```
──────────────────────── MODULE Config ────────────────────────
INSTANCE Naturals

INSTANCE FiniteSets

INSTANCE Sequences

INSTANCE TLC
───────────────────────────────────────────────────────────────

  An empty constant
CONSTANT Nil

  Transaction type constants
CONSTANTS
    TransactionChange,
    TransactionRollback

  Transaction status constants
CONSTANTS
    TransactionPending,
    TransactionValidating,
    TransactionApplying,
    TransactionComplete,
    TransactionFailed

  Configuration status constants
CONSTANTS
    ConfigurationPending,
    ConfigurationInitializing,
    ConfigurationUpdating,
    ConfigurationComplete,
    ConfigurationFailed

  The set of all nodes
CONSTANT Node

Target is the possible targets, paths, and values
Example: Target ≜ [
    target1 ↦ [
      path1 ↦ {"value1", "value2"},
      path2 ↦ {"value2", "value3"}],
    target2 ↦ [
      path2 ↦ {"value3", "value4"},
      path3 ↦ {"value4", "value5"}]]
CONSTANT Target
```

```
ASSUME $Nil \in$ STRING

ASSUME $TransactionPending \in$ STRING
ASSUME $TransactionValidating \in$ STRING
ASSUME $TransactionApplying \in$ STRING
ASSUME $TransactionComplete \in$ STRING
ASSUME $TransactionFailed \in$ STRING

ASSUME $ConfigurationPending \in$ STRING
ASSUME $ConfigurationInitializing \in$ STRING
ASSUME $ConfigurationUpdating \in$ STRING
ASSUME $ConfigurationComplete \in$ STRING
ASSUME $ConfigurationFailed \in$ STRING

ASSUME $\land IsFiniteSet(Node)$
       $\land \forall n \in Node :$
           $\land n \notin$ DOMAIN $Target$
           $\land n \in$ STRING

ASSUME $\land \forall t \in$ DOMAIN $Target :$
           $\land t \notin Node$
           $\land t \in$ STRING
           $\land \forall p \in$ DOMAIN $Target[t] :$
               $IsFiniteSet(Target[t][p])$
```

```
TYPE $TransactionType ::= type \in$
  $\{TransactionChange,$
   $TransactionRollback\}$

TYPE $TransactionStatus ::= status \in$
  $\{TransactionPending,$
   $TransactionValidating,$
   $TransactionApplying,$
   $TransactionComplete,$
   $TransactionFailed\}$

TYPE Transaction $\triangleq$ [
  $type ::= type \in TransactionType,$
  $index ::= index \in Nat,$
  revision ::= revision $\in Nat,$
  $atomic ::= atomic \in$ BOOLEAN ,
  $sync ::= sync \in$ BOOLEAN ,
  $changes ::=$ [ $target \in$ SUBSET (DOMAIN $Target) \mapsto$ [
      $path \in$ SUBSET (DOMAIN $Target[target]) \mapsto$ [
         $value ::= value \in$ STRING,
         $delete ::= delete \in$ BOOLEAN ]]],
  $rollback ::= index \in Nat,$
  $status ::= status \in TransactionStatus]$
```

```
TYPE ConfigurationStatus ::= status ∈
  {ConfigurationPending,
   ConfigurationInitializing,
   ConfigurationUpdating,
   ConfigurationComplete,
   ConfigurationFailed}
TYPE Configuration ≜ [
  id       ::= id ∈ STRING,
  revision ::= revision ∈ Nat,
  target ::= target ∈ STRING,
  paths ::= [ path ∈ SUBSET (DOMAIN Target[target]) ↦ [
      value ::= value ∈ STRING,
      index ::= index ∈ Nat,
      deleted ::= delete ∈ BOOLEAN ]],
  txIndex ::= txIndex ∈ Nat,
  syncIndex ::= syncIndex ∈ Nat,
  term     ::= term ∈ Nat,
  status ::= status ∈ ConfigurationStatus]
```

A sequence of transactions
Each transactions contains a record of 'changes' for a set of targets
VARIABLE *transaction*

A record of target configurations
Each configuration represents the desired state of the target
VARIABLE *configuration*

A record of target states
VARIABLE *target*

A record of target masters
VARIABLE *master*

VARIABLE *history*

$vars \triangleq \langle transaction, configuration, master, target, history \rangle$

```
SetMaster(n, t) ≜
   ∧ master[t].master ≠ n
   ∧ master' = [master EXCEPT ![t].term   = master[t].term + 1,
                              ![t].master = n]
   ∧ UNCHANGED ⟨transaction, configuration, target, history⟩

UnsetMaster(t) ≜
   ∧ master[t].master ≠ Nil
   ∧ master' = [master EXCEPT ![t].master = Nil]
   ∧ UNCHANGED ⟨transaction, configuration, target, history⟩
```

This section models the northbound *API* for the configuration service.

