
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 *leadership*

Per-node per-device election state
VARIABLE *mastership*

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

$nodeVars \triangleq \langle leadership, mastership \rangle$

Configuration variables

$configVars \triangleq \langle networkChange, deviceChange \rangle$

Device variables

$deviceVars \triangleq \langle deviceState \rangle$

State constraint variables

$constraintVars \triangleq \langle electionCount, configCount, connectionCount \rangle$

$vars \triangleq \langle nodeVars, configVars, deviceVars, constraintVars \rangle$

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

$SetNodeLeader(n, l) \triangleq$
 $\wedge leadership' = [leadership \text{ EXCEPT } ![n] = n = l]$
 $\wedge electionCount' = electionCount + 1$
 $\wedge \text{UNCHANGED } \langle mastership, configVars, deviceVars, configCount, connectionCount \rangle$

Set the master for device d on node n to l

$SetDeviceMaster(n, d, l) \triangleq$
 $\wedge mastership' = [mastership \text{ EXCEPT } ![n] = [mastership[n] \text{ EXCEPT } ![d] = n = l]]$
 $\wedge electionCount' = electionCount + 1$
 $\wedge \text{UNCHANGED } \langle leadership, configVars, deviceVars, configCount, connectionCount \rangle$

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

Enqueue network configuration change c

$$\begin{aligned} \text{SubmitChange}(c) &\triangleq \\ &\wedge \text{Cardinality}(\text{DOMAIN } c) > 0 \\ &\wedge \text{networkChange}' = \text{Append}(\text{networkChange}, [\\ &\quad \text{phase} \mapsto \text{Change}, \\ &\quad \text{changes} \mapsto c, \\ &\quad \text{value} \mapsto \text{Len}(\text{networkChange}), \\ &\quad \text{state} \mapsto \text{Pending}, \\ &\quad \text{attempt} \mapsto 0]) \\ &\wedge \text{configCount}' = \text{configCount} + 1 \\ &\wedge \text{UNCHANGED } \langle \text{nodeVars}, \text{deviceChange}, \text{deviceVars}, \text{electionCount}, \text{connectionCount} \rangle \end{aligned}$$

RollbackChange(c) \triangleq

$$\begin{aligned} &\wedge \text{networkChange}[c].\text{phase} = \text{Change} \\ &\wedge \text{networkChange}[c].\text{state} = \text{Complete} \\ &\wedge \text{networkChange}' = [\text{networkChange} \text{ EXCEPT } ![c].\text{phase} = \text{Rollback}, ![c].\text{state} = \text{Pending}] \\ &\wedge \text{configCount}' = \text{configCount} + 1 \\ &\wedge \text{UNCHANGED } \langle \text{nodeVars}, \text{deviceChange}, \text{deviceVars}, \text{electionCount}, \text{connectionCount} \rangle \end{aligned}$$

This section models a configuration change scheduler. The role of the scheduler is to determine when network changes can be applied and enqueue the relevant changes for application by changing their state from *Pending* to *Applying*. The scheduler supports concurrent application of non-overlapping configuration changes (changes that do not impact intersecting sets of devices) by comparing *Pending* changes with *Applying* changes.

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

$$\begin{aligned} \text{PriorNetworkChanges}(c) &\triangleq \\ &\{n \in \text{DOMAIN } \text{networkChange} : n < c\} \end{aligned}$$

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

$$\begin{aligned} \text{NetworkCompletedChanges}(c) &\triangleq \\ &\{d \in \text{DOMAIN } \text{networkChange}[c].\text{changes} : \\ &\quad \wedge c \in \text{DOMAIN } \text{deviceChange}[d] \\ &\quad \wedge \text{deviceChange}[d][c].\text{state} = \text{Complete}\} \end{aligned}$$

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

$$\begin{aligned} \text{NetworkChangesComplete}(c) &\triangleq \\ &\text{Cardinality}(\text{NetworkCompletedChanges}(c)) = \text{Cardinality}(\text{DOMAIN } \text{networkChange}[c].\text{changes}) \end{aligned}$$

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

$$\begin{aligned} \text{PriorIncompleteDevices}(c) &\triangleq \\ &\text{UNION } \{ \text{DOMAIN } \text{networkChange}[n].\text{changes} : \\ &\quad n \in \{n \in \text{PriorNetworkChanges}(c) : \neg \text{NetworkChangesComplete}(n)\} \} \end{aligned}$$

Return the set of all devices configured by network change c

$$\text{NetworkChangeDevices}(c) \triangleq \text{DOMAIN } \text{networkChange}[c].\text{changes}$$

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

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

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

$CanApplyNetworkChange(c) \triangleq$
 $\wedge Cardinality(ConnectedDevices(c) \cup NetworkChangeDevices(c)) = 0$
 $\wedge Cardinality(NetworkChangeDevices(c) \cap PriorIncompleteDevices(c)) = 0$

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

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.

