Tapestry5LookupDebate

I find it confusing that Tapestry has so many *rules* for finding things like specifications and templates.

At first, as the SpindleGuy I was not enamoured of the ability to prefix names with path parts like

```
mycomponents/MyComponent
```

I understand the utility for end users but, boy, from a tool developers pov it's hard to support.

In discussing/debating the implementation of Tapestry5 I first want to say that I embrace the above completely. I think it's possible to keep all the existing utility and, with a few tweaks, make tool development a whole lot easier.

The goal Howard has implied is that xml specifications are going to go away in Tapestry 5. While I'm uncertain that getting rid of them completely is the best goal (ex. getting rid of <component> tags leaves us with cruft in the templates no?) I m certain that specification objects that have Resource locations are going to stay around.

A lot of what I'm going to touch on is currently related to the locations of xml specification files and how Tapestry finds them. Assuming that these specs go away doesn't really change what I'm laying out as the specification objects will still be there, even if they are completely synthetic.

Let me add that I understand the current implementation was built to make thing easier for new and veteran users of Tapestry. Those addition are one of the reasons why adoption of Tapestry has accelerated the way it has. I just think that the some of the impacts of past decisions were missed and need to be addressed.

First, since backwards compatilibity is not an issue and names with path parts is here to stay I would propose that if at least the .application and .library files will stay around...

```
Remove <page> and <component-type> as they exist today.

Instead of a name mapped to a specification path, these should be replaced with two new (optional) tags:

<page-alias name="" full-name=""/>

and

<component-alias name="" full-name=""/>

where 'name' and 'full-name' have the same meaning in both tags.

name - a short name which is the alias of the full name (including path part) of the page or component in the
namespace.
       'name' is a simple identifier with no path parts.

full-name - the full name, including path parts, of the page or component in the namespace.
```

Tapestry 3 was xml-centric in that the specification was the lynchpin. Howard has already stated that, althought the implementation is incomplete in Tapestry 4, he sees the need for the names (incl path parts) of Pages/Components to become the lynchpin for the whole operation.

I think the above change is respectful of that intention. These two tags allow a user to optionally define a short alias for any page or component in the namespace.

Secondly, I've been again looking at the *rules* for locating pages and components. I think that with some simple tweaks these *rules* can be vastly simplified with little impact on the utility of Tapestry.

First, the Tapestry 4 'rules' for locating components (from ComponentSpecificationResolverImpl).

```
<ul>
 * <li>As declared in the application specification
 * <li><i>type</i>.jwc in the same folder as the application specification
 * <li><i>type</i> jwc in the WEB-INF/ <i>servlet-name </i> directory of the context root
 * <li><i>type</i>.jwc in WEB-INF
 * <li><i>type</i>.jwc in the application root (within the context root)
 * <li>By searching the framework namespace
 * <li>By searching for a named class file within the org.apache.tapestry.component-class-packages
 * property (defined within the namespace)
 * </ul>
 * The search for components in library namespaces is more abbreviated:
 * <li>As declared in the library specification
 * <li><i>type </i>.jwc in the same folder as the library specification
 * <li>By searching the framework namespace
 * </ul>
```

First, if the my first suggestion above for aliases was adopted we can cut out two rules right away:

```
<ul>
 <removed>
 * <li><i>type</i>.jwc in the same folder as the application specification
 * <li><i>type</i> jwc in the WEB-INF/ <i>servlet-name </i> directory of the context root
 * <li><i>type</i>.jwc in WEB-INF
 * <li><i>type</i>.jwc in the application root (within the context root)
 * <li>By searching the framework namespace
 * <li>By searching for a named class file within the org.apache.tapestry.component-class-packages
 * property (defined within the namespace)
 * </ul>
 * The search for components in library namespaces is more abbreviated:
 <removed>
 * <li><i>type </i>.jwc in the same folder as the library specification
 * <li>By searching the framework namespace
 * </ul>
```

What if the remaining could be further simplified, without losing any existing utility, to this...

```
<removed>
 * <li><i>type</i>.jwc in the same folder as the application specification
<removed>
<removed>
<removed>
 * <li>By searching the framework namespace
 * <li>By searching for a named class file within the org.apache.tapestry.component-class-packages
 * property (defined within the namespace)
 * </ul>
 * The search for components in library namespaces is more abbreviated:
 <removed>
 * <li><i>type </i>.jwc in the same folder as the library specification
 * <li>By searching the framework namespace
 * </ul>
```

Wouldn't that be nice? I think it can be done with some adopted conventions and with a change to where Tapestry currently creates a synthetic Application specification when the .application file is missing.

