



Tapestry: Java Web Components

Howard M. Lewis Ship
hlship@attbi.com

What If ...

- ◆ ...you could build a web application without even thinking about URLs?
- ◆ ... you could use HTML from your HTML developers as-is?
- ◆ ... your web application would localize itself (just add translations)?

What If ...

- ◆ ... your development team could work together *easily*?
- ◆ ... you could really *reuse code*?
- ◆ ... extending your application meant adding methods, not creating entire servlets?

What If ...

- ◆ ... this technology existed **now**, was **tested, documented, proven** and **ready to go**?



**Tapestry: Java Web
Components**

Concept

- ◆ Tapestry *reconceptualizes* web application development in terms of Java objects, methods and properties

Concept

- ◆ Removes URLs, query parameters, servlets, HttpSession from developer considerations
- ◆ Application pages cooperate in terms of Java objects and methods, not URLs and query parameters
- ◆ Still allows 100% control over HTML production

Concept

- ◆ Build complex apps from simple, reusable components
- ◆ "Bake in" best practices
 - Error recovery, reporting
 - Localization / Internationalization
 - Personalization
 - Clustering for fail over / scalability
 - Browser-safe HTML

Concept

- ◆ Reduce amount of Java coding
- ◆ Improve robustness of applications
- ◆ Support team development
- ◆ Simplify interaction between HTML producers and Java developers
- ◆ Alternative to JSPs; more dynamic, no code generation

Concept

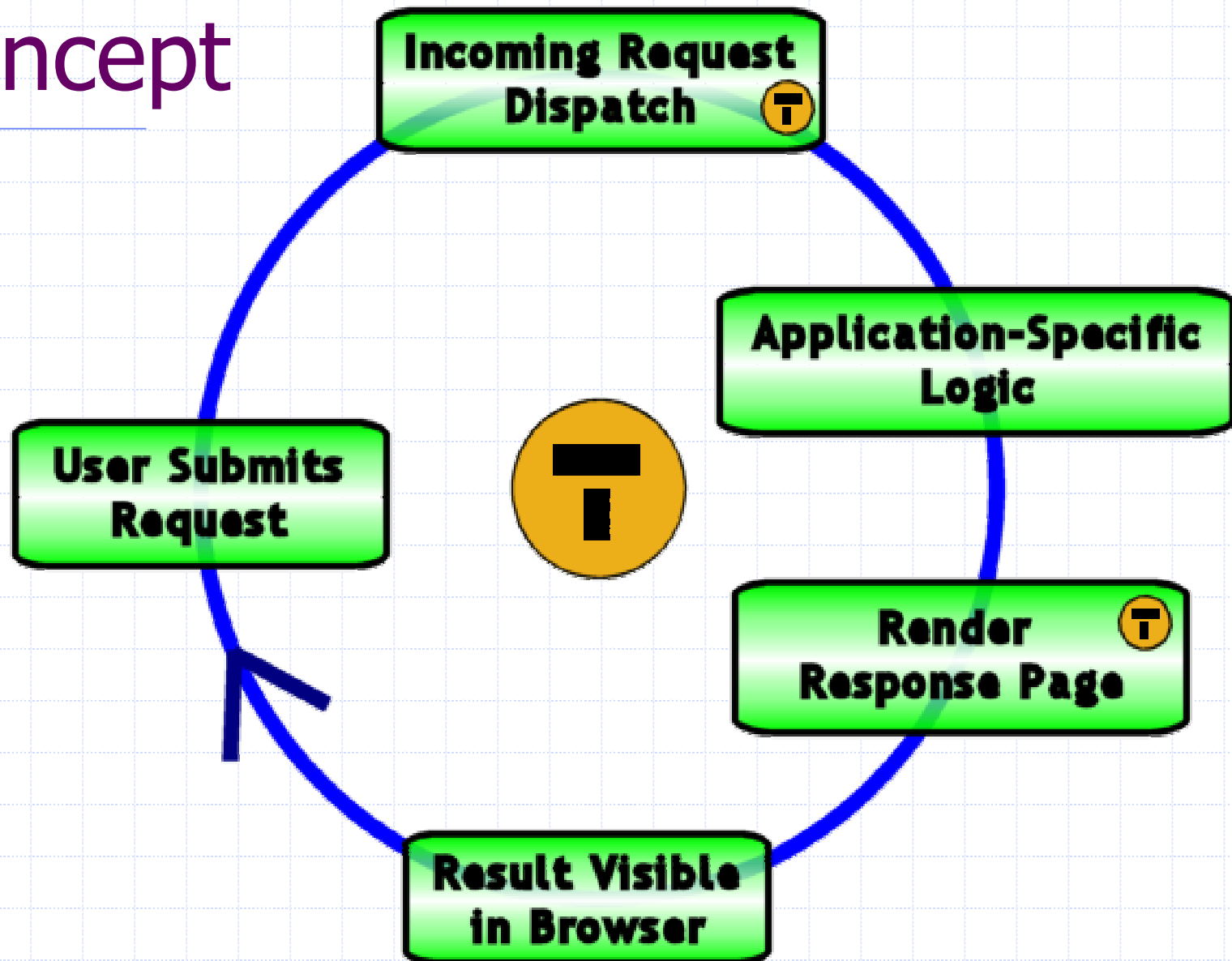
- ◆ Truly separate HTML and Java code
- ◆ **No** Java code in HTML template -- ever
- ◆ Normal, static HTML with some elements marked as "dynamic"
- ◆ Tapestry components provide dynamic behavior
- ◆ Still looks like normal HTML, no special tags

