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

INSTANCE FiniteSets

INSTANCE Sequences

INSTANCE TLC

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

 An empty constant
CONSTANT Nil

 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 ∈ STRING

ASSUME TransactionPending ∈ STRING
ASSUME TransactionValidating ∈ STRING
ASSUME TransactionApplying  ∈ STRING
```

$\textsc{assume}\ \mathit{TransactionComplete} \in \textsc{string}$
$\textsc{assume}\ \mathit{TransactionFailed} \in \textsc{string}$

$\textsc{assume}\ \mathit{ConfigurationPending} \in \textsc{string}$
$\textsc{assume}\ \mathit{ConfigurationInitializing} \in \textsc{string}$
$\textsc{assume}\ \mathit{ConfigurationUpdating} \in \textsc{string}$
$\textsc{assume}\ \mathit{ConfigurationComplete} \in \textsc{string}$
$\textsc{assume}\ \mathit{ConfigurationFailed} \in \textsc{string}$

$$\begin{array}{l}
\textsc{assume}\ \land \mathit{IsFiniteSet}(\mathit{Node}) \\
\qquad\quad\ \land \forall n \in \mathit{Node} : \\
\qquad\qquad\quad \land n \notin \textsc{domain}\ \mathit{Target} \\
\qquad\qquad\quad \land n \in \textsc{string}
\end{array}$$

$$\begin{array}{l}
\textsc{assume}\ \land \forall t \in \textsc{domain}\ \mathit{Target} : \\
\qquad\qquad\quad \land t \notin \mathit{Node} \\
\qquad\qquad\quad \land t \in \textsc{string} \\
\qquad\qquad\quad \land \forall p \in \textsc{domain}\ \mathit{Target}[t] : \\
\qquad\qquad\qquad \mathit{IsFiniteSet}(\mathit{Target}[t][p])
\end{array}$$

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

TYPE Transaction $\triangleq$ [
  $\mathit{id} ::= \mathit{id} \in \textsc{string},$
  $\mathit{index} ::= \mathit{index} \in \mathit{Nat},$
  revision ::= revision $\in \mathit{Nat},$
  $\mathit{atomic} ::= \mathit{atomic} \in \textsc{boolean}$,
  $\mathit{sync} ::= \mathit{sync} \in \textsc{boolean}$,
  $\mathit{changes} ::= [\ \mathit{target} \in \textsc{subset}\ (\textsc{domain}\ \mathit{Target}) \mapsto [$
    $\mathit{path} \in \textsc{subset}\ (\textsc{domain}\ \mathit{Target}[\mathit{target}]) \mapsto [$
      $\mathit{value} ::= \mathit{value} \in \textsc{string},$
      $\mathit{delete} ::= \mathit{delete} \in \textsc{boolean}\ ]]],$
  $\mathit{status} ::= \mathit{status} \in \mathit{TransactionStatus}]$

TYPE $\mathit{ConfigurationStatus} ::= \mathit{status} \in$
  $\{\mathit{ConfigurationPending},$
  $\mathit{ConfigurationInitializing},$
  $\mathit{ConfigurationUpdating},$
  $\mathit{ConfigurationComplete},$
  $\mathit{ConfigurationFailed}\}$

TYPE Configuration $\triangleq$ [
  $\mathit{id} ::= \mathit{id} \in \textsc{string},$
  revision ::= revision $\in \mathit{Nat},$

```
target ::= target ∈ STRING,
paths ::=  [ path ∈ SUBSET  (DOMAIN Target[target]) ↦ [
     value ::= value ∈ STRING,
     index ::= index ∈ Nat,
     deleted ::= delete ∈ BOOLEAN ]],
txIndex         ::= txIndex ∈ Nat,
syncIndex       ::= syncIndex ∈ Nat,
mastershipTerm ::= mastershipTerm ∈ Nat,
status          ::= status ∈ ConfigurationStatus]
```

A sequence of transactions

Each transactions contains a record of 'changes' for a set of targets

```
VARIABLE transactions
```

A record of target configurations

Each configuration represents the desired state of the target

```
VARIABLE configurations
```

A record of target states

```
VARIABLE targets
```

A record of target masters

```
VARIABLE masters

vars ≜ ⟨transactions, configurations, targets⟩
```

```
ChangeMaster(n, t) ≜
    ∧ masters[t].master ≠ n
    ∧ masters' = [masters EXCEPT ![t].term   = masters[t].term + 1,
                                 ![t].master = n]
    ∧ UNCHANGED ⟨transactions, configurations⟩
```

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

This crazy thing returns the set of all possible sets of valid changes

