Die Vergangenheit
Das Präsent
Die Zukunft des Java Web-Tier

Ed Burns

Senior Staff Engineer

Sun Microsystems Inc

Agenda

- Historical Perspective
  - A Web Application is a Distributed Application
- What You Need to Know for Web Development
  - It's Still Getting Easier
- JavaServer™ Faces: The Java Web UI
  - The Power of Community
- Web Application Trends
  - AJAX, Web 2.0, SOA, and many other buzzwords

A Web App is a Distributed App

- Why?
  - Multiple Computers
  - Interconnections Between Them
  - Shared State Amoung Them
- Today's production Web apps are extremely complex distributed applications.

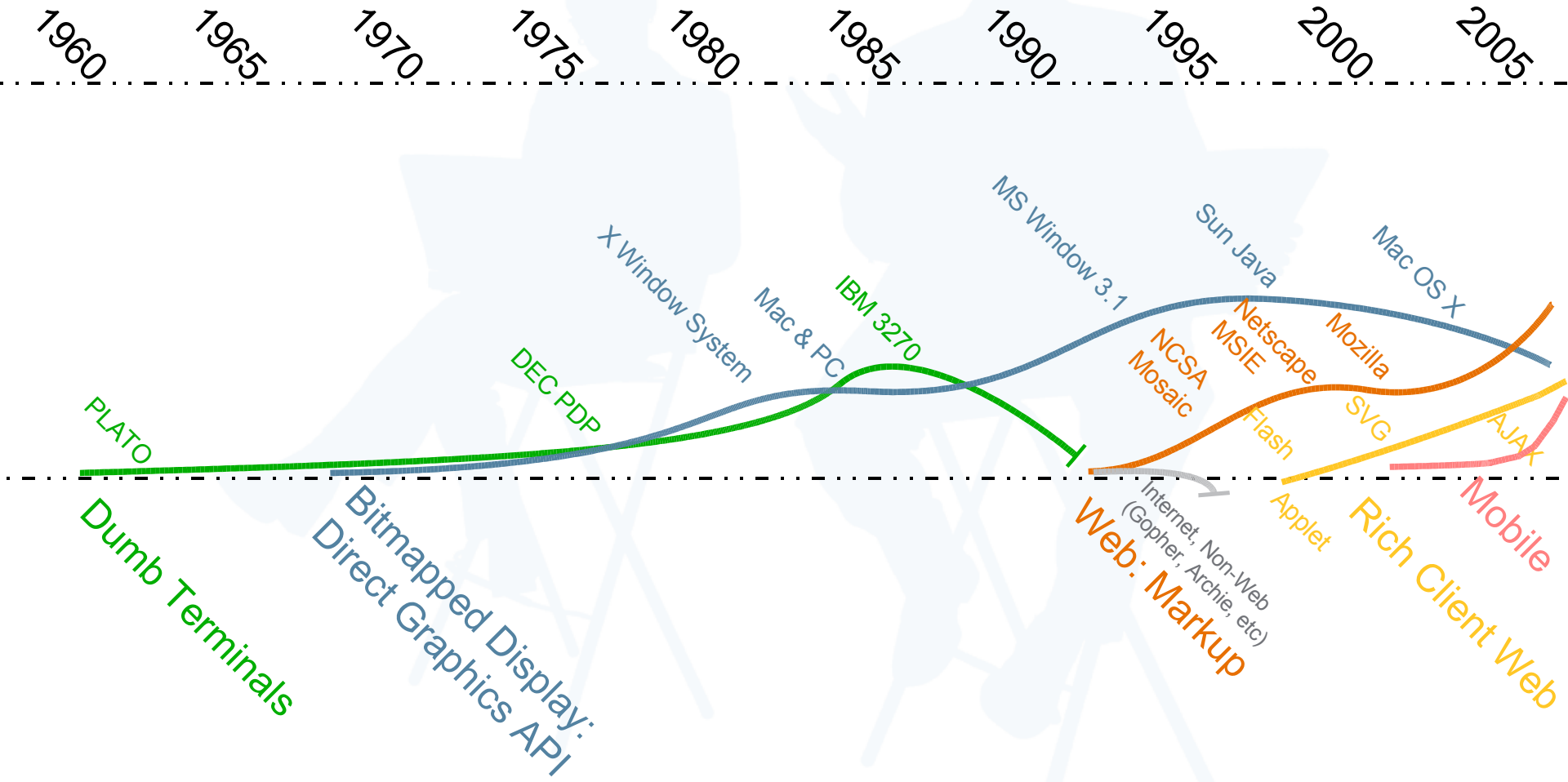
Yeah, So What?