$HasDeviceChange(d, c) \triangleq$
 $\vee d \notin \text{DOMAIN } networkChange[c].changes$
 $\vee \wedge d \in \text{DOMAIN } networkChange[c].changes$
 $\wedge c \in \text{DOMAIN } deviceChange[d]$
 $\wedge \vee deviceChange[d][c].attempt = networkChange[c].attempt$
 $\vee deviceChange[d].state = Complete$

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

$HasDeviceChanges(c) \triangleq$
 $Cardinality(\{d \in Device : HasDeviceChange(d, c)\}) \neq 0$

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
 $[x \in \{c\} \mapsto [$
 $\quad phase \mapsto networkChange[c].phase,$
 $\quad state \mapsto Pending,$
 $\quad value \mapsto networkChange[c].value,$
 $\quad attempt \mapsto networkChange[c].attempt]]$
 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

$$\begin{aligned}
& [x \in \{c\} \mapsto [\\
& \quad \text{phase} \mapsto \text{networkChange}[c].\text{phase}, \\
& \quad \text{state} \mapsto \text{Pending}, \\
& \quad \text{value} \mapsto \text{networkChange}[c].\text{value}, \\
& \quad \text{attempt} \mapsto \text{networkChange}[c].\text{attempt}] @@ \text{deviceChange}[d] \\
& \text{ELSE} \\
& \quad \text{deviceChange}[d] \\
& \text{Add or update device changes for the given network change} \\
& \text{CreateDeviceChanges}(c) \triangleq \\
& \quad \text{deviceChange}' = [d \in \text{DOMAIN } \text{deviceChange} \mapsto \text{CreateDeviceChange}(d, c)] \\
& \text{Rollback device change } c \text{ for device } d \\
& \text{RollbackDeviceChange}(d, c) \triangleq \\
& \quad \text{IF } c \in \text{DOMAIN } \text{deviceChange}[d] \text{ THEN} \\
& \quad \quad [\text{deviceChange}[d] \text{ EXCEPT } ![c].\text{phase} = \text{Rollback}, ![c].\text{state} = \text{Pending}] \\
& \quad \text{ELSE} \\
& \quad \quad \text{deviceChange}[d] \\
& \text{Roll back device changes} \\
& \text{RollbackDeviceChanges}(c) \triangleq \\
& \quad \text{deviceChange}' = [d \in \text{DOMAIN } \text{deviceChange} \mapsto \text{RollbackDeviceChange}(d, c)] \\
& \text{Return a boolean indicating whether the given device change is } \textit{Failed} \\
& \text{IsFailedDeviceChange}(d, c) \triangleq \\
& \quad \vee \text{deviceChange}[d][c].\text{phase} = \textit{Change} \\
& \quad \vee \text{deviceChange}[d][c].\text{attempt} = 0 \\
& \quad \vee \wedge \text{deviceChange}[d][c].\text{attempt} = \text{networkChange}[c].\text{attempt} \\
& \quad \quad \wedge \text{deviceChange}[d][c].\text{state} = \textit{Failed} \\
& \text{IsDeviceChangeRollbackFailed}(d, c) \triangleq \\
& \quad \wedge c \in \text{DOMAIN } \text{deviceChange}[d] \\
& \quad \wedge \text{deviceChange}[d][c].\text{attempt} = \text{networkChange}[c].\text{attempt} \\
& \quad \wedge \text{deviceChange}[d][c].\text{phase} = \textit{Rollback} \\
& \quad \wedge \text{deviceChange}[d][c].\text{state} = \textit{Failed} \\
& \text{IsDeviceChangeRollbackComplete}(d, c) \triangleq \\
& \quad \wedge c \in \text{DOMAIN } \text{deviceChange}[d] \\
& \quad \wedge \text{deviceChange}[d][c].\text{attempt} = \text{networkChange}[c].\text{attempt} \\
& \quad \wedge \text{deviceChange}[d][c].\text{phase} = \textit{Rollback} \\
& \quad \wedge \text{deviceChange}[d][c].\text{state} = \textit{Complete} \\
& \text{Return a boolean indicating whether the given device change is } \textit{Complete} \\
& \text{IsCompleteDeviceChange}(d, c) \triangleq \\
& \quad \wedge \text{deviceChange}[d][c].\text{phase} = \textit{Change} \\
& \quad \wedge \text{deviceChange}[d][c].\text{state} = \textit{Complete}
\end{aligned}$$

