```
MODULE Config
EXTENDS Naturals, FiniteSets, Sequences, TLC

  Indicates that a configuration change is waiting to be applied to the network
CONSTANT Pending

  Indicates that a configuration change has been applied to the network
CONSTANT Complete

  Indicates that a configuration change failed
CONSTANT Failed

  Indicates a change is a configuration
CONSTANT Change

  Indicates a change is a rollback
CONSTANT Rollback

  Indicates a device is connected
CONSTANT Connected

  Indicates a device is disconnected
CONSTANT Disconnected

  Indicates that an error occurred when applying a change
CONSTANT Error

  The set of all nodes
CONSTANT Node

  The set of all devices
CONSTANT Device

  An empty constant
CONSTANT Nil

  Per-node election state
VARIABLE leader

  Per-node per-device election state
VARIABLE master

  A sequence of network-wide configuration changes
  Each change contains a record of 'changes' for each device
VARIABLE networkChange

  A record of sequences of device configuration changes
  Each sequence is a list of changes in the order in which they
  are to be applied to the device
```

VARIABLE *deviceChange*

A record of device states - either Available or Unavailable

VARIABLE *deviceState*

A count of leader changes to serve as a state constraint

VARIABLE *electionCount*

A count of configuration changes to serve as a state constraint

VARIABLE *configCount*

A count of device connection changes to serve as a state constraint

VARIABLE *connectionCount*

Node variables

$$\mathit{nodeVars} \triangleq \langle \mathit{leader}, \mathit{master} \rangle$$

Configuration variables

$$\mathit{configVars} \triangleq \langle \mathit{networkChange}, \mathit{deviceChange} \rangle$$

Device variables

$$\mathit{deviceVars} \triangleq \langle \mathit{deviceState} \rangle$$

State constraint variables

$$\mathit{constraintVars} \triangleq \langle \mathit{electionCount}, \mathit{configCount}, \mathit{connectionCount} \rangle$$

$$\mathit{vars} \triangleq \langle \mathit{nodeVars}, \mathit{configVars}, \mathit{deviceVars}, \mathit{constraintVars} \rangle$$

This section models leader election for control loops and for devices. Leader election is modelled as a simple boolean indicating whether each node is the leader for the cluster and for each device. This model implies the ordering of leadership changes is irrelevant to the correctness of the spec.

Set the leader for node n to l

$$
\begin{aligned}
&\mathit{SetNodeLeader}(n, l) \triangleq \\
&\quad \land \mathit{leader}' = [\mathit{leader} \text{ EXCEPT } ![n] = n = l] \\
&\quad \land \mathit{electionCount}' = \mathit{electionCount} + 1 \\
&\quad \land \text{UNCHANGED } \langle \mathit{master}, \mathit{configVars}, \mathit{deviceVars}, \mathit{configCount}, \mathit{connectionCount} \rangle
\end{aligned}
$$

Set the master for device d on node n to l

$$
\begin{aligned}
&\mathit{SetDeviceMaster}(n, d, l) \triangleq \\
&\quad \land \mathit{master}' = [\mathit{master} \text{ EXCEPT } ![n] = [\mathit{master}[n] \text{ EXCEPT } ![d] = n = l]] \\
&\quad \land \mathit{electionCount}' = \mathit{electionCount} + 1 \\
&\quad \land \text{UNCHANGED } \langle \mathit{leader}, \mathit{configVars}, \mathit{deviceVars}, \mathit{configCount}, \mathit{connectionCount} \rangle
\end{aligned}
$$

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

Enqueue network configuration change c

```
$SubmitChange(c) \triangleq$
    $\land\ Cardinality(\textsc{domain}\ c) > 0$
    $\land\ networkChange' = Append(networkChange, [$
                                $phase \mapsto Change,$
                                $changes \mapsto c,$
                                $value \mapsto Len(networkChange),$
                                $state \mapsto Pending,$
                                $attempt \mapsto 0])$
    $\land\ configCount' = configCount + 1$
    $\land\ \textsc{unchanged}\ \langle nodeVars, deviceChange, deviceVars, electionCount, connectionCount \rangle$

$RollbackChange(c) \triangleq$
    $\land\ networkChange[c].phase = Change$
    $\land\ networkChange[c].state = Complete$
    $\land\ networkChange' = [networkChange\ \textsc{except}\ ![c].phase = Rollback, ![c].state = Pending]$
    $\land\ configCount' = configCount + 1$
    $\land\ \textsc{unchanged}\ \langle nodeVars, deviceChange, deviceVars, electionCount, connectionCount \rangle$
```

This section models the *NetworkChange* reconciler. The reconciler reconciles network changes when the change or one of its device changes is updated.

