Building Componentized Web Interfaces with Clay Instead of Tiles

Hermod Opstvedt
Chief Architect
DnB NOR ITU
A little history lesson
Building Componentized Web Interfaces with Clay Instead of Tiles

- **Struts**
  - Came about in June 2000.
  - Introduced by Craig R. McClanahan.

- **Tiles**
  - Came about in 2001.
  - Introduced by Cedric Dumoulin.
  - Was created as a templating framework for Struts.
  - Rewritten as a Struts plugin in July 2002.
  - Has become very popular in the Struts community, and also lately in the JSF community.
Building Componentized Web Interfaces with Clay Instead of Tiles

Java Server Faces

- Came about in as JSR 127 in May 2001.
  - Final release in May 2004
- Introduced by Craig R. McClanahan
- Reference implementation in June 2004.
- Now in 1.2 release as JSR 252.
Building Componentized Web Interfaces with Clay Instead of Tiles

- Shale
  - Came about in 2004.
  - Introduced by Craig R. McClanahan.
  - Currently in 1.3 snapshot.
Building Componentized Web Interfaces with Clay Instead of Tiles

- Clay
  
  ➢ Came about in 2005.
  ➢ Introduced by Gay VanMatre.
  ➢ Currently in 1.3 snapshot.
Building Componentized Web Interfaces with Clay Instead of Tiles

- First a little background on why you want to use templating.
- A web page actually consists of (normally) more than one distinct part, literally dividing the page into frames of information pieces. Each one of these again can be a candidate for reuse on any other Web page within the site. Also, within a Web site, keeping consistency with regards to look and feel is essential.
Building Componentized Web Interfaces with Clay Instead of Tiles

- What makes up a Web page?
Building Componentized Web Interfaces with Clay Instead of Tiles

- What makes up a Web page?
  - **Header**
    - Tells who we are.
    - Does not change during a session.
  - **Menu**
    - Gives the user choices of things to do/look at.
    - Does not change frequently during a session.
Building Componentized Web Interfaces with Clay Instead of Tiles

What makes up a Web page?

- Content area
  - Provides information to the user.
  - Constantly changes during a session.

- Footer
  - Provides some more static information.
  - Does not change frequently during a session.
Building Componentized Web Interfaces with Clay Instead of Tiles

- What makes up a Web page?
  - This means that $\frac{3}{4}$ of the parts of our Web page is fairly constant, and does not require change often.
  - Absolutely a candidate for breaking up into reusable parts.
  - Combine with Cascading Style Sheets (CSS) to further isolate effects of changes.
  - How do we go about doing that?
Building Componentized Web Interfaces with Clay Instead of Tiles

- How do we go about doing that?
  - Option 1:
    - Use a master page that we start with every time we want to provide some new content. Has all the static parts on it.
  - Advantages: Hardly any.
  - Disadvantages: Maintenance becomes a problem as our site grows. A change to, for instance, the header requires working through all our pages and changing the header part of each one of them.
How do we go about doing that?

Option 2:

- Create jsp parts of the static stuff. Create a master page that uses jsp:include of the jsp parts. We then use this master page as a starting point. (Struts templating)

Advantages: Not so vulnerable to changes to the static parts.

Disadvantages: Requires two pages for each new content. One with the content, and one which includes it. Also, a change in layout means working through all the pages to keep consistency.
Building Componentized Web Interfaces with Clay Instead of Tiles

Option 3:

• Use Templating.

Advantages: Immune to changes (Very loose coupling). Requires only one page to be created for each new content — The content itself.

Disadvantages: I don't see any (Negligible performance hit).
A simple Struts/Tiles application

- Start out by creating a standard Web application.
- Add struts jar files and dependencies.
- Add servlet/servlet-mapping for Struts servlet to web.xml.
- Create struts-config.xml file in WEB-INF directory of web application.
Building Componentized Web Interfaces with Clay Instead of Tiles