- Why does this classification matter?
  - Because History Matters
- To understand the current state of the Java Web-Tier, we must go back to the history of distributed applications, and of the Internet itself.
- Conclusion: We're in an exciting time now, lots of opportunity in the Participation Age

What Makes A Distributed App?

- Finding the best allocation of processing tasks to processing nodes
  - UI
  - Domain Logic
  - Application Logic
  - Data Persistence
  - Communication
  - Reliability (not covered in this talk, but still very important!)

Distributed App UI History



Distributed App Back End History

- Before J2EE, no open standards for how to build domain or application logic.
- Each vendor had their own story
- Many companies wrote their own domain logic framework, and many still do.
- This situation gave rise to the drive towards SOA and the “integration product” marketplace (ie TIBCO)
- Let's look within the last ten years

Distributed App Domain Logic History

- Java
  - J2EE
    - EJB
    - Spring (“J2EE without EJB”)
  - Keel
  - Pure POJO
- Non-Java
  - .NET
  - PHP/Smarty
  - PHP/client specific
  - Rails (For the Ruby platform, Rails does it all)
  - Python/Zope
  - Python/client specific

Distributed App Application Logic History

- Java
  - JSF
  - Struts
  - Tapestry
  - WebWork
  - Wicket
  - RIFE
- Non-Java
  - .NET (“full stack”)
  - PHP
  - Zope
  - Rails (again, Rails is “full stack”)

Distributed App Data Persistence History – aka ORM

- Java
  - J2EE Session Beans (pre-Java EE 5)
  - Java Persistence (Java EE 5)
  - Toplink
  - Hibernate
  - JDO
- Non-Java
  - PHP
  - Rails
  - Python -- ZODB

Folientitel

- Font: Arial, 32 pt
  - Font: Arial, 28 pt
    - Font: Arial, 24 pt
      - Font: Arial, 20 pt

- Lorem ipsum dolor sit amet, consectetur adipscing elit, sed diam nonnumy eiusmod tempor incidunt ut labore et dolore magna aliquam erat volupat.

Distributed App Communication Layer History

- Transport Layer
  - Mainframe
    - Proprietary
    - DECnet
    - IBM SNA
  - PC
    - NetBeui
    - AppleTalk
  - Unix – TCP Sockets

Distributed App Communication Layer History

- Pre-Internet
  - Proprietary communication protocols
- Post-Internet
  - HTTP
    - Web Services
      - WS-*
      - SOAP
      - REST

History Lesson Conclusion

- Over time, each processing task has seen disruptive technologies or market forces reduce the crowded marketplace down to a small number of players
- Several interesting battles and debates are being waged, with the winner likely to emerge in the next few years
- UI
  - AJAX vs Rich Client
  - Which Java Web Framework?
- Domain Logic Modeling
  - SOA vs Web Services
  - EJB vs POJO (Spring)
- Providing Full Stack Distributed App Development
  - Java EE vs .NET vs Rails
- Persistence
  - CMP vs JDO vs Java Persistence (Java EE 5)
- History has shown that more open technologies tend to win more often

Skills Required for Distributed Applications

- For each decade starting with the 1970's, ask the following questions about the skills required to be a lead developer on a distributed application project.
  a) What did you have to know? (languages, OS's, techniques, etc)
  b) How portable was knowledge to other environments and jobs?
  c) Is a Computer Science degree necessary to fill the role?
  d) On a scale of 1 to 10, how good are the tools available to help you?
     - 01 being you had to write machine code by hand
     - 10 being you could choose from a large and competitive marketplace of high quality tools that make it very easy to develop and maintain the app.

Distributed App Skills: 1970's

- What did you have to know?
  - Deep knowledge of machine hardware and OS. All of the distributed app processing tasks had proprietary implementations, ie, IBM CICS
  - COBOL, PL/I, Fortran, C, assembley language.
  - If you happened to work at PARC on the Alto, Smalltalk.