Concept

◆ Empower Java Developers

- Write less code
- Write interesting code
- All code written in IDE
- Provide standardized approach to common problems
- Break "tyranny of the URL"
 - ◆ Develop in terms of objects and properties, not URLs and query parameters

Concept



Concept

- ◆ Framework links response generation (including generating URLs for links and forms) to dispatch logic
- ◆ Developers provide pointer to application-specific code, executing when link clicked or form submitted
- ◆ Developer decides *what* and *when*, framework provides *how*

What is a Component?

◆ General

- Every framework has own definition of "component"
- Has to interact with application, other components
- "Black-box"; shouldn't need to know implementation, just the interface

What is a Component?

◆ Swing

- Responsible for drawing in 2d space
- Properties configured by application (push model)
- User interaction:
 - ◆ Event queue -> component
 - ◆ Application as event notification listener

What is a Component?

◆ Tapestry

- Renders a portion of (1 dimensional) HTML stream
- Has parameters that are "bound" to application properties (pull model)
- User interaction:
 - ◆ Servlet -> framework -> component
 - ◆ Application as delegate to component

What is a Component?

HTML Template

Hello,
Joe User.

Component Specification

```
<component id="insertUserName" type="Insert">  
  <binding name="value" property-path="user.name"/>  
</component>
```

HTML Output

Hello, Conan T. Barbarian.

Demo



History

- ◆ Open Source, LGPL
- ◆ Started January 2000
- ◆ Moved to SourceForge.net, July 2000
- ◆ 1.0 Release, May 2001
- ◆ Java Report Article, September 2001
- ◆ OnJava Article, November 2001
- ◆ 2.0 Release, April 2002
- ◆ > 25000 Downloads from SourceForge



Problems and Solutions

Problem: HTML vs. JSP

- ◆ HTML Producers don't know Java, JSP
- ◆ Java Developers weak on HTML (can't choose colors, either)
- ◆ Converting HTML to JSP
 - Add lots of cryptic stuff
 - Result is no longer usable in WYSIWYG HTML Editor
 - Changes require lots of producer/developer interaction
 - Producers must run full app to preview changes

Solution: Tapestry

- ◆ Minimal changes to HTML templates
 - Addition of jwcid attribute
 - Adds `` tags in some cases
- ◆ HTML still previews properly
- ◆ HTML Producers and Java Developers can work without interfering with each other

Problem: Complexity

- ◆ Requests are linked to pages (Sun Model 2)
 - Servlet (or Struts action) receives request
 - Updates "business model"
 - Invokes JSP to render response
 - Page-oriented dispatch model
- ◆ Components
 - Behavior of component defined by component
 - Component wants to handle incoming request
 - Component needs to know which page to render response

Problem: Complexity

- ◆ "Components" require special addressing
- ◆ Page dispatch logic
 - Additional query parameters to identify component
 - Ad-hoc code to "find" component, invoke methods
 - Similar code in each servlet or Action that includes component
- ◆ Servlet / Action for component
 - Receives request, finds component in standardized way
 - Needs to be configured with page to render response
 - JSP may require special setup, known to servlet