Return the set of all network changes prior to the given change

```
$PriorNetworkChanges(c) \triangleq$
    $\{n \in \textsc{domain}\ networkChange : n < c\}$
```

Return the set of all completed device changes for network change c

```
$NetworkCompletedChanges(c) \triangleq$
    $\{d \in \textsc{domain}\ networkChange[c].changes :$
        $\land\ c \in \textsc{domain}\ deviceChange[d]$
        $\land\ deviceChange[d][c].state = Complete\}$
```

Return a boolean indicating whether all device changes are complete for the given network change

```
$NetworkChangesComplete(c) \triangleq$
    $Cardinality(NetworkCompletedChanges(c)) = Cardinality(\textsc{domain}\ networkChange[c].changes)$
```

Return the set of all incomplete device changes prior to network change c

```
$PriorIncompleteDevices(c) \triangleq$
    $\textsc{union}\ \{\textsc{domain}\ networkChange[n].changes :$
                 $n \in \{n \in PriorNetworkChanges(c) : \neg NetworkChangesComplete(n)\}\}$
```

Return the set of all devices configured by network change c

```
$NetworkChangeDevices(c) \triangleq \textsc{domain}\ networkChange[c].changes$
```

Return the set of all connected devices configured by network change c

```
$ConnectedDevices(c) \triangleq \{d \in \textsc{domain}\ networkChange[c].changes : deviceState[d] = Connected\}$
```

Return a boolean indicating whether network change c can be applied
A change can be applied if its devices do not intersect with past device
changes that have not been applied

```
$CanApplyNetworkChange(c) \triangleq$
    $\land\ Cardinality(ConnectedDevices(c) \cap NetworkChangeDevices(c)) \neq 0$
    $\land\ Cardinality(NetworkChangeDevices(c) \cap PriorIncompleteDevices(c)) = 0$
    $\land\ \lor\ networkChange[c].attempt = 0$
        $\lor\ Cardinality(\{d \in \text{DOMAIN}\ networkChange[c].changes :$
                $\land\ deviceChange[d][c].attempt = networkChange[c].attempt$
                $\land\ deviceChange[d][c].phase = Rollback$
                $\land\ deviceChange[d][c].state\ \ = Complete\}) =$
                    $Cardinality(\text{DOMAIN}\ networkChange[c].changes)$
```

Return a boolean indicating whether a change exists for the given device
If the device is modified by the change, it must contain a device change
that's either *Complete* or with the same 'attempt' as the network change.

```
$HasDeviceChange(d,\ c) \triangleq$
    $\land\ c \in \text{DOMAIN}\ deviceChange[d]$
    $\land\ deviceChange[d][c].attempt = networkChange[c].attempt$
```

Return a boolean indicating whether device changes have been propagated
for the given network change

```
$HasDeviceChanges(c) \triangleq$
    $Cardinality(\{d \in \text{DOMAIN}\ networkChange[c].changes : HasDeviceChange(d,\ c)\}) =$
        $Cardinality(\text{DOMAIN}\ networkChange[c].changes)$
```

Add or update the given device changes for the given network change.
If a device change already exists, update the 'attempt' field.

```
$CreateDeviceChange(d,\ c) \triangleq$
    IF $Cardinality(\text{DOMAIN}\ deviceChange[d]) = 0$ THEN
        $[x \in \{c\} \mapsto [$
                $phase\ \ \ \ \mapsto networkChange[c].phase,$
                $state\ \ \ \ \ \mapsto Pending,$
                $value\ \ \ \ \mapsto networkChange[c].value,$
                $attempt \mapsto networkChange[c].attempt]]$
     ELSE
        IF $d \in \text{DOMAIN}\ networkChange[c].changes$ THEN
            IF $c \in \text{DOMAIN}\ deviceChange[d]$ THEN
                IF $deviceChange[d][c].state = Complete$ THEN
                    $deviceChange[d][c]$
                 ELSE
                    $[deviceChange[d]\ \text{EXCEPT}\ ![c].attempt = networkChange[c].attempt,$
                                                  $![c].state = Pending]$
            ELSE
                $[x \in \{c\} \mapsto [$
                    $phase\ \ \ \ \mapsto networkChange[c].phase,$
                 $state \quad \mapsto Pending,$
                 $value \quad \mapsto networkChange[c].value,$
                 $attempt \mapsto networkChange[c].attempt]]$ @@ $deviceChange[d]$
            ELSE
               $deviceChange[d]$
```

