ANYWARE TECHNOLOGIES

Advanced Cocoon Forms

Sylvain Wallez
http://apache.org/~sylvain

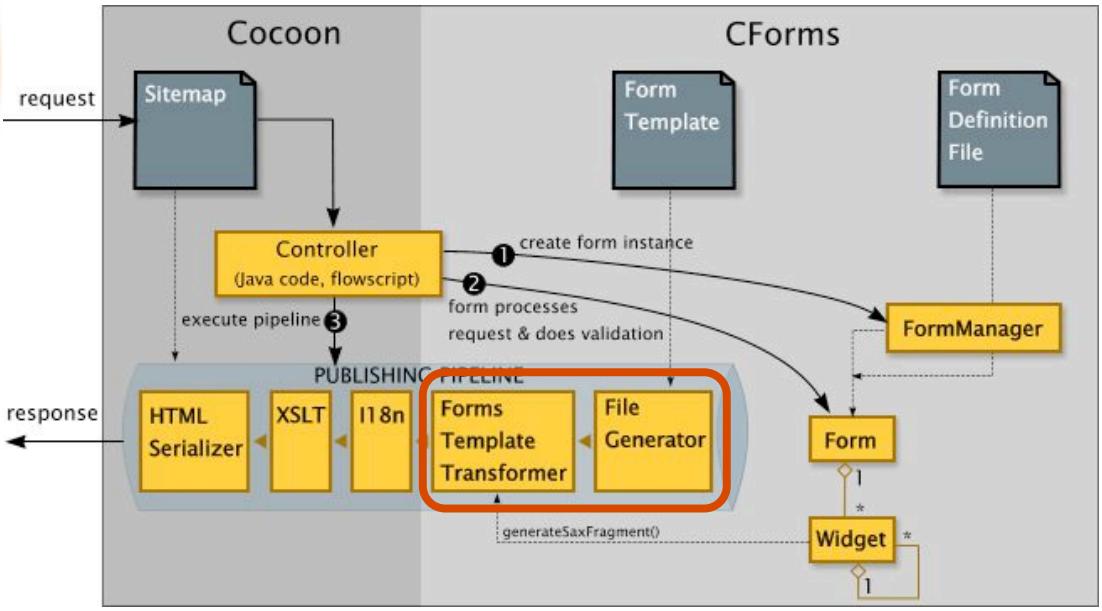
www.anyware-tech.com

Overview

- The classical Cocoon Forms pipeline
- Dynamic form templates
- Form variants: <fd:union>
- Recursive forms: <fd:class> and <fd:new>

The classical Cocoon Forms pipeline

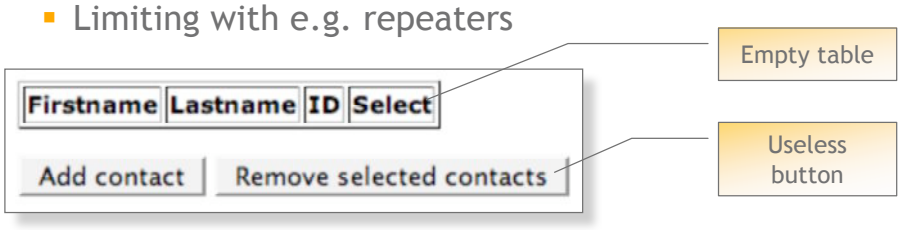
The big picture



The classical Cocoon Forms pipeline

Limitations

- The form layout cannot depend on form contents
  - →Forms transformer comes *after* template generation
  - Adequate for simple forms
  - Limiting with e.g. repeaters



  - →Empty repeater layout needs to be different
- Workaround: direct form access with JXTemplate
  - Limited to top-level widgets

Dynamic form templates

What is it?

- CForms template language implemented with JX macros
  - Import the macro file
  - Remove the forms transformer!

```
<jx:import
    uri="resource://org/apache/cocoon/forms/generation/jx-macros.xml"/>
```

(get it from SVN, will be in 2.1.6)

- Provides new variables within template elements
  - →Use them to access for state and build the template

Dynamic form templates

New JX variables

- Anywhere within <ft:form-template>
  - "form": the current form
- Within <ft:*> (widget, validation-error, struct…)
  - "widget": the current widget

```
<jx:choose>
  <jx:when test="${widget.getChild('contacts').getSize() == 0}">
    <p><strong><em>There are no contacts to display</em></strong></p>
  </jx:when>
  <jx:otherwise>
    <table border="1">
      <tr>
        <th><ft:repeater-widget-label id="contacts" widget-id="firstname"/></th>
        <th><ft:repeater-widget-label id="contacts" widget-id="lastname"/></th>
```

Dynamic form templates

New JX variables within <ft:repeater-widget>

- "repeater" is the repeater
- "widget" is the current row
- "repeaterLoop": the <jx:forEach> iterator
  - →"repeaterLoop.index": row number
  - →"repeaterLoop.first": at first row?
  - →"repeaterLoop.last": at last row?

