Dynamic forms and cross-field validation

Your Rating: Results: 13 rates

This concept captures improvements to the Magnolia forms/dialogs. They address various sides and pain points with the current UI framework, in order to gain flexibility, sanitize our own customizations and expose more easily the key benefits of Vaadin (ease and speed of such developments).

Most work is expected to be carried out through the following ticket/epic. This ticket, as well as linked support tickets hold several concrete scenarios.

MGNLUI-2542 - Getting issue details... STATUS

1. Use cases

We need to support such dynamic forms in a much easier way. Collecting some commons requirements:

- **populating** select options based on the value of another field
- **validating** a field depending on the value of another field (including within a composite field itself)
- **enabling/disabling fields** conditionally
- potentially custom handling of any field, via **plain Vaadin** code
- updating form buttons (enabling/disabling/relabeling)

2. General directions & goals

- **Exposing hooks on form-level**
  - currently only possible on field-level via `FieldFactories`
  - fields & factories are only aware about themselves, in isolation. They're not aware about other fields and this is harder to do than it should.
- Use more of Vaadin out of the box
  - less customizations
  - get rid of legacy Vaadin 6.x or client-side code
- **Open up APIs to offer more possibilities (data-binding, or dynamic/ cross-field behaviors)**
- Reduce technical debt

3. Key Concepts

⚠ Key concepts highlight the main direction. Details and realization are work-in-progress, and are being revised regularly.

A new Form “component”

- Focus on The Big Role
- Create a Vaadin form that can be bound to Magnolia content
- Most often produce fields from config / definitions
- Most often bound to JCR items, and saving the session

Separation of concerns

1. Field creation
   - Responsible for producing raw fields with UI presets, unbound
   - New `FieldFactory`: Creating `Field` from `FieldDefinition` => clear and simple
2. Data-binding
   - New binding ways, using `FieldGroup` (later V8 `Binder`) instead of set/get datasource in Factory
   - Responsible for binding, default-values, transforming, converter, validation setup
   - Item variation (i18n, p13n) through re-binding?
3. Layout
   - Responsible for laying out fields in order
   - Plain Vaadin form with tabsheets, no need to customize, code is more clear and easy to maintain

- ~ Action(s)
  - Form actions vs. Dialog actions (surrounding UI context)

Break free from MVP

In our experience, the (M)VP pattern is generally regarded as an obstacle. Creating custom forms, apps can be delicate at times, and almost all the time involves hefty boilerplate.
—Even on our side, dealing with evolution of short-lived APIs in a framework code base is a challenging exercise.

