MODULE *Config*

INSTANCE *Naturals*

INSTANCE *FiniteSets*

INSTANCE *Sequences*

INSTANCE *TLC*

An empty constant

CONSTANT *Nil*

Transaction type constants

CONSTANTS
Change,
Rollback

Transaction isolation constants

CONSTANTS
ReadCommitted,
Serializable

Phase constants

CONSTANTS
Initialize,
Validate,
Abort,
Commit,
Apply

$$
\begin{array}{l}
Phase \triangleq \\
\quad \text{LET } phases \triangleq \langle Initialize, \\
\qquad\qquad\qquad\qquad Validate, \\
\qquad\qquad\qquad\qquad Abort, \\
\qquad\qquad\qquad\qquad Commit, \\
\qquad\qquad\qquad\qquad Apply\rangle \\
\quad \text{IN } \quad [p \in \{phases[i] : i \in \text{DOMAIN } phases\} \mapsto \\
\qquad\quad \text{CHOOSE } i \in \text{DOMAIN } phases : phases[i] = p]
\end{array}
$$

Status constants

CONSTANTS
Pending,
Complete,
Failed

$$Status \triangleq$$

LET $statuses \triangleq \langle Pending,$
$Complete,$
$Failed\rangle$
IN $[s \in \{statuses[i] : i \in \text{DOMAIN } statuses\} \mapsto$
CHOOSE $i \in \text{DOMAIN } statuses : statuses[i] = s]$

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* $\triangleq$ [
*target*1 $\mapsto$ [*persistent* $\mapsto$ FALSE, *values* $\mapsto$ [
*path*1 $\mapsto$ {"*value*1", "*value*2"},
*path*2 $\mapsto$ {"*value*2", "*value*3"}]],
*target*2 $\mapsto$ [*persistent* $\mapsto$ TRUE, *values* $\mapsto$ [
*path*2 $\mapsto$ {"*value*3", "*value*4"},
*path*3 $\mapsto$ {"*value*4", "*value*5"}]]]

CONSTANT *Target*

Configuration update/rollback requests are tracked and processed through two data types. Transactions represent the lifecycle of a single configuration change request and are stored in an append-only log. Configurations represent the desired configuration of a *gNMI* target based on the aggregate of relevant changes in the Transaction log.

TYPE Type ::= *type* $\in$
{*Change*,
Rollback}

TYPE *Phase* ::= *phase* $\in$
{*Initialize*,
Validate,
Abort,
Commit,
Apply}

TYPE *Status* ::= *status* $\in$
{*Pending*,
Initializing,
Initialized,
Validating,
Validated,

```
    Committing,
    Committed,
    Applying,
    Applied,
    Synchronizing,
    Synchronized,
    Persisted,
    Failed}

TYPE Transaction ≜ [
  type      ::= type ∈ Type,
  index     ::= index ∈ Nat,
  isolation ::= isolation ∈ {IsolationDefault, IsolationSerializable}
  values ::= [
    target ∈ SUBSET (DOMAIN Target) ↦ [ path ∈ SUBSET (DOMAIN Target[target].values) ↦
      [
        value ::= value ∈ STRING,
        delete ::= delete ∈ BOOLEAN ]]],
  rollback ::= index ∈ Nat,
  targets ::= targets ∈ SUBSET (DOMAIN Target)
  phase     ::= phase ∈ Phase]
  status ::= status ∈ Status]

TYPE Proposal ≜ [
  type            ::= type ∈ Type,
  index           ::= index ∈ Nat,
  values          ::= [ path ∈ SUBSET (DOMAIN Target[target].values) ↦ [
      value ::= value ∈ STRING,
      delete ::= delete ∈ BOOLEAN ]],
  rollback        ::= index ∈ Nat,
  dependencyIndex ::= dependencyIndex ∈ Nat,
  rollbackIndex ::= rollbackIndex ∈ Nat,
  rollbackValues ::= [ path ∈ SUBSET (DOMAIN Target[target].values) ↦ [
      value ::= value ∈ STRING,
      delete ::= delete ∈ BOOLEAN ]],
  phase       ::= phase ∈ Phase]
  status      ::= status ∈ Status]

TYPE Configuration ≜ [
  id              ::= id ∈ STRING,
  target          ::= target ∈ STRING,
  values          ::= [ path ∈ SUBSET (DOMAIN Target[target]) ↦ [
      value ::= value ∈ STRING,
      index ::= index ∈ Nat,
      deleted ::= delete ∈ BOOLEAN ]],
  configIndex ::= configIndex ∈ Nat,
  configTerm      ::= configTerm ∈ Nat,
  proposedIndex ::= proposedIndex ∈ Nat,
  committedIndex ::= committedIndex ∈ Nat,
  appliedIndex ::= appliedIndex ∈ Nat,
  appliedTerm ::= appliedTerm ∈ Nat,
```

```
    appliedValues ::= [ path ∈ SUBSET (DOMAIN Target[target]) ↦ [
        value ::= value ∈ STRING,
        index ::= index ∈ Nat,
        deleted ::= delete ∈ BOOLEAN ]],
    status ::= status ∈ Status]
```

A transaction log. Transactions may either request a set of changes to a set of targets or rollback a prior change.