- Add Tiles plugin definition to struts-config.xml.
- Create tiles-defs.xml file in WEB-INF directory of web application.
web.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
<display-name>StrutsTiles</display-name>
<servlet>
  <servlet-name>action</servlet-name>
  <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
  <init-param>
    <param-name>config</param-name>
    <param-value>/WEB-INF/struts-config.xml</param-value>
  </init-param>
  <init-param>
    <param-name>debug</param-name>
  </init-param>
</servlet>
```
<init-param><param-name>detail</param-name><param-value>2</param-value></init-param><init-param><param-name>validate</param-name><param-value>true</param-value></init-param><load-on-startup>2</load-on-startup></servlet><servlet-mapping><servlet-name>action</servlet-name><url-pattern>*.do</url-pattern></servlet-mapping>
Building Componentized Web Interfaces with Clay Instead of Tiles

<jsp-config>
<taglib>
<taglib-uri>/WEB-INF/struts-bean.tld</taglib-uri>
<taglib-location>/WEB-INF/struts-bean.tld</taglib-location>
</taglib>
<taglib>
<taglib-uri>/WEB-INF/struts-html.tld</taglib-uri>
<taglib-location>/WEB-INF/struts-html.tld</taglib-location>
</taglib>
<taglib>
<taglib-uri>/WEB-INF/struts-logic.tld</taglib-uri>
<taglib-location>/WEB-INF/struts-logic.tld</taglib-location>
</taglib>
<taglib>
<taglib-uri>/WEB-INF/struts-nested.tld</taglib-uri>
<taglib-location>/WEB-INF/struts-nested.tld</taglib-location>
</taglib>
Building Componentized Web Interfaces with Clay Instead of Tiles

<taglib>
<taglib-uri>/WEB-INF/struts-template.tld</taglib-uri>
<taglib-location>/WEB-INF/struts-template.tld</taglib-location>
</taglib>
<taglib>
<taglib-uri>/WEB-INF/struts-tiles.tld</taglib-uri>
<taglib-location>/WEB-INF/struts-tiles.tld</taglib-location>
</taglib>
</jsp-config>
</web-app>
Building Componentized Web Interfaces with Clay Instead of Tiles

- **struts-config.xml**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD Struts Configuration 1.1//EN" "http://jakarta.apache.org/struts/dtds/struts-config_1_1.dtd">
<struts-config>
<!-- Data Sources -->
<data-sources></data-sources>
<!-- Form Beans -->
<form-beans></form-beans>
<!-- Global Exceptions -->
<global-exceptions></global-exceptions>
<!-- Global Forwards -->
<global-forwards></global-forwards>
<!-- Action Mappings -->
```
<action-mappings/>
<!-- Message Resources -->
<message-resources
parameter="css2006.clay.resources.ApplicationResources" />
<plug-in class="org.apache.struts.tiles.TilesPlugin">
<set-property property="definitions-config"
value="WEB-INF/tiles-defs.xml" />
<set-property property="definitions-debug" value="0" />
<set-property property="locale" value="true" />
<set-property property="definitions-parser-details" value="0" />
<set-property property="definitions-parser-validate"
value="true" />
<set-property property="moduleAware" value="false" />
</plug-in>
</struts-config>
Building Componentized Web Interfaces with Clay Instead of Tiles

- **tiles-defs.xml**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE tiles-definitions PUBLIC "+//Apache Software Foundation//DTD Tiles Configuration 1.1//EN" "http://jakarta.apache.org/struts/dtds/tiles-config_1_1.dtd">

<tiles-definitions>

</tiles-definitions>
```
Building Componentized Web Interfaces with Clay Instead of Tiles

- Define a layout

```
<table>
<thead>
<tr>
<th>Header</th>
</tr>
</thead>
<tbody>
<tr>
<td>LeftMenu</td>
</tr>
<tr>
<td>Content area</td>
</tr>
<tr>
<td>Footer</td>
</tr>
</tbody>
</table>
```
Building Componentized Web Interfaces with Clay Instead of Tiles

- Create a template jsp file to map the defined layout
  - Use tiles-get tag
  - Use div tags and positioning for layout
Building Componentized Web Interfaces with Clay Instead of Tiles

**template.jsp**

