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*

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

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

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

This crazy thing returns the set of all possible sets of valid changes
$ValidChanges \triangleq$
$\text{LET}\ allPaths \triangleq \text{UNION}\ \{(\text{DOMAIN}\ Target[t]) : t \in \text{DOMAIN}\ Target\}$
$allValues \triangleq \text{UNION}\ \{\text{UNION}\ \{Target[t][p] : p \in \text{DOMAIN}\ Target[t]\} : t \in \text{DOMAIN}\ Target\}$
IN
$\{targetPathValues \in \text{SUBSET}\ (Target \times allPaths \times allValues \times \text{BOOLEAN}\ ) :$
$\land \forall\, target \in \text{DOMAIN}\ Target :$
$\text{LET}\ targetIndexes \triangleq \{i \in 1\ ..\ Len(targetPathValues) : \land\ targetPathValues[i][1] = target\}$
IN $\lor\ Cardinality(targetIndexes) = 0$
$\lor\ \land\ Cardinality(targetIndexes) = 1$
$\land\ \text{LET}\ targetPathValue \triangleq targetPathValues[\text{CHOOSE}\ index \in targetIndexes : \text{TRUE}]$
$targetPath \triangleq targetPathValue[2]$

```
                    targetValue  ≜ targetPathValue[3]
            IN
                ∧ targetPath \ (DOMAIN Target[target]) = {}
                ∧ targetValue ∈ Target[target][targetPath]}

NextIndex ≜
  IF Len(transaction) = 0 THEN
    1
  ELSE
    (CHOOSE i ∈ DOMAIN transaction :
       ∀ j ∈ DOMAIN transaction :
         transaction[j].index < 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, targets⟩
```

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⟩
      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⟩
        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⟩
            ∨ ∧ ¬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]]
              ∧ transaction′ = [transaction EXCEPT ![t].status = TransactionComplete]
    ∧ UNCHANGED ⟨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.
            ∧ ∨ ∧ masters[configuration[c].target].term > configuration[c].term
                ∧ configuration′ = [configuration EXCEPT ![c].status = ConfigurationInitializing,
                                                         ![c].term   = masters[configuration[c].target].term]
               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.
              ∨ ∧ configuration[c].syncIndex < configuration[c].txIndex
                ∧ configuration′ = [configuration EXCEPT ![c].status = ConfigurationUpdating]
          ∨ ∧ configuration[c].status = ConfigurationInitializing
            ∧ masters[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 c.paths \ deletePaths
```

```
                targetPaths ≜ DOMAIN targets[configuration[c].target] \ deletePaths
            IN
                ∧ targets' = [targets EXCEPT ![configuration[c].target] =
                                [p ∈ configPaths ↦ [value ↦ configuration[c].paths[p]]]
                                  @@ [p ∈ targetPaths ↦ targets[configuration[c].target][p]]]
                Set the configuration's status to Complete
            ∧ configuration' = [configuration EXCEPT ![c].status    = ConfigurationComplete,
                                                     ![c].syncIndex = configuration[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
        ∨ ∧ configuration[c].status = ConfigurationUpdating
          ∧ 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.
          ∧ LET updatePaths ≜ {p ∈ DOMAIN configuration[c].paths :
                                  configuration[c].paths[p].index > configuration[c].syncIndex}
                deletePaths ≜ {p ∈ updatePaths : configuration[c].paths[p].deleted}
                configPaths ≜ updatePaths \ deletePaths
                targetPaths ≜ DOMAIN targets[configuration[c].target] \ deletePaths
            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 ![configuration[c].target] =
                              [p ∈ configPaths ↦ configuration[c].paths[p]]
                                @@ [p ∈ targetPaths ↦ targets[configuration[c].target][p]]]
          ∧ configuration' = [configuration EXCEPT ![c].status    = ConfigurationComplete,
                                                   ![c].syncIndex = configuration[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[configuration[c].target].master = Nil
          ∧ configuration' = [configuration EXCEPT ![c].status = ConfigurationPending]
    ∧ UNCHANGED ⟨transaction⟩

⊢

Init and next state predicates
Init ≜
    ∧ transaction = ⟨⟩
    ∧ configuration = [t ∈ Target ↦
```

$$
\begin{array}{l}
\qquad\qquad\qquad\qquad [id \quad\;\; \mapsto t, \\
\qquad\qquad\qquad\quad\; config \mapsto \\
\qquad\qquad\qquad\qquad\; [path \in \{\} \;\mapsto \\
\qquad\qquad\qquad\qquad\qquad [path \;\; \mapsto path, \\
\qquad\qquad\qquad\qquad\qquad\; value \;\; \mapsto Nil, \\
\qquad\qquad\qquad\qquad\qquad\; index \;\; \mapsto 0, \\
\qquad\qquad\qquad\qquad\qquad\; deleted \mapsto \text{FALSE}]]]] \\
\quad \land\, targets \;\; = [t \in Target \mapsto \\
\qquad\qquad\qquad\qquad [path \in \{\} \mapsto \\
\qquad\qquad\qquad\qquad\quad [value \mapsto Nil]]] \\
\quad \land\, masters = [t \in Target \mapsto [master \mapsto Nil,\; term \mapsto 0]]
\end{array}
$$

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

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

* Modification History
* Last modified *Tue Jan* 18 00:24:57 *PST* 2022 by *jordanhalterman*
* Created *Wed Sep* 22 13:22:32 *PDT* 2021 by *jordanhalterman*