```
$ValidValues(t, p) \triangleq$
  $\text{UNION } \{\{[value \mapsto v, delete \mapsto \text{FALSE}] : v \in Target[t][p]\}, \{[value \mapsto Nil, delete \mapsto \text{TRUE}]\}\}$

$ValidPaths(t) \triangleq$
  $\text{UNION } \{\{v \mathbin{@@} [path \mapsto p] : v \in ValidValues(t, p)\} : p \in \text{DOMAIN } Target[t]\}$

$ValidTargets \triangleq$
  $\text{UNION } \{\{p \mathbin{@@} [target \mapsto t] : p \in ValidPaths(t)\} : t \in \text{DOMAIN } Target\}$

$ValidPath(s, t, p) \triangleq$
  $\text{LET } value \triangleq \text{CHOOSE } v \in s : v.target = t \land v.path = p$
  $\text{IN}$
     $[value \mapsto value.value,$
      $delete \mapsto value.delete]$

$ValidTarget(s, t) \triangleq$
  $[p \in \{v.path : v \in \{v \in s : v.target = t\}\} \mapsto ValidPath(s, t, p)]$

$ValidChange(s) \triangleq$
  $[t \in \{v.target : v \in s\} \mapsto ValidTarget(s, t)]$

$ValidChanges \triangleq$
  $\text{LET } changeSets \triangleq \{s \in \text{SUBSET } ValidTargets :$
                             $\forall t \in \text{DOMAIN } Target :$
                               $\forall p \in \text{DOMAIN } Target[t] :$
                                 $Cardinality(\{v \in s : v.target = t \land v.path = p\}) \leq 1\}$
  $\text{IN}$
     $\{ValidChange(s) : s \in changeSets\}$

$NextIndex \triangleq$
  $\text{IF DOMAIN } transaction = \{\} \text{ THEN}$
     $1$
   $\text{ELSE}$
     $\text{LET } i \triangleq \text{CHOOSE } i \in \text{DOMAIN } transaction :$
            $\forall j \in \text{DOMAIN } transaction :$
              $i \geq j$
     $\text{IN} \quad i + 1$
```

Add a set of changes to the transaction log

```
$Change(c) \triangleq$
  $\land \quad transaction' = transaction \mathbin{@@} (NextIndex :> [type \mapsto TransactionChange,$
                                                      $index \mapsto NextIndex,$
                                                      $atomic \mapsto \text{FALSE},$
                                                      $sync \mapsto \text{FALSE},$
                                                      $changes \mapsto c,$
```

```
                                                    $sources \mapsto \langle\rangle,$
                                                    $status \mapsto TransactionPending])$
    $\land$ UNCHANGED $\langle configuration, master, target, history\rangle$

Add a rollback to the transaction log
$Rollback(t) \triangleq$
    $\land transaction[t].type = TransactionChange$
    $\land transaction' = transaction @@ (NextIndex :> [type \mapsto TransactionRollback,$
                                                   $index \mapsto NextIndex,$
                                                   $atomic \mapsto$ FALSE,
                                                   $sync \mapsto$ FALSE,
                                                   $rollback \mapsto t,$
                                                   $status \mapsto TransactionPending])$
    $\land$ UNCHANGED $\langle configuration, master, target, history\rangle$
```

```
This section models the Transaction log reconciler.