VARIABLE *transaction*

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*

$vars \triangleq \langle transaction, proposal, configuration, mastership, target \rangle$

This section models *mastership* for the configuration service.

Mastership is used primarily to track the lifecycle of individual configuration targets and react to state changes on the southbound. Each target is assigned a master from the *Node* set, and masters can be unset when the target disconnects.

Set node n as the master for target t

$SetMaster(n, t) \triangleq$
- $\land\ mastership[t].master \neq n$
- $\land\ mastership' = [mastership \text{ EXCEPT } ![t].term = mastership[t].term + 1,$
  $![t].master = n]$
- $\land$ UNCHANGED $\langle transaction, proposal, configuration, target \rangle$

$UnsetMaster(t) \triangleq$
- $\land\ mastership[t].master \neq Nil$
- $\land\ mastership' = [mastership \text{ EXCEPT } ![t].master = Nil]$
- $\land$ UNCHANGED $\langle transaction, proposal, configuration, target \rangle$

This section models configuration changes and rollbacks. Changes are appended to the transaction log and processed asynchronously.

$Value(s, t, p) \triangleq$
- LET $value \triangleq$ CHOOSE $v \in s : v.target = t \land v.path = p$
- IN

```
        [value  ↦ value.value,
         delete ↦ value.delete]

Paths(s, t) ≜
   [p ∈ {v.path : v ∈ {v ∈ s : v.target = t}} ↦ Value(s, t, p)]

Changes(s) ≜
   [t ∈ {v.target : v ∈ s} ↦ Paths(s, t)]

ValidValues(t, p) ≜
   UNION {{[value ↦ v, delete ↦ FALSE] : v ∈ Target[t].values[p]}, {[value ↦ Nil, delete ↦ TRUE]}}

ValidPaths(t) ≜
   UNION {{v @@ [path ↦ p] : v ∈ ValidValues(t, p)} : p ∈ DOMAIN Target[t].values}

ValidTargets ≜
   UNION {{p @@ [target ↦ t] : p ∈ ValidPaths(t)} : t ∈ DOMAIN Target}

The set of all valid sets of changes to all targets and their paths.
The set of possible changes is computed from the Target model value.
ValidChanges ≜
   LET changeSets ≜ {s ∈ SUBSET ValidTargets :
                       ∀ t ∈ DOMAIN Target   :
                         ∀ p ∈ DOMAIN Target[t].values :
                           Cardinality({v ∈ s : v.target = t ∧ v.path = p}) ≤ 1}
   IN
      {Changes(s) : s ∈ changeSets}

The next available index in the transaction log.
This is computed as the max of the existing indexes in the log to
allow for changes to the log (e.g. log compaction) to be modeled.
NextIndex ≜
   IF DOMAIN transaction = {} THEN
      1
    ELSE
      LET i ≜ CHOOSE i ∈ DOMAIN transaction :
            ∀ j ∈ DOMAIN transaction : i ≥ j
      IN    i + 1

Add a set of changes 'c' to the transaction log
RequestChange(c) ≜
   ∧ ∃ isolation ∈ {ReadCommitted, Serializable} :
       ∧ transaction' = transaction @@ (NextIndex :> [type      ↦ Change,
                                                      index     ↦ NextIndex,
                                                      isolation ↦ isolation,
                                                      values    ↦ c,
                                                      targets   ↦ {},
```

$$\begin{array}{ll} & phase \mapsto Initialize, \\ & status \mapsto Pending]) \end{array}$$

$\land$ UNCHANGED $\langle proposal, configuration, mastership, target \rangle$

Add a rollback of transaction 't' to the transaction log

$RequestRollback(t) \triangleq$

$\land \exists\, isolation \in \{ReadCommitted, Serializable\} :$
$\land\, transaction' = transaction \,@@\, (NextIndex :> [type \mapsto Rollback,$
$index \mapsto NextIndex,$
$isolation \mapsto isolation,$
$rollback \mapsto t,$
$targets \mapsto \{\},$
$phase \mapsto Initialize,$
$status \mapsto Pending])$

$\land$ UNCHANGED $\langle proposal, configuration, mastership, target \rangle$

This section models the Transaction log reconciler.

Transactions come in two flavors : − *Change* transactions contain a set of changes to be applied to a set of *targets* − *Rollback* transactions reference a prior change transaction to be reverted to the previous state

Transacations proceed through a series of phases:
* *Initialize* − create and link Proposals
* *Validate* − validate changes and rollbacks
* *Commit* − commit changes to Configurations
* *Apply* − commit changes to Targets

Reconcile a transaction

$ReconcileTransaction(n, i) \triangleq$

Initialize is the only transaction phase that's globally serialized. While in the *Initializing* phase, the reconciler checks whether the prior transaction has been *Initialized* before creating Proposals in the *Initialize* phase. Once all of the transaction's proposals have been *Initialized*, the transaction will be marked *Initialized*. If any proposal is *Failed*, the transaction will be marked *Failed* as well.