The adopted conventions would be:

```
1. By convention, Tapestry looks for components according to these rules (insert the above abbreviated list of
rules).

2. By convention, if you are going to deploy multiple Tapestry application in the same .war file and you have a
.application file, it is expected that you will place it in the folder /WEB-INF/name/ where name is the name
defined for the servlet in web.xml. This convention is not enforced at runtime. Its a recommended 'best
practice'.

3. By convention, if you do not have a .application file Tapestry will install a synthetic (invisible)
specification. The synthetic specification will have a path and that path will be:

   - /WEB-INF/name/name.application when the folder /WEB-INF/name/ exists and where name is the name defined
for the servlet in web.xml; or
   - /WEB-INF/name.application if the above folder /WEB-INF/name does not exist.
```

So let's look at the rules that would be removed and I'll argue why they should be removed and why thier removal is not catastrophic.

```
* <li>As declared in the application specification

and

* <li>As declared in the library specification
```

These two rules make no more sense if the `<component-alias name="" full-name=""/>` tag of my first suggestion are adopted. The old `<component-type>` tag mapped names to jwc files. If the 'name' of the component is to be truely more important than the xml file (and indeed if the xml is going away) then these new tags make more sense, making it easier to reference a component (with a possible long name). If the xml is gone or the path to the xml is no longer paramount, then the two rules above are no longer needed and can be dropped.

```
* <li><i>type</i> jwc in the WEB-INF/ <i>servlet-name </i> directory of the context root
```

I don't know if anyone has been paying attention to my ramblings (recently or at all) but Tapestry allows pages/components to exist in more than one namespace (in other words to cross namespace boundaries). While I'm not enmoured of this I can live with it if it's something that an end user would use *as a tool with full knowledge of the possible pitfalls and possible impacts*.

Rules like this one may cause components to cross namespace boundaries or they may not. It requires concious effort and understanding of the rule to 'prevent' boundary crossing. Quite the opposite of conciously using this 'feature' to solve some obscure domain problem. What happended to *the simplest choice is the right choice*?

Why does removing this rule not cause the world to end? Since the convention already is that that `/WEB-INF/servlet-name/` should be the location for of the application spec in the multiple-app-per-war scenario then this rule is moot even today as the previous rule (relative to the app spec location) would be hit and matched first. If the user follows the convention then not only is the rule not needed, the fact that it is there at all is just a mechanism to cause components to cross namespace boundaries. Removing the rule is is like a defensive measure, applications that don't follow the convention would break in a documentable, easily explainable way.

```
As a side note I want to state again that this crossing boundaries situation is a bad thing. Tapestry already
has a mechanism for sharing pages and components...libraries. Libraries are an explicit, documented, mechanism
for sharing. Namespace boundary crossing is undocumented and kinda hacky don't you think?

The  current implementation of Tapestry allows boundary crossing to occur implicity, without concious decision
or action on the part of a developer. We all know that the implementation of Tapestry changes significantly
from release to release. That means the conditions where boundary crossing could occur implicity might change
from release to release. This could affect backwards compatibility as a developer, who may unknowingly be
depending on this 'feature', may have thier application break in Tapestry v.X and it would be tough to diagnose
the problem.
```

Back to the rules...

```
* <li><i>type</i>.jwc in WEB-INF
```

There are two impacts to this rule, both are questionable:

*If the user has one app in the war and has no `.application` file, then Tapestry has already installed a synthetic app spec in `/WEB-INF` anyways and this rule would never be encountered at runtime. If the is an application file in `/WEB-INF` then same applies.

*In any other scenario related to the location of the `.application` file, whether there is one or many applications in the war, and then this rule is just a mechanism for components to cross namespace boundaries.

Like the previous rule, removing this one is doing the Tapestry user community a service (in my opinion) by eliminating a case where boundary crossing can occur implicity.

```
* <li><i>type</i>.jwc in the application root (within the context root)
```

I don't really understand why this rule was added in the first place. This rule makes it possible to place a spec in a location outside of `/WEB-INF` which means it's possible that the xml file itself would be served to a browser, thus exposing the implementation of the application. yikes! It's an unwritten rule (or worse its a written rule and I missed it in the docs) that these files never be served.

