Nexus design revisions

This document collects changes we would like to make to the Nexus API and utilities as we build more complex distributed systems with it.

Reverse the options-flattening revolution

The Nexus was originally designed with the expectation that Infusion would soon move away from the convention of specifying components to-be-constructed with a record a la

```
{
    type: "fluid.component",
    options: {
        myOption: "someValue"
    }
}
```

and instead write the contents of the "options" record at the top level (FLUID-5750). This change of convention has not occurred in the rest of the Infusion ecosystem, so the Nexus now operates a different convention than everyone else, creating confusion for programmers who are likely to use the Nexus alongside Kettle on the server and Infusion on the client. For the time being, the Nexus ought to either accept both conventions, or move fully to the more general one.

Whole-Nexus avatar

One envisioned role for the Nexus is to provide a live "avatar" of its component tree to clients, which they may bind to local data or user interface components. Currently, the API could only support this by clients manually opening a WebSocket to the model of every Nexus component. The Visible Nexus built for the early Nexus demos provided a version of this functionality, which could possibly be folded into the Nexus proper.

Nexus bottlers

The goal of the Nexus has been described as "connecting anything to anything", but so far this has been exemplified only through live, ephemeral data relationships where e.g. a sensor reading is turned into a tone or a meter. We have discussed multiple possible uses for a "accumulating" data relationships that persist data streams in various ways. For example, collecting sensor readings into chunks of time and writing the average value to a spreadsheet row, or collecting edits to a spreadsheet and committing an updated CSV or JSON version of the spreadsheet to a git repository. There may be an opportunity to demonstrate such a Nexus bottler component as part of the ongoing Accessibility Map of COVID-19 Assessment Centres project.

Allow binding to non-existent components

Currently opening a model binding to a path where there is no component crashes the Nexus server. This is undesirably brittle. We may simply want to reject the connection, but it would be more faithful to the model relay paradigm of Infusion to allow such model bindings to lay dormant until a component does exist at that path, and then begin working in the normal manner. Post-

~~FLUID-6504~~ - Creating a WebSocket Bind Model to a non-existent component crashes the Nexus RESOLVED we reject such connection attempts.

This issue is being tracked as FLUID-6543 - Allow model bindings to be established prior to bound components existing OPEN .

Add error messages to the model binding API (DONE)

The model binding API has so far been based on asymmetric messaging, where the client sends the equivalent of relative ChangeApplier arguments packaged as objects containing "value" and "path" members, and the server sends the updated value relative to the path the model was bound at, upon establishing the model binding and after each successful model update is processed.

If we want to deal coherently with malformed requests, we have to choose between

1. quietly consuming the malformed message without making any changes,
2. responding as before, with the (unchanged) model value,
3. trying to respond with some sort of error feedback.

3 feels nicer, and Kettle already has a scheme for sending "typed" messages. It should not complicate client implementations significantly, and would produce better feedback in interactive clients / client under development. Post-

~~FLUID-6504~~ - Creating a WebSocket Bind Model to a non-existent component crashes the Nexus RESOLVED the model binding WebSocket connection emits three types of messages, "initModel", "modelChanged", and "error".

Allow binding to a component's complete model (DONE)

Currently the API demands that at non-empty path into a component's model is specified when establishing a binding. This means it is not possible to bind to the entirety of a component's model if it has more than one top-level key, e.g.

```
model: {
    x: 23,
    y: 1
}
```

The Nexus should correctly interpret WebSockets opened at ws://{server}/bindModel/componentPath/ as addressing the entire model. This has been implemented with ~~FLUID-6516~~ - Allow Nexus model binding to empty model paths RESOLVED .

Add a GET endpoint for /components/path.to.component (DONE)

Currently, WebSocket model bindings are the only way to get information about a component out of a Nexus "from the outside". The need for this functionality has come up in testing Nexus servers, but it may become necessary in use cases as well, and is nicely symmetrical with the GET endpoint for /defaults/. This has been initially implemented with ~~FLUID-6511~~ - Revise Nexus endpoints for /components/ RESOLVED .

Theoretically, this endpoint should provide an externalization of the addressed component, i.e. serialized data that could later be used to re-establish the component to its state at the time of the request, or could be modified outside the Nexus and later re-inserted to change the system's state. However, currently, we only expect to use this to perform tests of whether a component path is currently occupied or not.

Construct components with PUT rather than POST (DONE)

Currently, the API endpoint for constructing a component is to send an HTTP POST request to the /components/{desired component path} . The traditional distinction between PUT and POST is that the former puts a resource at a requested location, while the latter determines the final location of the resource and sends it back to the requester. Component construction follows the former case, and should therefore use PUT. This has been implemented with ~~FLUID-6511~~ - Revise Nexus endpoints for /components/ RESOLVED .