Problem: Complexity

- ◆ Feature creep -- applications grow more and more complex
- ◆ Many pages have similar functionality
 - Navigation bars
 - Search fields
 - Look and Feel
- ◆ Looks the same, acts the same == component

Solution: Tapestry

- ◆ Everything is a component
- ◆ Framework handles component addressing
 - Builds and interprets URLs
 - Finds right component on incoming request, invokes component code
 - Complexity encapsulated by framework
 - ◆ Manages feature creep
 - ◆ No upper limit: Components per page or nesting level
 - Leverages component object model -- just plain works

Problem: Scalability

- ◆ J2EE Approach: Clustering multiple servers
- ◆ Data stored in HttpSession copied to other servers in cluster
- ◆ Lots of pitfalls
 - Forgetting to store values
 - Forgetting to delete unneeded values (bloat)
 - References getting copied (can duplicate objects)
 - EJB references must be converted to EJB Handles
- ◆ Creating HttpSession too early inefficient, too late is ... too late (lots of coding headaches)

Solution: Tapestry

- ◆ Single object, "engine", stores all server-side state
- ◆ Automatically stored into HttpSession at end of request cycle
- ◆ HttpSession created only when needed
- ◆ Run with or without Cookies equally well
- ◆ State appears as JavaBeans properties
 - No code to move values in or out of HttpSession

Problem: Team Development / Integration

- ◆ Lots of developers, each contributing "their piece"
- ◆ Interface a combination of:
 - Servlets or Struts Actions
 - URL query parameters
 - Java code inside JSP
 - JSP tags
- ◆ Different developers, different "interfaces"
- ◆ Lack of standards for behavior, naming, errors

Problem: Team Development / Integration

- ◆ Code that builds URL, interprets URL can easily get out of sync
- ◆ Hard to get a "component" to work inside a "page"
 - Setup code inside Struts Action
 - More code inside JSP
 - JSP includes
 - Things have to be named "just so"
- ◆ Evolving a component requires changes to everywhere component is used

Solution: Tapestry

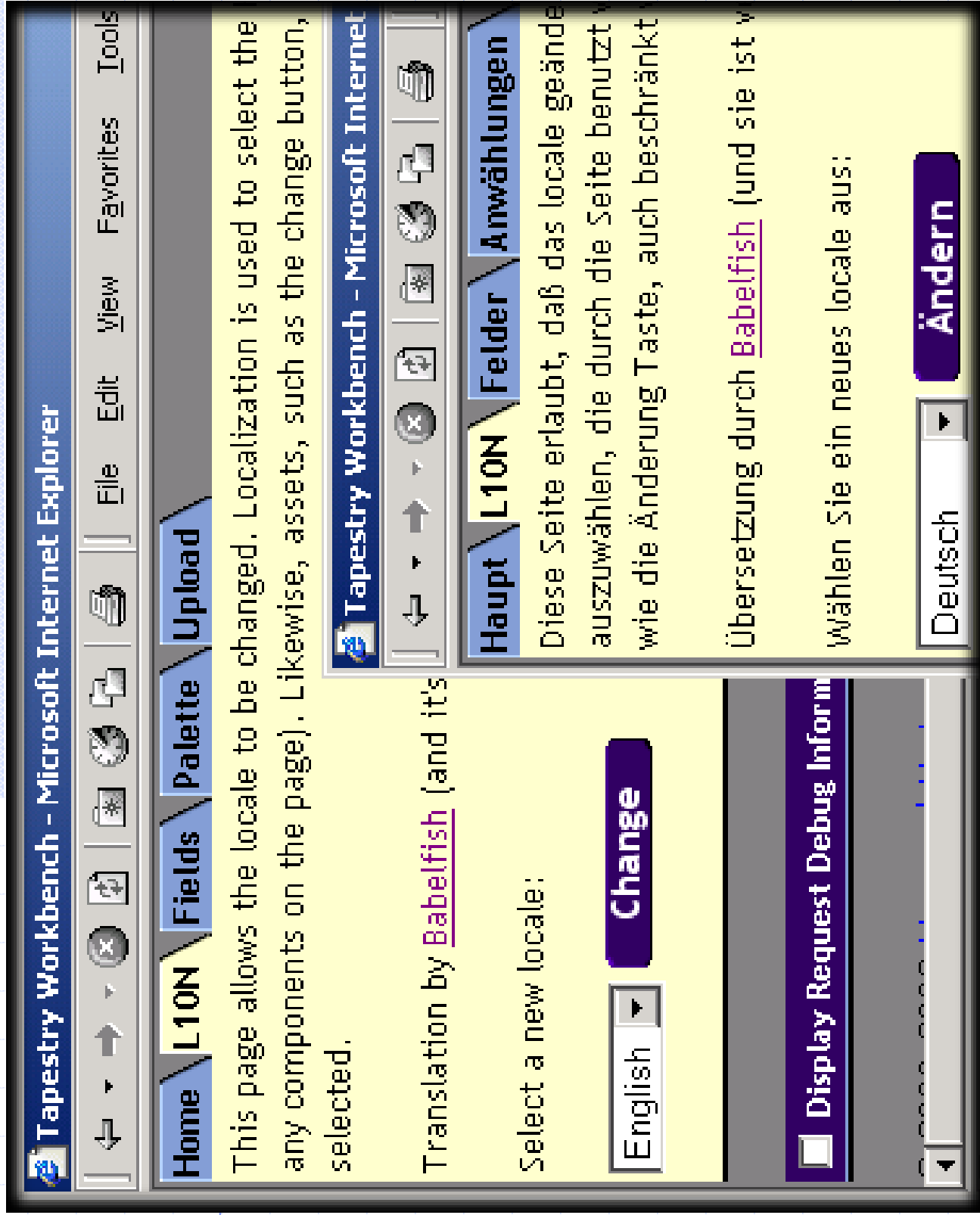
- ◆ Everything is a component
- ◆ Components have named parameters that are "bound" to page properties
- ◆ Components can read or set properties of page via parameters
- ◆ Consistent approach, easy to document
- ◆ Eclipse Plugin for Tapestry