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

$$\begin{aligned} IsCompleteDeviceRollback(d, c) &\triangleq \\ &\wedge deviceChange[d][c].phase = Rollback \\ &\wedge deviceChange[d][c].state = Complete \end{aligned}$$

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

$$\begin{aligned} HasCompleteDeviceChanges(c) &\triangleq \\ &Cardinality(\{d \in DOMAIN \ deviceChange : \\ &\quad \wedge c \in DOMAIN \ deviceChange[d] \\ &\quad \wedge IsCompleteDeviceChange(d, c)\}) \neq 0 \end{aligned}$$

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

$$\begin{aligned} HasFailedDeviceChanges(c) &\triangleq \\ &Cardinality(\{d \in DOMAIN \ deviceChange : \\ &\quad \wedge c \in DOMAIN \ deviceChange[d] \\ &\quad \wedge IsFailedDeviceChange(d, c)\}) \neq 0 \end{aligned}$$

Return a boolean indicating whether all device changes have been rolled back for the given network change

$$\begin{aligned} DeviceChangesRolledBack(c) &\triangleq \\ &Cardinality(\{d \in DOMAIN \ deviceChange : \\ &\quad \wedge c \in DOMAIN \ deviceChange[d] \\ &\quad \wedge IsCompleteDeviceRollback(d, c)\}) = \\ &\quad Cardinality(DOMAIN \ networkChange[c].changes) \end{aligned}$$

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

$$\begin{aligned} DeviceChangesComplete(c) &\triangleq \\ &Cardinality(\{d \in DOMAIN \ deviceChange : \\ &\quad \wedge c \in DOMAIN \ deviceChange[d] \\ &\quad \wedge IsCompleteDeviceChange(d, c)\}) = \\ &\quad Cardinality(DOMAIN \ networkChange[c].changes) \end{aligned}$$

Reconcile a network change state

$$\begin{aligned} ReconcileNetworkChange(n, c) &\triangleq \\ &\wedge leadership[n] = TRUE \\ &\wedge networkChange[c].state = Pending \\ &\quad \text{Create device changes if necessary} \\ &\wedge \vee \wedge \neg HasDeviceChanges(c) \\ &\quad \wedge CreateDeviceChanges(c) \\ &\quad \wedge UNCHANGED \langle networkChange \rangle \\ &\vee \wedge HasDeviceChanges(c) \\ &\quad \text{Reconcile a change} \\ &\wedge \vee \wedge networkChange[c].phase = Change \\ &\quad \text{If a device change failed rollback any applied changes} \\ &\quad \wedge \vee \wedge HasFailedDeviceChanges(c) \\ &\quad \quad \wedge RollbackDeviceChanges(c) \\ &\quad \quad \wedge UNCHANGED \langle networkChange \rangle \end{aligned}$$

If device changes have been rolled back, increment the attempt
 $\vee \wedge DeviceChangesRolledBack(c)$
 $\wedge networkChange' = [networkChange \text{ EXCEPT } !c].attempt = networkChange[c].attempt + 1]$
 If device changes are complete, complete the network change
 $\vee \wedge DeviceChangesComplete(c)$
 $\wedge networkChange' = [networkChange \text{ EXCEPT } !c].state = Complete]$
 $\wedge \text{UNCHANGED } \langle deviceChange \rangle$
 Reconcile a rollback
 $\vee \wedge networkChange[c].phase = Rollback$
 $\wedge networkChange' = [networkChange \text{ EXCEPT } !c].state = Complete]$
 $\wedge \text{UNCHANGED } \langle nodeVars, deviceVars, constraintVars \rangle$

This section models the *DeviceChange* reconciler.