$\land\ \lor\ \land\, transaction[i].phase = Initialize$
$\land\ \lor\ \land\, transaction[i].status = Pending$

Serialize transaction initialization

$\land\ \lor\, i - 1 \notin$ DOMAIN $transaction$
$\lor\, Phase[transaction[i-1].phase] > Phase[Initialize]$
$\lor\, transaction[i-1].status \neq Pending$

If the transaction's targets are not yet set, create proposals and add targets to the transaction state.

$\land\ \lor\ \land\, transaction[i].targets = \{\}$

If the transaction is a change, the targets are taken from the change values.

$$
\begin{array}{l}
\land \lor \land transaction[i].type = Change \\
\quad \land transaction' = [transaction \text{ EXCEPT } ![i].targets = \text{DOMAIN } transaction[i].values] \\
\quad \land proposal' = [t \in \text{DOMAIN } proposal \mapsto \\
\qquad \text{IF } t \in \text{DOMAIN } transaction[i].values \text{ THEN} \\
\qquad\quad proposal[t] \mathbin{@@} (i :> [type \mapsto Change, \\
\qquad\qquad\qquad\qquad\qquad index \mapsto i, \\
\qquad\qquad\qquad\qquad\qquad values \mapsto transaction[i].values[t], \\
\qquad\qquad\qquad\qquad\qquad dependencyIndex \mapsto 0, \\
\qquad\qquad\qquad\qquad\qquad rollbackIndex \mapsto 0, \\
\qquad\qquad\qquad\qquad\qquad rollbackValues \mapsto \langle\rangle, \\
\qquad\qquad\qquad\qquad\qquad phase \mapsto Initialize, \\
\qquad\qquad\qquad\qquad\qquad status \mapsto Pending]) \\
\qquad \text{ELSE} \\
\qquad\quad proposal[t]]
\end{array}
$$

If the transaction is a rollback, the targets affected are the targets of the change transaction being rolled back.

$$
\begin{array}{l}
\quad \lor \land transaction[i].type = Rollback \\
\quad\quad \land \lor \land transaction[i].rollback \in \text{DOMAIN } transaction \\
\quad\quad\quad\quad \land transaction[transaction[i].rollback].type = Change \\
\quad\quad\quad\quad \land transaction' = [transaction \text{ EXCEPT } ![i].targets = \\
\qquad\qquad\qquad\qquad\qquad\qquad \text{DOMAIN } transaction[transaction[i].rollback].values] \\
\quad\quad\quad\quad \land proposal' = [t \in \text{DOMAIN } proposal \mapsto \\
\qquad\qquad\qquad\qquad \text{IF } t \in \text{DOMAIN } transaction[transaction[i].rollback].values \text{ THEN} \\
\qquad\qquad\qquad\qquad\quad proposal[t] \mathbin{@@} (i :> [type \mapsto Rollback, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad index \mapsto i, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad rollback \mapsto transaction[i].rollback, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad dependencyIndex \mapsto 0, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad rollbackIndex \mapsto 0, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad rollbackValues \mapsto \langle\rangle, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad phase \mapsto Initialize, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad status \mapsto Pending]) \\
\qquad\qquad\qquad\qquad \text{ELSE} \\
\qquad\qquad\qquad\qquad\quad proposal[t]] \\
\quad\quad\quad \lor \land \lor \land transaction[i].rollback \in \text{DOMAIN } transaction \\
\qquad\qquad\qquad\qquad\quad \land transaction[transaction[i].rollback].type = Rollback \\
\qquad\qquad\qquad\qquad \lor transaction[i].rollback \notin \text{DOMAIN } transaction \\
\quad\quad\quad\quad \land transaction' = [transaction \text{ EXCEPT } ![i].status = Failed] \\
\quad\quad\quad\quad \land \text{UNCHANGED } \langle proposal \rangle \\
\lor \land transaction[i].targets \neq \{\}
\end{array}
$$

If all proposals have been *Complete*, mark the transaction *Complete*.

$$
\begin{array}{l}
\quad \land \lor \land \forall\, t \in transaction[i].targets : proposal[t][i].status = Complete \\
\quad\quad\quad \land transaction' = [transaction \text{ EXCEPT } ![i].status = Complete] \\
\quad\quad\quad \land \text{UNCHANGED } \langle proposal \rangle
\end{array}
$$

If any proposal has been *Failed*, mark the transaction *Failed*.

