PRIMIX SOLUTIONS

Core Labs

Tapestry: Java Web Components Whitepaper

CORE LABS

Tapestry: Java Web Components Whitepaper

© Primix Solutions
One Arsenal Marketplace
Phone (617) 923-6639 • Fax (617) 923-5139

Tapestry contact information:

Howard Ship <hship@primix.com>
http://sourceforge.net/projects/tapestry

Tapestry Whitepaper

Sun's J2EE (Java 2 Enterprise Edition) achitecture is fundamentally a component-based architecture. At the core is the Enterpise JavaBeans specification, which defines enterprise-wide remote components as the location for business logic and data.

However, Sun's JavaServer Pages (JSPs), which are granted special status with in the J2EE specifications as the preferred method of delivering web content, are fundamentally flawed. They simply are not components. JavaServer Pages are stateless (they are really Java Servlets in disguise), and so have no properties. They also have no interface, beyond the HTTP request parameters they respond to. They cannot be easily re-used, since they are not "black boxes" ... intergrating them into an application requires an intimate understanding of their internals.

In addition, servlets and JSPs suffer from a phenominon I've termed "the tyranny of the URL". In effect, the URL *is* the interface to the JSP, in that HTTP parameters encode whatever information is required by the JSP. If the name of the JSP or any of its parameters change in any way, URLs created by referencing JSPs will be broken ... and it won't be evident that there's a problem until runtime.

Because there are no standards for JSP and parameter naming and usage, different members of a project team can and will approach similar problems in different ways. This creates a huge amount of impedance between different portions of the same application (created by different developers), causing an integration migraine.

JSP developers are expected to assemble their HTML on a character-by-character basis. GUI development, typified by the Swing library, has long moved over to a component model … where objects acting as components work together. JSP development is still a very procedural operation: start at the top of the page and work to the end … it's the equivalent of creating a complicated GUI as a single monolithic object, painting the screen pixel by pixel.

Tapestry is a framework and component architecture for creating web applications. It is designed to meet the above mentioned issues head on, as well as other less evident issues in web application design.

Tapestry is built upon the Java Servlet 2.2 API and is designed to work within any application server. To the application server, a Tapestry application appears as a single Java servlet.

Tapestry is an open-source project, released under the Lesser GNU Public License. It may be freely intergrated into both open-source and proprietary systems, subject to the requirements of the License.

Tapestry in a nutshell

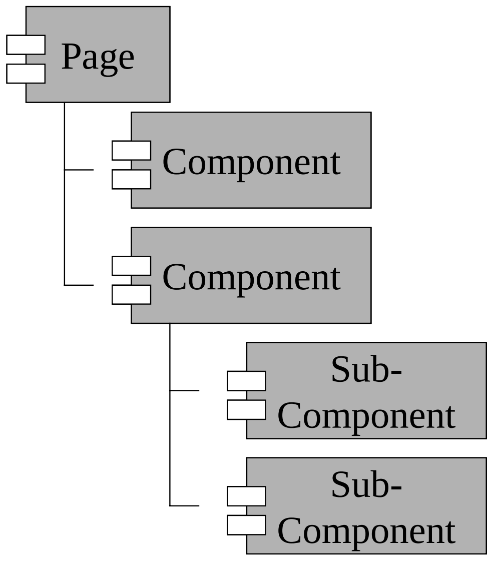
At the root of Tapestry is the component. A Tapestry component consists of a specification, an HTML template and a Java class. Tapestry makes use of the Model View Controller design pattern at several levels. Within a component, the HTML template is the View, the Java class is the Controller, which itself provides access to the Model data.

Components exist primarily to provide dynamic content when rendering an HTML page that will ultimately be displayed in a client web browser. A few components (related to hyperlinks, HTML forms and HTML form elements) are also involved in responding to requests from the web browser.

Components have *parameters* (which are described in their specification) that define the source of data needed by the component. The ultimate source of this data comes from the Controller or, through it, from some form of business object, such as an Enterprise JavaBean, or data from a database.

Components have *bindings* which are matched to their parameters. Bindings specify, in terms of a JavaBeans property, where the value for a parameter will be obtained when needed.

As an example, the Insert component is used to insert a bit of text into the HTML response. It has a parameter, named value, which is the string to insert. The value may be bound to a property of the page object which contains the Insert component. During rendering, the binding will extract the property, by executing the correct accessor method. The value returned by the page is then inserted into the stream of HTML destined for the client web browser.



Pages are built from components, which may themselves be built from other components. Pages are themselves components (complete with specifications and HTML templates), with some special responsibilities for organizing all the components. Each page has a unique name within the application. Each component has a unique name within its container.