Distributed App Skills: 1970's

- How portable was that knowledge?
  - Conceptual portability at best
  - Each vendor was “full-stack” down to the hardware.
  - Extremely homogeneous environments
  - Not uncommon for people to make a career out of one architecture.

Distributed App Skills: 1970's

- Was a Computer Science degree or equivalent experience required?
  - Yes, but degree programs not as widespread. Many vendors offered full training to users and employees.
- How well did tools help?
  - 2: Excepting Smalltalk, nothing really resembling a tool by today's standards existed, though compilers were a great help.

Distributed App Skills: 1980's

- What did you have to know?
  - Still had to know the OS, but market forces reduced the number of popular OS's.
  - C, C++, Pascal, SQL, X Window System
  - Proprietary mainframe technologies still in very widespread use.
- Was the knowledge portable?
  - Much more than in 1970's due to ANSI and other standards bodies.

Distributed App Skills: 1980's

- CS Degree or equivalent required?
  - Yes, but it was easier to get one than in the 70's
- How much help were tools?
  - 3: Each of the vendor stack products had decent tools to help getting the job done, but still nothing like today.

Distributed App Skills: 1990's pre-Web

- What did you have to know?
  - C++, CORBA, SQL, OS specific GUI Toolkits
  - MS Visual Basic significantly lowered the barriers to entry for building a distributed app
  - PowerBuilder was a popular full stack solution
- How portable was the knowledge?
  - Most vendors (except Microsoft) at least giving lip-service to code portability.
  - Much more portable than in the past.
  - Cross platform UI frameworks: Galaxy and Qt

Distributed App Skills: 1990's pre-Web

- Did you need a CS degree?
  - For the first time, thanks to Visual Basic, no, you did not.
  - Enterprise quality apps still required CS degree level skill.
- How good were the tools?
  - 6: Generally very good. The kinds of systems people were building were getting complex enough so that customers demanded better tools, so having them became a competitive advantage.

Distributed App Skills: 1990's post-Web

- What did you have to know?
  - All the pre-web knowledge was still important.
  - Borland Delphi was a good competitor to VB
  - Java shown at Third International WWW Conference in Darmstadt 4/17/1995
  - CGI, then a few years later, PHP, Perl, Python, and then LAMP

Distributed App Skills: 1990's post-Web

- How portable was it?
  - At this point, MS is the last one adamantly singing the non-portable tune, and even they have to admit the open-ness of the Web is unstoppable
  - Open standards pervade all of the processing tasks of the distributed app
  - Java's portability was more of a promise than a fact initially, but that has changed.

Distributed App Skills: 1990's post-Web

- Did you need a CS degree?
  - Definately not. Many dropped out and started dot-coms.
- How good were the tools?
  - 2: Tools for Web application development were largely non-existent at this point. Non-Web distributed application development continued to improve.

Distributed App Skills: 2000-2005

- The Web is the platform. Non-Web distributed Apps on the decline. What do you need to know?
  - HTML, CSS, and a programming environment (Java, Ruby, PHP, Python, .NET, etc.)
  - Web Services and/or SOA of some kind
- How portable is this knowledge?
  - Very, more than ever in the history of computing

Distributed App Skills: 2000-2005

- Do you need a CS degree?
  - No, but it sure helps when you get to the enterprise level.
- How good are the tools?
  - 8: Tools like Sun Java Studio Creator and Microsoft Visual Studio lead the way for ease of development of web applications

Historical Skills Analysis Conclusion

- The underlying problems of building a distributed application are still hard, but there are many high quality tools and technologies to help with all of the processing tasks.
  - Guide you down the path by constraining choice (wizards, generators)
  - Provide application and design patterns
  - Provide components to re-use
  - Graphical application development
  - Technologies that acknowledge the importance of role based development

JavaServer™ Faces in Perspective

- Complete the Java EE full stack picture by bringing a component-based stateful UI framework to the Java EE platform
  1. Tool friendly
  2. UI Component state and lifecycle
  3. Ready for use with Web Browsers
  4. Leverage JavaBeans patterns
  5. Validation, including client-side
  6. Client-device independent
  7. Fully I18N, L10N, and A11Y

Sharing: The Java Community Process

The JCP holds the responsibility for the development of Java technology. As an open, inclusive organization of active members and non-member public input, it primarily guides the development and approval of Java technical specifications. Anyone can join the JCP and have a part in its process, and you don't even have to join to contribute as a public participant.

