SimplifiedSpecificationsProposal

NOTE: This information only applies to Tapestry 3. Current versions do not require page and component specifications. See http://tapestry.apache.org/component-templates.html.

Problem Description

Tapestry page and component specifications are still too verbose. Tapestry 3.0 allowed much of the content of specifications to be moved into the HTML template, in the form of implicit components. However, implicit components are primarily used by beginners just learning Tapestry, or in isolated cases (trivial component references). In most real Tapestry applications, the separation of concerns mandated by Tapestry 2.3 and earlier (which did not have implicit components) is maintained, and the specifications continue to be quite verbose.

Proposed Solution

The 3.1 DTD will be a simplification of the 3.0 DTD. This will affect several different areas. The goal is to reduce the size of specifications, to rationalize naming, and to improve consistency.

Note that it is **expressly** required that Tapestry 3.0 DTDs be fully supported in Tapestry 3.1. This *may* be extended to Tapestry 2.3 DTDs as well.

<binding> and prefixes

In HTML templates, a prefix system is used to differentiate the different types of bindings:

HTML prefix	*3.0 specification equivalent
ognl:	`<binding>`
message:	`<message-binding`
no prefix	`<static-binding>`

This discord has been confusing for new users of Tapestry (those who came on in 3.0). Even for experienced Tapestry developers (myself included), it is common to make mistakes ... either using the wrong attribute (value instead of expression) or including a prefix in the specification.

For Tapestry 3.1, the three binding elements will be collapsed down to a single element, `<binding>`:

```
<binding name="..." value="..."/>

OR

<binding name="...">
  ...
</binding>
```

The value will be interpreted just as Tapestry 3.0 interprets HTML template attributes; an `ognl:` prefix will indicate an expression, thus:

```
<component id="inputName" type="ValidField">
  <binding name="displayName" value="message:name-label"/>
  <binding name="value" value="ognl:name"/>
  <binding name="validator" value="bean:nameValidator"/>
</component>

<component id="loop" type="Foreach">
  <binding name="source" value="ognl:items"/>
  <binding name="value" value="ognl:item"/>
  <binding name="element" value="tr"/>
  <binding name="index" value="ognl:index"/>
  <binding name="class">
    ognl:

      index % 2 == 0 ? "even" : "odd"
  </binding>
</component>
```

This example demonstrates the use of prefixes, including a proposed `bean:` prefix for a new type of binding for accessing JavaBeans (equivalent to the `beans` property. Once the basic mechanism for supporting prefixes is in place, it will be extremely easy to add new prefixes ... even application-specific prefixes.

The example also shows a long format binding, (`class`), is represented.

<inherited-binding>

In Tapestry 3.1, all component parameters will have a connected parameter property (see the ComponentParametersProposal). This will eliminate the need for the `<inherited-binding>` element.

<bean> and <set-property>

The `<bean>` element is used to define a managed JavaBean within a page or component ("managed bean" is an improved term over the one used in existing documentation, "helper bean").

Beans typically will require some configuration. Today, that configuration is only allowed as OGNL expressions, via the `<set-property>` element.

For 3.1, the `<set-property>` element will be replaced with a simpler `<set>` element, that operates in the same way as a `<binding>` element, i.e.:

```
<bean name="stringValidator">
  <set property="required" value="true"/>
  <set property="clientScriptingEnabled">
    ognl:clientScriptingEnabled
  </set>
</bean>
```

<external-asset>, <context-asset>, <private-asset>

As with `<binding>`, a prefix system will be instituted.

*3.0 element_'	prefix
`<context-asset>`	context:
`<external-asset>`	http:, https:, ftp:
`<private-asset>`	private:

No prefix will indicate a relative path. Some of the semantics need to be worked out (basically, private assets should be relative to the specification file, and context assets should be relative to the web context directory). Generally, the prefix will be omitted for context assets.

Example usage:

```
  <asset name="stylesheet" path="context:styles/standard.css"/>
  <asset name="icon" path="images/page-icon.png"/>
```

It will be possible to define additional prefixes for external assets via a HiveMind configuration.

<property> vs. <property-specification>

The Tapestry 3.0 `<property>` element is used to define meta-data. This element is very infrequently used. It will be renamed to `<meta>`.

The Tapestry 3.0 `<property-specification>` element is used to define a new property on a page or component. It is used very frequently and will be renamed to `<property>`.

<property[-specification]> simplifications

The Tapestry 3.0 `<property-specification>` element has a number of problems:

- It violates the DontRepeatYourself principal, since the Java type for the property is often in the specification and in the Java class (as abstract accessor methods)
- It requires a Java `type` to be specified, even when the type is irrelevant (such as when a property is only accessed via OGNL, which is typeless)
- It must be specified, even for transient properties that are fully defined in the Java class as abstract accessor methods.

In 3.1 the `<property>` element will be used to augment the information provided in the Java class.

Any abstract accessor method in the Java class will result in a concrete property in the fabricated subclass. This will eliminate the need for most `<property>` elements.

The `type` attribute can be omitted. If a value is specified, it must match the actual type defined by the Java class. If the Java class does not provide abstract accessors, then `type` will simply default to `java.lang.Object`.

- Note: Which makes me wonder if type is_ ever" useful?

A `<property>` element will still be required for any persistent properties. Again, `type` will typically be omitted here.

Discussion

HowardLewisShip: An outstanding question for me is how to handle `<extension>` and `<service>` in the application specification. I believe all services will be specified as HiveMind services contributed into a configuration point. Extensions are also somewhat irrelevant in the HiveMind scheme of things. Also, perhaps we can remove the `engine-class` element from `<application>`, since it will now be very, very rare to subclass `BaseEngine`.

In addition, I have thought of abandoning having a DTD in 3.1, which would allow an alternate format for bindings, which somewhat resembles Ant:

```
<component id="inputName" type="ValidField">
  <displayName value="message:name-label"/>
  <value value="ognl:name"/>
  <validator value="bean:nameValidator"/>
</component>

<component id="loop" type="Foreach">
  <source value="ognl:items"/>
  <value value="ognl:item"/>
  <element value="tr"/>
  <index value="ognl:index"/>
  <class>
    ognl:

      index % 2 == 0 ? "even" : "odd"
  </class>
</component>
```

That is, inside a `<component>` element, each element identifies a parameter of the component to bind. Saves a small amount of typing. Abandoning the DTD will allow other changes, such as changing `<page-specification>` to `<page>` and `<component-specification>` to `<component>`. That is, we have much more freedom to interpret elements based on context (`<component>` as a root element is different than `<component>` as a child element).

Status

The work on <binding> has taken place (including removal of <static-binding>, <message-binding> and <inherited-binding>).

<service> has been removed. A warning has been added for use of <service> in a 3.0 DTD specification.

Geoff threatened to kill me if I eliminated the DTD entirely. This was not unexpected.