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,
```

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

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*

$vars \triangleq \langle transactions,\ configurations,\ targets \rangle$

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

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 $\lor\ Cardinality(targetIndexes) = 0$
$\lor\ \land\ Cardinality(targetIndexes) = 1$
$\land\ \text{LET}\ targetPathValue \triangleq targetPathValues[\text{CHOOSE}\ index \in targetIndexes : \text{TRUE}]$
$targetPath \triangleq targetPathValue[2]$

$$\begin{aligned}
&\quad targetValue \triangleq targetPathValue[3]\\
&\text{IN}\\
&\quad \land\ targetPath \setminus (\text{DOMAIN } Target[target]) = \{\}\\
&\quad \land\ targetValue \in Target[target][targetPath]\}
\end{aligned}$$

Add a set of changes to the transaction log

$$\begin{aligned}
&Change \triangleq\\
&\quad \land \exists\, changes \in ValidChanges :\\
&\qquad \land\ transactions' = Append(transactions, [index \mapsto Len(transactions) + 1,\\
&\qquad\qquad\qquad\qquad\qquad\qquad\quad atomic \mapsto \text{FALSE},\\
&\qquad\qquad\qquad\qquad\qquad\qquad\quad sync \mapsto \text{FALSE},\\
&\qquad\qquad\qquad\qquad\qquad\qquad\quad changes \mapsto changes,\\
&\qquad\qquad\qquad\qquad\qquad\qquad\quad status \mapsto TransactionPending])\\
&\quad \land \text{UNCHANGED } \langle configurations, targets\rangle
\end{aligned}$$

This section models the Transaction log reconciler.

Reconcile the transaction log

$$\begin{aligned}
&ReconcileTransaction(n, i) \triangleq\\
&\qquad \text{If the transaction is } \textit{Pending}\text{, begin validation if the prior transaction}\\
&\qquad \text{has already been applied. This simplifies concurrency control in the controller}\\
&\qquad \text{and guarantees transactions are applied to the configurations in sequential order.}\\
&\quad \land\ \lor\ \land transactions[i].status = TransactionPending\\
&\qquad\quad \land\ \lor\ \land transactions[i].index - 1 \in \text{DOMAIN } transactions\\
&\qquad\qquad\qquad\ \land transactions[transactions[i].index - 1].status \in \{TransactionComplete, TransactionFailed\}\\
&\qquad\qquad\ \lor transactions[i].index - 1 \notin \text{DOMAIN } transactions\\
&\qquad\quad \land transactions' = [transactions \text{ EXCEPT } ![transactions[i].index].status = TransactionValidating]\\
&\qquad\quad \land \text{UNCHANGED } \langle configurations\rangle\\
&\qquad \text{If the transaction is in the } \textit{Validating} \text{ state, compute and validate the}\\
&\qquad \text{Configuration for each target.}\\
&\qquad \lor\ \land transactions[i].status = TransactionValidating\\
&\qquad\quad \text{If validation fails any target, mark the transaction } \textit{Failed}.\\
&\qquad\quad \text{If validation is successful, proceed to } \textit{Applying}.\\
&\qquad\quad \land \exists\, valid \in \text{BOOLEAN } :\\
&\qquad\qquad\quad \lor\ \land valid\\
&\qquad\qquad\qquad\quad \land transactions' = [transactions \text{ EXCEPT } ![transactions[i].index].status = TransactionApplying]\\
&\qquad\qquad\quad \lor\ \land \neg valid\\
&\qquad\qquad\qquad\quad \land transactions' = [transactions \text{ EXCEPT } ![transactions[i].index].status = TransactionFailed]\\
&\qquad\quad \land \text{UNCHANGED } \langle configurations\rangle\\
&\qquad \text{If the transaction is in the } \textit{Applying} \text{ state, update the Configuration for each}\\
&\qquad \text{target and } \textit{Complete} \text{ the transaction.}\\
&\qquad \lor\ \land transactions[i].status = TransactionApplying\\
&\qquad\quad \land\ \lor\ \land transactions[i].atomic\\
&\qquad\qquad\qquad\ \textit{TODO}\text{: Apply atomic transactions here}\\
&\qquad\qquad\quad \land transactions' = [transactions \text{ EXCEPT } ![transactions[i].index].status = TransactionComplete]
\end{aligned}$$

```
                    ∧ UNCHANGED ⟨configurations⟩
                ∨ ∧ ¬transactions[i].atomic
                  Add the transaction index to each updated path
                  ∧ configurations′ = [
                        t ∈ DOMAIN Target ↦ [
                          configurations[t] EXCEPT
                            !.paths = [path ∈ DOMAIN transactions[i].changes ↦
                              transactions[i].changes[path] @@ [index ↦ transactions[i].index] @@ configurations[
                            !.txIndex = transactions[i].index,
                            !.status  = ConfigurationPending]]
                  ∧ transactions′ = [transactions EXCEPT ![transactions[i].index].status = TransactionComplete]
    ∧ UNCHANGED ⟨targets⟩
```

This section models the Configuration reconciler.

```
ReconcileConfiguration(n, c) ≜
    ∧ ∨ ∧ 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.
      ∧ ∨ ∧ masters[configurations[c].target].term > configurations[c].mastershipTerm
          ∧ configurations′ = [configurations EXCEPT ![c].status          = ConfigurationInitializing,
                                                     ![c].mastershipTerm = masters[configurations[c].tar
        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.
        ∨ ∧ configurations[c].syncIndex < configurations[c].txIndex
          ∧ configurations′ = [configurations EXCEPT ![c].status = ConfigurationUpdating]
    ∨ ∧ configurations[c].status = ConfigurationInitializing
      ∧ masters[configurations[c].target].master = n
        Merge the configuration paths with the target paths, removing paths
        that have been marked deleted
      ∧ LET deletePaths ≜ {p ∈ DOMAIN configurations[c].paths : configurations[c].paths[p].deleted}
        IN
          ∧ targets′ = [targets EXCEPT ![configurations[c].target] =
                [p ∈ (DOMAIN c.paths \ deletePaths) ↦ [value ↦ configurations[c].paths[p]]] @@
                [p ∈ (DOMAIN targets[configurations[c].target] \ deletePaths) ↦ targets[configurations[c].tar
           Set the configuration's status to Complete
          ∧ configurations′ = [configurations 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
    ∨ ∃ n ∈ Node :
```

$\lor \exists\, t \in \text{DOMAIN } \mathit{Target} :$
$\quad \mathit{ChangeMaster}(n,\, t)$
$\lor \exists\, t \in \text{DOMAIN } \mathit{transactions} :$
$\quad \mathit{ReconcileTransaction}(n,\, t)$
$\lor \exists\, t \in \text{DOMAIN } \mathit{configurations} :$
$\quad \mathit{ReconcileConfiguration}(n,\, t)$

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

* Modification History
* Last modified *Mon Jan* 17 23:06:50 *PST* 2022 by *jordanhalterman*
* Created *Wed Sep* 22 13:22:32 *PDT* 2021 by *jordanhalterman*