MODULE *Proposal*

EXTENDS *Configuration*, *Mastership*

INSTANCE *Naturals*

INSTANCE *FiniteSets*

LOCAL INSTANCE *TLC*

Transaction type constants
CONSTANTS
ProposalChange,
ProposalRollback

Phase constants
CONSTANTS
ProposalCommit,
ProposalApply

Status constants
CONSTANTS
ProposalInProgress,
ProposalComplete,
ProposalFailed

CONSTANT *TraceProposal*

A record of per-target proposals
VARIABLE *proposal*

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

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

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

Reconcile a proposal

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

Only the master can process proposals for the target.

$\land\ mastership.master = n$

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

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

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

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

$\land\ configuration.committed.index = i - 1$

For Change proposals validate the set of requested changes.

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

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.

$\land\ \lor$ LET $rollbackIndex \triangleq configuration.committed.revision$

$rollbackValues \triangleq [p \in \text{DOMAIN } proposal[i].change.values \mapsto$
IF $p \in \text{DOMAIN } configuration.committed.values$ THEN
$configuration.committed.values[p]$
ELSE
$[index \mapsto 0,\ value \mapsto Nil]]$

$changeValues \triangleq [p \in \text{DOMAIN } proposal[i].change.values \mapsto$
$proposal[i].change.values[p]\ @@\ [index \mapsto i]]$

IN $\land\ configuration' = [configuration$ EXCEPT $!.committed.index = i,$
$!.committed.revision = i,$
$!.committed.values = changeValues]$

$\land\ proposal' = [proposal$ EXCEPT $![i].change = [$
$index \mapsto i,$
$values \mapsto changeValues],$
$![i].rollback = [$
$index \mapsto rollbackIndex,$
$values \mapsto rollbackValues],$
$![i].state = ProposalComplete]$

A proposal can fail validation at this point, in which case the proposal is marked failed.

$\lor\ \land\ proposal' = [proposal$ EXCEPT $![i].state = ProposalFailed]$
$\land$ UNCHANGED $\langle configuration \rangle$

For Rollback proposals, validate the rollback changes which are

proposal being rolled back.

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

Rollbacks can only be performed on Change type proposals.

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

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

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

Record the changes required to roll back the target proposal and the index to
which the configuration is being rolled back.

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

Note: these two changes must be implemented as an atomic, idempotent update.
Implementations should check if the configuration has already been updated and
skip the configuration update if the committed index is $\geq$ the proposal index.

IN $\land configuration' = [configuration$ EXCEPT $!.committed.index = i,$
$!.committed.revision = changeI$
$!.committed.values = changeV$
$\land proposal' = [proposal$ EXCEPT $![i].change = [$
$index \mapsto changeIndex,$
$values \mapsto changeValues],$
$![i].state = ProposalComplete]$

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

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

Note: these two changes must be implemented as an atomic, idempotent update.
Implementations should check if the configuration has already been updated and
skip the configuration update if the committed index is $\geq$ the proposal index.

$\land configuration' = [configuration$ EXCEPT $!.committed.index = i]$
$\land proposal' = [proposal$ EXCEPT $![i].state = ProposalFailed]$

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

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

Note: these two changes must be implemented as an atomic, idempotent update.
Implementations should check if the configuration has already been updated and
skip the configuration update if the committed index is $\geq$ the proposal index.

$\land configuration' = [configuration$ EXCEPT $!.committed.index = i]$
$\land proposal' = [proposal$ EXCEPT $![i].state = ProposalFailed]$

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

Once the proposal is committed, update the configuration's commit index
and move to the apply phase.

$\lor \land proposal[i].state = ProposalComplete$
$\land proposal' = [proposal$ EXCEPT $![i].phase = ProposalApply,$
$![i].state = ProposalInProgress]$
$\land$ UNCHANGED $\langle configuration, target \rangle$

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

$\lor \land proposal[i].phase = ProposalApply$
$\land configuration.applied.index = i - 1$
$\land proposal[i].state = ProposalInProgress$
Process the proposal once the prior proposal has been applied.
$\land i - 1 \in \text{DOMAIN } proposal \Rightarrow$
$\lor \land proposal[i-1].phase = ProposalCommit$
$\land proposal[i-1].state = ProposalFailed$
$\lor \land proposal[i-1].phase = ProposalApply$
$\land proposal[i-1].state \in \{ProposalComplete, ProposalFailed\}$
Verify the applied term is the current *mastership* term to ensure the configuration has been synchronized following restarts.
$\land configuration.applied.term = mastership.term$
Verify the node's connection to the target.
$\land node[n].connected$
$\land target.running$
Model successful and failed target update requests.
$\land \lor \land target' = [target \text{ EXCEPT } !.values = proposal[i].change.values]$
Note: these two changes must be implemented as an atomic, idempotent update. Implementations should check if the configuration has already been updated and skip the configuration update if the applied index is $\geq$ the proposal index.
$\land \text{LET } index \triangleq proposal[i].change.index$
$values \triangleq proposal[i].change.values \mathbin{@@} configuration.applied.values$
$\text{IN } configuration' = [configuration \text{ EXCEPT } !.applied.index = i,$
$!.applied.revision = index,$
$!.applied.values = values]$
$\land proposal' = [proposal \text{ EXCEPT } ![i].state = ProposalComplete]$
If the proposal could not be applied, update the configuration's applied index and mark the proposal Failed.
Note: these two changes must be implemented as an atomic, idempotent update. Implementations should check if the configuration has already been updated and skip the configuration update if the applied index is $\geq$ the proposal index.
$\lor \land configuration' = [configuration \text{ EXCEPT } !.applied.index = i]$
$\land proposal' = [proposal \text{ EXCEPT } ![i].state = ProposalFailed]$
$\land \text{UNCHANGED } \langle target \rangle$
$\land \text{UNCHANGED } \langle mastership, node \rangle$

Formal specification, constraints, and theorems.

$InitProposal \triangleq$
$\land proposal = [$
$i \in \{\} \mapsto [$
$type \mapsto ProposalChange,$
$change \mapsto [$
$index \mapsto 0,$

```
            values ↦ [p ∈ {} ↦ [index ↦ 0, value ↦ Nil, delete ↦ FALSE]]],
        rollback ↦ [
            index  ↦ 0,
            values ↦ [p ∈ {} ↦ [index ↦ 0, value ↦ Nil, delete ↦ FALSE]]],
        phase    ↦ ProposalCommit,
        state    ↦ ProposalInProgress]]
  ∧ Trace!Init

NextProposal ≜
  ∨ ∃ n ∈ Nodes :
      ∃ i ∈ DOMAIN proposal :
        Trace!Step(ReconcileProposal(n, i), [node ↦ n, index ↦ 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*