$$
\quad \lor \land \exists\, t \in transaction[i].targets : proposal[t][i].status = Failed
$$

```
                    ∧ transaction′ = [transaction EXCEPT ![i].status = Failed]
                    ∧ UNCHANGED ⟨proposal⟩
        Once the transaction has been Initialized, proceed to the Validate phase.
        If any of the transaction's proposals depend on a Serializable transaction,
        verify the dependency has been Validated to preserve serializability before
        moving the transaction to the Validate phase.
    ∨ ∧ transaction[i].status = Complete
      ∧ ∀ t ∈ transaction[i].targets :
          ∨ proposal[t][i].dependencyIndex = 0
          ∨ transaction[proposal[t][i].dependencyIndex].isolation ≠ Serializable
          ∨ Phase[transaction[proposal[t][i].dependencyIndex].phase] > Phase[Validate]
          ∨ ∧ transaction[proposal[t][i].dependencyIndex].phase = Validate
            ∧ transaction[proposal[t][i].dependencyIndex].status ∈ {Complete, Failed}
      ∧ transaction′ = [transaction EXCEPT ![i].phase  = Validate,
                                           ![i].status = Pending]
      ∧ UNCHANGED ⟨proposal⟩
    ∨ ∧ transaction[i].status = Failed
      ∧ transaction′ = [transaction EXCEPT ![i].phase  = Abort,
                                           ![i].status = Pending]
      ∧ UNCHANGED ⟨proposal⟩
∨ ∧ transaction[i].phase = Validate
  ∧ ∨ ∧ transaction[i].status = Pending
        Move the transaction's proposals to the Validating state
      ∧ ∨ ∧ ∃ t ∈ transaction[i].targets : proposal[t][i].phase ≠ Validate
          ∧ proposal′ = [t ∈ DOMAIN proposal ↦
                          IF t ∈ transaction[i].targets THEN
                            [proposal[t] EXCEPT ![i].phase  = Validate,
                                                ![i].status = Pending]
                          ELSE
                            proposal[t]]
          ∧ UNCHANGED ⟨transaction⟩
        If all proposals have been Complete, mark the transaction Complete.
        ∨ ∧ ∀ t ∈ transaction[i].targets : proposal[t][i].status = Complete
          ∧ transaction′ = [transaction EXCEPT ![i].status = Complete]
          ∧ UNCHANGED ⟨proposal⟩
        If any proposal has been Failed, mark the transaction Failed.
        ∨ ∧ ∃ t ∈ transaction[i].targets : proposal[t][i].status = Failed
          ∧ transaction′ = [transaction EXCEPT ![i].status = Failed]
          ∧ UNCHANGED ⟨proposal⟩
    Once the transaction has been Validated, proceed to the Commit phase.
    If any of the transaction's proposals depend on a Serializable transaction,
    verify the dependency has been Committed to preserve serializability before
    moving the transaction to the Commit phase.
  ∨ ∧ transaction[i].status = Complete
    ∧ ∀ t ∈ transaction[i].targets :
```

```
                \/ proposal[t][i].dependencyIndex = 0
                \/ transaction[proposal[t][i].dependencyIndex].isolation # Serializable
                \/ Phase[transaction[proposal[t][i].dependencyIndex].phase] > Phase[Commit]
                \/ /\ transaction[proposal[t][i].dependencyIndex].phase = Commit
                   /\ transaction[proposal[t][i].dependencyIndex].status \in {Complete, Failed}
             /\ transaction' = [transaction EXCEPT ![i].phase  = Commit,
                                                   ![i].status = Pending]
             /\ UNCHANGED <proposal>
       \/ /\ transaction[i].phase = Commit
          /\ \/ /\ transaction[i].status = Pending
                Move the transaction's proposals to the Committing state
                /\ \/ /\ \E t \in transaction[i].targets : proposal[t][i].phase # Commit
                      /\ proposal' = [t \in DOMAIN proposal |->
                                        IF t \in transaction[i].targets THEN
                                           [proposal[t] EXCEPT ![i].phase  = Commit,
                                                               ![i].status = Pending]
                                        ELSE
                                           proposal[t]]
                      /\ UNCHANGED <transaction>
                   If all proposals have been Complete, mark the transaction Complete.
                   \/ /\ \A t \in transaction[i].targets : proposal[t][i].status = Complete
                      /\ transaction' = [transaction EXCEPT ![i].status = Complete]
                      /\ UNCHANGED <proposal>
             Once the transaction has been Committed, proceed to the Apply phase.
             If any of the transaction's proposals depend on a Serializable transaction,
             verify the dependency has been Applied to preserve serializability before
             moving the transaction to the Apply phase.
