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 *targets*

A record of target masters
VARIABLE *masters*

VARIABLE *history*

$vars \triangleq \langle transaction, configuration, masters, targets, history \rangle$

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

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 ∧ PrintT(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 ∧ PrintT(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, masters, targets, 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 ↦ [
                  configuration[r] EXCEPT
                    !.paths = [path ∈ DOMAIN transaction[t].changes ↦
                      transaction[t].changes[path] @@ [index ↦ transaction[t].index]]
                        @@ configuration[t].paths,
                    !.txIndex = transaction[t].index,
                    !.status  = ConfigurationPending]]
            ∧ history′ = [r ∈ DOMAIN Target ↦ Append(history[r], configuration′[r])]
            ∧ transaction′ = [transaction EXCEPT ![t].status = TransactionComplete]
    ∧ UNCHANGED ⟨masters, targets⟩

⊢

This section models the Configuration reconciler.
ReconcileConfiguration(n, c) ≜
    ∧ ∨ ∧ configuration[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.
```

$\land \lor \land masters[configuration[c].target].term > configuration[c].term$
$\land configuration' = [configuration \text{ EXCEPT } ![c].status = ConfigurationInitializing,$
$![c].term = masters[configuration[c].target].term]$
$\land history' = [history \text{ EXCEPT } ![c] = Append(history[c], configuration'[c])]$
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.
$\lor \land configuration[c].syncIndex < configuration[c].txIndex$
$\land configuration' = [configuration \text{ EXCEPT } ![c].status = ConfigurationUpdating]$
$\land history' = [history \text{ EXCEPT } ![c] = Append(history[c], configuration'[c])]$
$\land \text{UNCHANGED } \langle targets \rangle$
$\lor \land configuration[c].status = ConfigurationInitializing$
$\land masters[configuration[c].target].master = n$
Merge the configuration paths with the target paths, removing paths that have been marked deleted
$\land \text{LET } deletePaths \triangleq \{p \in \text{DOMAIN } configuration[c].paths : configuration[c].paths[p].deleted\}$
$configPaths \triangleq \text{DOMAIN } configuration[c].paths \setminus deletePaths$
$targetPaths \triangleq \text{DOMAIN } targets[configuration[c].target] \setminus deletePaths$
IN
$\land targets' = [targets \text{ EXCEPT } ![configuration[c].target] =$
$[p \in configPaths \mapsto [value \mapsto configuration[c].paths[p]]]$
$@@ [p \in targetPaths \mapsto targets[configuration[c].target][p]]]$
Set the configuration's status to *Complete*
$\land configuration' = [configuration \text{ EXCEPT } ![c].status = ConfigurationComplete,$
$![c].syncIndex = configuration[c].txIndex]$
$\land history' = [history \text{ EXCEPT } ![c] = Append(history[c], configuration'[c])]$
If the configuration is 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*
$\lor \land configuration[c].status = ConfigurationUpdating$
$\land masters[configuration[c].target].master = n$
Compute the set of updated and deleted paths by comparing their indexes to the target s last *sync* index.
$\land \text{LET } updatePaths \triangleq \{p \in \text{DOMAIN } configuration[c].paths :$
$configuration[c].paths[p].index > configuration[c].syncIndex\}$
$deletePaths \triangleq \{p \in updatePaths : configuration[c].paths[p].deleted\}$
$configPaths \triangleq updatePaths \setminus deletePaths$
$targetPaths \triangleq \text{DOMAIN } targets[configuration[c].target] \setminus deletePaths$
IN
Update the target paths by adding/updating paths that have changed and removing paths that have been deleted since the last *sync*.
$\land targets' = [targets \text{ EXCEPT } ![configuration[c].target] =$

$$
\begin{array}{l}
\qquad [p \in configPaths \mapsto configuration[c].paths[p]] \\
\qquad\quad @@ [p \in targetPaths \mapsto targets[configuration[c].target][p]]] \\
\quad \land configuration' = [configuration \text{ EXCEPT } ![c].status \quad = ConfigurationComplete, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad ![c].syncIndex = configuration[c].txIndex] \\
\quad \land history' = [history \text{ EXCEPT } ![c] = Append(history[c],\ configuration'[c])] \\
\quad \text{If the configuration is not already } \textit{ConfigurationPending} \text{ and mastership} \\
\quad \text{has been lost revert it. This can occur when the connection to the} \\
\quad \text{target has been lost and the mastership is no longer valid.} \\
\quad \textit{TODO}\text{: We still need to model mastership changes} \\
\lor \land configuration[c].status \neq ConfigurationPending \\
\quad \land masters[configuration[c].target].master = Nil \\
\quad \land configuration' = [configuration \text{ EXCEPT } ![c].status = ConfigurationPending] \\
\quad \land history' = [history \text{ EXCEPT } ![c] = Append(history[c],\ configuration'[c])] \\
\quad \land \text{UNCHANGED } \langle targets \rangle \\
\land \text{UNCHANGED } \langle transaction,\ masters \rangle
\end{array}
$$

Init and next state predicates

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

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

$$\lor \exists\, t \in \text{DOMAIN } configuration :$$
$$\quad ReconcileConfiguration(n,\ t)$$

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

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

* Modification History
* Last modified *Tue Jan* 18 11:52:37 *PST* 2022 by *jordanhalterman*
* Created *Wed Sep* 22 13:22:32 *PDT* 2021 by *jordanhalterman*