JSF in the JCP

- June 2001
  - J2EE lacks real UI Framework story
  - JSP alone doesn't cut-it: just a view description technology
  - Struts already very popular
  - Other non-JCP technologies gaining popularity
    - WebWork, Barracuda, Cocoon, Tapestry
  - Bring the power of the JCP together to provide a best-of-breed Web UI Framework story for Java EE

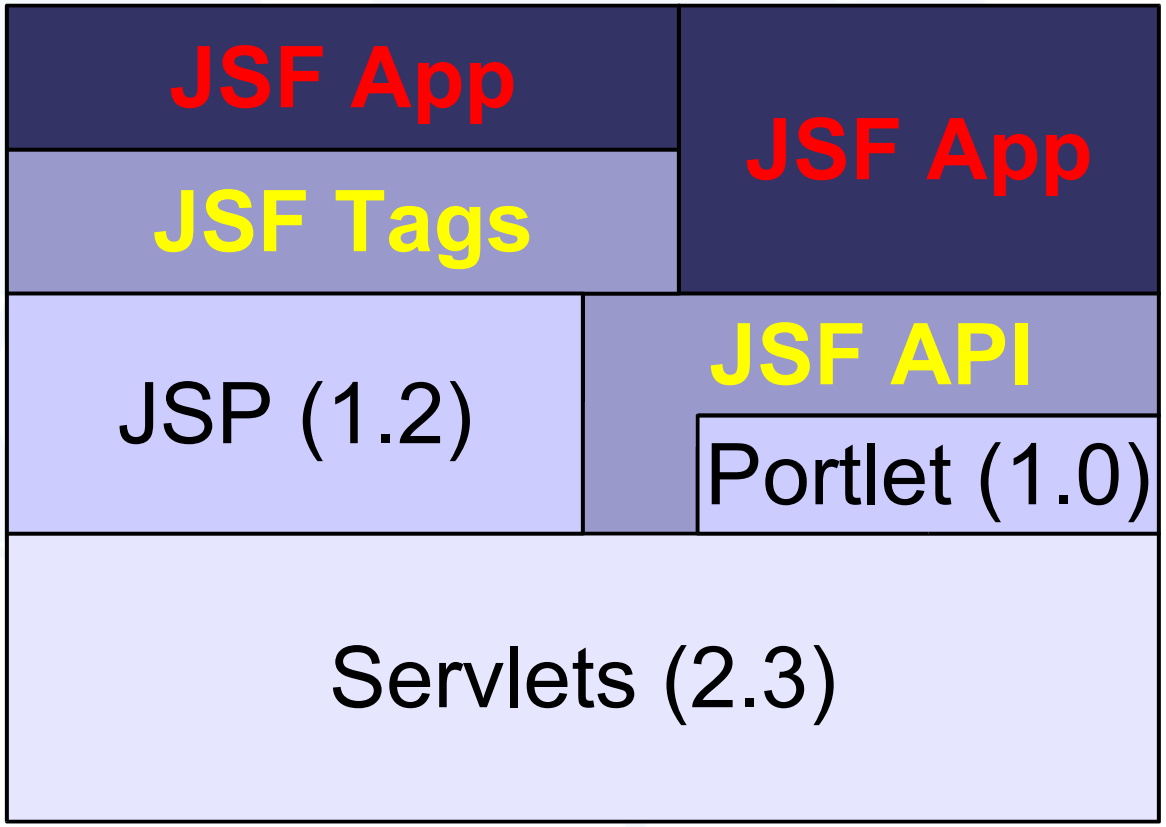
JSF in the JCP

- June 2001
  - J2EE lacks real UI Framework story
  - JSP alone doesn't cut-it: just a view description technology
  - Struts already very popular
  - Other non-JCP technologies gaining popularity
    - WebWork, Barracuda, Cocoon, Tapestry
  - Bring the power of the JCP together to provide a best-of-breed Web UI Framework story for Java EE