Problem: Internationalization

- ◆ Applications need to be localized
- ◆ Hodge podge of techniques
 - Struts `<bean:write>` everywhere
 - ◆ Single message file
 - ◆ Obscures HTML for HTML producers
 - Complex linking between translated pages
 - ◆ More pages == more complexity == more bugs
 - ◆ Lots of code everywhere to "find" right translation

Solution: Tapestry

- ◆ Each page, or component *may* have localized templates
- ◆ Tapestry *automatically* selects correct template
- ◆ Can use message files instead
- ◆ Can mix and match
- ◆ Can localize assets (images, stylesheets, etc.) as well



Problem: Debugging

- ◆ JSPs convert to ugly, unreadable code
- ◆ Hard to get IDE to "recognize" Java code from JSPs
- ◆ Errors in JSPs result in blank page or cryptic error message
- ◆ Thrown exceptions not reported well
 - Usually, only outermost exception displayed, when innermost exception most helpful
 - With Struts, exception often "lost" and not reported

Solution: Tapestry

- ◆ No code generation! Ever!
- ◆ Multiple layers of exception handling and reporting
- ◆ Exceptions get very complete report

Solution: Tapestry

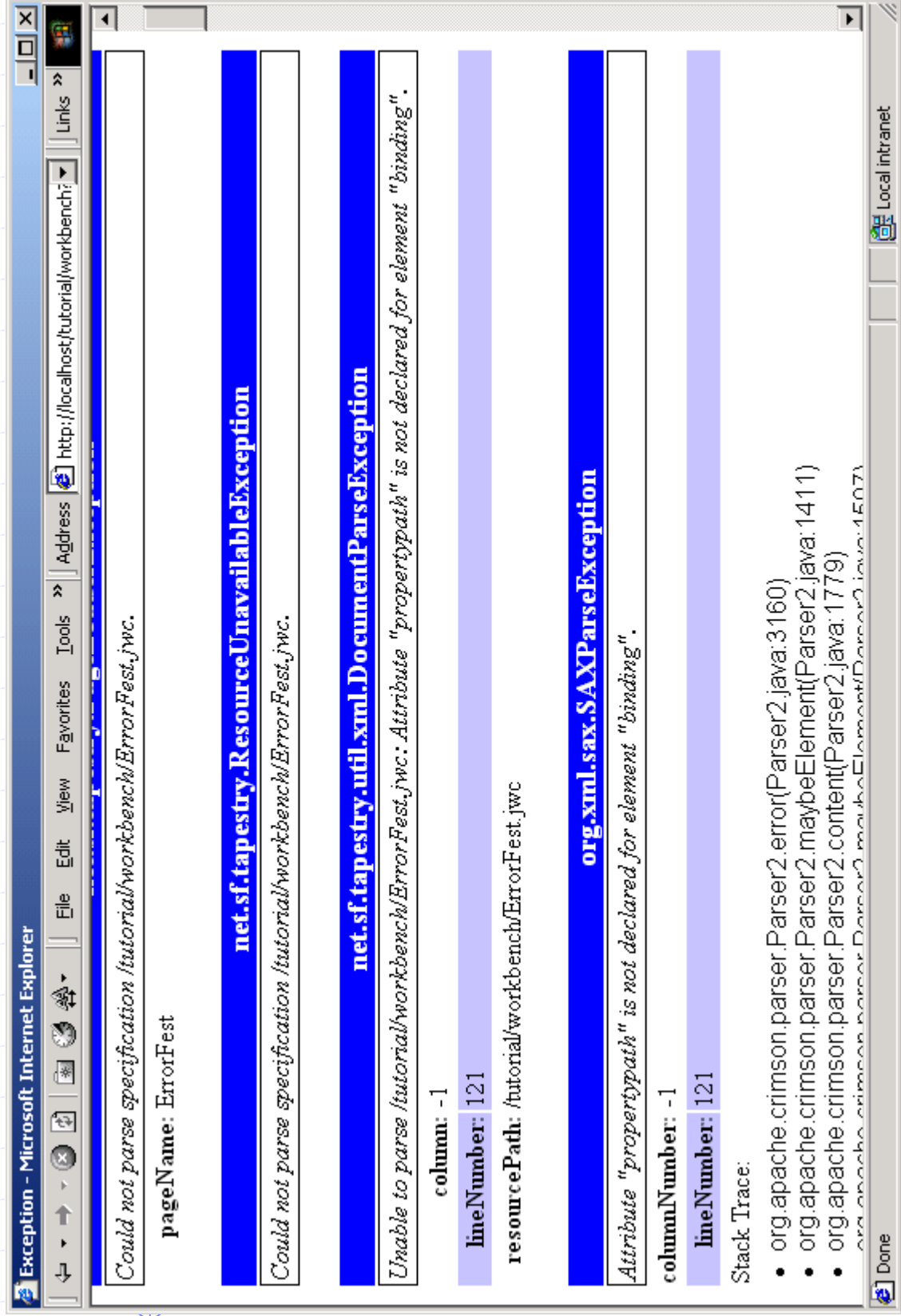
◆ Exception report:

- All exceptions (outermost to innermost)
- Properties of exceptions
- Stack trace of innermost exception
- Complete rundown of all objects:
 - ◆ HttpServletRequest
 - ◆ HttpSession
 - ◆ ServletContext
 - ◆ JVM System Properties

Solution: Tapestry

◆ Example:

- Bad XML in Tapestry specification file
- Result: Exception report includes SAXException with exact position of error