$ReconcileChange(n, i) \triangleq$
    If the transaction is Pending, begin validation if the prior transaction
    has already been applied. This simplifies concurrency control in the controller
    and guarantees transactions are applied to the configurations in sequential order.
    $\lor \land transaction[i].status = TransactionPending$
      $\land \lor \land i - 1 \in$ DOMAIN $transaction$
              $\land transaction[i - 1].status \in \{TransactionComplete, TransactionFailed\}$
         $\lor i - 1 \notin$ DOMAIN $transaction$
      $\land transaction' = [transaction$ EXCEPT $![i].status = TransactionValidating]$
      $\land$ UNCHANGED $\langle configuration, history\rangle$
    If the transaction is in the Validating state, compute and validate the
    Configuration for each target.
    $\lor \land transaction[i].status = TransactionValidating$
       If validation fails any target, mark the transaction Failed.
       If validation is successful, proceed to Applying.
      $\land \exists valid \in$ BOOLEAN :
           IF $valid$ THEN
              $\land transaction' = [transaction$ EXCEPT $![i].status = TransactionApplying]$
           ELSE
              $\land transaction' = [transaction$ EXCEPT $![i].status = TransactionFailed]$
      $\land$ UNCHANGED $\langle configuration, history\rangle$
    If the transaction is in the Applying state, update the Configuration for each
    target and Complete the transaction.
    $\lor \land transaction[i].status = TransactionApplying$
       Update the target configurations, adding the transaction index to each updated path
      $\land configuration' = [$
            $t \in$ DOMAIN $Target \mapsto$
              IF $t \in$ DOMAIN $transaction[i].changes$ THEN
```

```
                [$configuration[t]$ EXCEPT
                    $!.paths = [p \in$ DOMAIN $transaction[i].changes[t] \mapsto$
                                  $[index \mapsto transaction[i].index,$
                                   $value \mapsto transaction[i].changes[t][p].value,$
                                   $deleted \mapsto transaction[i].changes[t][p].delete]]$
                           @@ $configuration[t].paths,$
                    $!.txIndex = transaction[i].index,$
                    $!.status = ConfigurationPending]$
              ELSE
                $configuration[t]]$
      $\land history' = [r \in$ DOMAIN $Target \mapsto Append(history[r], configuration'[r])]$
      $\land transaction' = [transaction$ EXCEPT
          $![i].status = TransactionComplete,$
          $![i].sources = [t \in$ DOMAIN $transaction[i].changes \mapsto$
              LET $updatePaths \triangleq \{p \in$ DOMAIN $transaction[i].changes[t]:$
                                         $\neg transaction[i].changes[t][p].delete\}$
              IN $[p \in updatePaths \cap$ DOMAIN $configuration[t].paths \mapsto configuration[t].paths[p]]]]$

$ReconcileRollback(n, i) \triangleq$
    If the transaction is Pending, begin validation if the prior transaction
    has already been applied. This simplifies concurrency control in the controller
    and guarantees transactions are applied to the configurations in sequential order.
  $\lor \land transaction[i].status = TransactionPending$
     $\land \lor \land i - 1 \in$ DOMAIN $transaction$
           $\land transaction[i - 1].status \in \{TransactionComplete, TransactionFailed\}$
        $\lor i - 1 \notin$ DOMAIN $transaction$
     $\land transaction' = [transaction$ EXCEPT $![i].status = TransactionValidating]$
     $\land$ UNCHANGED $\langle configuration, history\rangle$
    If the transaction is in the Validating state, validate the rollback.
