DYNAMIC UI POC

Making a Dynamic UI for Middleware Providers
(aka *Simplifying Middleware Providers*)

MOTIVATIONS

- Middleware has many providers to implement
  - Fuse
  - JDG
  - BRMS
  - And more…

FASTER

- Can we do this FASTER?
  - It took us a long time to implement this for EAP (partially)
  - Doing this for every ‘Provider for Middleware’ is time consuming
  - Can we simplify the models to make this process more efficient for Providers essentially implementing multiple providers?

FASTER - YES!

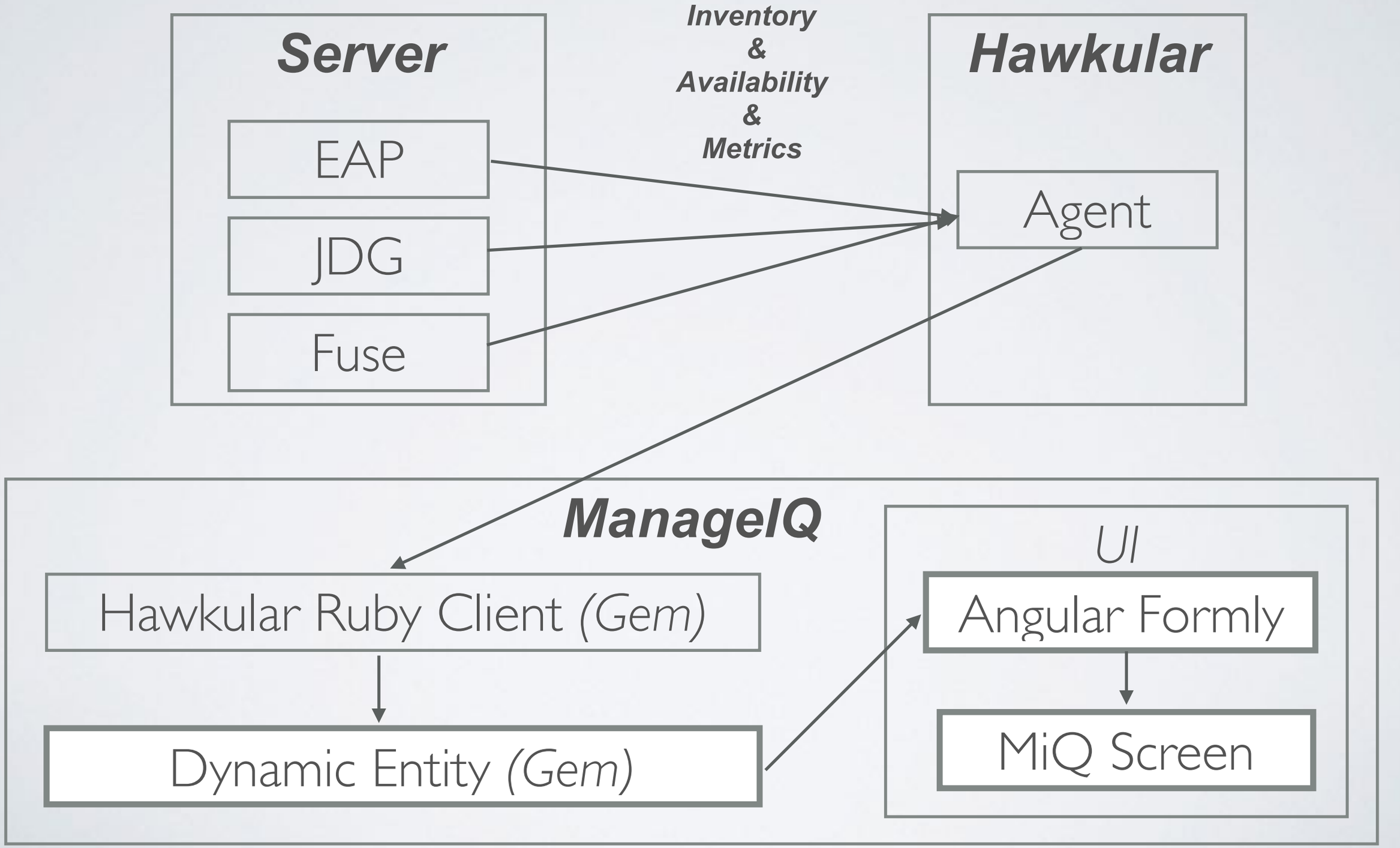
- Utilizing a more generic UI and a more generic API we can simplify this problem and create screens (aka *Components*) that are common to all Middleware 'Providers'

- Declarative screen definitions for specific situations (i.e., providers)