```html
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html"%>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean"%>
<html:html>
 <HEAD>
  <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
       pageEncoding="ISO-8859-1"%>
  <META http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
  <META http-equiv="Content-Style-Type" content="text/css">
  <TITLE>CSS StrutsTiles sample</TITLE>
  <TITLE></TITLE>
 </HEAD>
```
<BODY>
  <tiles:get name="header" flush="true" />
  <TABLE>
    <TBODY>
      <TR>
        <TD width="20%"><tiles:get name="leftmenu" flush="true" /></TD>
        <TD><tiles:get name="content" flush="true" /></TD>
      </TR>
      <TR>
        <TD colspan="2"><tiles:get name="footer" flush="true" /></TD>
      </TR>
    </TBODY>
  </TABLE>
</BODY>
<html:html>
Building Componentized Web Interfaces with Clay Instead of Tiles

- We added 4 tiles definitions in our template:
  - header
  - leftmenu
  - content
  - footer

- Now we need to define the template as a master layout in the tiles-defs.xml file
Add:

```html
<!-- Master definition -->
<!-- Main page layout used as a root for other pages definitions. -->
<definition name="site.mainLayout" path="/jsp/template.jsp">
  <put name="header" value="/jsp/defaultHeader.jsp"/>
  <put name="leftmenu" value="/jsp/defaultLeftNav.jsp"/>
  <put name="content" value="site.frontpage.body"/>
  <put name="footer" value="/jsp/defaultFooter.jsp"/>
</definition>
```

...
Building Componentized Web Interfaces with Clay Instead of Tiles

- Note that we did not define a fixed jsp for the content name. This is because we want to override this definition for each content.

- Next create the jsp files that will represent the header, leftmenu and footer. These are normally relative static files and therefore we defined them within the master definition.
Building Componentized Web Interfaces with Clay Instead of Tiles

- defaultHeader.jsp
  <h1>CSS 2006 Clay header</h1>

- defaultLeftNav.jsp
  <div id="menu">
    <div id="button" class="button">
      <ul>
        <li><a href="index.do">Home</a></li>
        <li><a href="page2.do">Page2</a></li>
        <li><a href="page3.do">Page3</a></li>
      </ul>
    </div>
  </div>
Building Componentized Web Interfaces with Clay Instead of Tiles

- defaultFooter.jsp

<p>Colorado Software Summit 2006</p>
Building Componentized Web Interfaces with Clay Instead of Tiles

- Define the Struts actions

...  

<!-- Action Mappings -->  

<action-mappings>  

<action path="/index" forward="site.frontpage"/>  
<action path="/page2" forward="site.page2"/>  
<action path="/page3" forward="site.page3"/>  

</action-mappings>  

...
Add the tiles definitions:

```xml
...<definition name="site.frontpage" extends="site.mainLayout">
  <put name="content" value="/jsp/welcome.jsp"/>
</definition>
<definition name="site.page2" extends="site.mainLayout">
  <put name="content" value="/jsp/page2.jsp"/>
</definition>
<definition name="site.page3" extends="site.mainLayout">
  <put name="content" value="/jsp/page3.jsp"/>
</definition>
...
Building Componentized Web Interfaces with Clay Instead of Tiles

- And finally deploy and run it on a server, and open a browser on the front-page:
Building Componentized Web Interfaces with Clay Instead of Tiles

- Now lets create a Clay implementation.
- But first a little bit about JSF, Shale and Clay
- What is JSF?
  - JavaServer Faces (JSF) is a new standard Java framework for building Web applications.
  - Provides a component-centric approach to developing Java Web user interfaces.
  - Has integrated the Model-View-Controller (MVC) design pattern into its architecture
Building Componentized Web Interfaces with Clay Instead of Tiles

- Is a Java standard through Java Community Process (JCP).
- Well suited for implementing “drag and drop” functionality in various IDE’s.
Building Componentized Web Interfaces with Clay Instead of Tiles

- What is Shale?
  - An extension of JavaServer Faces
  - Utilizes existing front controller architecture
  - Utilizes extension points to add functionality
  - View Lifecycle Events
  - Web Flow
  - Ajax Support
  - Validation
  - Spring and Tiles Integration
  - Parameterized Subtrees
Building Componentized Web Interfaces with Clay Instead of Tiles

