MODULE *Proposals*

EXTENDS *Configurations*, *Southbound*

INSTANCE *Naturals*

INSTANCE *FiniteSets*

LOCAL INSTANCE *TLC*

Transaction type constants

CONSTANTS
ProposalChange,
ProposalRollback

Phase constants

CONSTANTS
ProposalInitialize,
ProposalValidate,
ProposalAbort,
ProposalCommit,
ProposalApply

Status constants

CONSTANTS
ProposalInProgress,
ProposalComplete,
ProposalFailed

CONSTANTS
ProposalValid,
ProposalInvalid

CONSTANTS
ProposalSuccess,
ProposalFailure

A record of per-target proposals

VARIABLE *proposal*

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

```
LOCAL NextState ≜
  [proposals      ↦ proposal′,
   configurations ↦ configuration′,
   targets        ↦ target′,
   masterships    ↦ mastership′]

LOCAL Trace ≜ INSTANCE Trace WITH
  Module    ← "Proposals",
  InitState ← InitState,
  NextState ← NextState

──────────────────────────────────────────────────────────────────────

 Reconcile a proposal
ReconcileProposal(n, t, i) ≜
  ∧ ∨ ∧ proposal[t][i].phase = ProposalInitialize
      ∧ proposal[t][i].state  = ProposalInProgress
      ∧ proposal′ = [proposal EXCEPT ![t] =
            [proposal[t] EXCEPT ![i].state  = ProposalComplete,
                                ![i].dependency.index = configuration[t].proposal.index]]
      ∧ configuration′ = [configuration EXCEPT ![t].proposal.index = i]
      ∧ UNCHANGED ⟨target⟩
    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 = ProposalValidate
      ∧ proposal[t][i].state  = ProposalInProgress
      ∧ configuration[t].commit.index = proposal[t][i].dependency.index
        For Change proposals validate the set of requested changes.
      ∧ ∨ ∧ proposal[t][i].type = ProposalChange
          ∧ 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 ∈ {ProposalValid, ProposalInvalid} :
                 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 = ProposalValid
                   ∧ proposal′ = [proposal EXCEPT ![t] =
                         [proposal[t] EXCEPT ![i].rollback = [index  ↦ rollbackIndex,
                                                              values ↦ rollbackValues],
                                             ![i].state    = ProposalComplete]]
```

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

For *Rollback* proposals, validate the rollback changes which are proposal being rolled back.

$\lor \land proposal[t][i].type = ProposalRollback$

Rollbacks can only be performed on *Change* type proposals.

$\land \lor \land proposal[t][proposal[t][i].rollback.index].type = ProposalChange$

Only roll back the change if it's the lastest change made to the configuration based on the configuration index.

$\land \lor \land configuration[t].config.index = proposal[t][i].rollback.index$
$\land \text{LET } changeIndex \triangleq proposal[t][proposal[t][i].rollback.index].rollback.index$
$changeValues \triangleq proposal[t][proposal[t][i].rollback.index].rollback.values$
$rollbackValues \triangleq proposal[t][proposal[t][i].rollback.index].change.values$
$\text{IN } \exists r \in \{ProposalValid, ProposalInvalid\} :$

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.

$\lor \land r = ProposalValid$
$\land proposal' = [proposal \text{ EXCEPT } ![t] =$
$[proposal[t] \text{ EXCEPT } ![i].change = [index \mapsto changeIndex,$
$values \mapsto changeValues],$
$![i].change = [index \mapsto proposal[t][i].change.ind$
$values \mapsto changeValues],$
$![i].state = ProposalComplete]]$
$\lor \land r = ProposalInvalid$
$\land proposal' = [proposal \text{ EXCEPT } ![t] =$
$[proposal[t] \text{ EXCEPT } ![i].state = ProposalFailed]]$

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

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

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

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

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

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

$\lor \land proposal[t][i].phase = ProposalCommit$
$\land proposal[t][i].state = ProposalInProgress$

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

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

$!\,[t].commit.index \;= i]$

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

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

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

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

$\land\ proposal[t][i].state \;= ProposalInProgress$

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

$\land\ configuration[t].target.term \;= mastership[t].term$

$\land\ mastership[t].master = n$

Model successful and failed target update requests.

$\land\ \exists\, r \in \{ProposalSuccess,\ ProposalFailure\} :$

$\lor\ \land\ r = ProposalSuccess$

$\land\ target' = [target \text{ EXCEPT } ![t] = proposal[t][i].change.values \mathbin{@@} target[t]]$

$\land\ configuration' = [configuration \text{ EXCEPT }$
$\quad ![t].target.index \;= i,$
$\quad ![t].target.values = proposal[t][i].change.values$
$\quad\quad \mathbin{@@} configuration[t].target.values]$

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

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

$\lor\ \land\ r = ProposalFailure$

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

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

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

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

$\land\ proposal[t][i].state \;= ProposalInProgress$

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.

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

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

$\land\ \text{UNCHANGED } \langle proposal \rangle$

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.

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

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

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

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

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.

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

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

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

$!\,[t].target.index \quad = i]$

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

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

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

Formal specification, constraints, and theorems.

$InitProposal \triangleq$

$\land\ proposal = [t \in \text{DOMAIN } Target \mapsto$
$[i \in \{\} \mapsto$
$[phase \mapsto ProposalInitialize,$
$state \mapsto ProposalInProgress]]]$

$\land\ Trace!\,Init$

$NextProposal \triangleq$

$\lor\ \exists\, n \in Node :$
$\exists\, t \in \text{DOMAIN } proposal :$
$\exists\, i \in \text{DOMAIN } proposal[t] :$
$Trace!\,Step(\text{“Reconcile”}, ReconcileProposal(n,\ t,\ i), [node \mapsto n,\ target \mapsto t,\ index \mapsto i])$

---

* Modification History
* Last modified Sun *Feb* 20 10:07:54 *PST* 2022 by *jordanhalterman*
* Created Sun *Feb* 20 10:07:16 *PST* 2022 by *jordanhalterman*