Dynamic form templates

New JX variables within <ft:repeater-widget>

```
<ft:repeater-widget id="contacts">
  <tr class="forms-row-${repeaterLoop.index % 2}">
    <td><ft:widget id="firstname"/></td>
    <td><ft:widget id="lastname"/></td>
    <td><ft:widget id="ID"/></td>
    <td>
      <jx:choose>
        <jx:when test="${repeaterLoop.first}">
          <img src="resources/blank_btn.gif"/>
        </jx:when>
        <jx:otherwise>
          <ft:widget id="up">
            <fi:styling type="image" src="resources/move_up.gif"/>
          </ft:widget>
        </jx:otherwise>
      </jx:choose>
    </td>...
```

Alternating CSS classes

Don't display "up" in first row

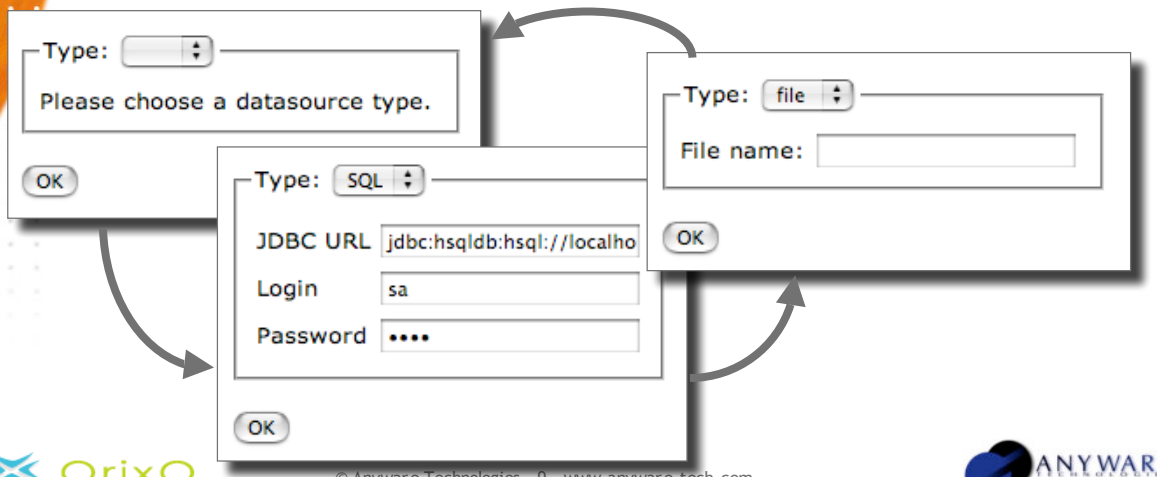
Firstname	Lastname	ID	Select
Harry	Potter	2	
Anakin	Skywalker	1	
		3	

Add contact | Remove selected contacts

Form variants with <fd:union>

Use case: a datasource chooser

- Some common data (e.g. name)
- Some data that depend on the datasource type
  - SQL: JDBC URL, credentials
  - File: file name



Form variants with <fd:union>

Form definition

```
<fd:field id="sourcetype">
  <fd:datatype base="string"/>
  <fd:selection-list>
    <fd:item value=""/>
    <fd:item value="SQL"/>
    <fd:item value="file"/>
  </fd:selection-list>
</fd:field>

<fd:union id="datasource" case="sourcetype">
  <fd:widgets>
    <fd:struct id="SQL">
      <fd:widgets>
        <fd:field id="jdbc-url">
          <fd:datatype base="string"/>
        </fd:field>          ...
      </fd:widgets>
    </fd:struct>
    <fd:struct id="file">
      <fd:widgets>
        <fd:field id="filename">
          <fd:datatype base="string"/>
        </fd:field>
      </fd:widgets>
    </fd:struct>
  </fd:widgets>
</fd:union>
```