             \/ /\ transaction[i].status = Complete
                /\ \A t \in transaction[i].targets :
                      \/ proposal[t][i].dependencyIndex = 0
                      \/ transaction[proposal[t][i].dependencyIndex].isolation # Serializable
                      \/ Phase[transaction[proposal[t][i].dependencyIndex].phase] > Phase[Apply]
                      \/ /\ transaction[proposal[t][i].dependencyIndex].phase = Apply
                         /\ transaction[proposal[t][i].dependencyIndex].status \in {Complete, Failed}
                /\ transaction' = [transaction EXCEPT ![i].phase  = Apply,
                                                      ![i].status = Pending]
                /\ UNCHANGED <proposal>
       \/ /\ transaction[i].phase = Apply
          /\ transaction[i].status = Pending
             Move the transaction's proposals to the Applying state
          /\ \/ /\ \E t \in transaction[i].targets : proposal[t][i].phase # Apply
                /\ proposal' = [t \in DOMAIN proposal |->
                                  IF t \in transaction[i].targets THEN
                                     [proposal[t] EXCEPT ![i].phase  = Apply,
                                                         ![i].status = Pending]
```

```
                                ELSE
                                  proposal[t]]
            ∧ UNCHANGED ⟨transaction⟩
        If all proposals have been Complete, mark the transaction Complete.
        ∨ ∧ ∀ t ∈ transaction[i].targets : proposal[t][i].status = Complete
          ∧ transaction' = [transaction EXCEPT ![i].status = Complete]
          ∧ UNCHANGED ⟨proposal⟩
        If any proposal has been Failed, mark the transaction Failed.
        ∨ ∧ ∃ t ∈ transaction[i].targets : proposal[t][i].status = Failed
          ∧ transaction' = [transaction EXCEPT ![i].status = Failed]
          ∧ UNCHANGED ⟨proposal⟩
    The Aborting state is used to clean up transactions that have failed during
    the Initializing or Validating phases.
    ∨ ∧ transaction[i].phase = Abort
      ∧ transaction[i].status = Pending
        Move the transaction's proposals to the Aborting state
      ∧ ∨ ∧ ∃ t ∈ transaction[i].targets : proposal[t][i].phase ≠ Abort
        ∧ proposal' = [t ∈ DOMAIN proposal ↦
                        IF t ∈ transaction[i].targets THEN
                          [proposal[t] EXCEPT ![i].phase  = Abort,
                                              ![i].status = Pending]
                        ELSE
                          proposal[t]]
        ∧ UNCHANGED ⟨transaction⟩
      If all proposals have been Complete, mark the transaction Complete.
      ∨ ∧ ∀ t ∈ transaction[i].targets : proposal[t][i].status = Complete
        ∧ transaction' = [transaction EXCEPT ![i].status = Complete]
        ∧ UNCHANGED ⟨proposal⟩
  ∧ UNCHANGED ⟨configuration, mastership, target⟩

Reconcile a proposal
ReconcileProposal(n, t, i) ≜
  ∧ ∨ ∧ proposal[t][i].phase = Initialize
      ∧ proposal[t][i].status = Pending
      ∧ proposal' = [proposal EXCEPT ![t] = [
            proposal[t] EXCEPT ![i] = [
              status          ↦ Complete,
              dependencyIndex ↦ configuration[t].proposedIndex] @@ proposal[t][i]]]
      ∧ configuration' = [configuration EXCEPT ![t].proposedIndex = 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 = Validate
      ∧ proposal[t][i].status = Pending
```

```
∧ configuration[t].committedIndex = proposal[t][i].dependencyIndex
    For Change proposals validate the set of requested changes.
∧ ∨ ∧ proposal[t][i].type = Change
    ∧ LET rollbackIndex ≜ configuration[t].configIndex
          rollbackValues ≜ [p ∈ DOMAIN proposal[t][i].values ↦
                              IF p ∈ DOMAIN configuration[t].values THEN
                                 configuration[t].values[p]
                              ELSE
                                 [value ↦ Nil,
                                  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].rollbackIndex  = rollbackIndex,
                                                 ![i].rollbackValues = rollbackValues,
                                                 ![i].status         = Complete]]
           ∨ ∧ r = Invalid
             ∧ proposal′ = [proposal EXCEPT ![t] = [
                              proposal[t] EXCEPT ![i].status = 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].type = Change
        Only roll back the change if it's the lastest change made
        to the configuration based on the configuration index.
      ∧ ∨ ∧ configuration[t].configIndex = proposal[t][i].rollback
          ∧ LET rollbackIndex ≜ proposal[t][proposal[t][i].rollback].rollbackIndex
