as much as I hate working with model 2 frameworks. I love RIFE for the looks of the site, the documentation, and all the neat little sub-projects. What I like less is the fact that it delivers a whole stack for you to use — personally, I prefer frameworks that are focused on one thing — although having that whole stack that might be an advantage too sometimes (à la RoR).

### *Seam*

Not really a web framework. More an extension on JSF. If you go for the EJB/JSF/JBoss stack, it's probably a good idea to use Seam. I wonder why JBoss does not seem to align to JSR 227/ 235. It seems that data binding is an important aspect of Seam after all.

I guess Seam can make your life easier when you decide to go for the stack. Yet another layer to add though, and I am not sure where you get to the point that annotations actually get in your way more than they help you.

### *Tapestry*

Nice, and it got much better with version 4. I almost introduced it as the strategic framework for the company I was working for two years ago, but a colleague (Johan Compagner — also a Wicket contributor), convinced me to take a look at Wicket. He did two projects with Tapestry and had a lot of issues with it, and we both liked Wicket's programmer-centric approach better than Tapestry's "pragmatic" approach. Tapestry is easier to scale than Wicket, and the templates are more flexible in use than are Wicket's. The price for that is high, though: things like additional configuration, abstract classes,  "rewinding," and no completely automatic state handling. A lot in Tapestry brings up the "premature optimization"' red flag for me (e.g., page pooling).

### *Trails*

Never used it, so I don't know much about it.

### *Spring Web Flow*

The framework itself is okay, but Spring Web Flow is exactly the kind of framework stacking I don't like. If I wanted to build a workflow-based application, I would use a framework that is focused on that, like jBMP (which is really great, by the way). Otherwise, I would just pick a framework that supports the kind of navigation I am looking for.

To be honest, I don't like all those "flow"-oriented solutions. They impose a documented oriented and procedural way of programming. Look at the mail client you are using or at any other app with any UI complexity (including gmail, flickr, amazon)... Are you really thinking in pages? I like to work like I do with Wicket. Sometimes you go from page 1 to page 2, but a lot of times, you have a page with tabs and borders and I don't know what on it, and you just want to replace that left panel with the search results of the query from another part of the page and reuse another panel in some other page without having to worry about what the flow looks like. And I'm glad I don't have to "flow" my Swing applications, although I wouldn't be surprised if people proposed it .

### *WebWork, Spring MVC, Struts*

Model 2 frameworks suck. I used them for years, I am a committer for one (Maverick), but I totally lost my belief in them (and so did the guy who started Maverick, by the way). I have seen projects being drained when the complexity of the UI rose. Programmers relied too much on ad-hoc session usage (try clustering that) and other hacks to work around their problems. Model 2 frameworks are highly procedural, and programmers don't learn object-orientation properly from using them. I would rather hire someone who coded Swing for a few years than someone who primarily worked with model 2 frameworks because I would expect the Swing guy to be a better coder in general. Working with model 2 frameworks means going from hack to hack and back again, and the amount of copy 'n paste code it generally results in is something that makes any serious coder sad.

If I had to pick *any* model 2 framework, I would probably pick Stripes, which at least got some of the most annoying aspects of model 2 out of the way.

## 4. What is the future of your web framework? What's coming that'll make it easier for users to develop with? Do you support Ajax natively? If not, are you planning on adding support?

We plan to go over to Java 5 completely soon. Our current version is good enough for people to use, and we plan to support it for a long time, but we want even more strong typing on the Java side with generics. For the rest, we improve, improve, improve by using it for our projects and discussing it with users on the mailing lists and on our IRC channel.

Some additions of the last few months: enhanced Spring support as a core project, sophisticated authorization support with a role-based reference implementation, native AJAX support included in our core project, URL mounting (nice URLs), stateless pages, etc.

We haven't been doing a lot of marketing lately, but our user base and participation grows, and we are still spending most of our waking hours on improving the framework.

Also, the writing of "Wicket In Action" (Manning) is currently in progress and scheduled to be published by the end of this summer.

## 5. Are there myths about your framework you'd like to challenge? If so, which ones?

There is this myth that scalability always has to be concern #1. It certainly isn't for most applications I've worked on in the past (they generally had a few hundred to a few thousand users), and the scalability issues I experienced were usually related to (over)usage of the database without a proper caching strategy and/or query optimization, concurrency problems, etc. Furthermore, the fact that Wicket uses server-side memory doesn't mean it doesn't scale at all. We made sure it clusters out of the box, and if you play by the rules, the memory load is very acceptable and predictable. At my former employee's office, we benchmarked Wicket applications and compared it with some model 2 applications and found that those model 2 applications, which roughly had the same complexity, ended up using way more server-side memory (more than 1MB in some cases) than our Wicket applications — Largely because those model 2 applications were relying heavily on ad-hoc session usage to solve the more difficult UI issues.

## 6. What do you think of Ruby on Rails?

I like what I have seen from Ruby, and if I ever have more time, I'd love to play with it more. I wasn't too impressed by the elegance of RoR, but I might have been looking at the wrong stuff. I think Ruby is interesting, and if it weren't for Wicket, I think I would probably use it by now or at least would have given it a serious try.

That said, I would miss the enormous supply of APIs/libraries we have for Java today, and some of RoR's focuses might be a bit over-pragmatic, biting you in the long term. Also, I am spoiled by those great Java IDEs with all their things like documentation hovers, code completion, (call) hierarchy selection, etc. I miss that whenever I play around with Ruby.