- What is Clay?
  - Lets you graft a component subtree onto an existing component tree.
  - HTML Views.
    - Define your views in HTML.
    - Tie HTML elements to JSF components with a jsfid attribute.
    - Replaces static HTML elements that have jsfid attributes with their component counterpart.
Building Componentized Web Interfaces with Clay Instead of Tiles

- Meta-data Inheritance and Tiles-like Composition.
  - Define a component as an extension of an existing component and then override or add attributes.
  - Similar to the inheritance mechanism built into Tiles.
- Symbols.
  - Map managed beans to symbols.
  - Reuse a view for many different managed beans.
Building Componentized Web Interfaces with Clay Instead of Tiles

- Enough background – Lets try it out.
  ➢ Start out by creating a standard Web application
  ➢ Add the JSF, Shale and Clay libraries.
  ➢ Configure the JSF, Shale and Clay stuff in web.xml.
    • Add Filter and filter mappings for MyFaces JSF implementation.
    • Add Filter and filter mappings for Shale.
    • Add Servlet configuration for FacesServlet.
    • Add necessary config parameters.
Building Componentized Web Interfaces with Clay Instead of Tiles

- Create faces-config.xml in WEB-INF
- For a complete description of setting up Shale and JSF visit:
  
  ➢ http://shale.apache.org/using.html
Configuring clay:

- Create chain-config.xml in WEB-INF
- Add ClayViewHandlerCommand definition to it.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<catalogs>
<!-- Define preprocessing command chain for Shale to execute -->
<catalog name="shale">
  <chain name="preprocess">
    <!-- This command is only needed for full clay html views with myfaces -->
    <command className="org.apache.shale.clay.faces.ClayViewHandlerCommand" />
    <!-- This filter command wakes up the watchdog monitoring the Clay configuration files for change. -->
    <command className="org.apache.shale.clay.config.beans.ConfigDefinitionsWatchdogFilter" includes="\S*\.*\.*\.*html,\S*\.*jsp,\S*\.*xml" />
  </chain>
</catalog>
</catalogs>
```
Building Componentized Web Interfaces with Clay Instead of Tiles

- Create clay-config.xml in WEB-INF
- This is the file that will contain our clay component definitions etc.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<view>

</view>
```
Building Componentized Web Interfaces with Clay Instead of Tiles

- So now that the infrastructure is in place, we start creating the actual Web application.
- First we need to create the master template.
  - Remember that we are only dealing with html at this point, so the template may come from our master html guru.
Building Componentized Web Interfaces with Clay Instead of Tiles

- template.html

```html
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"/>
<title><span jsfid="outputText" value="@title"></span></title>
<link href="/theme/styles.css" rel="stylesheet" type="text/css" />
<link href="/theme/styles.css" rel="stylesheet" type="text/css" />
<link href="../theme/styles.css" rel="stylesheet" type="text/css" />
<style type="text/css">
/*<![CDATA[*/
html {overflow-x:auto; overflow-y:hidden;}
/*]]>*/
</style>
</head>
<body>
<div id="head"><span jsfid="clay" clayjsfid="@headerContent" allowbody="false">Header Content</span></div>
<div id='logo'>&nbsp;</div>
<div id="left"><span jsfid="clay" clayjsfid="@leftContent" allowbody="false">Left Content</span></div>
<div id='content'><span jsfid="clay" clayjsfid="@bodyContent" allowbody="false">Body Content</span></div>
<div id='pad1'>&nbsp;</div>
<div id="footer"><span jsfid="clay" clayjsfid="@footerContent" allowbody="false">Footer Content</span></div>
</body>
</html>
```
Building Componentized Web Interfaces with Clay Instead of Tiles

- **What have we replaced?**
  - All references to `tiles:get` have been replaced by `span` tags.
    - Ie `<tiles:get name="top" flush="true" />`
    - Replaced with `<span jsfid="clay" clayjsfid="@headerContent" allowbody="false">Header Content</span>`
  - Notice that there are some non-standard html attributes in the span tag
Building Componentized Web Interfaces with Clay Instead of Tiles

- **jsfid**
  - Indicates which type of JSF component.

