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

CONSTANT *Target*

ASSUME *Nil* $\in$ STRING

ASSUME *TransactionPending* $\in$ STRING
ASSUME *TransactionValidating* $\in$ STRING
ASSUME *TransactionApplying* $\in$ STRING

```
ASSUME TransactionComplete ∈ STRING
ASSUME TransactionFailed ∈ STRING

ASSUME ConfigurationPending ∈ STRING
ASSUME ConfigurationInitializing ∈ STRING
ASSUME ConfigurationUpdating ∈ STRING
ASSUME ConfigurationComplete ∈ STRING
ASSUME ConfigurationFailed ∈ STRING

ASSUME ∧ IsFiniteSet(Node)
       ∧ ∀ n ∈ Node :
           ∧ n ∉ DOMAIN Target
           ∧ n ∈ STRING

ASSUME ∧ ∀ t ∈ DOMAIN Target :
           ∧ t ∉ Node
           ∧ t ∈ STRING
           ∧ ∀ p ∈ DOMAIN Target[t] :
               IsFiniteSet( Target[t][p])
```

```
TYPE TransactionStatus ::= status ∈
  {TransactionPending,
   TransactionValidating,
   TransactionApplying,
   TransactionComplete,
   TransactionFailed}

TYPE Transaction ≜ [
  id       ::= id ∈ STRING,
  index    ::= index ∈ Nat,
  revision ::= revision ∈ Nat,
  atomic   ::= atomic ∈ BOOLEAN ,
  sync     ::= sync ∈ BOOLEAN ,
  changes  ::= [ target ∈ SUBSET (DOMAIN Target) ↦ [
      path ∈ SUBSET (DOMAIN Target[target]) ↦ [
        value  ::= value ∈ STRING,
        delete ::= delete ∈ BOOLEAN ]]],
  status   ::= status ∈ TransactionStatus]

TYPE ConfigurationStatus ::= status ∈
  {ConfigurationPending,
   ConfigurationInitializing,
   ConfigurationUpdating,
   ConfigurationComplete,
   ConfigurationFailed}

TYPE Configuration ≜ [
  id       ::= id ∈ STRING,
  revision ::= revision ∈ Nat,
```

$target ::= target \in \text{STRING}$,
$paths ::= [\ path \in \text{SUBSET}\ (\text{DOMAIN}\ Target[target]) \mapsto [$
$value ::= value \in \text{STRING}$,
$index ::= index \in Nat$,
$deleted ::= delete \in \text{BOOLEAN}\ ]]$,
$txIndex ::= txIndex \in Nat$,
$syncIndex ::= syncIndex \in Nat$,
$term ::= term \in Nat$,
$status ::= status \in ConfigurationStatus]$

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*

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

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

This section models the northbound *API* for the configuration service.
$ValidValues(t, p) \triangleq$
$\text{UNION}\ \{\{[value \mapsto v, delete \mapsto \text{FALSE}] : v \in Target[t][p]\}, \{[value \mapsto Nil, delete \mapsto \text{TRUE}]\}\}$

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

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

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

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

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

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

ValidChanges ≜
    LET changeSets ≜ {s ∈ SUBSET ValidTargets :
                          ∀ t ∈ DOMAIN Target   :
                            ∀ p ∈ DOMAIN Target[t] :
                              Cardinality({v ∈ s : v.target = t ∧ v.path = p}) ≤ 1}
    IN
        {ValidChange(s) : s ∈ changeSets}


NextIndex ≜
    IF Len(transaction) = 0 THEN
        1
     ELSE
        LET i ≜ CHOOSE i ∈ DOMAIN transaction :
              ∀ j ∈ DOMAIN transaction :
                 transaction[j].index ≤ transaction[i].index
        IN   transaction[i].index + 1

 Add a set of changes to the transaction log
Change ≜
    ∧ ∃ changes ∈ ValidChanges :
        ∧ transaction′ = Append(transaction, [index   ↦ NextIndex,
                                              atomic  ↦ FALSE,
                                              sync    ↦ FALSE,
                                              changes ↦ changes,
                                              status  ↦ TransactionPending])
    ∧ UNCHANGED ⟨configuration, master, target, history⟩
```

This section models the Transaction log reconciler.

```
 Reconcile the transaction log
ReconcileTransaction(n, t) ≜
        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.
    ∧  ∨  ∧ transaction[t].status = TransactionPending
```