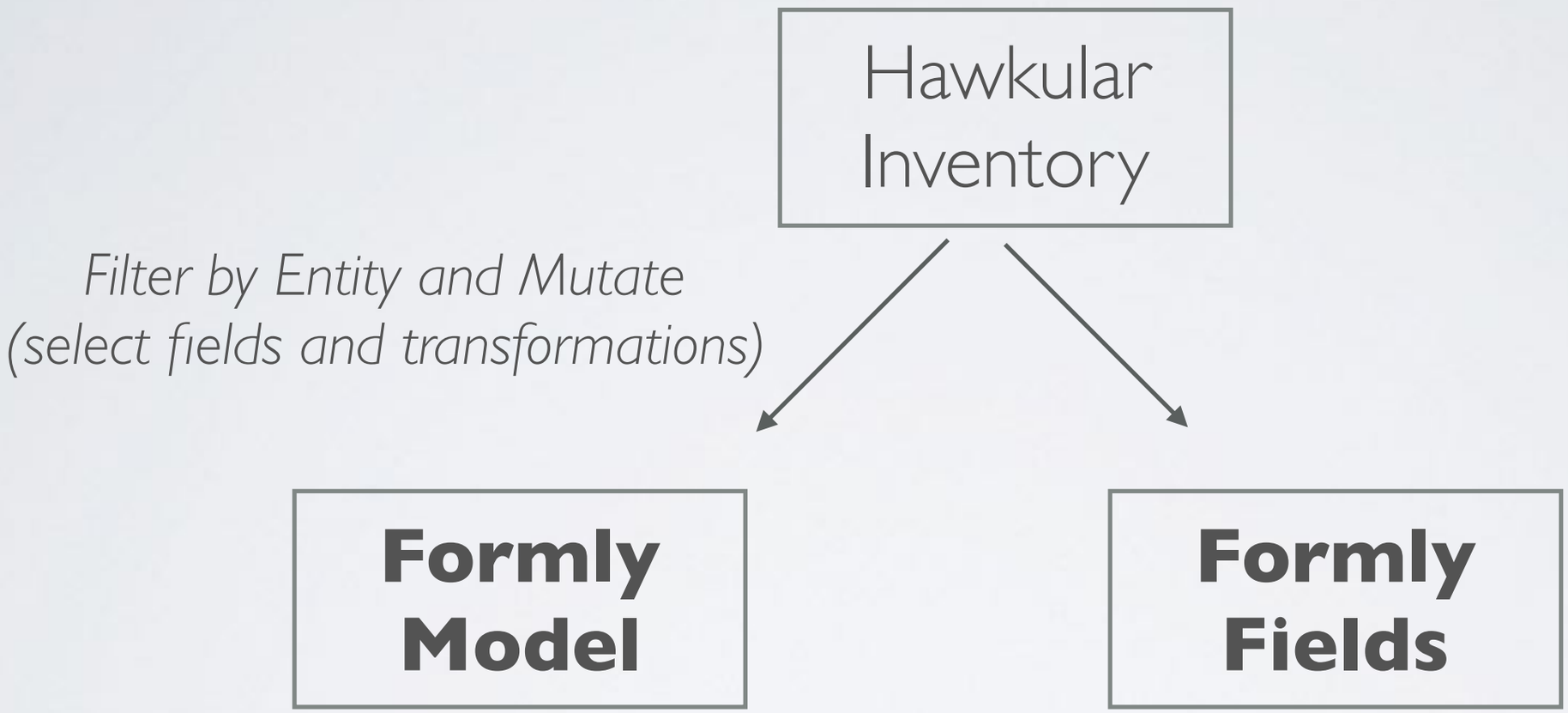
- Send me your data and how I should display it

ARCHITECTURE

NOTE: Ignoring reverse flow of operations



ANGULAR FORMLY MODEL



Angular Formly takes the model and the UI representation, and melds them into a formatted UI

WHAT ARE WE DOING HERE?

- **Now, the backend determines how the screens look!**

- Inversions of control — UI is now determined **generically** (and **dynamically**)

GENERIC BACKEND

- By utilising a generic backend API retrieving inventory from Hawkular
  - https://github.com/ManageIQ/manageiq-providers-hawkular/pull/74
  - https://github.com/cfcosta/dynamic-entity-poc/

```
def dynamic_ui
  render json:
ManageIQ::Providers::Hawkular::Connection.new.fetch_all_resources
```

GENERIC UI

- In combination with the generic backend API, a generic frontend library is needed to represent *specific* generic entities.

- Together, they produce a **Dynamic/Generic UI** — adapting to various inputs automagically — *resistant to the changes in data by only adding declarative view definitions*

ANGULAR FORMLY

- Angular Formly (http://angular-formly.com) is one such library that is able to abstract forms away from concrete implementations

- Replace your HTML with Javascript(*declarative model*)

- Backend driven forms become entirely possible through the UI given the declaratively driven representations of both the *data* and *field* representations

UI VIEW

- What does the HAML view look like:

```
- @angular_form = true
%generic-properties-component{"entity" ⇒ "WildFly Server [Local DMR]",
                              "fields" ⇒ "'id', 'feed', 'product', 'bind_address', 'version'"}


:javascript
  miq_bootstrap('generic-properties-component');
```