- **Clayjsfid**
  - Names the Clay component to use

- **allowBody**
  - Tells Clay that the content between the start and end tag should be rendered or not.
These are ignored by browsers, so if we open up template.html in a browser it will look very similar to the end result, with the exception of the left menu. We could even write the template so that it had a mockup for that.
Building Componentized Web Interfaces with Clay Instead of Tiles
Building Componentized Web Interfaces with Clay Instead of Tiles

- So now that we have a template in place we need to define the Clay components them self.

- We start out by creating the content of the components, in our case content for:
  - @headerContent
  - @leftContent
  - @bodyContent
  - @footerContent
Building Componentized Web Interfaces with Clay Instead of Tiles

- We create these as html files, containing only the html fragment we need. These are in our case exactly the same as the Tiles jsp files, with the exception of defaultLeftNav.html. In our Clay version we define the menu using Clay components.
We use the standard commandLink component for our navigation. This component is already defined for us in the clay-config that is part of Clay.
Next we need to define the components in the clay-config.xml file. We start by declaring the basic structure, as we did with Tiles version

```
<component jsfid="baseLayout" extends="clay" id="base">
  <attributes>
    <set name="clayJsfid" value="/templates/standard.html" />
  </attributes>
  <symbols>
    <set name="title" value="Hello World" />
    <set name="leftContent" value="/pages/defaultLeftNav.html" />
    <set name="headerContent" value="/pages/defaultHeader.html" />
    <set name="bodyContent" value="/pages/defaultBody.html" />
    <set name="footerContent" value="/pages/defaultFooter.html" />
  </symbols>
</component>
```
So now we have our base definition in place. Note the symbol attributes we have defined. These will replace the @... attributes in the template.html file when Clay processes it.

Next we need to define the various pages or views that we want our application to have. We do this by creating a new xml file: clay-views-config.xml.
Building Componentized Web Interfaces with Clay Instead of Tiles

- We have already defined this in web.xml as the value for the context-param: org.apache.shale.clay.FULLXML_CONFIG_FILES
- This is how we tell Clay that we want to declare our views (or composition thereof). If we did not do this, we would have had to create an xml file for each and every view.
Building Componentized Web Interfaces with Clay Instead of Tiles

- First we define page1:

  <component jspid="/page1.xml" extends="baseLayout">
    <symbols>
      <set name="title" value="Page 1" />
      <set name="bodyContent" value="/pages/page1Body.html" />
    </symbols>
  </component>

- Here we tell clay what to substitute for the @bodyContent symbol.
Building Componentized Web Interfaces with Clay Instead of Tiles

- Next, we need to create the /pages/page1Body.html file, which has the same content as the Tiles version, except for the name in the header.

<h3>Clay template application - Page 1</h3>
<p>This is Page 1 content</p>
Before we deploy and run our web application on the server, we create an entry to our application: index.jsp. This file simply redirects to our Clay view.

```jsp
<% String redirectURL = "/page1.xml";
    response.sendRedirect(redirectURL);
%>
Building Componentized Web Interfaces with Clay Instead of Tiles

CSS 2006 Clay header
Next we define the other views in clay-views.xml and create the bodycontent files for them

... 
<component jsfid="/page2.xml" extends="baseLayout"> 
  <symbols> 
    <set name="title" value="Page 2" /> 
    <set name="bodyContent" value="/pages/page2Body.html" /> 
  </symbols> 
</component> 

<component jsfid="/page3.xml" extends="baseLayout"> 
  <symbols> 
    <set name="title" value="Page 3" /> 
    <set name="bodyContent" value="/pages/page3Body.html" /> 
  </symbols> 
</component> 
...
Building Componentized Web Interfaces with Clay Instead of Tiles

- page2Body.html

  <h3>Clay template application - Page 2</h3>
  <p>This is Page 2 content</p>

- page3Body.html

  <h3>Clay template application - Page 3</h3>
  <p>This is Page 3 content</p>
There one item that we have not payed attention to: the menu (defaultLeftNav.html).

This uses standard JSF components, in our case it’s the components from the MyFaces tomahawk library. To use these components we need to add a Clay definition file for these components.
Building Componentized Web Interfaces with Clay Instead of Tiles

- This file comes along with the Clay distribution: tomahawk-1_1_3-config.xml
- This file declares most of the Tomahawk components as Clay components, ready for us to use.
- In our menu we use the the t:commandLink component. This has the following declaration
Building Componentized Web Interfaces with Clay Instead of Tiles

<component jsfid="t:commandLink"
  componentType="org.apache.myfaces.HtmlCommandLink"
  extends="baseAction">
  <attributes>
    <set name="id" bindingType="VB" />
    <set name="binding" bindingType="VB" />
    <set name="accesskey" bindingType="VB" />
    <set name="charset" bindingType="VB" />
    <set name="coords" bindingType="VB" />
    <set name="hreflang" bindingType="VB" />
    <set name="rel" bindingType="VB" />
    <set name="rev" bindingType="VB" />
    <set name="shape" bindingType="VB" />
    <set name="tabindex" bindingType="VB" />
    <set name="type" bindingType="VB" />
    <set name="onblur" bindingType="VB" />
    <set name="onfocus" bindingType="VB" />
    <set name="target" bindingType="VB" />
    <set name="enabledOnUserRole" bindingType="VB" />
    <set name="visibleOnUserRole" bindingType="VB" />
    <set name="forceId" bindingType="VB" />
    <set name="forceIdIndex" bindingType="VB" />
  </attributes>
</component>
Building Componentized Web Interfaces with Clay Instead of Tiles

- We only override the following attributes:
  - action
  - allowBody
  - Immediate

- In order for the navigation to work we also need to declare the navigation rules in the faces-config.xml file
Building Componentized Web Interfaces with Clay Instead of Tiles

