Notes on Model Transformation Semantics

These notes were taken as part of a special GPII Architecture meeting themed on the Model Transformation system held on 9/4/14.

The following JIRAs are strategic for the future evolution of the model transformation semantic:

http://issues.fluidproject.org/browse/FLUID-5133 - invertibility
http://issues.fluidproject.org/browse/FLUID-5294 - "input" and "value" consistency
http://issues.fluidproject.org/browse/FLUID-5250 - nested transforms
http://issues.fluidproject.org/browse/FLUID-5300 - extra wrapper paths for valueMapper

It seems that many people who write nested transforms have a different expectation for the meaning of material supplied as arguments to the transforms.

https://github.com/fluid-project/infusion/pull/452

At line 551 is an example of this expectation:

https://github.com/fluid-project/infusion/pull/452/files#diff-f773e3637036f9b46ce86a278dea00a4R551

In addition to this, the valueMapper transform operates its own scheme for evaluating nesting. This requires, as well as "literalValue" the use of "outputValue" in order to wrap values which are being output. This also doesn't share implementation code with the other transforms (e.g. condition, linearScale. etc).

There's a further problem with the valueMapper in that it operates a custom scheme for undefined values. It can directly output the value "undefined" by means of directives such as undefinedOutput - however we have generally adopted a standard where "undefined output means no output" and the new model relay system depends on this.

This "dead" pull from Kasper also contains tests relevant to this problem)

https://github.com/fluid-project/infusion/pull/455/files

Right now the requirement is that the user needs to wrap the material in "literalValue" to get it interpreted as pure JSON material.

Two major global issues of consistency -
i) what is the semantic for material encountered as a nested transform
ii) what is the semantic for undefined values encountered as inputs, and also as outputs

Current documentation: http://wiki.gpii.net/index.php/Architecture_-_Available_transformation_functions
Currently

- everything is interpreted by the framework
- if you want something *not* interpreted, you need to use the transformer or keyword literalValue
- special cases
  - a string as a key that is not a keyword ("transform" or "literalValue") or immediately inside a transform it is considered an output path (the location in your output document for the transformation)
  - a string as a key is considered an input path (a reference to the model you are transforming)
- In general, transformations pass their result one level up, unless "outputPath" is explicitly given

This implies that the transform system at the moment has just two parsing states -

i) the "standard" parsing state where it is what the previous paragraph describes as "interpreting" - that is, it is interpreting keys as paths and also interpreting transform blocks
ii) the "literalValue" parsing state where it is ignoring everything.

If we actually want to do something about the status quo there seems no way out other than creating at least one more parsing state and possibly two more - and I imagine we would have to seriously consider if the extra complexity was worth it.

But there seems to be some value to having a "literal" state that still interprets transforms.

DECISIONS:

- We will drop the current value/input synonym scheme and instead ONLY allow "input". **KASPER** will create a JIRA for this and perhaps even fix it
- We will continue with and clarify our existing decision that "an undefined output value means no output" - this is reasonable because undefined can't appear as a legitimate value in any document used for interchange
- As for "undefined input" - there seems to be a wide class of transforms for which "undefined input" implies that the transform is inactivated and produces no output. For example every conventional "value mapping" transform like fluid.linearScale is like this. However there seem to be

some transforms which are special - fluid.condition when receiving undefined as its condition input treats it as "false". We should vet all the transforms that we have to make sure we understand what all these cases are and can document them as global rules.

Example: https://github.com/GPII/universal/blob/master/testData/solutions/win32.json#L345-L387