FORMLY IMPLEMENTATION

- By providing
  - Model Data
  - Field Representation (visual)
- We can generalize forms
  - By linking the Model to a visual field representation (via **key**)
  - Then both can be provided generically (and independently if needed)

HAWKULAR INVENTORY


```
[
  {
    id: "Local DMR~~",
    name: "WildFly Server [Local DMR]",
    feed: "f5b9c34222b7",
    properties: {
      suspend_state: "RUNNING",
      bound_address: "127.0.0.1",
      running_mode: "NORMAL",
      home_directory: "/opt/jboss/wildfly",
      version: “10.1.0.Final",
      node_name: "f5b9c34222b7",
      server_state: "running",
      product_name: "WildFly Full",
      hostname: "f5b9c34222b7",
      uuid: "76b9acc4-d023-4019-bf87-c792f992f02d",
      name: "f5b9c34222b7",
      immutable: "true",
      in_container: "true",
      os_name: "Linux",
      java_vm_name: "OpenJDK 64-Bit Server VM",
      container_id: "f5b9c34222b72fb110f17c8f70c06ab88494a2e5e8468ae78486b22f73e065b3",
      machine_id: null
    },
    type_path: "/t;hawkular/f;f5b9c34222b7/rt;WildFly Server",
    path: "/t;hawkular/f;f5b9c34222b7/r;Local%20DMR~~",
    metrics: null,
    bind_address: "127.0.0.1",
    product: "WildFly Full",
    version: "10.1.0.Final"
  }
]
```

FORMLY FORM

- We can replace the entire **static** form (all of the form html/css)representation with a one-line dynamic one

```
<formly-form model="vm.mwModel" fields=“vm.mwFields"></formly-form>
```

- Which can change *dynamically* based on the *json* sent to it *(even special cases)*

FORMLY FIELDS

- The Models are the models (linked by the **key** fields)
- Fields are visual representations of the model as specified by json:


```
[
  {
    "key": "id",
    "type": "mw-input",
    "templateOptions": {
      "label": "Id"
    },
    "extras": {},
    "data": {},
    "validation": {
      "messages": {},
      "errorExistsAndShouldBeVisible": false
    },
    "id": "formly_1_mw-input_id_0",
    "name": "formly_1_mw-input_id_0",
    "initialValue": "Local~~",
    "formControl": {
      "$viewValue": "Local~~",
      "$modelValue": "Local~~",
      "$validators": {},
      "$asyncValidators": {},
      "$parsers": [],
      "$formatters": [
        null
      ],
      "$viewChangeListeners": [],
      "$untouched": true,
      "$touched": false,
      "$pristine": true,
      "$dirty": false,
      "$valid": true,
      "$invalid": false,
      "$error": {},
      "$name": "formly_1_mw-input_id_0",
      "$options": {}
    }
  },
```

FORMLY TEMPLATES

- The templates are totally customisable
- But are attached at the Angular **run** phase:
  - ManageIQ.angular.app.run(**function**(formlyConfig) {

```
  formlyConfig.setType({
    name: 'mw-input',
    template: '<label>{{to.label}}</label><input ng-
model="model[options.key]" readonly style="margin-left:15px;" />'
  });
});
```

UI INTERNALS

- The Models json is filtered by **Entity** (for example: '*WildFly Server [Local]*')
- And the Entity Fields are specified
  - For example: ['id', 'feed', 'product', 'bindaddress'];

FORMLY REUSE

- So now, with a Formly View, we can effectively use the same view for multiple providers by just altering the **model** and **field** representation

```
<formly-form model="vm.mwModel" fields="vm.mwFields"></formly-form>
```

- Reuse becomes just altering the json sent to the form view — Aka ***Generic UI***

- Can we easily/dynamically accommodate other UI representations — Yes!

FORMLY ADVANTAGES

- Declarative (in javascript not form view)
- Dynamic
- Extendable templates (totally customisable)
- Validation
- Parsers/Formatters
- Angular 2 version

FORMLY ISSUES

- Where should the label names (could be i18n) be stored, and how do they get sent to the UI.
- Ideally, since this is declarative every piece of data should be sent to the UI (no logic in the UI). The UI just displays what it receives.

FORMLY STYLES

- Formly, has types and templates each of which can be easily customized for specialized visual representations:
    - Bootstrap
    - Angular Material
    - Ionic
    - *PatternFly (not created yet; should do high value targets first)*
- *Custom special grouping of components and styles can also be created as a 'type' or 'template'*

FORMLY EXTRAS

- Has Angular 2 version as well
- Supports validations (even between fields)
- Interactive expressions (dynamically hide/show fields)
- Parsers and formatters
- Custom templates for Bootstrap, etc…
- Really no limitations

RESOURCES

- Server-side Dynamic Entity: https://github.com/ManageIQ/manageiq-providers-hawkular/pull/74
- Angular Formly: http://angular-formly.com/

FUTURE?

- Can we go forward with this architecture?
  - Pros/Cons
- New Hawkular Inventory v4.x
- Inclusion for MiQ Re-arch via 5.0