```xml
<faces-config>
  ...
  <navigation-rule>
    <from-view-id>*</from-view-id>
    <navigation-case>
      <from-outcome>home</from-outcome>
      <to-view-id>/page1.xml</to-view-id>
    </navigation-case>

    <navigation-case>
      <from-outcome>page2</from-outcome>
      <to-view-id>/page2.xml</to-view-id>
    </navigation-case>

    <navigation-case>
      <from-outcome>page3</from-outcome>
      <to-view-id>/page3.xml</to-view-id>
    </navigation-case>
  </navigation-rule>
  ...
</faces-config>
```
Building Componentized Web Interfaces with Clay Instead of Tiles

- So when we now start our web application, we have a fully functional navigation too.
- If we sum up a little, we have an application that solely consists of files with .html markup with some added parameters to the tags.
- This means that we can leave the gui work to the guys who really know gui design/interaction and as programmers concentrate on functionality.
Defining our own Clay components.

- If we want to make our components we define them much like we have done so far, as components.
- Let's create reusable Label component.
Building Componentized Web Interfaces with Clay Instead of Tiles

- We define it as follows:

  ```
  <component jsfid="widgetsLabel" extends="baseLabel">
    <attributes>
      <set name="value" value="@label" />
      <set name="for" value="@property" />
    </attributes>
  </component>
  ```

- We can now use this in a .html file as follows:

  ```
  <span jsfid="clay" clayjsfid="widgetslabel" label="Name" property="#{myBean.textvalue}">A label</span>
  ```
Some more samples – Create a .html component, in this case a reusable news component (/template/news.html).

```html
<div class="story">
    <h3 class="newshead"><span jsfid="newsHeader">Story Title</span></h3>
    <p><span jsfid="newsBody">News body.</span></p>
</div>

<component jsfid="news" extends="clay" id="news">
    <attributes>
        <set name="clayJsfid" value="/template/news.html" />
    </attributes>
</component>

...<component jsfid="newsHeader" extends="outputText" allowBody="false">
    <attributes>
        <set name="escape" value="false" />
        <set name="value" value="#{@managed-bean-name.heading}" />
    </attributes>
</component>
...
<component jsfid="newsBody" extends="outputText allowBody="false">
  <attributes>
    <set name="escape" value="false" />
    <set name="value" value="#{@managed-bean-name.body}" />
  </attributes>
</component>

- And we can then reuse this using:
  &lt;span jsfid="clay" clayjsfid="news">A news</span>
Building Componentized Web Interfaces with Clay Instead of Tiles

- Time permitting, some more advanced samples.
Building Componentized Web Interfaces with Clay Instead of Tiles

- References:
  - http://shale.apache.org
Building Componentized Web Interfaces with Clay Instead of Tiles