MODULE *Proposal*

INSTANCE *Naturals*

INSTANCE *FiniteSets*

LOCAL INSTANCE *TLC*

Transaction type constants

CONSTANTS
Change,
Rollback

Phase constants

CONSTANTS
Initialize,
Validate,
Abort,
Commit,
Apply

Status constants

CONSTANTS
InProgress,
Complete,
Failed

CONSTANTS
Valid,
Invalid

CONSTANTS
Success,
Failure

The set of all nodes

CONSTANT *Node*

Target is the set of all targets and their possible paths and values.

Example:

```
Target ≜
  [target1 ↦
    [persistent ↦ FALSE, values ↦ [
      path1 ↦ {"value1", "value2"},
      path2 ↦ {"value2", "value3"}]],
  target2 ↦
    [persistent ↦ TRUE, values ↦ [
```

$path2 \mapsto \{$"$value3$", "$value4$"$\}$,
$path3 \mapsto \{$"$value4$", "$value5$"$\}]]]$

CONSTANT *Target*

A record of per-target proposals
VARIABLE *proposal*

A record of per-target configurations
VARIABLE *configuration*

A record of target states
VARIABLE *target*

A record of target *masterships*
VARIABLE *mastership*

LOCAL $InitState \triangleq$
$[proposals \mapsto proposal,$
$configurations \mapsto configuration,$
$targets \mapsto target,$
$masterships \mapsto mastership]$

LOCAL $NextState \triangleq$
$[proposals \mapsto proposal',$
$configurations \mapsto configuration',$
$targets \mapsto target',$
$masterships \mapsto mastership']$

LOCAL $Trace \triangleq$ INSTANCE *Trace* WITH
Module $\leftarrow$ "Proposal",
InitState $\leftarrow InitState$,
NextState $\leftarrow NextState$

Reconcile a proposal
$Reconcile(n, t, i) \triangleq$
$\land \lor \land proposal[t][i].phase = Initialize$
$\land proposal[t][i].state = InProgress$
$\land proposal' = [proposal$ EXCEPT $![t] =$
$[proposal[t]$ EXCEPT $![i].state = Complete,$
$![i].dependency.index = configuration[t].proposal.index]]$
$\land configuration' = [configuration$ EXCEPT $![t].proposal.index = i]$
$\land$ UNCHANGED $\langle target \rangle$
While in the *Validate* phase, validate the proposed changes.
If validation is successful, the proposal also records the changes

```
required to roll back the proposal and the index to which to roll back.
∨ ∧ proposal[t][i].phase = Validate
  ∧ proposal[t][i].state  = InProgress
  ∧ configuration[t].commit.index = proposal[t][i].dependency.index
    For Change proposals validate the set of requested changes.
  ∧ ∨ ∧ proposal[t][i].type = Change
      ∧ LET rollbackIndex  ≜ configuration[t].config.index
            rollbackValues ≜ [p ∈ DOMAIN proposal[t][i].change.values ↦
                               IF p ∈ DOMAIN configuration[t].config.values THEN
                                 configuration[t].config.values[p]
                               ELSE
                                 [delete ↦ TRUE]]
        Model validation successes and failures with Valid and Invalid results.
        IN  ∃ r ∈ {Valid, Invalid} :
              If the Change is Valid, record the changes required to roll
              back the proposal and the index to which the rollback changes
              will roll back the configuration.
            ∨ ∧ r = Valid
              ∧ proposal' = [proposal EXCEPT ![t] =
                              [proposal[t] EXCEPT ![i].rollback = [index  ↦ rollbackIndex,
                                                                   values ↦ rollbackValues],
                                                  ![i].state    = Complete]]
            ∨ ∧ r = Invalid
              ∧ proposal' = [proposal EXCEPT ![t] =
                              [proposal[t] EXCEPT ![i].state = Failed]]
    For Rollback proposals, validate the rollback changes which are
    proposal being rolled back.
  ∨ ∧ proposal[t][i].type = Rollback
      Rollbacks can only be performed on Change type proposals.
    ∧ ∨ ∧ proposal[t][proposal[t][i].rollback.index].type = Change
        Only roll back the change if it's the lastest change made
        to the configuration based on the configuration index.
      ∧ ∨ ∧ configuration[t].config.index = proposal[t][i].rollback.index
          ∧ LET changeIndex    ≜ proposal[t][proposal[t][i].rollback.index].rollback.index
                changeValues   ≜ proposal[t][proposal[t][i].rollback.index].rollback.values
                rollbackValues ≜ proposal[t][proposal[t][i].rollback.index].change.values
            IN  ∃ r ∈ {Valid, Invalid} :
                  If the Rollback is Valid, record the changes required to
                  roll back the target proposal and the index to which the
                  configuration is being rolled back.
                ∨ ∧ r = Valid
                  ∧ proposal' = [proposal EXCEPT ![t] =
                                  [proposal[t] EXCEPT ![i].change = [index  ↦ changeIndex,
                                                                     values ↦ changeValues],
                                                      ![i].change = [index  ↦ proposal[t][i].change.ind
```

$values \mapsto changeValues]$,
$![i].state = Complete]]$

$\lor \land r = Invalid$
$\land proposal' = [proposal \text{ EXCEPT } ![t] =$
$[proposal[t] \text{ EXCEPT } ![i].state = Failed]]$

If the *Rollback* target is not the most recent change to the configuration,
fail validation for the proposal.