In a JavaServer Pages application, there is no application per-se, just a collection of servlets and JSPs that act together. The application is implied, expressed as emergent behavior from a collection of loosely coupled elements.

Tapestry has a much clearer definition of what an application is, starting with a specification that defines the logical name of all pages in the application, and a Java class, the application engine, that acts as a global resource for all pages in the application.

By imposing this structure, Tapestry can take over much of the "plumbing" of a web application ... breaking the tyranny of the URL. The framework is responsible for generating the URLs for inclusion in an HTML response, and later responds to those URLs, performing the necessary dispatching to change to a different page within the application, or to notify a particular component that it was invoked.

Simplified HTML Templates

One of Tapestry's design goals is to have minimal impact on the HTML. Unlike JavaServer Pages and other scripting solutions, Tapestry introduces just a single special HTML-like tag into its templates, a `<jwc>` tag (which stands for *J*ava *W*eb *C*omponent). The tag is a placeholder for dynamic content generated by a Tapestry component.

Tapestry components may span a portion of the HTML template (by using standard HTML open and close tags), wrapping around static HTML text and other Tapestry components. The outermost component controls when, if and even how many times the wrapped content is rendered as part of the HTML response. This allows for tremendous flexibility. Some Tapestry components provide control logic, acting as conditional or looping elements.

Other Tapestry components combine the content they wrap with their own HTML template. This allows for such uses as consistent navigation borders across an application.

Rich Server-Side State

Tapestry provides rich support for per-client server-side state. Each page, or component on a page, may persist any number of properties between request cycles. Some of this state is references to business objects, including Enterprise JavaBeans. Other state is more directly related to presentation.

The Tapestry application selects the strategy used to save this data ... in memory, in a database or in a session Enterprise JavaBean.

Seperating the page's persistent state (a collection of property names and values) from the page instance (a particular Java object) allows Tapestry to pool page instances, sharing them between different client sessions. This makes efficient use of server-side resources, since a relatively small number of pooled page instances can support a large number of concurrent clients.

Localization / Internationalization

Support for localization and internationalization can be difficult in a JavaServer Pages environment. Because of how JSPs reference each other, it becomes difficult when creating URLs to know which JSP to create a linkage to ... the French version, the German version, etc. This leads to a large amount of cut-and-paste work, where the primary version, usually in English, is cloned and translated to create the secondary language versions. This cut-and-paste includes not only the page's content, but all the embedded Java code and JSP tags.

Tapestry approaches this issue by keeping the control logic separate from the presentation. This control logic is the Tapestry component specification and corresponding Java class; the presentation is the HTML template. Under Tapestry, a single component may have multiple HTML templates translated to different languages; Tapestry simply searches for the most appropriate HTML template available, matching it against the language specified in the client's web browser (the HTTP protocol includes a request header that identifies the client's preferred language).

Of course, there is still a limited amount of cutting and pasting, as with JavaServer Pages, but virtually all of the template is actual page content ... there's no Java code on the page, just the minimal `<jwc>` tags.

Robust Error Handling

Tapestry has extremely robust error handling; its single-servlet architecture allows an ideal place to catch unexpected exceptions and handle them. By contrast, JSPs have no single common control point, requiring that every page implement an ad-hoc exception handling strategy.

The default behavior when an exception is thrown is to display a special exception page, that shows the exceptions thrown, stack trace, and a wide collection of information about the request, the session and the Java Virtual Machine.

Tapestry applications are free to change this behavior. A fully-featured Tapestry application could capture this information on the server, in a database or flat file, and automatically notify some form of support organization, in addition to providing an HTML response to the client.

Load balancing and Fail-over

Load balancing and fail-over are features usually provided by the application server. Typically, this is accomplished by serializing Java objects related to one client and broadcasting this information to other servers in a cluster, so that any server in the cluster is equally capable of handling an incoming request.

Tapestry is designed to be a "good citizen" inside an application server. All aspects of the code are designed to handle being serialized in one Java Virtual Machine (JVM) and de-serialized in another. There is also an effort to keep the amount of client data small, though on a large application this may involve using a database or Enterprise JavaBeans so that the data won't be stored in-memory.

Future Directions for Tapestry

Tapestry is currently focused on providing HTML content; a future enhancement will be to support other kinds of content, such as XML and WML.

Cross-browser compatibility is left largely in the hands of the application developer. Adding the ability to identify the client web browser and it's capabilities, and improving the HTML created by various components to account for browser differences will be a huge improvement.

Additional tool support and/or IDE integration will enhance Tapestry's usefulness. The ability to edit HTML templates, Java classes and Tapestry component specifications side-by-side will speed application development.