    A transaction can only be rolled back if:
    1. The source transaction is in the log
    2. The source transaction was applied successfully (did not fail validation)
    3. The source transaction is the most recent change for each path is modified
  $\lor \land transaction[i].status = TransactionValidating$
     $\land \lor \land transaction[transaction[i].rollback].status = TransactionComplete$
           $\land \lor \land transaction[i].rollback \in$ DOMAIN $transaction$
                    Determine whether the source transaction is the most recent change
                    by comparing the configuration path indexes to the transaction index.
                 $\land$ LET $canRollback \triangleq \forall t \in$ DOMAIN $transaction[transaction[i].rollback].changes:$
                                              $\forall p \in$ DOMAIN $transaction[transaction[i].rollback].changes[t]:$
                                                $configuration[t].paths[p].index = transaction[i].rollback$
                   IN
                     IF $canRollback$ THEN
                         $\land transaction' = [transaction$ EXCEPT $![i].status = TransactionApplying]$
                      ELSE
                    $\land\ transaction' = [transaction\ \text{EXCEPT}\ ![i].status = TransactionFailed]$
             If the source transaction failed, fail the rollback.
             $\lor\ \land\ transaction[i].rollback \notin \text{DOMAIN}\ transaction$
               $\land\ transaction' = [transaction\ \text{EXCEPT}\ ![i].status = TransactionFailed]$
          $\lor\ \land\ transaction[transaction[i].rollback].status = TransactionFailed$
            $\land\ transaction' = [transaction\ \text{EXCEPT}\ ![i].status = TransactionFailed]$
       $\land\ \text{UNCHANGED}\ \langle configuration, history\rangle$
    If the *transcation* is in the Applying state, roll back the Configuration for
    each target and Complete the transaction.
    $\lor\ \land\ transaction[i].status = TransactionApplying$
       Target configurations are rolled back by reverting to the source paths/values
       stored in the transaction when it was applied.
       $\land\ configuration' = [$
           $t \in \text{DOMAIN}\ Target \mapsto$
             $\text{IF}\ t \in \text{DOMAIN}\ transaction[transaction[i].rollback].changes\ \text{THEN}$
               $\text{LET}\ adds \triangleq \{p \in \text{DOMAIN}\ transaction[transaction[i].rollback].changes[t]:$
                                 $\land\ p \notin \text{DOMAIN}\ transaction[transaction[i].rollback].sources[t]$
                                 $\land\ \neg transaction[transaction[i].rollback].changes[t][p].delete\}$
                   $updates \triangleq \{p \in \text{DOMAIN}\ transaction[transaction[i].rollback].changes[t]:$
                                 $\land\ p \in \text{DOMAIN}\ transaction[transaction[i].rollback].sources[t]$
                                 $\land\ \neg transaction[transaction[i].rollback].changes[t][p].delete\}$
                   $removes \triangleq \{p \in \text{DOMAIN}\ transaction[transaction[i].rollback].changes[t]:$
                                 $\land\ p \in \text{DOMAIN}\ transaction[transaction[i].rollback].sources[t]$
                                 $\land\ transaction[transaction[i].rollback].changes[t][p].delete\}$
                   $changes \triangleq adds \cup updates \cup removes$
                   $unchanges \triangleq \text{DOMAIN}\ configuration[t].paths \setminus changes$
               $\text{IN}$
                 $[configuration[t]\ \text{EXCEPT}$
                   $!.paths = [p \in unchanges \mapsto configuration[t].paths[p]]$
                               $@@\ [p \in updates \cup removes \mapsto$
                                   $transaction[transaction[i].rollback].sources[t][p]],$
                   $!.txIndex = transaction[i].index,$
                   $!.status\ \ = ConfigurationPending]$
             $\text{ELSE}$
               $configuration[t]]$
       $\land\ history' = [r \in \text{DOMAIN}\ Target \mapsto Append(history[r], configuration'[r])]$
       $\land\ transaction' = [transaction\ \text{EXCEPT}\ ![i].status = TransactionComplete]$

Reconcile the transaction log
$ReconcileTransaction(n, i) \triangleq$
  $\land\ \lor\ \land\ transaction[i].type = TransactionChange$
             $\land\ ReconcileChange(n, i)$
        $\lor\ \land\ transaction[i].type = TransactionRollback$
             $\land\ ReconcileRollback(n, i)$
  $\land\ \text{UNCHANGED}\ \langle master, target\rangle$
```

```
This section models the Configuration reconciler.