```
ValidChanges ≜
   LET allPaths ≜ UNION {(DOMAIN Target[t]) : t ∈ DOMAIN Target}
       allValues ≜ UNION {UNION {Target[t][p] : p ∈ DOMAIN Target[t]} : t ∈ DOMAIN Target}
   IN
      {targetPathValues ∈ SUBSET (Target × allPaths × allValues × BOOLEAN ) :
         ∧ ∀ target ∈ DOMAIN Target :
            LET targetIndexes ≜ {i ∈ 1 . . Len(targetPathValues) : ∧ targetPathValues[i][1] = target}
            IN  ∨ Cardinality(targetIndexes) = 0
                ∨ ∧ Cardinality(targetIndexes) = 1
                  ∧ LET targetPathValue ≜ targetPathValues[CHOOSE index ∈ targetIndexes : TRUE]
                        targetPath      ≜ targetPathValue[2]
                $targetValue \triangleq targetPathValue[3]$
            IN
                $\land\ targetPath \setminus (\text{DOMAIN } Target[target]) = \{\}$
                $\land\ targetValue \in Target[target][targetPath]\}$

 Add a set of changes to the transaction log
$Change \triangleq$
    $\land\ \exists\ changes \in ValidChanges :$
        $\land\ transactions' = Append(transactions, [index \mapsto Len(transactions) + 1,$
                                                      $atomic \mapsto \text{FALSE},$
                                                      $sync \mapsto \text{FALSE},$
                                                      $changes \mapsto changes,$
                                                      $status \mapsto TransactionPending])$
    $\land\ \text{UNCHANGED } \langle configurations, targets \rangle$
```

```
This section models the Transaction log reconciler.

 Reconcile the transaction log
$ReconcileTransaction(n, t) \triangleq$
    LET $tx \triangleq transactions[t]$
    IN      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.
        $\land$    $\lor\ \land\ tx.status = TransactionPending$
                  $\land\ \lor\ \land\ tx.index - 1 \in \text{DOMAIN } transactions$
                            $\land\ transactions[tx.index - 1].status \in \{TransactionComplete, TransactionFailed\}$
                      $\lor\ tx.index - 1 \notin \text{DOMAIN } transactions$
                  $\land\ transactions' = [transactions \text{ EXCEPT } ![tx.index].status = TransactionValidating]$
                  $\land\ \text{UNCHANGED } \langle configurations \rangle$
            If the transaction is in the Validating state, compute and validate the
            Configuration for each target.
            $\lor\ \land\ tx.status = TransactionValidating$
                 If validation fails any target, mark the transaction Failed.
                 If validation is successful, proceed to Applying.
                  $\land\ \exists\ valid \in \text{BOOLEAN} :$
                      $\lor\ \land\ valid$
                          $\land\ transactions' = [transactions \text{ EXCEPT } ![tx.index].status = TransactionApplying]$
                      $\lor\ \land\ \neg valid$
                          $\land\ transactions' = [transactions \text{ EXCEPT } ![tx.index].status = TransactionFailed]$
                  $\land\ \text{UNCHANGED } \langle configurations \rangle$
            If the transaction is in the Applying state, update the Configuration for each
            target and Complete the transaction.
            $\lor\ \land\ tx.status = TransactionApplying$
                  $\land\ \lor\ \land\ tx.atomic$
                              TODO: Apply atomic transactions here
```

```
                $\land\ transactions' = [transactions\ \text{EXCEPT}\ ![tx.index].status = TransactionComplete]$
                $\land\ \text{UNCHANGED}\ \langle configurations \rangle$
              $\lor\ \land\ \neg tx.atomic$
                Add the transaction index to each updated path
                $\land\ configurations' = [$
                    $r \in \text{DOMAIN}\ Target \mapsto [$
                      $configurations[r]\ \text{EXCEPT}$
                        $!.paths = [path \in \text{DOMAIN}\ tx.changes \mapsto$
                          $tx.changes[path]\ @@\ [index \mapsto tx.index]]\ @@\ configurations[t].paths,$
                        $!.txIndex = tx.index,$
                        $!.status\ \ = ConfigurationPending]]$
                $\land\ transactions' = [transactions\ \text{EXCEPT}\ ![tx.index].status = TransactionComplete]$
      $\land\ \ \text{UNCHANGED}\ \langle targets \rangle$
```

