Currently, we define applications as a pair of a set of resource logics and a set of transaction functions (for full details, see the RMv2 report). The resource logics define the transition rules of the application's state, and the transaction functions define some specific actions which can change that state (~ writes).

We currently lack a standardized (well-defined) way for applications to define how their state can be read

. I propose that:

- We rename the current "application interface" (set of transaction functions) to the ApplicationWriteInterface

.

- We create an ApplicationReadInterface

defined as part of an application definition (so an application is now a triple of (application logic, application write interface, application read interface).

- The ApplicationReadInterface

consists of a set of declarative projections

of the application's state, i.e. functions of type ApplicationState -> T

for some arbitrary T

.

Note that here ApplicationState

can include both resources (with resource logics defined in the ApplicationLogic

) and also other non-linear state written to blob storage (further details to be worked out here).

For example, a read interface for the kudos application could include functions which:

- compute the current balance of a specific denomination owned by a specific user
- compute the overall ownership distribution of a specific denomination (subject to available information)
- compute the current balances for all denominations owned by a specific user and use known price data to convert them to other denominations (as UoA)
- … etc

I propose that this read interface is entirely declarative

- i.e. it says nothing about how

the projection functions should be computed, but merely defines what they are. Separately, we will need to figure out how to compute different projection functions and how to efficiently route updates to interested agents. Hopefully this will map somewhat naturally to our existing publish-subscribe system.

Definitions of application composition and related operations should be able to be updated in a straightforward manner.

As a side note, this is not altogether dissimilar from a "model-view-controller" design pattern, where the resource logics are the model, the transaction functions (write interface) are the controller, and the projection functions (read interface) are the view.

cc @vveiln @degregat @jonathan @Michael @mariari for input.