JSF Version History

- 1.0 on 11 March 2004
- 1.1 on 27 May 2004

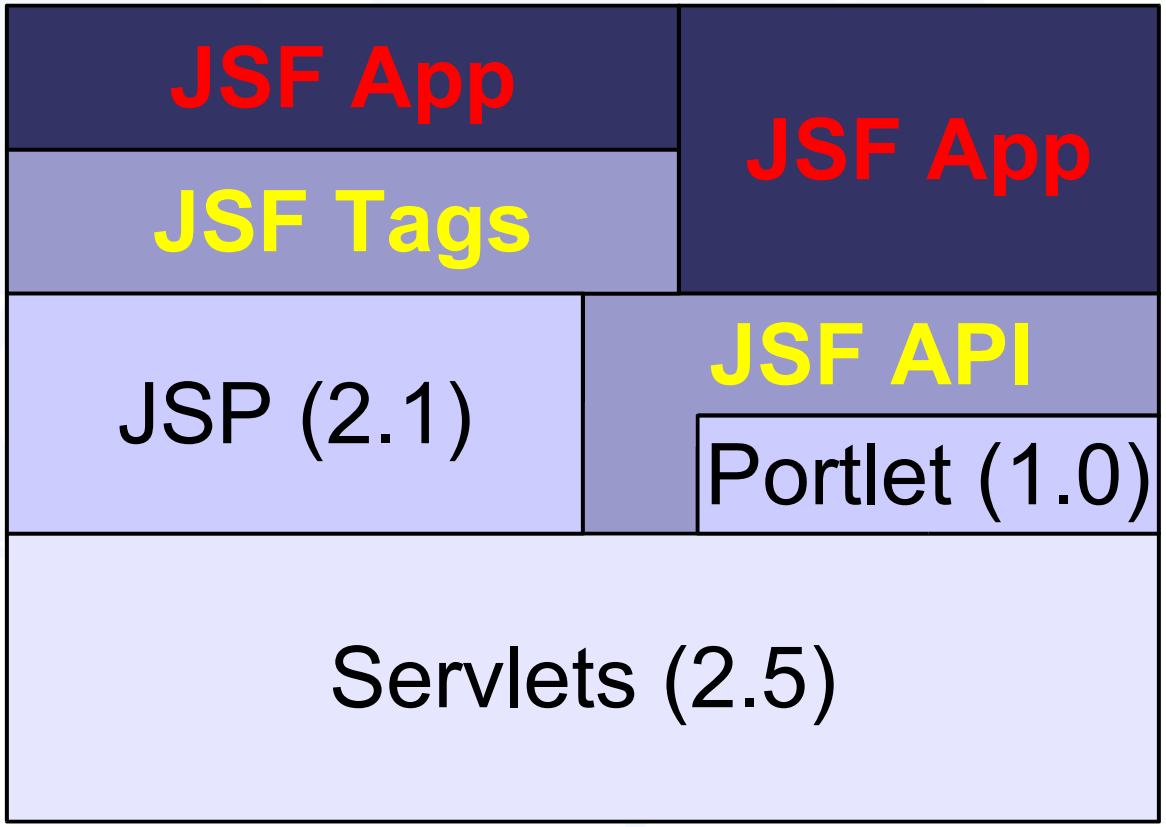
- Based on J2EE 1.3
- Implementation bundled with J2EE 1.4 SDK



JSF Version History

- 1.2 in Proposed Final Draft 25 August 2005

- A core part of Java EE 5
- Implementation bundled with Glassfish, Sun's open-source App Server



JSF Metrics for Success as of October 2005

- Real World Deployments
  - FedEx presented its JSF vision at JavaOne 2005
  - ADP using it in revenue generating products
- Ecosystem Strength
  - Developer Tool Vendors: IBM, BEA, Oracle, Sun and many other smaller players as well
  - Component Vendors: BusinessObjects, Otrix, etc.
  - Job Market (158 JSF job listings on Monster.com)
  - Books available (6 currently, more in the works)

JSF Metrics for Success as of October 2005

- Vitality of the Community
  - Forums: 21,341 messages in 5,650 topics
  - Contributors
    - Sun Impl has two active non-Sun committers
    - Popular Apache MyFaces Implementation has about 20 committers
  - Integration with other frameworks and technologies
    - Spring
    - Seam (JSF + Hibernate)
  - Websites
    - jsfcentral.com, jsftutorials.net, jsffaq.com