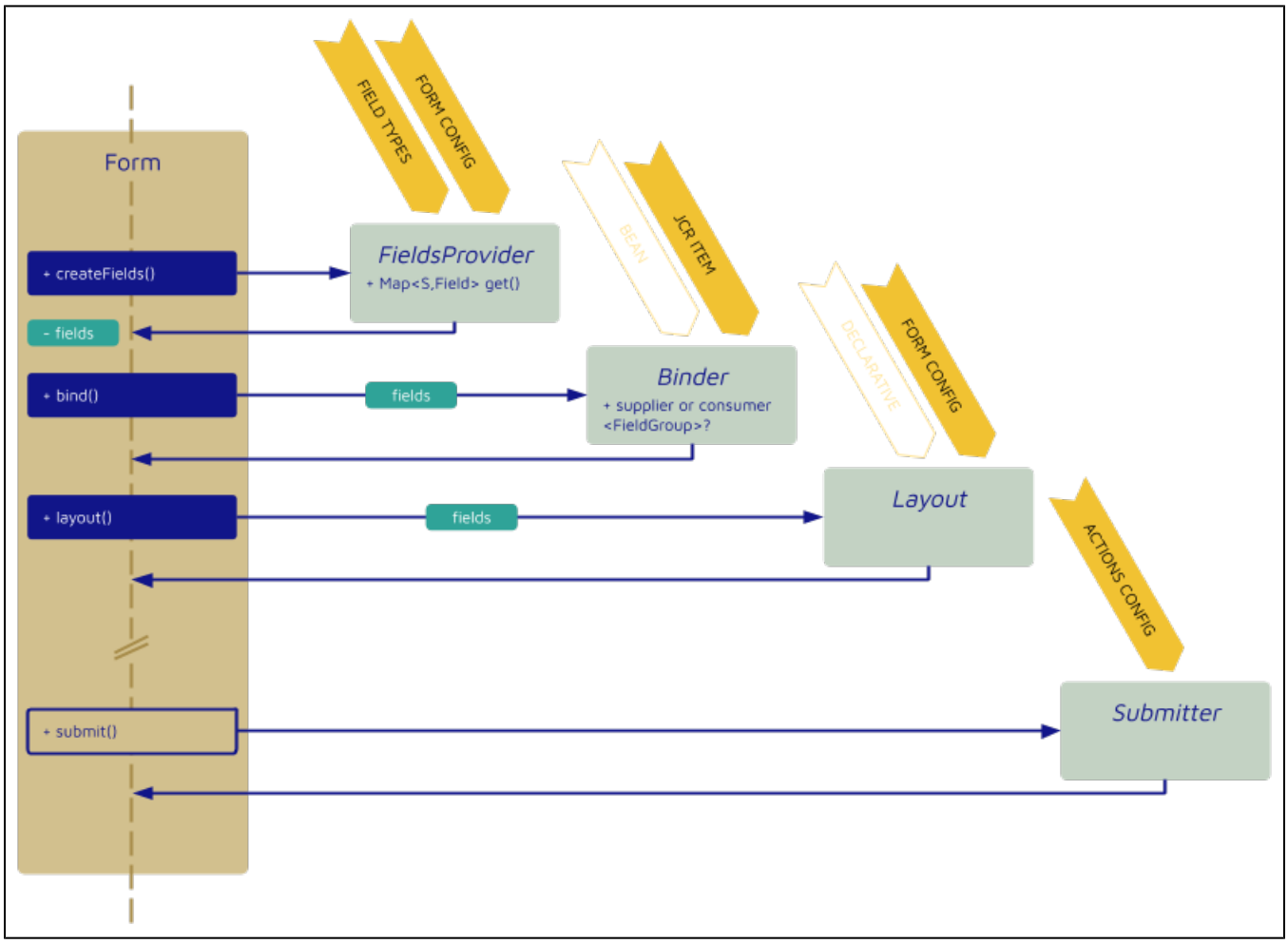
This Vaadin blog post *Is MVP a Best Practice* highlights common caveats of the pattern. This somewhat prevents Magnolia developers from leveraging Vaadin to its full extent.

- The split between view and presenter is rather artificial; in practice, coupling remains high.
- Presenter is typically not Vaadin-free: it hosts Vaadin data APIs, while View has Vaadin ui APIs

- Every single Vaadin binding or interaction required additional View +/- Listener API
- Unit-tests became slightly too atomic, with little value
- No UI-tests below full-fledged integration

Instead, we embrace a more domain-driven, component approach.

- `Form` is a magnolia `View` itself
  - at this stage, `Form` may not even need an interface
- Remove artificial-rigid-public intercom between presenters & views
- No specific FormView interface + FormViewImpl



The Form component

Flexibility

- Primarily through delegation to functional interfaces
- Ideally also through decoration (TBD)
- Alternatively through extension (config typically mandates implClass)
- Less through custom field-factories
- Less through complex fields

Configuration

- Support current common cases ootb, without config changes
- Gained flexibility is not necessarily exposed through config at first
- Rather whoever opens the form/dialog is responsible for its setup
  - *e.g.* `OpenEditDialogAction`
  - entails providing sugar / builders with good defaults
- Introduce simplified + more advanced config gradually
  - *e.g.* fields on root level / less nesting, layout config
- No intent to expose config on technical terms
  - *e.g.* no defs for bindings, delegates, generics, all sorts of highly technical classes
- No intent for declarative config of interaction logic between fields

4. PoC phase

The PoC phase was conducted in September–October 2016. Refer to child-page Dynamic forms - PoC phase for details and outcome.

5. Productization phase

Based on the PoC results, we first collected the following tasks, in order to bring new forms on par with the current Magnolia forms.