Add or update device changes for the given network change

```
$CreateDeviceChanges(c) \triangleq$
   $deviceChange' = [d \in$ DOMAIN $deviceChange \mapsto CreateDeviceChange(d, c)]$
```

Rollback device change c for device d

```
$RollbackDeviceChange(d, c) \triangleq$
  IF $\land\ c \in$ DOMAIN $deviceChange[d]$
     $\land\ \lor\ deviceChange[d][c].phase = Change$
        $\lor\ \land\ deviceChange[d][c].phase = Rollback$
           $\land\ deviceChange[d][c].state \ = Failed$
   THEN
      $[deviceChange[d]$ EXCEPT $![c].phase = Rollback, ![c].state = Pending]$
   ELSE
      $deviceChange[d]$
```

Roll back device changes

```
$RollbackDeviceChanges(c) \triangleq$
   $deviceChange' = [d \in$ DOMAIN $deviceChange \mapsto RollbackDeviceChange(d, c)]$
```

Return a boolean indicating whether the given device change is *Failed*

```
$IsFailedDeviceChange(d, c) \triangleq$
   $\land\ c \in$ DOMAIN $deviceChange[d]$
   $\land\ deviceChange[d][c].attempt = networkChange[c].attempt$
   $\land\ deviceChange[d][c].state = Failed$
```

Return a boolean indicating whether the given device change is *Complete*

```
$IsCompleteDeviceChange(d, c) \triangleq$
   $\land\ c \in$ DOMAIN $deviceChange[d]$
   $\land\ deviceChange[d][c].attempt = networkChange[c].attempt$
   $\land\ deviceChange[d][c].phase = Change$
   $\land\ deviceChange[d][c].state \ = Complete$
```

Return a boolean indicating whether any device change is *Failed* for the given network change

```
$HasFailedDeviceChanges(c) \triangleq$
   $Cardinality(\{d \in$ DOMAIN $networkChange[c].changes :$
      $IsFailedDeviceChange(d, c)\}) \neq 0$
```

Return a boolean indicating whether all device changes are *Complete* for the given network change

```
$DeviceChangesComplete(c) \triangleq$
   $Cardinality(\{d \in$ DOMAIN $networkChange[c].changes :$
     $IsCompleteDeviceChange(d, c)\}) =$
```

```
          Cardinality(DOMAIN networkChange[c].changes)

 Reconcile a network change state
ReconcileNetworkChange(n, c) ≜
    ∧ leader[n]
    ∧ networkChange[c].state = Pending
    ∧ ∨ ∧ ¬HasDeviceChanges(c)
          ∧ CreateDeviceChanges(c)
          ∧ UNCHANGED ⟨networkChange⟩
      ∨ ∧ HasDeviceChanges(c)
          ∧ ∨ ∧ networkChange[c].phase = Change
                ∧ ∨ ∧ CanApplyNetworkChange(c)
                      ∧ networkChange' = [networkChange EXCEPT
                            ![c].attempt = networkChange[c].attempt + 1]
                      ∧ UNCHANGED ⟨deviceChange⟩
                  ∨ ∧ DeviceChangesComplete(c)
                      ∧ networkChange' = [networkChange EXCEPT
                            ![c].state = Complete]
                      ∧ UNCHANGED ⟨deviceChange⟩
                  ∨ ∧ HasFailedDeviceChanges(c)
                      ∧ RollbackDeviceChanges(c)
                      ∧ UNCHANGED ⟨networkChange⟩
             TODO
            ∨ ∧ networkChange[c].phase = Rollback
                ∧ networkChange' = [networkChange EXCEPT
                      ![c].state     = Complete]
                ∧ UNCHANGED ⟨deviceChange⟩
    ∧ UNCHANGED ⟨nodeVars, deviceVars, constraintVars⟩
```

```
This section models the DeviceChange reconciler.
ReconcileDeviceChange(n, d, c) ≜
    ∧ master[n][d]
    ∧ deviceChange[d][c].state = Pending
    ∧ deviceChange[d][c].attempt > 0
    ∧ ∨ ∧ deviceState[d] = Connected
          ∧ deviceChange' = [deviceChange EXCEPT
                ![d] = [deviceChange[d] EXCEPT ![c].state = Complete]]
      ∨ ∧ deviceState[d] = Disconnected
          ∧ deviceChange' = [deviceChange EXCEPT
                ![d] = [deviceChange[d] EXCEPT ![c].state = Failed]]
    ∧ UNCHANGED ⟨nodeVars, networkChange, deviceVars, constraintVars⟩
```

