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
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 $NextState \triangleq$
$[proposals \mapsto proposal',$
$configurations \mapsto configuration',$

$targets \mapsto target'$,
$masterships \mapsto mastership']$

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

Reconcile a proposal

$ReconcileProposal(n,\ t,\ i) \triangleq$

$\land \lor \land proposal[t][i].phase = ProposalInitialize$
$\land proposal[t][i].state = ProposalInProgress$
$\land proposal' = [proposal$ EXCEPT $![t] =$
$[proposal[t]$ EXCEPT $![i].state = ProposalComplete]]$
$\land configuration' = [configuration$ EXCEPT $![t].proposed.index = i]$
$\land$ 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.

$\lor \land proposal[t][i].phase = ProposalValidate$
$\land proposal[t][i].state = ProposalInProgress$
$\land configuration[t].index = i - 1$

For Change proposals validate the set of requested changes.

$\land \lor \land proposal[t][i].type = ProposalChange$
$\land$ LET $rollbackIndex \triangleq configuration[t].committed.index$
$rollbackValues \triangleq [p \in$ DOMAIN $proposal[t][i].change.values \mapsto$
IF $p \in$ DOMAIN $configuration[t].committed.values$ THEN
$configuration[t].committed.values[p]$
ELSE
$[delete \mapsto$ TRUE$]]$

If all the change values are valid,record the changes required to roll
back the proposal and the index to which the rollback changes
will roll back the configuration.

IN
$\lor \land \forall\, v \in proposal[t][i].change.values : v.valid$
$\land proposal' = [proposal$ EXCEPT $![t] =$
$[proposal[t]$ EXCEPT $![i].rollback = [index \mapsto rollbackIndex,$
$values \mapsto rollbackValues]$,
$![i].state = ProposalComplete]]$
$\lor \land \exists\, v \in proposal[t][i].change.values : \neg v.valid$
$\land proposal' = [proposal$ EXCEPT $![t] =$
$[proposal[t]$ 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.

$\land \lor \land configuration[t].committed.index = proposal[t][i].rollback.index$

$\land$ LET $changeIndex \triangleq proposal[t][proposal[t][i].rollback.index].rollback.index$

$changeValues \triangleq proposal[t][proposal[t][i].rollback.index].rollback.values$

$rollbackValues \triangleq proposal[t][proposal[t][i].rollback.index].change.values$

Record the changes required to roll back the target proposal and the index to

which the configuration is being rolled back.

IN $\land proposal' = [proposal$ EXCEPT $![t] =$

$[proposal[t]$ EXCEPT $![i].change = [index \mapsto changeIndex,$

$values \mapsto changeValues],$

$![i].change = [index \mapsto proposal[t][i].change.index,$

$values \mapsto changeValues],$

$![i].state = ProposalComplete]]$

If the Rollback target is not the most recent change to the configuration,

fail validation for the proposal.

$\lor \land configuration[t].committed.index \neq proposal[t][i].rollback.index$

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

If a Rollback proposal is attempting to roll back another Rollback,

fail validation for the proposal.

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

$\land proposal' = [proposal$ EXCEPT $![t] =$

$[proposal[t]$ EXCEPT $![i].state = ProposalFailed]]$

$\land$ 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.

$\land configuration[t].index = i - 1$

$\land configuration' = [configuration$ EXCEPT $![t].committed.values = proposal[t][i].change.values,$

$![t].committed.index = proposal[t][i].change.index,$

$![t].index = i]$

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

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

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

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

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

$\land configuration[t].applied.index = i - 1$

$\land configuration[t].applied.term = mastership[t].term$

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

Model successful and failed target update requests.

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

$\lor \land r = ProposalSuccess$
$\land target' = [target \text{ EXCEPT } ![t] = proposal[t][i].change.values \mathbin{@@} target[t]]$
$\land configuration' = [configuration \text{ EXCEPT } ![t].applied.index = i,$
$![t].applied.values = proposal[t][i].change.values \mathbin{@@} configuration[t].applied.values]$
$\land proposal' = [proposal \text{ EXCEPT } ![t] = [proposal[t] \text{ EXCEPT } ![i].state = ProposalComplete]]$

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

$\lor \land r = ProposalFailure$
$\land configuration' = [configuration \text{ EXCEPT } ![t].applied.index = i]$
$\land proposal' = [proposal \text{ EXCEPT } ![t] = [proposal[t] \text{ EXCEPT } ![i].state = ProposalFailed]]$
$\land \text{UNCHANGED } \langle target \rangle$

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

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

$\land \lor \land configuration[t].index = i - 1$
$\land configuration' = [configuration \text{ EXCEPT } ![t].index = i]$
$\land \text{UNCHANGED } \langle proposal \rangle$

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

$\lor \land configuration[t].index \geq i$
$\land configuration[t].applied.index = i - 1$
$\land configuration' = [configuration \text{ EXCEPT } ![t].applied.index = i]$
$\land proposal' = [proposal \text{ EXCEPT } ![t] = [proposal[t] \text{ EXCEPT } ![i].state = ProposalComplete]]$

If both the configuration's index and *applied.index* match the previous proposal index, update the index and *applied.index* and mark the proposal Complete for the Abort phase.

$\lor \land configuration[t].index = i - 1$
$\land configuration[t].applied.index = i - 1$
$\land configuration' = [configuration \text{ EXCEPT } ![t].index = i,$
$![t].applied.index = i]$
$\land proposal' = [proposal \text{ EXCEPT } ![t] = [proposal[t] \text{ EXCEPT } ![i].state = ProposalComplete]]$

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

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

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 *Fri Apr* 21 19:15:11 *PDT* 2023 by *jhalterm*
* Last modified *Mon Feb* 21 01:24:12 *PST* 2022 by *jordanhalterman*
* Created Sun *Feb* 20 10:07:16 *PST* 2022 by *jordanhalterman*