                rollbackValues ≜ proposal[t][proposal[t][i].rollback].rollbackValues
            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].rollbackIndex  = rollbackIndex,
                                                       ![i].rollbackValues = rollbackValues,
                                                       ![i].status         = Complete]]
                 ∨ ∧ r = Invalid
                   ∧ proposal′ = [proposal EXCEPT ![t] = [
                                    proposal[t] EXCEPT ![i].status = Failed]]
```

```
                    If the Rollback target is not the most recent change to the configuration,
                    fail validation for the proposal.
                  ∨ ∧ configuration[t].configIndex ≠ proposal[t][i].rollback
                    ∧ proposal' = [proposal EXCEPT ![t] = [proposal[t] EXCEPT ![i].status = Failed]]
                If a Rollback proposal is attempting to roll back another Rollback,
                fail validation for the proposal.
              ∨ ∧ proposal[t][proposal[t][i].rollback].type = Rollback
                ∧ proposal' = [proposal EXCEPT ![t] = [
                      proposal[t] EXCEPT ![i].status  = Failed]]
  ∧ UNCHANGED ⟨configuration, target⟩
While in the Commit state, commit the proposed changes to the configuration.
∨ ∧ proposal[t][i].phase = Commit
  ∧ proposal[t][i].status = Pending
   Only commit the proposal if the prior proposal has already been committed.
  ∧ configuration[t].committedIndex = proposal[t][i].dependencyIndex
     If the proposal is a change, commit the change values and set the configuration
     index to the proposal index.
  ∧ ∨ ∧ proposal[t][i].type = Change
      ∧ configuration' = [configuration EXCEPT ![t].values         = proposal[t][i].values,
                                               ![t].configIndex    = i,
                                               ![t].committedIndex = i]
     If the proposal is a rollback, commit the rollback values and index. This
     will cause the configuration index to be reverted to the index prior to
     the transaction/proposal being rolled back.
    ∨ ∧ proposal[t][i].type = Rollback
      ∧ configuration' = [configuration EXCEPT ![t].values         = proposal[t][i].rollbackValues,
                                               ![t].configIndex    = proposal[t][i].rollbackIndex,
                                               ![t].committedIndex = i]
  ∧ proposal' = [proposal EXCEPT ![t] = [proposal[t] EXCEPT ![i].status = Complete]]
  ∧ UNCHANGED ⟨target⟩
While in the Apply phase, apply the proposed changes to the target.
∨ ∧ proposal[t][i].phase = Apply
  ∧ proposal[t][i].status = Pending
  ∧ configuration[t].appliedIndex = proposal[t][i].dependencyIndex
  ∧ configuration[t].appliedTerm = mastership[t].term
  ∧ mastership[t].master = n
   Model successful and failed target update requests.
  ∧ ∃ r ∈ {Success, Failure} :
      ∨ ∧ r = Success
           If the proposal is a change, apply the change values to the target
           and update the configuration's applied index and values.
        ∧ ∨ ∧ proposal[t][i].type = Change
            ∧ target' = [target EXCEPT ![t] = proposal[t][i].values @@ target[t]]
            ∧ configuration' = [configuration EXCEPT
                  ![t].appliedIndex = i,
```

```
                        ![t].appliedValues = proposal[t][i].values @@ configuration[t].appliedValues]
                  If the proposal is a rollback, apply the rollback values and update the
                  configuration's applied values with the rolled back values.
                 ∨ ∧ proposal[t][i].type = Rollback
                   ∧ target' = [target EXCEPT ![t] = proposal[t][i].rollbackValues @@ target[t]]
                   ∧ configuration' = [configuration EXCEPT
                         ![t].appliedIndex  = i,
                         ![t].appliedValues = proposal[t][i].rollbackValues @@ configuration[t].appliedValues]
               ∧ proposal' = [proposal EXCEPT ![t] = [proposal[t] EXCEPT ![i].status = Complete]]
            If the proposal could not be applied, update the configuration's applied index
            and mark the proposal Failed.
           ∨ ∧ r = Failure
             ∧ configuration' = [configuration EXCEPT ![t].appliedIndex = i]
             ∧ proposal' = [proposal EXCEPT ![t] = [proposal[t] EXCEPT ![i].status = Failed]]
             ∧ UNCHANGED ⟨target⟩
     ∨ ∧ proposal[t][i].phase = Abort
       ∧ proposal[t][i].status = Pending
       ∧ configuration' = [configuration EXCEPT ![t].committedIndex = i]
       ∧ proposal' = [proposal EXCEPT ![t] = [proposal[t] EXCEPT ![i].status = Complete]]
       ∧ UNCHANGED ⟨target⟩
  ∧ UNCHANGED ⟨transaction, mastership⟩
```