- Savings: Didn't have to start up debugger, restart application, reproduce activity, analyze in debugger (five - ten minutes minimum)





Full Disclosure: Costs of Using Tapestry

Learning Curve

- ◆ Servlets, JSPs, Struts Actions are *procedural*
 - Start at top, work to bottom
 - Monolithic generated Java class
 - Most developers, including Java developers, think procedurally
 - Struts is "push" oriented -- store values into request or session for later access by JSP

Learning Curve

- ◆ Tapestry is object oriented
 - Many objects working together
 - More like event-driven GUI
 - Pull oriented: properties are drawn from known objects
 - Most page-page interaction in Java code
 - Must “unlearn” limitations of JSPs

Performance

◆ Surprisingly similar

◆ JSPs

- Slow initial start up (create and compile class, load class)
- Slight edge under extreme load

◆ Tapestry

- Quicker initial startup (parsing templates, specifications)

Performance

◆ Comparison

- Not a fair comparison ... Tapestry does more
- Still, neck-and-neck response time in tests
- Tapestry *may be* faster than Struts

◆ Conclusion

- Differences irrelevant compared to costs of EJB access, database

JSP / Tapestry Interaction

- ◆ Tapestry needs to "run the show"
 - Needs to generate entire page
 - Links should be back into Tapestry
 - Can't "include" JSP easily
 - *Can* redirect to JSP to render response

JSP / Tapestry Interaction

- ◆ Would need special taglib to reference Tapestry pages
- ◆ Not practical for JSP forms to be handled using Tapestry form components
 - Could process query parameters in developer code



Lightspeed Introduction to Tapestry

A fast-paced overview of basic concepts and components.