$ReconcileConfiguration(n, c) \triangleq$
    $\land \lor \land configuration[c].status = ConfigurationPending$
            $\land master[configuration[c].target].master \neq Nil$
              If the configuration is marked ConfigurationPending and mastership
              has changed (indicated by an increased mastership term), mark the
              configuration ConfigurationInitializing to force full re-synchronization.
            $\land \lor \land master[configuration[c].target].term > configuration[c].term$
                  $\land configuration' = [configuration$ EXCEPT $![c].status = ConfigurationInitializing,$
                                                              $![c].term \ \ = master[configuration[c].target].term]$
                  $\land history' = [history$ EXCEPT $![c] = Append(history[c], configuration'[c])]$
              If the configuration is marked ConfigurationPending and the values have
              changed (determined by comparing the transaction index to the last sync
              index), mark the configuration ConfigurationUpdating to push the changes
              to the target.
               $\lor \land master[configuration[c].target].term = configuration[c].term$
                  $\land configuration[c].syncIndex < configuration[c].txIndex$
                  $\land configuration' = [configuration$ EXCEPT $![c].status = ConfigurationUpdating]$
                  $\land history' = [history$ EXCEPT $![c] = Append(history[c], configuration'[c])]$
            $\land$ UNCHANGED $\langle target \rangle$
         $\lor \land configuration[c].status = ConfigurationInitializing$
            $\land master[configuration[c].target].master = n$
             Merge the configuration paths with the target paths, removing paths
             that have been marked deleted
            $\land$ LET $deletePaths \triangleq \{p \in$ DOMAIN $configuration[c].paths : configuration[c].paths[p].deleted\}$
                    $configPaths \triangleq$ DOMAIN $configuration[c].paths \setminus deletePaths$
                    $targetPaths \triangleq$ DOMAIN $target[configuration[c].target] \setminus deletePaths$
              IN
                 $\land target' = [target$ EXCEPT $![configuration[c].target] =$
                       $[p \in configPaths \mapsto [value \mapsto configuration[c].paths[p]]]$
                          $@@ [p \in targetPaths \mapsto target[configuration[c].target][p]]]$
                 Set the configuration's status to Complete
            $\land configuration' = [configuration$ EXCEPT $![c].status \quad = ConfigurationComplete,$
                                                      $![c].syncIndex = configuration[c].txIndex]$
            $\land history' = [history$ EXCEPT $![c] = Append(history[c], configuration'[c])]$
         If the configuration is marked ConfigurationUpdating, we only need to
         push paths that have changed since the target was initialized or last
         updated by the controller. The set of changes made since the last
         synchronization are identified by comparing the index of each path-value
         to the last synchronization index, syncIndex
         $\lor \land configuration[c].status = ConfigurationUpdating$
            $\land master[configuration[c].target].master = n$
             Compute the set of updated and deleted paths by comparing
```

```
  their indexes to the target s last sync index.
  $\land$ LET $updatePaths \triangleq \{p \in$ DOMAIN $configuration[c].paths :$
                      $configuration[c].paths[p].index > configuration[c].syncIndex\}$
          $deletePaths \triangleq \{p \in updatePaths : configuration[c].paths[p].deleted\}$
          $configPaths \triangleq updatePaths \setminus deletePaths$
          $targetPaths \triangleq$ DOMAIN $target[configuration[c].target] \setminus deletePaths$
    IN
        Update the target paths by adding/updating paths that have changed and
        removing paths that have been deleted since the last sync.