Somebody help me out here. Why should this rule remain?

So, that covers all the removed rules. This does not eliminate the possibility that components will cross namespace boundaries. But it makes crossing an explicit action on the part of a Tapestry developer rather than an something that happens implictly due to the way Tapestry is implemented.

Ok, so how is it still explicity possible a component to cross boundaries? If the above changes are accepted I think there is just one way:

```
* <li>By searching for a named class file within the org.apache.tapestry.component-class-packages
```

The packages declared by using the meta property `org.apache.tapestry.component-class-packages` may overlap. They may overlap with the packages declared in the `org.apache.tapestry.component-class-packages` of other namespaces. But, that is an explicit choice made by the developer and not an implicit rule enforced by Tapestry.

Page Lookup

Most of the arguments made for Component lookup apply also to Page lookup. But there are some cases where boundary crossing is impossible to eliminate (and indeed may be desired by some. but not me). That case is described below and as it will definitely still exist, it should be explicity explained in the Tapestry documentation, warning of possible pitfalls.

Here are the proposed, modified, page lookup rules.

see PageSpecificationResolverImpl for the unmodified rules.

```
 <removed>
 * <li><i>simple-name </i>.page in the same folder as the application specification
 <removed>
 <removed>
 <removed>
 * <li><i>simple-name </i>.html as a template in the application root, for which an implicit
 * specification is generated
 * <li>By searching the framework namespace
 * <li>By invoking
 * {@link org.apache.tapestry.resolver.ISpecificationResolverDelegate#findPageSpecification(IRequestCycle,
INamespace, String)}
 * </ul>
 * <p>
 * Pages in a component library are searched for in a more abbreviated fashion:
 * <ul>
 <removed>
 * <li><i>simple-name </i>.page in the same folder as the library specification
 * <li>By searching the framework namespace
 * <li>By invoking
 * {@link org.apache.tapestry.resolver.ISpecificationResolverDelegate#findPageSpecification(IRequestCycle,
INamespace, String)}
```

Each of the removed rules above corresponds exactly to a rule I proposed be removed from the set of rules for component lookup. Apart from the rule removed due to my suggested new tag `<page-alias name="" full-name=""/>`, each removed rule eliminated a case where boundary crossing (this time for pages) could occur implicitly.

The remaining page rules allow boundary crossing to occur. The first is implicit and can onlu occur when there is more than one Tapestry application in a war file.

```
* <li><i>simple-name </i>.html as a template in the application root, for which an implicit
 * specification is generated
```

Can't remove this rule or the notion of specless page is a goner. This rule means that pretty much every template in the context is implicity a page in every application namespace in the war file. This needs to be documented.

The last rule:

```
 * <li>By invoking
 * {@link org.apache.tapestry.resolver.ISpecificationResolverDelegate#findPageSpecification(IRequestCycle,
INamespace, String)}
```

Is a user supplied class used to resolve pages when all else fails. Since the developer of the application supplies an implementation of `ISpecificationResolverDelegate`, no implicit boundary crossing can occur anyways. The implementation of Tapestry is free from responsibility.

And that's it for lookup rules. As a lark, I changed my local copy of Tapestry 3 to conform to my suggestions (not the alias tags though). At work here we have a Tapestry 3 application, otherwise I would have hacked the Tapestry 4 code. This app has over 300 pages and components (in total) and runs without any error I have been able to find so far. I suspect the same would be true if I had hacked up the Tapestry 4 source code. Perhaps not a good example as the application is proprietary and I can't share any implementation details. But one could take this as a sign that my suggested changes are not catastrophic.

There is one more issue. In Tapestry 3.0.2 or 3.0.3 a change was made to allow libraries to be defined in the context instead of the traditional place (the classpath). In Tapestry 3 I don't see this as a big deal but in Tapestry 4 this in effect makes every library page/component implicity a member of every application namespace in the war file. That's not good. I would suggest that libraries longer be allowed to be defined in the context.

Although, I may be missing something here. Help me out. Why was this change made? In other words, why is it a good thing that libraries can be defined in the context?

This is the end of the comments I wanted to talk about.

Rip away.