Basics

- ◆ Tapestry is highly "pluggable"; limitations can always be worked around
- ◆ Tapestry uses large numbers of objects; and uses pools and caches for efficiency
- ◆ Tapestry is very structured; things are usually "defined" or "declared" (often in XML)
- ◆ Tapestry doesn't hide the fact that it is built on servlets and HTTP

Basics

◆ Leverages JavaBeans properties

- Adds "property paths"
 - ◆ Dotted name sequence
 - ◆ Ex: "page.user.name" ==
getPage().getUser().getName()
- Reads *and writes* properties
- Properties are not type-safe
- Always uses actual, not declared, type
- Properties are not the same as attributes
 - ◆ Just in time calculation
 - ◆ Synthetic properties

Top Down View

- ◆ Application = servlet + engine + pages
- ◆ Servlet loads *application specification* on initialization
- ◆ Application Specification:
 - Name of application
 - Java class for engine
 - Name and specification path for each page

Top Down View

◆ Engine

- Central "brain" for running application
- Located or created by servlet
- Servlet defers to engine
- Engine works with pages and components to process requests, render responses

Top Down View

◆ Page

- Has unique name (in application specification)
- Instantiates as particular Java class
 - ◆ Extends BasePage, implements IPage
- Special type of component (see next)
- Has HTML template (may be localized)
- Has *component specification*
 - ◆ Lists id, type of components contained on page
 - ◆ Sets up (binds) parameters of components

Top Down View

◆ Components

- Uses component specification, just like Page
- May be container or containee or both
- Class extends `AbstractComponent` or `BaseComponent`, implements `IComponent`
- May have HTML template, or may generate HTML directly in code
- Sets id, type, parameters of any contained components

Top Down View

◆ Components

- About 30 pre-defined components
 - ◆ Various types of links
 - ◆ Loops, conditionals
 - ◆ Forms, form fields
 - ◆ Images, Rollovers
 - ◆ Bunch of special-purpose

HTML Templates

- ◆ Goal: Take output from HTML producer and use it *as is*
 - Minor markups to make content "dynamic"
 - May still be edited by HTML producer without corrupting it
- ◆ Uses 'fake' attribute, jwcid
 - Can be added to any existing tag
 - Tag is replaced with dynamic content
 - Doesn't really matter what HTML element is used
 - jwcid attribute matched against component specification

HTML Templates

HTML Template

Hello, ``John Q. Student``,
today is ``Jan 1 2002``.

Component Specification

```
<component id="insertUserName" type="Insert">  
  <binding name="value" property-path="user.fullName"/>  
</component>  
  
<component id="insertDate" type="Insert">  
  <binding name="value" property-path="date"/>  
  <binding name="format" property-path="dateFormat"/>  
</component>
```

HTML Output

Hello, Conan T. Barbarian,
today is June 23, 2003.

HTML Templates

- ◆ Tapestry components can wrap around text and *other components*
- ◆ Some components replace wrapped content completely
 - Insert component (previous page)
 - Wrapped text exists as sample, for WYSIWYG editor
- ◆ Some components *integrate* wrapped text & components with their *own template*
 - Kind of like putting JSP include before and after content
 - Works recursively, no depth limit!

Basic Interaction

- ◆ Simple examples showing the basics
 - Page Component
 - Direct Component
 - Form and TextField Components

Page Component

- ◆ Creates a link to a new page

- ◆ Parameters

- "page", name of page to go to
- "disabled", (optional) can disable link
 - ◆ If true, link doesn't render, but content does

Page Component

HTML Template

Click `here` for the next page.

Component Specification

```
<component id="nextPage" type="Page">  
  <static-binding name="page">NextPage</static-binding>  
</component>
```

HTML Output

Click `here` for the next page.