        $\land target' = [target$ EXCEPT $![configuration[c].target] =$
              $[p \in configPaths \mapsto configuration[c].paths[p]]$
                 $@@ [p \in targetPaths \mapsto target[configuration[c].target][p]]]$
  $\land configuration' = [configuration$ EXCEPT $![c].status = ConfigurationComplete,$
                                              $![c].syncIndex = configuration[c].txIndex]$
  $\land history' = [history$ EXCEPT $![c] = Append(history[c], configuration'[c])]$
 If the configuration is not already ConfigurationPending and mastership
 has been lost revert it. This can occur when the connection to the
 target has been lost and the mastership is no longer valid.
 TODO: We still need to model mastership changes
 $\lor \land configuration[c].status \neq ConfigurationPending$
   $\land master[configuration[c].target].master = Nil$
   $\land configuration' = [configuration$ EXCEPT $![c].status = ConfigurationPending]$
   $\land history' = [history$ EXCEPT $![c] = Append(history[c], configuration'[c])]$
   $\land$ UNCHANGED $\langle target \rangle$
$\land$ UNCHANGED $\langle transaction, master \rangle$
```

Init and next state predicates

```
$Init \triangleq$
   $\land transaction = \langle\rangle$
   $\land configuration = [t \in$ DOMAIN $Target \mapsto$
                          $[target \mapsto t,$
                           $paths \mapsto$
                              $[path \in \{\} \mapsto$
                                  $[path \mapsto path,$
                                   $value \mapsto Nil,$
                                   $index \mapsto 0,$
                                   $deleted \mapsto$ FALSE$]],$
                           $txIndex \mapsto 0,$
                           $syncIndex \mapsto 0,$
                           $term \mapsto 0,$
                           $status \mapsto ConfigurationPending]]$
   $\land target = [t \in$ DOMAIN $Target \mapsto$
                  $[path \in \{\} \mapsto$
```

```
                    $[value \mapsto Nil]]]$
    $\land\ master = [t \in \text{DOMAIN } Target \mapsto [master \mapsto Nil,\ term \mapsto 0]]$
    $\land\ history = [t \in \text{DOMAIN } Target \mapsto \langle\rangle]$

$Next \triangleq$
    $\lor\ \exists\, c \in ValidChanges :$
        $Change(c)$
    $\lor\ \exists\, t \in \text{DOMAIN } transaction :$
        $Rollback(t)$
    $\lor\ \exists\, n \in Node :$
        $\exists\, t \in \text{DOMAIN } Target :$
          $SetMaster(n,\ t)$
    $\lor\ \exists\, t \in \text{DOMAIN } Target :$
        $UnsetMaster(t)$
    $\lor\ \exists\, n \in Node :$
        $\exists\, t \in \text{DOMAIN } transaction :$
            $ReconcileTransaction(n,\ t)$
    $\lor\ \exists\, n \in Node :$
        $\exists\, c \in \text{DOMAIN } configuration :$
            $ReconcileConfiguration(n,\ c)$

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

$Inv \triangleq$
    $\land\ \forall\, a,\ b \in \text{DOMAIN } transaction :$
        $transaction[a].index > transaction[b].index \Rightarrow$
            $(transaction[a].status \in \{TransactionComplete,\ TransactionFailed\} \Rightarrow$
                $transaction[b].status \in \{TransactionComplete,\ TransactionFailed\})$
    $\land\ \forall\, t \in \text{DOMAIN } Target :$
        $\forall\, c \in \text{DOMAIN } history[t] :$
            $\land\ configuration[t].txIndex \geq history[t][c].txIndex$
            $\land\ configuration[t].syncIndex \geq history[t][c].syncIndex$
```

* Modification History
* Last modified *Tue Jan* 18 16:30:29 *PST* 2022 by *jordanhalterman*
* Created *Wed Sep* 22 13:22:32 *PDT* 2021 by *jordanhalterman*