Removal of State Alignment and Explicit State Recording

This proposal is to remove the explicit concepts of state, and state alignment, from the CDL specification. As defined in the CDL specification, the principle concept is the interaction. It is possible for participants to understand their 'state' in terms of a choreography, based on the observations related to the interactions that have occurred with other participants.

The benefit of removing the need to explicitly encode choreography states, and the 'state alignment' mechanism that attempts to ensure different participants share this same notion of state, is that it simplifies the choreography definition into a description of the flow of business relevant messages (interactions).

Variables should only be used to store channel instances, or information that will contribute to the clarity of the business protocol definition. They should not be used to help implement a state machine.

To illustrate the point, we can look at an example described in the CDL specification (section 2.5.2.1):

"..... for example when an order is sent from a Buyer to a Seller the outcomes could be one of the following State Changes:

Buyer.OrderState = Sent' Seller.OrderState = Received

Buyer.OrderState = SendFailure' Seller.OrderState not set

Buyer.OrderState = AckReceived' Seller.OrderState = OrderAckSent"

This example shows that an interaction could result in a number of different states. The current CDL specification would require any following CDL statements to apply a condition over these state variables to determine which course of action to take.

However, if we recast this example based purely on observing interactions, we could represent it as:

```
<choreography>

        <sequence>
                <interaction operation="placeOrder" />

                <!-- Any interactions that were based on the order
                        being sent, could be placed here (if
                        sequential) -->

                <!-- If these interactions were based on the order
                        being received, but needed to be
                        performed before the 'acknowledgeOrder'
                        was returned, then this could be
                        achieved by the interaction declaring
                        that "quality of service" it required.
                        It would be up to the binding to ensure
                        that it was appropriate to deliver the
                        required "qos" -->

                <interaction operation="acknowledgeOrder" />

                <!-- Any further interactions that depended upon
                        the 'acknowledgeOrder' being returned
                        could be placed here -->

        </sequence>

        <exception name="sendFailure" >

                <!-- This exception block could include whatever
                        interaction the buyer does next to enable the
                        seller to observe the fact that it had a failure.
                        It may be that no further interaction will be
                        performed, in which case this section would
                        remain blank? -->

        </exception>
</choreography>
```

In the above example, we use the exception block to represent the fact that the "placeOrder" interaction failed. This would enable any alternative path initiated by such a failure to be modelled as a separate flow of interactions.

ISSUES:

Should interactions simply be one-way - requiring two interactions to be defined for a request/response?

How should information contained in the messages be referenced?

Should only the recipient role in an interaction be able to use the contents of the message in a decision that would affect a subsequent interaction? Or should both roles that participated in the interaction have visibility of the same message?

Should variables be associated with roles? - especially as the 'assign' statement in the current CDL is role specific.

Related Issue: Improved Scoping of Exception Handlers

Possibly a separate proposal, but one of the areas that would need to be addressed, if this proposal was accepted, is to enable exception paths to be scoped at different levels within a choreography, instead of just at the top. Currently this means that at any

point where a choreography may need to defined an alternate path, this has to be defined as a separate choreography. This approach could lead to an unnecessarily large number of small/simple choreographies.

An alternative approach would be to enable the exception scope to be defined at any level within a choreography, to allow more comprehensive flows to be described in a single choreography.

Related Issue: React Mechanism

Currently if a workunit defines a guard, this is described as having an 'active interest' in one or more variables being set. The issue with this is that it expresses an implicit synchronization mechanism based around data, which is difficult to handle from a model verification perspective.

Any dependencies between interactions defined in CDL should be explicitly defined based on the flow of the choreography - not based on one choreography thread's interactions having a side effect on other pending choreography threads.

This may require that conditional expressions are revisited in CDL, to possibly provide a 'switch/case' based construct.