```
    ∧ ∨ ∧ transaction[t].index − 1 ∈ DOMAIN transaction
        ∧ transaction[transaction[t].index − 1].status ∈ {TransactionComplete, TransactionFailed}
      ∨ transaction[t].index − 1 ∉ DOMAIN transaction
    ∧ transaction′ = [transaction EXCEPT ![t].status = TransactionValidating]
    ∧ UNCHANGED ⟨configuration, history⟩
  If the transaction is in the Validating state, compute and validate the
  Configuration for each target.
∨ ∧ transaction[t].status = TransactionValidating
    If validation fails any target, mark the transaction Failed.
    If validation is successful, proceed to Applying.
    ∧ ∃ valid ∈ BOOLEAN :
        ∨ ∧ valid
          ∧ transaction′ = [transaction EXCEPT ![t].status = TransactionApplying]
        ∨ ∧ ¬valid
          ∧ transaction′ = [transaction EXCEPT ![t].status = TransactionFailed]
    ∧ UNCHANGED ⟨configuration, history⟩
  If the transaction is in the Applying state, update the Configuration for each
  target and Complete the transaction.
∨ ∧ transaction[t].status = TransactionApplying
  ∧ ∨ ∧ transaction[t].atomic
      TODO: Apply atomic transactions here
      ∧ transaction′ = [transaction EXCEPT ![t].status = TransactionComplete]
      ∧ UNCHANGED ⟨configuration, history⟩
    ∨ ∧ ¬transaction[t].atomic
      Add the transaction index to each updated path
      ∧ configuration′ = [
          r ∈ DOMAIN Target ↦
            IF r ∈ DOMAIN transaction[t].changes THEN
              [configuration[r] EXCEPT
                !.paths = [p ∈ DOMAIN transaction[t].changes[r] ↦
                             [index   ↦ transaction[t].index,
                              value   ↦ transaction[t].changes[r][p].value,
                              deleted ↦ transaction[t].changes[r][p].delete]]
                          @@ configuration[r].paths,
                !.txIndex = transaction[t].index,
                !.status  = ConfigurationPending]
            ELSE
              configuration[r]]
      ∧ history′ = [r ∈ DOMAIN Target ↦ Append(history[r], configuration′[r])]
      ∧ transaction′ = [transaction EXCEPT ![t].status = TransactionComplete]
∧ UNCHANGED ⟨master, target⟩
```

This section models the Configuration reconciler.

```
ReconcileConfiguration(n, c) ≜
  ∧ ∨ ∧ configuration[c].status = ConfigurationPending
      ∧ master[configuration[c].target].master ≠ 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.
      ∧ ∨ ∧ master[configuration[c].target].term > configuration[c].term
          ∧ configuration' = [configuration EXCEPT ![c].status = ConfigurationInitializing,
                                                   ![c].term   = master[configuration[c].target].term]
          ∧ 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.
        ∨ ∧ master[configuration[c].target].term = configuration[c].term
          ∧ configuration[c].syncIndex < configuration[c].txIndex
          ∧ configuration' = [configuration EXCEPT ![c].status = ConfigurationUpdating]
          ∧ history' = [history EXCEPT ![c] = Append(history[c], configuration'[c])]
      ∧ UNCHANGED ⟨target⟩
    ∨ ∧ configuration[c].status = ConfigurationInitializing
      ∧ master[configuration[c].target].master = n
      Merge the configuration paths with the target paths, removing paths
      that have been marked deleted
      ∧ LET deletePaths ≜ {p ∈ DOMAIN configuration[c].paths : configuration[c].paths[p].deleted}
            configPaths ≜ DOMAIN configuration[c].paths \ deletePaths
            targetPaths ≜ DOMAIN target[configuration[c].target] \ deletePaths
        IN
          ∧ target' = [target EXCEPT ![configuration[c].target] =
                [p ∈ configPaths ↦ [value ↦ configuration[c].paths[p]]]
                  @@ [p ∈ targetPaths ↦ target[configuration[c].target][p]]]
            Set the configuration's status to Complete
      ∧ configuration' = [configuration EXCEPT ![c].status    = ConfigurationComplete,
                                               ![c].syncIndex = configuration[c].txIndex]
      ∧ 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
    ∨ ∧ configuration[c].status = ConfigurationUpdating
      ∧ 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.
      ∧ LET updatePaths ≜ {p ∈ DOMAIN configuration[c].paths :
                              configuration[c].paths[p].index > configuration[c].syncIndex}
```

$$deletePaths \triangleq \{p \in updatePaths : configuration[c].paths[p].deleted\}$$
$$configPaths \triangleq updatePaths \setminus deletePaths$$
$$targetPaths \triangleq \text{DOMAIN } target[configuration[c].target] \setminus deletePaths$$

IN

Update the target paths by adding/updating paths that have changed and removing paths that have been deleted since the last *sync*.

$$\land\ target' = [target \text{ EXCEPT } ![configuration[c].target] =$$
$$[p \in configPaths \mapsto configuration[c].paths[p]]$$
$$@@\ [p \in targetPaths \mapsto target[configuration[c].target][p]]]$$
$$\land\ configuration' = [configuration \text{ EXCEPT } ![c].status = ConfigurationComplete,$$
$$![c].syncIndex = configuration[c].txIndex]$$
$$\land\ history' = [history \text{ EXCEPT } ![c] = Append(history[c], configuration'[c])]$$

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

$$\lor\ \land\ configuration[c].status \neq ConfigurationPending$$
$$\land\ master[configuration[c].target].master = Nil$$
$$\land\ configuration' = [configuration \text{ EXCEPT } ![c].status = ConfigurationPending]$$
$$\land\ history' = [history \text{ EXCEPT } ![c] = Append(history[c], configuration'[c])]$$
$$\land\ \text{UNCHANGED } \langle target \rangle$$
$$\land\ \text{UNCHANGED } \langle transaction, master \rangle$$

Init and next state predicates

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

$$
\begin{array}{l}
Next \triangleq \\
\quad \lor Change \\
\quad \lor \exists\, n \in Node : \\
\qquad \lor \exists\, t \in \text{DOMAIN } Target : \\
\qquad\quad ChangeMaster(n,\, t) \\
\qquad \lor \exists\, t \in \text{DOMAIN } transaction : \\
\qquad\quad ReconcileTransaction(n,\, t) \\
\qquad \lor \exists\, t \in \text{DOMAIN } configuration : \\
\qquad\quad ReconcileConfiguration(n,\, t)
\end{array}
$$

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

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

* Modification History
* Last modified *Tue Jan* 18 12:53:43 *PST* 2022 by *jordanhalterman*
* Created *Wed Sep* 22 13:22:32 *PDT* 2021 by *jordanhalterman*