JSF in Comparison

- **Lots** of other Java Web UI Frameworks
  - A total of 50 at last count!
  - About four that are popular: Struts, JSF, Tapestry, WebWork
  - Web Framework Smackdown at JavaOne 2005
  - Two kinds of web frameworks
    - Request Based: Struts, WebWork, etc
    - Component Based: JSF, Tapestry, Wicket, etc

JSF Compared to Struts

- Flexibility
  - Struts form-beans can span pages, but the concept gets muddied when you do that. Faces Managed-Beans span pages just fine
  - Faces has client device independence, Struts doesn't
  - Struts tags aren't as well suited to complex widgets such as trees and tab

JSF Compared to Struts

- Model Tier Access
  - Struts uses commons-beanutils for bean hierarchy navigation
  - JSF uses the ValueBinding API.
  - Struts can create "FormBeans" for you. With DynaActionForms, you can pre-configure the initial properties of the form.
  - JSF has a much richer bean creation story. JSF Integrates well with Spring, don't know about Struts and Spring integration

JSF Compared to Struts

- Components and Events
  - Struts has no notion of components, but the struts-faces integration library allows you to use the JSF component model, and keep your Struts based back end logic.
  - Since Struts has no notion of components, it has no notion of component state. Faces has an excellent state management story supporting saving the state in the client or on the server.
  - Faces brings a JavaBeans like event model to the web, Struts has nothing similar to this.
  - JSF has dataTable support, struts does not, but you can approximate it with Struts + JSTL.
  - JSF was intended from the beginning to create a market for third party components.

JSF Compared to Struts

- Conversion and Validation
  - Both have support for validation.
  - Both support type Conversion, but the Faces story is more powerful
  - Struts Action class tightly coupled to ActionServlet, can call its methods. Nothing in JSF calls the FacesServlet.
  - Struts DynaActionForm instances can be author automatically. There is no support in the JSF framework to author backing beans automatically.
- Request Processing and Navigation
  - JSF uses logical outcomes from a java method to feed into a rule base. Struts uses the retruned ActionForward instance.
  - Struts Action concept is similar to what method bindings, and listeners give you in JSF.
  - In Struts, The ActionForm bean is passed to the Action and the action can do with it what it wants. In JSF, the ValueBinding mechanism exposes the entire managed-bean namespace to anywhere in the app that needs it.

JSF Compared to Tapestry

- Flexibility
  - Tapestry 3.x doesn't have as many extension points as JSF
  - Tapestry 4.0 is utterly extensible
  - Tapestry doesn't support client-device independence
  - Tapestry's use of OGNL implies more flexibility, because OGNL is more flexible than the Unified Expression Language used by JSF
  - Tapestry's use of HTML as the view description technology allows greater flexibility in choice of tools for UI design
  - Faces has the Facelets sub-project to provide the same HTML capability as Tapestry.
  - Both integrate well with Spring

JSF Compared to Tapestry

- Model Tier Access
  - Tapestry uses OGNL (Same as WebWork), JSF uses Unified EL
  - JSF has an Inversion of Control container, Tapestry 3.x does not. Tapestry 4.x will have IoC

JSF Compared to Tapestry

- Components and Events
  - JSF Event model very close to JavaBeans event model
  - Tapestry event model is more page-focused
  - Faces has a more well defined component lifecycle

JSF Compared to Tapestry

- Conversion and Validation
  - JSF has notion of Conversion as separate from Validation, Tapestry puts these two concepts together
  - Tapestry's validation scheme isn't as extensible as JSF's
  - Tapestry has client side validation, while in JSF that feature is implementation specific
- Request Processing and Navigation
  - Tapestry navigation defined in code. JSF navigation defined in XML

Web Application Trends

- Continuing focus on the Developer
  - Ease of Development was major driver in Java SE and EE 5
  - Frameworks like Ruby on Rails raise the bar for Ease of Development
- AJAX
- Consolidation among Web Frameworks
  - Best AJAX may be deciding factor
- Web-based vs Rich Client technologies
  - JavaWebStart is great!

Web Application Trends

- SOA is here to stay
  - Call it REST if you like, but it's still SOA
- Dynamic Languages (Scripting) vs. Strongly Typed Languages
  - Groovy and JavaScript will be important to Java
- Web 2.0
  - Collaboration
  - Reportability

Value increases as Use Increases (network effect)
Ease of operations

Web Application Trends

- Thanks for listening!