Widget discriminating the variants (must be "string")

Reference to the discrimitating widget

One child for each possible value (struct is a container)

Form variants with <fd:union>

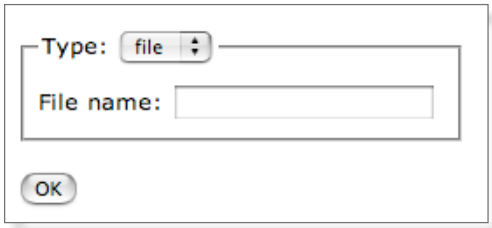
Form template

```
<fieldset>
  <legend>Type:
    <ft:widget id="sourcetype">
      <fi:styling submit-on-change="true"/>
    </ft:widget>
  </legend>

  <ft:union id="datasource">
    <ft:case id="">
      Please choose a datasource type.
    </ft:case>
    <ft:case id="SQL">
      <ft:struct id="SQL">
        JDBC URL: <ft:widget id="jdbc-url"/>
      </ft:struct>
    </ft:case>
    <ft:case id="file">
      <ft:struct id="file">
        File name: <ft:widget id="filename"/>
      </ft:struct>
    </ft:case>
  </ft:union>
</fieldset>
```

Immediately post changes

Template snippets for each variant



ft:struct simply enters the container

Form variants with <fd:union>

Binding

```
<fb:context path="config/datasource">

  <fb:value id="sourcetype" path="@type"/>

  <fb:union id="datasource" path=".">
    <fb:case id="SQL" path=".">
      <fb:struct id="SQL" path=".">
        <fb:value id="jdbc-url" path="URL"/>
        <fb:value id="login" path="login"/>
        <fb:value id="password" path="password"/>
      </fb:struct>
    </fb:case>
    <fb:case id="file" path=".">
      <fb:struct id="file" path=".">
        <fb:value id="file" path="file/@path"/>
      </fb:struct>
    </fb:case>
  </fb:union>
</fb:context>
```

Bind the discriminant widget first!

Variant-specific binding

```
<config>
  <datasource type="SQL">
    <URL>jdbc:hsqldb:hqsl:...</URL>
    <login>sa</login>
    <password>foo</password>
  </datasource>
</config>
```

- Constraint: the discriminant's binding cannot depend on the variant structure
  - → If needed, <fb:javascript> binding

Recursive forms: <fd:class> & <fd:new>

Overview

- <fd:class> defines a "definition snippet"
- <fd:new> instanciates the class (eh!)
  - It inlines the class contents
- Use cases
  - Reusable form snippets
    - Limited to a single form (for now)
  - Recursive forms
    - →A <fd:class> can instanciate itself!

Recursive forms: <fd:class> & <fd:new>

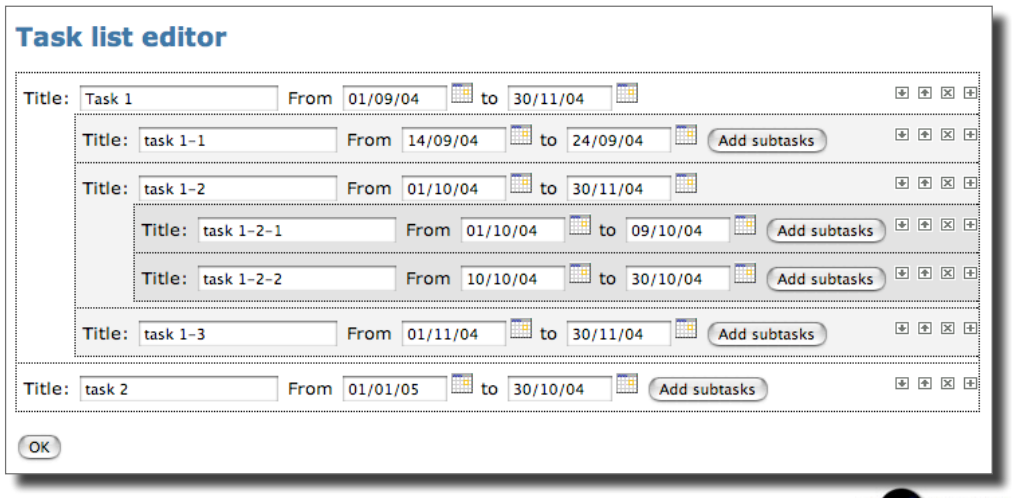
Recursive forms: when?