This section models the Configuration reconciler.

```
ReconcileConfiguration(n, t) ≜
  ∧ ∨ ∧ Target[t].persistent
      ∧ configuration[t].status ≠ Complete
      ∧ configuration' = [configuration EXCEPT ![t].status = Complete]
      ∧ UNCHANGED ⟨target⟩
    ∨ ∧ ¬Target[t].persistent
      ∧ ∨ mastership[t].term > configuration[t].configTerm
        ∨ ∧ mastership[t].term = configuration[t].configTerm
          ∧ mastership[t].master = Nil
      ∧ configuration' = [configuration EXCEPT ![t].configTerm = mastership[t].term,
                                               ![t].status     = Pending]
      ∧ UNCHANGED ⟨target⟩
    ∨ ∧ configuration[t].status = Pending
      ∧ mastership[t].term = configuration[t].configTerm
      ∧ mastership[t].master = n
      ∧ target' = [target EXCEPT ![t] = configuration[t].appliedValues]
      ∧ configuration' = [configuration EXCEPT ![t].appliedTerm = mastership[t].term,
                                               ![t].status      = Complete]
  ∧ UNCHANGED ⟨proposal, transaction, mastership⟩
```

Formal specification, constraints, and theorems.

$$
\begin{array}{l}
\mathit{Init} \triangleq \\
\quad \land\ \mathit{transaction} = \langle\rangle \\
\quad \land\ \mathit{proposal} = [t \in \text{DOMAIN } \mathit{Target} \mapsto \\
\qquad\qquad [p \in \{\} \mapsto [\mathit{phase} \mapsto \mathit{Initialize}, \\
\qquad\qquad\qquad\qquad \mathit{status} \mapsto \mathit{Pending}]]] \\
\quad \land\ \mathit{configuration} = [t \in \text{DOMAIN } \mathit{Target} \mapsto \\
\qquad\qquad\qquad [\mathit{target} \mapsto t, \\
\qquad\qquad\qquad\ \mathit{status} \mapsto \mathit{Pending}, \\
\qquad\qquad\qquad\ \mathit{values} \mapsto \\
\qquad\qquad\qquad\quad [\mathit{path} \in \{\} \mapsto \\
\qquad\qquad\qquad\qquad [\mathit{path} \mapsto \mathit{path}, \\
\qquad\qquad\qquad\qquad\ \mathit{value} \mapsto \mathit{Nil}, \\
\qquad\qquad\qquad\qquad\ \mathit{index} \mapsto 0, \\
\qquad\qquad\qquad\qquad\ \mathit{deleted} \mapsto \text{FALSE}]], \\
\qquad\qquad\qquad\ \mathit{configIndex} \mapsto 0, \\
\qquad\qquad\qquad\ \mathit{configTerm} \mapsto 0, \\
\qquad\qquad\qquad\ \mathit{proposedIndex} \mapsto 0, \\
\qquad\qquad\qquad\ \mathit{committedIndex} \mapsto 0, \\
\qquad\qquad\qquad\ \mathit{appliedIndex} \mapsto 0, \\
\qquad\qquad\qquad\ \mathit{appliedTerm} \mapsto 0, \\
\qquad\qquad\qquad\ \mathit{appliedValues} \mapsto \\
\qquad\qquad\qquad\quad [\mathit{path} \in \{\} \mapsto \\
\qquad\qquad\qquad\qquad [\mathit{path} \mapsto \mathit{path}, \\
\qquad\qquad\qquad\qquad\ \mathit{value} \mapsto \mathit{Nil}, \\
\qquad\qquad\qquad\qquad\ \mathit{index} \mapsto 0, \\
\qquad\qquad\qquad\qquad\ \mathit{deleted} \mapsto \text{FALSE}]]]] \\
\quad \land\ \mathit{target} = [t \in \text{DOMAIN } \mathit{Target} \mapsto \\
\qquad\qquad [\mathit{path} \in \{\} \mapsto \\
\qquad\qquad\quad [\mathit{value} \mapsto \mathit{Nil}]]] \\
\quad \land\ \mathit{mastership} = [t \in \text{DOMAIN } \mathit{Target} \mapsto [\mathit{master} \mapsto \mathit{Nil},\ \mathit{term} \mapsto 0]]
\end{array}
$$

$$
\begin{array}{l}
\mathit{Next} \triangleq \\
\quad \lor \exists\, c \in \mathit{ValidChanges} : \\
\qquad \mathit{RequestChange}(c) \\
\quad \lor \exists\, t \in \text{DOMAIN } \mathit{transaction} : \\
\qquad \mathit{RequestRollback}(t) \\
\quad \lor \exists\, n \in \mathit{Node} : \\
\qquad \exists\, t \in \text{DOMAIN } \mathit{Target} : \\
\qquad\quad \mathit{SetMaster}(n,\ t) \\
\quad \lor \exists\, t \in \text{DOMAIN } \mathit{Target} : \\
\qquad \mathit{UnsetMaster}(t) \\
\quad \lor \exists\, n \in \mathit{Node} : \\
\qquad \exists\, t \in \text{DOMAIN } \mathit{transaction} : \\
\qquad\quad \mathit{ReconcileTransaction}(n,\ t)
\end{array}
$$

$\lor \exists\, n \in Node :$
$\quad \exists\, t \in \text{DOMAIN } proposal :$
$\quad\quad \exists\, i \in \text{DOMAIN } proposal[t] :$
$\quad\quad\quad ReconcileProposal(n,\, t,\, i)$
$\lor \exists\, n \in Node :$
$\quad \exists\, c \in \text{DOMAIN } configuration :$
$\quad\quad ReconcileConfiguration(n,\, c)$

$Spec \triangleq Init \land \Box[Next]_{vars}$

$Order \triangleq$
$\land\ \forall\, i,\, j \in \text{DOMAIN } transaction :$
$\quad \lor\, j \leq i$
$\quad \lor\, Phase[transaction[i].phase] \geq Phase[transaction[j].phase]$
$\quad \lor\, transaction[j].status = Failed$
$\land\ \forall\, t \in \text{DOMAIN } proposal :$
$\quad \forall\, i,\, j \in \text{DOMAIN } proposal[t] :$
$\quad\quad \lor\, j \leq i$
$\quad\quad \lor\, Phase[proposal[t][i].phase] \geq Phase[proposal[t][j].phase]$
$\quad\quad \lor\, proposal[t][i].status = Failed$