$\lor \land configuration[t].config.index \neq proposal[t][i].rollback.index$
$\land proposal' = [proposal \text{ EXCEPT } ![t] = [proposal[t] \text{ EXCEPT } ![i].state = Failed]]$

If a *Rollback* proposal is attempting to roll back another *Rollback*,
fail validation for the proposal.

$\lor \land proposal[t][proposal[t][i].rollback.index].type = Rollback$
$\land proposal' = [proposal \text{ EXCEPT } ![t] =$
$[proposal[t] \text{ EXCEPT } ![i].state = Failed]]$

$\land \text{UNCHANGED } \langle configuration, target \rangle$

While in the *Commit* state, commit the proposed changes to the configuration.

$\lor \land proposal[t][i].phase = Commit$
$\land proposal[t][i].state = InProgress$

Only commit the proposal if the prior proposal has already been committed.

$\land configuration[t].commit.index = proposal[t][i].dependency.index$
$\land configuration' = [configuration \text{ EXCEPT } ![t].config.values = proposal[t][i].change.values,$
$![t].config.index = proposal[t][i].change.index,$
$![t].commit.index = i]$
$\land proposal' = [proposal \text{ EXCEPT } ![t] = [proposal[t] \text{ EXCEPT } ![i].state = Complete]]$
$\land \text{UNCHANGED } \langle target \rangle$

While in the *Apply* phase, apply the proposed changes to the target.

$\lor \land proposal[t][i].phase = Apply$
$\land proposal[t][i].state = InProgress$
$\land configuration[t].target.index = proposal[t][i].dependency.index$
$\land configuration[t].target.term = mastership[t].term$
$\land mastership[t].master = n$

Model successful and failed target update requests.

$\land \exists\, r \in \{Success, Failure\} :$
$\lor \land r = Success$
$\land target' = [target \text{ EXCEPT } ![t] = proposal[t][i].change.values \mathbin{@@} target[t]]$
$\land configuration' = [configuration \text{ EXCEPT }$
$![t].target.index = i,$
$![t].target.values = proposal[t][i].change.values$
$\mathbin{@@} configuration[t].target.values]$
$\land proposal' = [proposal \text{ EXCEPT } ![t] = [proposal[t] \text{ EXCEPT } ![i].state = Complete]]$

If the proposal could not be applied, update the configuration's applied index
and mark the proposal *Failed*.

$\lor \land r = Failure$
$\land configuration' = [configuration \text{ EXCEPT } ![t].target.index = i]$
$\land proposal' = [proposal \text{ EXCEPT } ![t] = [proposal[t] \text{ EXCEPT } ![i].state = Failed]]$

$\land$ UNCHANGED $\langle target \rangle$

$\lor \land proposal[t][i].phase = Abort$

$\land proposal[t][i].state = InProgress$

The *commit.index* will always be greater than or equal to the *target.index*. If only the *commit.index* matches the proposal's *dependency.index*, update the *commit.index* to enable commits of later proposals, but do not mark the *Abort* phase *Complete* until the *target.index* has been incremented.

$\land \lor \land configuration[t].commit.index = proposal[t][i].dependency.index$

$\land configuration' = [configuration \text{ EXCEPT } ![t].commit.index = i]$

$\land$ UNCHANGED $\langle proposal \rangle$

If the configuration's *target.index* matches the proposal's *dependency.index*, update the *target.index* and mark the proposal *Complete* for the *Abort* phase.

$\lor \land configuration[t].commit.index \geq i$

$\land configuration[t].target.index = proposal[t][i].dependency.index$

$\land configuration' = [configuration \text{ EXCEPT } ![t].target.index = i]$

$\land proposal' = [proposal \text{ EXCEPT } ![t] = [proposal[t] \text{ EXCEPT } ![i].state = Complete]]$

If both the configuration's *commit.index* and *target.index* match the proposal's *dependency.index*, update the *commit.index* and *target.index* and mark the proposal *Complete* for the *Abort* phase.

$\lor \land configuration[t].commit.index = proposal[t][i].dependency.index$

$\land configuration[t].target.index = proposal[t][i].dependency.index$

$\land configuration' = [configuration \text{ EXCEPT } ![t].commit.index = i,$
$\qquad ![t].target.index = i]$

$\land proposal' = [proposal \text{ EXCEPT } ![t] = [proposal[t] \text{ EXCEPT } ![i].state = Complete]]$

$\land$ UNCHANGED $\langle target \rangle$

$\land$ UNCHANGED $\langle mastership \rangle$

Formal specification, constraints, and theorems.

$Init \triangleq$

$\land proposal = [t \in \text{DOMAIN } Target \mapsto$
$\qquad [i \in \{\} \mapsto$
$\qquad\quad [phase \mapsto Initialize,$
$\qquad\quad\ state \mapsto InProgress]]]$

$\land Trace!Init$

$Next \triangleq$

$\lor \exists\, n \in Node :$

$\quad \exists\, t \in \text{DOMAIN } proposal :$

$\quad\quad \exists\, i \in \text{DOMAIN } proposal[t] :$

$\quad\quad\quad Trace!Step(\text{“Reconcile”}, Reconcile(n, t, i), [node \mapsto n, target \mapsto t, index \mapsto i])$

---

* Modification History

* Last modified Sun *Feb* 20 08:17:38 *PST* 2022 by *jordanhalterman*

\ * Created Sun *Feb* 20 02:20:56 *PST* 2022 by *jordanhalterman*