Page Component

- ◆ Page component creates the full and correct URL
- ◆ Writes an `<a>` tag, wrapping around the text ("here")
- ◆ Could wrap around images or other components
- ◆ Could easily use dynamic, not static, target page
 - Example: multi-page wizard
 - Example: page selected based on user's role
- ◆ Can easily disable link; useful for locking out options which aren't available

Direct Component

- ◆ Creates a link, like Page
- ◆ When clicked, a *listener method*, provided by the application, is invoked by the component
- ◆ Delegation pattern
- ◆ No limitations on listener method; can invoke methods, EJBs, etc.
- ◆ Default is to render the page containing the component for the response
- ◆ Listener can easily choose a different response page

Direct Component

HTML Template

```
You may <a jwcid="enroll">Enroll in  
<span jwcid="insertCourseName">Course Name</span></a>.
```

Component Specification

```
<component id="enroll" type="Direct">  
  <binding name="listener"  
    property-path="listeners.processEnroll"/>  
</component>  
  
<component id="insertCourseName" type="Insert">  
  <binding name="value" property-path="courseName"/>  
</component>
```

HTML Output

```
You may <a href="...">Enroll in  
20th Century Comedy</a>.
```

Direct Component

Java Code

```
public String getCourseName ()
{
    return ...;
}

public void processEnroll (IRequestCycle cycle)
{
    // Invoke methods to handle enrolling

    cycle.setPage ("EnrollConfirm");
}
```

Direct Component

- ◆ What is "listeners.processEnroll"?
 - Little bit of JavaBeans properties magic
 - "listeners" is a bridge between the specification and the Java code
 - Scan class for certain methods
 - Exposes those methods as properties
 - Looks for
 - ◆ `public void method(IRequestCycle)`
- ◆ This is how the Direct component calls back into application specific code to do stuff

Direct Component

- ◆ Not even a whiff of a URL
- ◆ Listener method selected new page, "EnrollConfirm", as response page (overriding default)

Form and TextField Components

- Forms can be very dynamic in Tapestry
- Form component must wrap around other components
- Form coordinates things during render and again, when form submitted
- Form components *read* properties on render, *update* properties when form submitted
- Form's listener invoked *after* all components have updated their properties
- Tapestry includes validation framework for text fields

Form and TextField Components

HTML Template

```
<form jwcid="form">  
Enter your name: <input type="text" jwcid="inputName"/>  
<br><input type=submit>  
</form>
```

Component Specification

```
<component id="form" type="Form">  
  <binding name="listener" property-path="listeners.submit"/>  
</component>  
  
<component id="inputName" type="TextField">  
  <binding name="text" property-path="name"/>  
</component>
```

HTML Output (Partial)

```
<form method="post" action="...">  
Enter your name: <input type="text" name="inputName">  
<br><input type=submit>  
</form>
```

Form and TextField Components

Java Code

```
public String getName()
{
    return name;
}

public void setName(String value)
{
    name = value;
}

public void submit(IRequestCycle)
{
    // Do something with name, it has been set

    ...

    // Choose a page to render response
    cycle.setPage(...);
}
```

Form and TextField Components

- ◆ Additional components for
 - Text area
 - Checkbox
 - Radio
 - Select
- ◆ Can get complex, with looping and conditionals inside the Form
- ◆ Note: names of form, elements generated automatically
 - Ensured to be unique



Wrap Up

Other Topics

- ◆ Tapestry Inspector
- ◆ Managing server-side state
- ◆ Assets
- ◆ Contributed components
- ◆ Spindle: Tapestry plugin for Eclipse
- ◆ Sabertooth: O/R mapping framework

Other Resources

◆ Tapestry Home Page

- <http://tapestry.sf.net>

◆ Tapestry Mailing List

- <http://lists.sourceforge.net/lists/listinfo/tapestry-developer>

◆ OnJava.com

- <http://www.onjava.com/pub/a/onjava/2001/11/21/tapestry.html>

Summary

- ◆ Components are powerful, flexible
 - Code generation not needed or desirable
 - Component object model allows much higher complexity
 - Complexity managed by framework, not developers
 - Specifications are almost self-documenting

Summary

- ◆ URLs are bad
 - Hidden away, handled by framework
- ◆ HTML: Producers vs. Developers
 - Integration easier
 - WYSIWYG maintained
 - Updates almost painless

Summary

◆ **Less code == less bugs ==
good**