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 $TransactionComplete \in$ STRING
ASSUME $TransactionFailed \in$ STRING

ASSUME $ConfigurationPending \in$ STRING
ASSUME $ConfigurationInitializing \in$ STRING
ASSUME $ConfigurationUpdating \in$ STRING
ASSUME $ConfigurationComplete \in$ STRING
ASSUME $ConfigurationFailed \in$ STRING

ASSUME $\land IsFiniteSet(Node)$
$\land \forall\, n \in Node :$
$\land\ n \notin$ DOMAIN $Target$
$\land\ n \in$ STRING

ASSUME $\land \forall\, t \in$ DOMAIN $Target :$
$\land\ IsFiniteSet(Target[t])$
$\land\ t \notin Node$
$\land\ t \in$ STRING

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

TYPE Transaction $\triangleq$ [
id ::= $id \in$ STRING,
$index$ ::= $index \in Nat$,
revision ::= revision $\in Nat$,
$atomic$ ::= $atomic \in$ BOOLEAN ,
$sync$ ::= $sync \in$ BOOLEAN ,
$changes$ ::= [$target \in$ SUBSET (DOMAIN $Target$) $\mapsto$ [
$path \in$ SUBSET (DOMAIN $Target[target]$) $\mapsto$ [
$value$ ::= $value \in$ STRING,
$delete$ ::= $delete \in$ BOOLEAN]]],
$status$ ::= $status \in TransactionStatus$]

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

TYPE Configuration $\triangleq$ [
id ::= $id \in$ STRING,
revision ::= revision $\in Nat$,
$target$ ::= $target \in$ 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*

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

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

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

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

$ValidChanges \triangleq$
$\quad \text{LET } allPaths \triangleq \text{UNION } \{(\text{DOMAIN } Target[t]) : t \in \text{DOMAIN } Target\}$
$\qquad allValues \triangleq \text{UNION } \{\text{UNION } \{Target[t][p] : p \in \text{DOMAIN } Target[t]\} : t \in \text{DOMAIN } Target\}$
$\quad \text{IN}$
$\qquad \{targetPathValues \in \text{SUBSET } (Target \times allPaths \times allValues \times \text{BOOLEAN }) :$
$\qquad\quad \land\ \forall\, target \in \text{DOMAIN } Target :$
$\qquad\qquad \text{LET } targetIndexes \triangleq \{i \in 1\,..\,Len(targetPathValues) : \land\ targetPathValues[i][1] = target\}$
$\qquad\qquad \text{IN} \quad \lor\ Cardinality(targetIndexes) = 0$
$\qquad\qquad\qquad\quad \lor\ \land\ Cardinality(targetIndexes) = 1$
$\qquad\qquad\qquad\qquad\quad \land\ \text{LET } targetPathValue \triangleq targetPathValues[\text{CHOOSE } index \in targetIndexes : \text{TRUE}]$
$\qquad\qquad\qquad\qquad\qquad\qquad\quad targetPath \triangleq targetPathValue[2]$
$\qquad\qquad\qquad\qquad\qquad\qquad\quad targetValue \triangleq targetPathValue[3]$

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

$ReconcileTransaction(n, i) \triangleq$

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.

$\land\ \lor\ \land\ transactions[i].status = TransactionPending$
$\land\ \lor\ \land\ transactions[i].index > 1$
$\land\ transactions[transactions[i].index - 1].status \in \{TransactionComplete, TransactionFailed\}$
$\lor\ transactions[i].index = 1$
$\land\ transactions' = [transactions\ \text{EXCEPT}\ ![transactions[i].index].status = TransactionValidating]$
$\land\ \text{UNCHANGED}\ \langle configurations\rangle$

If the transaction is in the *Validating* state, compute and validate the
Configuration for each target.

$\lor\ \land\ transactions[i].status = TransactionValidating$

If validation fails any target, mark the transaction *Failed*.
If validation is successful, proceed to *Applying*.

$\land \exists\, valid \in \text{BOOLEAN}\ :$
$\lor\ \land\ valid$
$\land\ transactions' = [transactions\ \text{EXCEPT}\ ![transactions[i].index].status = TransactionApplying]$
$\lor\ \land \neg valid$
$\land\ transactions' = [transactions\ \text{EXCEPT}\ ![transactions[i].index].status = TransactionFailed]$
$\land\ \text{UNCHANGED}\ \langle configurations\rangle$

If the transaction is in the *Applying* state, update the Configuration for each
target and *Complete* the transaction.

$\lor\ \land\ transactions[i].status = TransactionApplying$
$\land\ \lor\ \land\ transactions[i].atomic$

TODO: Apply atomic transactions here

$\land\ transactions' = [transactions\ \text{EXCEPT}\ ![transactions[i].index].status = TransactionComplete]$
$\land\ \text{UNCHANGED}\ \langle configurations\rangle$

```
        ∧ ∨ ∧ ¬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 :
        ∨ ∃ t ∈ DOMAIN Target :
```

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

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

* Modification History
* Last modified *Fri Jan* 14 15:33:11 *PST* 2022 by *jordanhalterman*
* Created *Wed Sep* 22 13:22:32 *PDT* 2021 by *jordanhalterman*