$Consistency \triangleq$
$\forall\, t \in \text{DOMAIN } target :$
LET
Compute the transaction indexes that have been applied to the target
$appliedIndexes \triangleq \{i \in \text{DOMAIN } transaction :$
$\land\, transaction[i].type = Change$
$\land\, i \in \text{DOMAIN } proposal[t]$
$\land\, proposal[t][i].phase = Apply$
$\land\, proposal[t][i].status = Complete$
$\land\, t \in \text{DOMAIN } transaction[i].values$
$\land \neg\exists\, j \in \text{DOMAIN } transaction :$
$\land\, j > i$
$\land\, transaction[j].type = Rollback$
$\land\, transaction[j].rollback = i$
$\land\, transaction[j].phase = Apply$
$\land\, transaction[j].status = Complete\}$
Compute the set of paths in the target that have been updated by transactions
$appliedPaths \triangleq \text{UNION } \{\text{DOMAIN } transaction[i].values[t] : i \in appliedIndexes\}$
Compute the highest index applied to the target for each path
$pathIndexes \triangleq [p \in appliedPaths \mapsto \text{CHOOSE } i \in appliedIndexes :$
$\forall\, j \in appliedIndexes :$
$\land\, i \geq j$
$\land\, p \in \text{DOMAIN } transaction[i].values]$
Compute the expected target configuration based on the last indexes applied to the target for each path.

$expectedConfig \triangleq [p \in \text{DOMAIN } pathIndexes \mapsto transaction[pathIndexes[p]].values[p]]$

IN

$target[t] = expectedConfig$

$Isolation \triangleq$

$\forall\, i, j \in \text{DOMAIN } transaction :$
$\lor\ j \leq i$
$\lor\ transaction[i].targets \cap transaction[j].targets = \{\}$
$\lor\ transaction[i].isolation \neq Serializable$
$\lor\ \land\ \lor\ \land\ transaction[i].phase \in \{Commit,\ Abort\}$
$\land\ transaction[i].status \in \{Complete,\ Failed\}$
$\lor\ Phase[transaction[i].phase] > Phase[Commit]$
$\lor\ Phase[transaction[j].phase] < Phase[Commit]$
$\land\ \lor\ \land\ transaction[i].phase \in \{Apply,\ Abort\}$
$\land\ transaction[i].status \in \{Complete,\ Failed\}$
$\lor\ Phase[transaction[j].phase] < Phase[Apply]$
$\lor\ transaction[j].status = Failed$

THEOREM $Safety \triangleq Spec \Rightarrow \Box(Order \land Consistency \land Isolation)$

$Completion \triangleq$

$\land\ \forall\, i \in \text{DOMAIN } transaction :$
$\land\ transaction[i].phase = Commit$
$\land\ transaction[i].status \in \{Complete,\ Failed\}$
$\land\ \forall\, i \in \text{DOMAIN } transaction :$
$\land\ transaction[i].phase = Apply$
$\land\ transaction[i].status \in \{Complete,\ Failed\}$
$\land\ \forall\, t \in \text{DOMAIN } proposal :$
$\forall\, i \in \text{DOMAIN } proposal[t] :$
$\land\ proposal[t][i].phase = Commit$
$\land\ proposal[t][i].status \in \{Complete,\ Failed\}$
$\land\ \forall\, t \in \text{DOMAIN } proposal :$
$\forall\, i \in \text{DOMAIN } proposal[t] :$
$\land\ proposal[t][i].phase = Apply$
$\land\ proposal[t][i].status \in \{Complete,\ Failed\}$

THEOREM $Liveness \triangleq Spec \Rightarrow \Diamond Completion$

Type assumptions.

ASSUME $Nil \in \text{STRING}$

ASSUME $\forall\, phase \in Phase : phase \in \text{STRING}$

ASSUME $\forall\, status \in Status : status \in \text{STRING}$

ASSUME $\land\ IsFiniteSet(Node)$

$\land \forall\, n \in \mathit{Node} :$
$\quad \land\ n \notin \text{DOMAIN}\ \mathit{Target}$
$\quad \land\ n \in \text{STRING}$

ASSUME $\land \forall\, t \in \text{DOMAIN}\ \mathit{Target} :$
$\quad \land\ t \notin \mathit{Node}$
$\quad \land\ t \in \text{STRING}$
$\quad \land\ \mathit{Target}[t].\mathit{persistent} \in \text{BOOLEAN}$
$\quad \land \forall\, p \in \text{DOMAIN}\ \mathit{Target}[t].\mathit{values} :$
$\quad\quad \mathit{IsFiniteSet}(\mathit{Target}[t].\mathit{values}[p])$

* Modification History
* Last modified *Mon Feb* 07 02:17:42 *PST* 2022 by *jordanhalterman*
* Created *Wed Sep* 22 13:22:32 *PDT* 2021 by *jordanhalterman*