Tapestry4

> NOTE: This is outdated information that applies only to Tapestry 4.

Tapestry 4.0

Work on Tapestry 3.0 has *finally* come to a close, and the focus is now on what comes next.

Hopefully, Tapestry 4.0 will not drag on for as long as Tapestry 3.0 (a year and a half!). More information is available on the Tapestry31Status page.

Woops! Guess it did! – Howard

Tapestry is now using HiveMind's build system ... here are notes on BuildingTapestry.

A number of ideas are floating around, rough areas in 3.0 (or even earlier) that need to be addressed.

Reworking Component Parameters

Component parameters are just not quite there. The idea of a component *direction* just doesn't work out well. Users need too much of an understanding of how components render to understand which direction works (i.e, can't use `in` if the parameter is accessed inside a component listener method). Then there are degenerate cases, such as an `auto` parameter that is connected to a `in` parameter (in an enclosing component).

Direction `auto` is one step on the way. But it has limitations: the primitive types it can operate with are limited, and parameters must be required. In addition, it can be less efficient that `in`, because it will evaluate an OGNL expression inside the fabricated accessor method *every time*.

What's needed is to improve direction `auto` with a notion of caching: how long can it cache the data if it has already retrieved it? I think likely options are **no-cache**, **render**, **page-render** or **request**. `render` means only as long as the component itself is rendering (or any components it encloses), that is, until the end of the component's `renderComponent()` method. `page-render` means until the current page finishes rendering. `request` means until the end of the current request cycle.

Components may need to know whether they are currently rendering or not ... when invoked by a listener method, `render` or `page-render` don't make sense.

I would think that `render` is the best default for a component parameter cache.

Update: Parameters have been *fixed* and are better and more efficient than ever. Parameter direction is gone. There isn't a choice in terms of caching, its the equivalent of `render` above, and I think that is more than sufficient.

Modularized Applications

Tapestry 3.0 expects all Tapestry pages and components in the application to be within a single directory. This causes some scalability problems ... when your application has dozens or hundreds of pages, its not good to have them all in a single directory.

Further, it leads to a basic incompatibility with J2EE DeclarativeSecurity, which is based on mapping paths (effectively, folders) to different security zones.

It will take a lot of work to address this; not just in terms of how pages are named, but even more so in terms of how URLs are constructed and parsed.

On top of this is the desire to maintain WYSIWYG preview ... this will likely entail having the Shell component render a <base> tag.

Update: This has started, and page names can now have one or more folder names, i.e., `admin/threads/ThreadAdmin`. This would locate a page specification as `/WEB-INF/admin/threads/ThreadAdmin.page` and an HTML template in the same folder, or as `/admin/threads/ThreadAdmin.html`. The Shell component now renders a <base> tag.

Improved Testing Story

Testing Tapestry pages and components is too hard. Because the classes are often abstract, it isn't easy to instantiate them for testing purposes.

Tapestry has a fairly good test suite that needs to be *productized*: documented, improved, stabilized.

Update: The Creator can instantiate abstract components, a HiveMind 1.1-alpha supports mocked classes via the EasyMock class extension. So unit testing pages and components, and even the integration of different pages and components is feasible (and widely used inside Tapestry's own tests). However, providing a good, efficient, *integration* test framework would still be a huge boon ... the start of one is there, but I've switched gears to use EasyMock instead.

HiveMind Integration

The DTD will be extended to make it easy to access HiveMind objects, using <inject> to inject services, configuration or whatever into pages and components as read-only properties. The <extension> and <service> elements will be deprecated.

Update: Yep, that's in there, and works great.

Offline Content Generation (4.1?)

This has come up ... the idea of using Tapestry to generate *static* web pages. This is a technique used by many high-volume web sites ... much of the content is neither fully dynamic nor personalized. It changes at fixed intervals and might as well be a static web page that is auto-magically updated. An alternative is to use your preferred http server's error handling mechanism to ask your app server to generate the static page (when possible): if the page is here, ok the http server serves it, otherwise it redirects the request to your app server (which in turn saves the final response). If you want to update, delete the static files. ...

Portlet Support

Supporting porlets in Tapestry 4.0 is gaining an ever higher priority.