DEV-419 - Getting issue details... STATUS

- Validate a high-level architecture with clear component responsibilities
    - expose 1. field creation, 2. data-binding, 3. layouting
    - produce architecture diagrams
- Assess definition changes to support new ways of configuring forms
    - especially moving towards a separation between bindings and layout
- Envision repackaging
- ~~Draft eventual backward compatibility~~ (dedicated ticket for that)
- Iterate from Sang's PoC
- Keep in mind / draft check if form is dirty or not
- Provide validation out of the box (upon #submit)
- Provide approach for default binding & saving of JCR items (not in the centralized Form, either extended or delegated)

DEV-441 - Getting issue details... STATUS

- ~~sort out the problem with marking component visible / invisible in JavaScript extensions~~
- Safest bet => **Roll back to a GWT-based AbstractExtension** where we have more power
    - mostly to listen to the target component state changes
- use plain Vaadin validation state (state.errorMessage && state.hideErrors)

DEV-442 - Getting issue details... STATUS

- Finalize pure-Vaadin implementation of Oanh's PoC.
- From the Vaadin data-binding perspective:
    - Refine generic-typing or leave it open: move away from Items, in favor of domain types, e.g. CompositeField<Contact>
    - Challenging with Magnolia "loose types"
    - Challenging with JCR adapters in particular
- No Magnolia definition into the field instance, use Vaadin Orientation enum for layout direction.

DEV-443 - Getting issue details... STATUS

- Finalize pure-Vaadin implementation of Oanh's PoC.
- From the Vaadin data-binding perspective:
    - Refine generic-typing or leave it open: MultivalueFields would in turn be typed with Collection types (List, Set, ordered/unordered, etc.)
    - Challenging with JCR adapters in particular
    - Should support one level of nesting of arbitrary fields, including CompositeFields
- No Magnolia definition into the field instance.
- Externalize new item handlers, in line with AccessControlListField

DEV-444 - Getting issue details... STATUS

- Show all tabs as Vaadin Extension
- Port the styling—quickly estimate effort against restart from Valo
- Focus first field, via plain Vaadin SelectedTabChangeListener?

DEV-445 - Getting issue details... STATUS

- Reuse and style Vaadin Windows instead?
- How about dynamic resizing of dialog height?
- How to treat form actions (commit) vs. dialog actions (close/cancel)

DEV-446 - Getting issue details... STATUS

- again, maybe as Form component extension?
- + jump to next error

DEV-420 - Getting issue details... STATUS

- Start off a common branch in UI with results from previous PoCs
    - Proposing to start in isolation with new UI submodule, to better control UI inter-dependencies and keep new packages apart
    - Still when it's mature enough, the goal would be to relocate the package/module and relocate the legacy stuff as well.
- Provide a Vaadin test webapp within the UI, to allow testing the real thing outside a full-fledged Magnolia webapp
- Eventually consider UI-testability of it
    - Provide test-bed for UI tests early
    - Think page objects

F4. [TOFILE] Reimplement i18n support within new Form

- Generalize "variation" switching
- aka item datasource decoration somehow
- so we can support p13n variant-switching similarly

F5. [TOFILE] Refine support for backward compatibility

- a. binding strategies would cater to backward compatibility with transformers?
- b. high-level switch to use fully old dialogs / transformers / FormBuilder.... vs. new APIs

F9. [TOFILE] Reimplement help-text bubbles the same way as validation bubbles

F11. [TOFILE] Write UI tests for our 5 user scenarios

- Can only be done once the basic stones are in place
- Prove form-level agility (binding upon field(s) constructed)

F12. [TOFILE] Proof-test new APIs against Vaadin 8

- Ideally, only the data-binding "part" should be affected, not field creation nor layouting

TODO:

- [ ] File in JIRA / MGNLUI
- [x] Estimate
- [ ] Flesh out, break out into even smaller tasks
- [ ] Add relative task dependencies
- [ ] Set deadline for stabilization

Then reevaluate progress after 2–3w.

Unprocessed ideas

- binding strategies: default, transformerDelegate...
- expose layouting (default from fieldDef + tabs, declarative, or other new mechanism?, responsive forms?)
- save actions bundled in Form itself by default, can be delegated to configured action(s)
    - does validate ootb
- extensibility: generate wrapper implementations for component interfaces

Related wishful improvements

—i.e. keep out as long as we can / later scope:

- Valo-based theme switch
- Generate concrete FieldDefinitions (& builders) from our interfaces
- Plain Java Decorators (add/remove field by user-based form decorator)
    - supplanting blossom
- Documentation generation (via pegdown)