GT2003Sylvain

Sylvain Wallez - Flowscript and Woody: forms and web applications made easy.

Given the amount of energy that Sylvain has put into Woody recently, this is a much anticipated talk. Historical?

Sylvain is the "Grandfather of the Interpreted Sitemap Engine"

Hey, I'm not that old: "father" is enough 😉 --SylvainWallez

Was going to talk about Flowscript and Woody but thinks he has too much material has been presented already - will concentrate on Woody. Questions about Flowscript are ok though.

Sylvain's Company uses Cocoon for everything:

- Embedded systems for cars and automation
- Documentation for Aircraft manufacturers
- Websites (Hey!!)

He's currently concentrating on Woody.

Woody Intro

- Cocoon started as a web publishing framework
- Today: general-purpose framework
- Tomorrow: Forms handling, validation features, web applications!

Previous forms modules: Precept, XMLForm have died due to lack of community support. XMLForms: XPath injection problem requires writing a shielding bean.

Woody was started by Outerthought: Bruno, Marc, Steven.

Woody principles

Nice slide on "the big picture" (historians trace it to Bruno Dumon). Me loves grafix 😉

- Form Template
- Form Definition file
- Bindings to map form values to application data
- Form Object Model, composed of Widgets
    - represent a piece of data on a form
    - can load, parse validate themselves
    - some widgets are non-terminal (support for tables, rows)
    - plugable datatypes (you can add your own)
    - plugable validators (you can add your own)

(Diagram, sample of Form Definition)

(Diagram, sample of Form Template)

Form template contains Woody template tags that refer to widgets in the Form Object Model Add your woody tags to whatever XML you want, transform it with the WoodyTransformer to access the properties in your view

Woody Widgets

Available widgets

- wd:form - main form widget
- wd:field - "atomic" input field
- wd:booleanfield - boolean input (checkbox etc.)
- wd:multivaluefield - multiple selections in a list
- wd:repeater - collection of widgets
- wd:output - non-modifiable value
- wt:group - visual grouping
- wt:choice - selection list
- wd:action - action button (intra form)
- wd:submit - form submit (exits form)

Typically an "add row" button will be action, "OK" will be submit They're all defined in cocoon.xconf, add your own if needed

Widgets take care of their own parsing, you get high-level typed data without effort.

Defining a datatype

- base type - java type (string, long, decimal, ...)
- converter - how is the data converted to a java object according to the needs of the form (locale-based formatting etc.)
- validator - adds more controls to the value and give meaningful error messages

Converter: component that can read, parse and format a data type. Formatting and parsing can be dependent on locale.

Validation

- built in validators: length, range, regexp, creditcard, assert, email
- Validators are Components, you can add your own
- You can use a sequence of validators in a widget
- Each validator can output its own messages

Selection lists

- Can be external and dynamic! Built from a different Cocoon pipeline.
- Can also be dynamic - you can get your list from a pipeline
- There's a switch to turn on/off caching

Repeater Widget

- Repeats a number of child widgets
- Can be used to manage collections
- Can be nested within each other
- Can be used for addition and removal of rows

The View

The Woody template Transformer takes the Woody form definition and a Form Template, which can come from any Generator.

(Diagram of woody template before and after transformation by the WoodyTransformer)

Role of the Woody transformer

Expands widget definitions into the form template. Widgets are responsible for their own xml representation

Resulting output contains all you need to present the form, but in a presentation-independent way.

"wt" Woody Template elements are expanded into "wi" Woody Instance elements. Output goes to styling You can provide styling hints

Repeater-widget automatically numbers generated elements, so that they can be identified when the form is submitted.

Higher-level styling

- Instance-only widgets are used for form layout only, they are not present in the model. Shows cool wi:styling with layout="columns" example.
- Plays well with CSS, needs little or no other coding

Event handling

Has been added to Woody about two weeks ago.

- Form definition supports event listeners on widgets
- Server-side JavaScript, can modify the form when a control is activated on the client (Dynamic selection list sample)
  - event.newValue
  - event.source
  - event.oldValue (previous value of the widget)
- can call flowscript functions from your JS event handlers

Shows the "car selector" example from the current Woody samples.

Woody Binding: Linking the Form to Application data

- Use XPath to map form to application data. Shows an example binding to XML

Woody Action

For simple forms that have no complex logic, Woody can be used without flowscript.

Integration with FlowScript

- Provides Form class, constructor takes form definition file

- Form.showForm() is like sendPageAndWait(), waits for action of user on form and automatically redisplays it until the form is valid.

Load the form definition into the form Add the Binding document to the form Load the data

(network downtime - SubEthaEdit connection lost here)

Putting it all together

- A powerful forms framework
  - Rich datatypes, validation rules
  - Easy extension
  - Event handling
  - Powerful built-in stylesheets
  - Fancy visual widgets (Calendar, Tabs etc.)
- Community development
  - Started by Outerthought
  - Other people have jumped in
- Woody will be **the** Cocoon Forms?
- client-side validation planned

Question

- Do Woody and XForms have some compatiblity
  - Could you use Woody to send XForms to XForms clients?

A: Not compatible due to Woody working Server-side and XForms being client-side

- Multimedia rendering?

A: Because you can transform your form after the WoodyTransformer does it's job, any type of XML can be output.