```
This section models the Configuration reconciler.
$ReconcileConfiguration(n, c) \triangleq$
  $\land\ \lor\ \land\ configurations[c].status = ConfigurationPending$
              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.
        $\land\ \lor\ \land\ masters[configurations[c].target].term > configurations[c].mastershipTerm$
                $\land\ configurations' = [configurations\ \text{EXCEPT}\ ![c].status = ConfigurationInitializing,$
                                                          $![c].mastershipTerm = masters[configurations[c].targ$
              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.
              $\lor\ \land\ configurations[c].syncIndex < configurations[c].txIndex$
                $\land\ configurations' = [configurations\ \text{EXCEPT}\ ![c].status = ConfigurationUpdating]$
    $\lor\ \land\ configurations[c].status = ConfigurationInitializing$
        $\land\ masters[configurations[c].target].master = n$
        Merge the configuration paths with the target paths, removing paths
        that have been marked deleted
        $\land\ \text{LET}\ deletePaths \triangleq \{p \in \text{DOMAIN}\ configurations[c].paths : configurations[c].paths[p].deleted\}$
          $\text{IN}$
            $\land\ targets' = [targets\ \text{EXCEPT}\ ![configurations[c].target] =$
                $[p \in (\text{DOMAIN}\ c.paths \setminus deletePaths) \mapsto [value \mapsto configurations[c].paths[p]]]\ @@$
                $[p \in (\text{DOMAIN}\ targets[configurations[c].target] \setminus deletePaths) \mapsto targets[configurations[c].targ$
            Set the configuration's status to Complete
            $\land\ configurations' = [configurations\ \text{EXCEPT}\ ![c].status\ \ = ConfigurationComplete,$
                                                      $![c].syncIndex = configurations[c].txIndex]$
      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
      ∨ ∧ configurations[c].status = ConfigurationUpdating
        ∧ masters[configurations[c].target].master = n
         Compute the set of updated and deleted paths by comparing
         their indexes to the target s last sync index.
        ∧ LET updatedPaths ≜ {p ∈ DOMAIN configurations[c].paths : configurations[c].paths[p].index > conf
              deletedPaths ≜ {p ∈ updatedPaths : configurations[c].paths[p].deleted}
          IN
             Update the target paths by adding/updating paths that have changed and
             removing paths that have been deleted since the last sync.
            ∧ targets' = [targets EXCEPT ![configurations[c].target] =
                  [p ∈ updatedPaths \ deletedPaths ↦ configurations[c].paths[p]] @@
                  [p ∈ DOMAIN targets[configurations[c].target] \ deletedPaths ↦ targets[configurations[c].targ
        ∧ configurations' = [configurations EXCEPT ![c].status    = ConfigurationComplete,
                                                   ![c].syncIndex = configurations[c].txIndex]
        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
      ∨ ∧ c.status ≠ ConfigurationPending
        ∧ masters[configurations[c].target].master = Nil
        ∧ configurations' = [configurations EXCEPT ![c].status = ConfigurationPending]
    ∧ UNCHANGED ⟨transactions⟩
```

```
Init and next state predicates
Init ≜
    ∧ transactions = ⟨⟩
    ∧ configurations = [t ∈ Target ↦
                           [id     ↦ t,
                            config ↦
                               [path ∈ {} ↦
                                   [path    ↦ path,
                                    value   ↦ Nil,
                                    index   ↦ 0,
                                    deleted ↦ FALSE]]]]
    ∧ targets  = [t ∈ Target ↦
                     [path ∈ {} ↦
                         [value ↦ Nil]]]
    ∧ masters = [t ∈ Target ↦ [master ↦ Nil, term ↦ 0]]

Next ≜
    ∨ Change
    $\lor \exists n \in Node :$
        $\lor \exists t \in \textsc{domain}\ Target :$
            $ChangeMaster(n, t)$
        $\lor \exists t \in \textsc{domain}\ transactions :$
            $ReconcileTransaction(n, t)$
        $\lor \exists t \in \textsc{domain}\ configurations :$
            $ReconcileConfiguration(n, t)$

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

* Modification History
* Last modified *Mon Jan* 17 23:09:38 *PST* 2022 by *jordanhalterman*
* Created *Wed Sep* 22 13:22:32 *PDT* 2021 by *jordanhalterman*