- Useful for deeply nested data structures
  - Document sections and sub-sections
  - IDE-like webapps (XML editors)
  - Tree structures
- Works well with <fd:union> and <fd:repeater>
  - Contents of a node is often not fixed
  - Contents is often a collection

Recursive forms: <fd:class> & <fd:new>

Use case: project breakdown

- A task is defined by
  - →A title, start date and end date
  - →An optional list of subtasks



Recursive forms: <fd:class> & <fd:new>

Form definition

List of tasks

```
<fd:class id="task-class">
  <fd:widgets>
    <fd:struct id="task">
      <fd:widgets>
        <fd:field id="title">
          <fd:datatype base="string"/>
        </fd:field>
        <fd:field id="start">
          <fd:datatype base="date"/>
        </fd:field>
        <fd:field id="end">
          <fd:datatype base="date"/>
        </fd:field>
        <fd:new id="tasklist-class"/>
      </fd:widgets>
      <fd:row-action id="add"
          action-command="add-after"/>
      <!-- other actions snipped -->
    </fd:struct>
  </fd:widgets>
</fd:class>
.../...
```

Subtasks

Action for "tasks" repeater (inlined)

```
.../...
<fd:class id="tasklist-class">
  <fd:widgets>
    <fd:repeater id="tasks">
      <fd:widgets>
        <fd:new id="task-class"/>
      </fd:widgets>
    </fd:repeater>
    <fd:repeater-action id="addsub"
        repeater="tasks"
        action-command="add-row">
      <fd:label>Add subtasks</fd:label>
    </fd:repeater-action>
  </fd:widgets>
</fd:class>

<fd:new id="tasklist-class"/>
```

Top-level task list

Recursive forms: <fd:class> & <fd:new>

Form template

```
<ft:class id="task-class">
  <div class="section>
    <ft:struct id="task">
      <span class="actions">
        <ft:widget id="down"/>
        <ft:widget id="up"/>
        <ft:widget id="delete"/>
        <ft:widget id="add"/>
      </span>
      Title: <ft:widget id="title"/>
      From <ft:widget id="start/>
      to <ft:widget id="end/>
      <ft:new id="tasklist-class"/>
    </ft:struct>
  </div>
</ft:class>
```

Indentation using CSS

Row actions

Subtasks

```
<ft:class id="tasklist-class">
  <jx:choose>
    <jx:when
      test="${widget.getChild('tasks').getSize() > 0}">
      <ft:repeater-widget id="tasks">
        <ft:new id="task-class"/>
      </ft:repeater-widget>
    </jx:when>
    <jx:otherwise>
      <ft:widget id="addsub"/>
    </jx:otherwise>
  </jx:choose>
</ft:class>

<ft:new id="tasklist-class"/>
```

Dynamic layout (jx-macros)

Starting point

Recursive forms: <fd:class> & <fd:new>

Binding

```
<fb:context path="project">

  <fb:class id="tasklist-class">
    <fb:simple-repeater id="tasks"
        parent-path="tasks" row-path="task">
      <fb:new id="task-class"/>
    </fb:simple-repeater>
  </fb:class>

  <fb:class id="task-class">
    <fb:struct id="task" path=".">
      <fb:value id="title" path="title"/>
      <fb:value id="start" path="@start">
        <fd:convertor datatype="date"
            type="formatting" style="short"/>
      </fb:value>
      <fb:value id="end" path="@end">
        <fd:convertor datatype="date"
            type="formatting" style="short"/>
      </fb:value>
      <fb:new id="tasklist-class"/>
    </fb:struct>
  </fb:class>

  <fb:new id="tasklist-class"/>
</fb:context>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<project>
  <tasks>
    <task end="11/30/04" start="9/1/04">
      <title>Task 1</title>
      <tasks>
        <task end="9/24/04" start="9/14/04">
          <title>task 1-1</title>
        </task>
        <task end="11/30/04" start="10/1/04">
          <title>task 1-2</title>
        </task>
        <task end="11/30/04" start="11/1/04">
          <title>task 1-3</title>
        </task>
      </tasks>
    </task>
    <task end="10/30/04" start="1/1/05">
      <title>task 2</title>
    </task>
  </tasks>
</project>
```

Conclusion

Can other form frameworks do that?

No. Or at least not so easily.

Cocoon Forms rulez :-)

… and there's more to come!

Many thanks to all CForms contributors and
especially to Tim Larson for <fd:union> and <fd:class>