Notes on valueMapper Requirements

Discussion with Cindy Li on 22/7/14 summarising previous week's work on improving the use of model relay/transformations in Metadata Editing component.

The Obnoxious Metadata Editing Problem:

We have a "busy input model" containing terms from an array in which all kind of stuff is inserted:
http://www.w3.org/wiki/WebSchemas/Accessibility
We want to author the set of values for "accessibilityHazard" in groups. For example, a group of two values are "flashing" and "noFlashingHazard".
Our own extension to this schema is documented here: http://wiki.fluidproject.org/display/fluid/Encoding+Accessibility+Metadata+for+Floe+-+Tables+and+Examples
These should be represented as a set of radio buttons which allows selection between 3 states - "yes", "no" and "not sure". This can be tested out at the demo at
http://metadata.floeproject.org/demos/html/resource.html?name=Climate_Change_Impacts

MODEL 1:

```
accessibilityHazard ["flashing", "some-irrelevant-value"]
```

MODEL 2:

- some choices, but in order to get off the ground with our filtering task it seems sensible to start with `fluid.arrayToSetMembership` as in the current code - this would yield:

```
accessibilityHazard {
    "flashing": true,
    "some-irrelevant-value": true
    }
```

UI MODEL:

We could render the radio buttons either from an array of booleans + repetition expander, or a single value (or array holding single value) using `fluid.selection.inputs`

The UI itself will enforce mutual exclusion between the 3 states - but it won't deal by itself with the problem of initial rendering if none of the values in the set are incoming. Therefore we explicitly need to image, for example, an empty document:

```
    accessibilityHazard: {}
```

to

- "unknown" OR
- ["unknown"] OR
- {unknown: true} -> mapped to [false, false, true] "by some means"

[These are the 3 possibilities that the renderer would accept in order to produce the initial UI state for the checkboxes)

Motivating requirement and draft configuration

We require the POSITIVE APPEARANCE of a fresh value when propagated from left to right. This seems to require the use of some "active case matching" which rules out simple passive solutions like fluid.filter or our various "array to object massaging transforms".
This seems to bring us back to the valueMapper - at least, in its "enhanced incarnation" that we were considering last week.
We would like the output space value to be "standardised" so that we can reuse the same UI module for multiple pairs of these values. So we will allocate a "lump of model" named after the set of values, and use standard values "yes"/"no"/"unknown" so that the same "lump of UI" code can handle multiple sets of such options

Draft configuration for valueMapper

```
{
    type: "fluid.transforms.valueMapper",
    defaultInputValue: true,
    outputPath: "soundHazard",
    options: [ {
        inputPath: "flashing",
        outputValue: "yes
        }, {
        inputPath: "noFlashingHazard",
        outputValue: "no"
        }, {
        inputValue: {
            "flashing": false,
            "noFlashingHazard", false
        },
        outputValue: "unknown"
    }
]
```

States for a valueMapper:

i) Exactly one preferred match -> THAT RULE is chosen
ii) No case matches
iii) Ambiguous match - the best n matches are equally good for some n > 1

The current implementation matches treated ii) and iii) the same and pushed out no value (undefined). There was a value called "defaultOutputValue" that was used only in the case there WAS an exact match, and its block supplied no value.
We made a "trial upgrade" in order to treat ii) and iii) as pushing out "defaultOutputValue". This was supplied for FLUID-5473 in pull request https://github.com/fluid-project/infusion/pull/551 . This now seems undesirable and we should avoid overloading defaultOutputValue in this way.
This seems dangerous - it seems safer to allocate a new name for the value pushed out in "no match"/"ambiguous match" cases
Perhaps we can get away with declaring there is no need for a new value for case iii) - since "no match" really is a subset of the cases of "ambiguous match" - given every rule still matches equally well, that is, not at all.
In fact we CANNOT - because of the increased ambiguity caused by computing the inverse of this transform.
It will ALWAYS be difficult to compute the inverse - because there is actually special structure here. It is not merely the fact that some rules match and others don't - there is a special DISTINGUISHED inverse value corresponding to

```
{
    flashing: false,
    noFlashingHazard: false
}
```

that we NEED TO REPRODUCE, which is not currently represented anywhere in the structure of the rules.
But actually this is a mess, and involves dumping lots of random named values in the top level configuration. Really we should reuse all the infrastructure we have for individual rules.

The draft above addresses these two issues at once - that is, a) the undesirability of providing special semantics for different matching states for the valueMapper - as well as b) providing explicit characterisation of the required inverse value