Explanation of the state tables

This will be expanded into a more formal and complete explanation later.

The state tables deal with the transitions and permitted message sending and receiving of the two sides. The individual roles within each side (especially the enroller, resigner on the service/participant side) do not have distinct state tables, but what they can do and when is determined by the state table for that side as a whole. (E.g. a participant cannot send a spontaneous VOTE until ENROLL/no-rsp has been sent or ENROLLED received, even though the Participant is not necessarily the Enroller).

There are two state tables, one for superior (Coordinator, Composer), one for inferior (Participant, Atom coordinator as inferior). States for the superior have an upper-case letter, for the inferior a lower-case letter. For convenience, both tables are divided into three – the sub-table for normal states (with numbers 4 and below), the sub-table for disrupted and recovery states (with numbers between 5 and 9) and the sub-table when queried by receipt of a *_STATUS message that requests a reply (numbers 10 and higher)

The persistent state changes (equivalent to logging in a regular transaction system) are modelled as "decide to xxx". These events model the making of a persistent record of the decision – persistent in that it will survive at least some failures that otherwise lose state information. In some cases, an implementation may not need to make an active change to have a persistent record of a decision – that an (inferior) implementation that had "decided to vote ready", and recorded a timeout in the persistent information for that decision, if it always updated that decision record on applying the opposite (non-timeout) result, could treat the presence of an expired record as a record of "decide to cancel" (or confirm as the case may be).

The event "decide to prepare" is considered semi-persistent. A superior is not allowed to decide to confirm if there is an outstanding prepare. This means that it has to remember if it has sent one, even if there is a failure that might cause the PREPARE to be lost, as long as it is still attempting to progress the atom to confirmation. However, if the atom is to be cancelled, there is no necessity to remember the sending of PREPARE over all failures.

In the case of a hierarchic tree, the "decide to confirm" and "decide to cancel" decisions of a superior (sub-coordinator) may in fact be the receipt of a confirm or cancel instruction from its own superior, without change of local persistent information (which would be a sub-coordinator record, pointing both up and down the tree).

Failure events are modelled as "disruption". A disruption implies the loss of some some state (and possibly of some messages in flight). From some states, there are several possible states that may arise after an implentation has recovered, depending on how much state information was preserved. The overall requirements are that the implementations MUST preserve the same information as in normal two-phase commit (vote ready decision and order commit decision), but MAY preserve any amount of information, as is appropriate. This is modelled by the multiple disruption levels – these identify states which a failure may move the endpoint to in general. For

a particular instance, the disruption level appropriate to a failure depends on what state information survives.

Disruption I is the most severe, corresponding to loss of all optional state information. The highest level for any state would typically be an appropriate abstraction for a communication failure.

Several of the states and many of the allowed transitions only occur as a result of failures and recovery attempts. They assume the following properties hold:

a) Messages in opposite directions can cross each other to any degree
b) messages are not lost unless there is a disruption of some level on at least one side
c) if a message is lost, all messages sent after it by the side that receives the disruption are also lost, and all undelivered messages sent before it to that side are also lost.
d) Messages that are delivered are delivered in the order they were sent – they cannot overtake each other.
e) Any messages arriving after a SUPERIOR_STATUS/reply-requested or INFERIOR_STATUS/reply_requested is not processed until a message has been issued by the side receiving the query.(i.e. there are no receive events in states numbered 10 and higher). The inbound message will be processed after the query has been responded to.

Given these constraints, receipt of a message corresponding to an empty cell indicates a protocol error (i.e. the violation of the peer table or violation of the assumptions above).

Mis-ordering of messages (d) could possibly be dealt with by just ignoring a messages that arrive where the current state says they cannot. However, the misordering will probably invalidate c) as well.