CubistForms

This is a for-discussion-and-revision document, not a statement of fact. So, please jump in, discuss, revise, and thoroughly refactor 🙂

The page layout is intentionally rough to encourage perfectionists to contribute 😉

First let's express the needs from the standpoint of various people:

Form Definition Designer (i.e. the one on the 'cubist' spot, looking at it from all viewpoints, needing to supporting hooks to enable the needs of the others.) He does seem to have an own agenda for that:

- definition of the HTTP-interface
    - names and 'styles' of request-parameters supported on this form
- easy management through definition reuse and definition repositories
- specifying datatyping and validation
- local event-handling (with no effects outside the form scope)
- providing hooks to the processing (actions, submit with or without validation,...)
- extensibility through custom widgets (probably not), datatypes and convertors.
- ability to dynamically change the label or the i18n key of the label.

User (What does the end user care about?)

- Data Entry
  *field (string, number, date, all with various formatting)
    - "required" indicator *"help" link *selection lists **"please choose" message **default selection(s) **option sets
      *toggle (checkbox, set of radio buttons)
      *text entry (textarea, htmlarea, other web-based editors)
      *grid (table)
- Navigation
  *buttons (trigger action, validate, submit form)
  *drill-down/help/etc. (expand inline, floating pane, spawned window, replaced page)
- Buglets
  *Page scroll position is lost after a round-trip to the server.

Visual/HTML Designer (How do we implement this, and how do we make it look the way we want?)

- Data Entry
  *field (input)
  *toggle (input type="checkbox", input type="radio")
  *text entry (whole list)
  *grid
- Navigation
  *buttons (all types)
  *drill-down/help/etc.
  *expand inline
  **div + css + js
  **"choice" + POST + no js needed
  *floating pane
  **div + css + js
  **"choice" + POST + no js needed
  *spawned window (GET/POST)
  *replaced page (GET/POST)
- Non-editable Data Display
  *field (with all its types and various formatting)
  *toggle (all types)
  *grid (table)
- Other Stuff

Binding Programmer (How do we interface with the data sources and data sinks?)

- Dynamic selection lists tied to live db queries, DAO's, lists, hashes, etc.
- For a Repeater part of the identity may be:
  *Fields we want to expose to the user for display and/or update.
  *Fields that we want to keep private to the form model (never sent to client).
  *This implies the need for a "private" widget attribute. (see below for details)

Form Logic/Event Handler Programmer

- What can/cannot be dynamically referenced and modified?

- How do you manually update a selection list.
- How do you make a selection list automatically follow an existing live list datasource (e.g. query, list, hash).
- We need support for a "binding direction"-like attribute for widgets:
  *input-only (for passwords and other sensitive user data that should not be echoed back to the user)
  *input/output (normal, editable widgets)
  *output-only (for display only, readFromRequest() disabled to prevent both null-reset and modification by user)
  *private (for data not to be sent to or modified by the user, such as part of a Repeater binding identity keyset)

Things to check before we are finished:

- Do all widget types have corresponding bindings and templates? (mpo: considering this thread http://marc.theaimsgroup.com/?l=xml-cocoon-users&m=108194795730710&w=2 we should find matching bindings not for each widget, but for each 'value-setting-type' of widget.
- Can all values and attributes be referenced and manipulated from all environments where appropriate?
  *(e.g. java, js, event handlers, bindings, templates)?
- Does the transformer have any compelling use-cases left where it is better than the JXGenerator approach?
  *If desired, can the JXGenerator be extended to handle id/extends/defines and choice? (probably yes)

Now let's toss out some more ideas to refine:

Other Random Thoughts:

- BooleanField
  *Even if we use a custom class to implement BooleanField's, why do we expose this in the XML definitions? (mpo: the reason is that the request-parameters are different, I like the fact that the form-definition editor is in control of the HTTP-interface of his form, after all cforms can be used without flow but even without the forms generator or transformer. Controlling this technical interface is essential IMHO)
  *We need support for both Booleans (true/false) and "Trileans" (true/false/null).
- Struct
  *Rename "Struct" to "Composite", "Group", or whatever make the most sense to us.
- Convertor
  *Rename to "Converter"? (Check relative hit-counts on google)
- Repeater
  *Use an established terms for this concept: "Array", "Grid", "List", ...
  *Do we need more direct support for other variants, such as sparse arrays and multi-dimensional arrays?
- MultiValueField versus Repeater
  *We have single value widgets (Field, BooleanField, Output)
  *We have multi-value widgets (MultiValueField, Repeater)
  *And the progression is something like this:
  *Field->MultiValueField->Repeater
  *What benefits does MultiValueField give us over Repeater?
  *Four less lines to type each time. (ouch)
  *One less widget id to create.
  *What do we pay for this?
  *One more concept to learn/teach/maintain in the model, binding, and template. (main point)
  *Only works like a list of Field-type widgets, not BooleanField, Repeater, etc.
  *No support in the binding for identity and Repeater's set of binding strategies.
  *What is missing to drop MultiValueField in favor of using Repeater?
  *(Not suggesting to do this drop, just looking at the possibility to see if there are any benefits.)
  *Design Repeater selection-list to at least match features with the MultiValueField selection list.
  *Other questions:
  *Is Repeater's auto-numbered row wrappers too much overhead when replacing MultiValueField's?
  *Should we use a common XML syntax, while backing it by the more efficient MultiValueField-like class?
  *An alternate idea would be to make the Repeater semantics more integrated:
  *Eliminate Repeater and MultiValueField and instead allow any type of widget to be "repeated".
  *For example: (*Not* suggesting syntax, just *very rough* semantics to get ideas flowing)
  **MultiValueField: <fd:field id="some-id" multi="true"/>
  **Repeater: <fd:composite id="some-id" multi="true"/>
    - mpo: on this integrated syntax I somewhat have the same reflex as on the 'booleanfield' since multivalue and repeater lead to different request-parameter handling (x=1&x=2 versus x.1.a=i&x.2.a=j) I would welcome explicit split syntax in the definition file. But the essence of the above still holds: 'allowing repetitive values' seems to be an aspect of widgets that we could abstract out. See also discussion here: http://marc.theaimsgroup.com/?l=xml-cocoon-dev&m=108209653422163&w=2