This section models device states. Devices begin in the Unavailable state and can only be configured while in the Available state.

Set device d state to *Connected*

```
$ConnectDevice(d) \triangleq$
    $\land\ deviceState' = [deviceState\ \text{EXCEPT}\ ![d] = Connected]$
    $\land\ connectionCount' = connectionCount + 1$
    $\land\ \text{UNCHANGED}\ \langle nodeVars,\ configVars,\ electionCount,\ configCount \rangle$
```

Set device d state to *Disconnected*

```
$DisconnectDevice(d) \triangleq$
    $\land\ deviceState' = [deviceState\ \text{EXCEPT}\ ![d] = Disconnected]$
    $\land\ connectionCount' = connectionCount + 1$
    $\land\ \text{UNCHANGED}\ \langle nodeVars,\ configVars,\ electionCount,\ configCount \rangle$
```

Init and next state predicates

```
$Init \triangleq$
    $\land\ leader\ \ = [n \in Node \mapsto \text{FALSE}]$
    $\land\ master = [n \in Node \mapsto [d \in Device \mapsto \text{FALSE}]]$
    $\land\ networkChange = \langle\rangle$
    $\land\ deviceChange = [d \in Device \mapsto [x \in \{\} \mapsto [phase \mapsto Change,\ state \mapsto Pending]]]$
    $\land\ deviceState = [d \in Device \mapsto Disconnected]$
    $\land\ electionCount = 0$
    $\land\ configCount = 0$
    $\land\ connectionCount = 0$

$Next \triangleq$
    $\lor\ \exists\, d \in \text{SUBSET}\ Device :$
        $SubmitChange([x \in d \mapsto 1])$
    $\lor\ \exists\, c \in \text{DOMAIN}\ networkChange :$
        $RollbackChange(c)$
    $\lor\ \exists\, n \in Node :$
        $\exists\, l \in Node :$
          $SetNodeLeader(n,\ l)$
    $\lor\ \exists\, n \in Node :$
        $\exists\, d \in Device :$
          $\exists\, l \in Node :$
            $SetDeviceMaster(n,\ d,\ l)$
    $\lor\ \exists\, n \in Node :$
        $\exists\, c \in \text{DOMAIN}\ networkChange :$
          $ReconcileNetworkChange(n,\ c)$
    $\lor\ \exists\, n \in Node :$
        $\exists\, d \in Device :$
          $\exists\, c \in \text{DOMAIN}\ deviceChange[d] :$
            $ReconcileNetworkChange(n,\ c)$
    $\lor\ \exists\, n \in Node :$
```

```
        $\exists\, d \in \mathit{Device} :$
          $\exists\, c \in \textsc{domain}\ \mathit{deviceChange}[d] :$
            $\mathit{ReconcileDeviceChange}(n,\ d,\ c)$
    $\lor\ \exists\, d \in \mathit{Device} :$
        $\mathit{ConnectDevice}(d)$
    $\lor\ \exists\, d \in \mathit{Device} :$
        $\mathit{DisconnectDevice}(d)$

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

\* Modification History
\* Last modified *Fri Dec* 13 17:43:05 *PST* 2019 by *jordanhalterman*
\* Created *Fri Sep* 27 13:14:24 *PDT* 2019 by *jordanhalterman*