$ReconcileDeviceChange(n, d, c) \triangleq$
 $\wedge deviceChange[d][c].state = Pending$
 $\wedge deviceChange[d][c].attempt > 0$
 $\wedge \vee \wedge deviceState[d] = Connected$
 $\wedge deviceChange' = [deviceChange \text{ EXCEPT } ![d] = [deviceChange[d] \text{ EXCEPT } ![c].state = Complete]]$
 $\vee \wedge deviceState[d] = Disconnected$
 $\wedge deviceChange' = [deviceChange \text{ EXCEPT } ![d] = [deviceChange[d] \text{ EXCEPT } ![c].state = Failed]]$
 $\wedge \text{UNCHANGED } \langle nodeVars, networkChange, deviceVars, constraintVars \rangle$

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

Set device d state to *Disconnected*
 $DisconnectDevice(d) \triangleq$
 $\wedge deviceState' = [deviceState \text{ EXCEPT } ![d] = Disconnected]$
 $\wedge connectionCount' = connectionCount + 1$
 $\wedge \text{UNCHANGED } \langle nodeVars, configVars, electionCount, configCount \rangle$

Init and next state predicates

$$\begin{aligned}
Init &\triangleq \\
&\wedge \textit{leadership} = [n \in \textit{Node} \mapsto \text{FALSE}] \\
&\wedge \textit{mastership} = [n \in \textit{Node} \mapsto [d \in \textit{Device} \mapsto \text{FALSE}]] \\
&\wedge \textit{networkChange} = \langle \rangle \\
&\wedge \textit{deviceChange} = [d \in \textit{Device} \mapsto [x \in \{\} \mapsto [\textit{phase} \mapsto \textit{Change}, \textit{state} \mapsto \textit{Pending}]]] \\
&\wedge \textit{deviceState} = [d \in \textit{Device} \mapsto \textit{Disconnected}] \\
&\wedge \textit{electionCount} = 0 \\
&\wedge \textit{configCount} = 0 \\
&\wedge \textit{connectionCount} = 0 \\
\\
Next &\triangleq \\
&\vee \exists d \in \text{SUBSET } \textit{Device} : \\
&\quad \textit{SubmitChange}([x \in d \mapsto 1]) \\
&\vee \exists c \in \text{DOMAIN } \textit{networkChange} : \\
&\quad \textit{RollbackChange}(c) \\
&\vee \exists n \in \textit{Node} : \\
&\quad \exists l \in \textit{Node} : \\
&\quad \quad \textit{SetNodeLeader}(n, l) \\
&\vee \exists n \in \textit{Node} : \\
&\quad \exists d \in \textit{Device} : \\
&\quad \exists l \in \textit{Node} : \\
&\quad \quad \textit{SetDeviceMaster}(n, d, l) \\
&\vee \exists n \in \textit{Node} : \\
&\quad \exists c \in \text{DOMAIN } \textit{networkChange} : \\
&\quad \quad \textit{ReconcileNetworkChange}(n, c) \\
&\vee \exists n \in \textit{Node} : \\
&\quad \exists d \in \textit{Device} : \\
&\quad \exists c \in \text{DOMAIN } \textit{deviceChange}[d] : \\
&\quad \quad \textit{ReconcileNetworkChange}(n, c) \\
&\vee \exists n \in \textit{Node} : \\
&\quad \exists d \in \textit{Device} : \\
&\quad \exists c \in \text{DOMAIN } \textit{deviceChange}[d] : \\
&\quad \quad \textit{ReconcileDeviceChange}(n, d, c) \\
&\vee \exists d \in \textit{Device} : \\
&\quad \textit{ConnectDevice}(d) \\
&\vee \exists d \in \textit{Device} : \\
&\quad \textit{DisconnectDevice}(d) \\
\\
Spec &\triangleq Init \wedge \Box[Next]_{vars}
\end{aligned}$$

```

\ * Modification History
\ * Last modified Thu Dec 12 17:34:06 PST 2019 by jordanhalterman
\ * Created Fri Sep 27 13:14:24 PDT 2019 by jordanhalterman

```