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

Set the master for device d on node n to l
$SetDeviceMaster(n,\ d,\ l) \triangleq$
$\quad \wedge\ mastership' = [mastership \text{ EXCEPT } ![n] = [mastership[n] \text{ EXCEPT } ![d] = n = l]]$
$\quad \wedge\ electionCount' = electionCount + 1$
$\quad \wedge\ \text{UNCHANGED } \langle leadership,\ configVars,\ deviceVars,\ configCount,\ connectionCount \rangle$

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

Enqueue network configuration change c

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

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

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

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

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

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

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

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

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

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

Return the set of all devices configured by network change c

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

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

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

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.

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

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

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

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

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

```
                    phase    ↦ networkChange[c].phase,
                    state    ↦ Pending,
                    value    ↦ networkChange[c].value,
                    attempt ↦ networkChange[c].attempt]] @@ deviceChange[d]
            ELSE
                deviceChange[d]

    Add or update device changes for the given network change
CreateDeviceChanges(c) ≜
    ∧ deviceChange′ = [d ∈ DOMAIN deviceChange ↦ CreateDeviceChange(d, c)]

    Return a boolean indicating whether the given device change is Failed
IsFailedDeviceChange(d, c) ≜
    ∧ c ∈ DOMAIN deviceChange[d]
    ∧ ∨ deviceChange[d][c].attempt = 0
      ∨ ∧ deviceChange[d][c].attempt = networkChange[c].attempt
        ∧ deviceChange[d][c].state = Failed

    Return a boolean indicating whether the given device change is Complete
IsCompleteDeviceChange(d, c) ≜
    ∧ c ∈ DOMAIN deviceChange[d]
    ∧ deviceChange[d][c].state = Complete

    Return a boolean indicating whether any device change is Failed for the given network change
HasFailedDeviceChanges(c) ≜
    Cardinality({d ∈ DOMAIN deviceChange : IsFailedDeviceChange(d, c)}) ≠ 0

    Return a boolean indicating whether all device changes are Complete for the given network change
DeviceChangesComplete(c) ≜
    Cardinality({d ∈ DOMAIN deviceChange : IsCompleteDeviceChange(d, c)}) = Cardinality(DOMAIN networ

    Reconcile a network change state
ReconcileNetworkChange(n, c) ≜
    ∧ leadership[n] = TRUE
    ∧ networkChange[c].state = Pending
        Create device changes if necessary
    ∧ ∨ ∧ ¬HasDeviceChanges(c)
        ∧ CreateDeviceChanges(c)
        ∧ UNCHANGED ⟨networkChange⟩
      ∨ ∧ HasDeviceChanges(c)
            Reconcile a change
        ∧ ∨ ∧ networkChange[c].phase = Change
            ∧ ∨ ∧ HasFailedDeviceChanges(c)
                ∧ CanApplyNetworkChange(c)
                ∧ networkChange′ = [networkChange EXCEPT ![c].attempt = networkChange[c].attempt +
                ∧ UNCHANGED ⟨deviceChange⟩
              ∨ ∧ DeviceChangesComplete(c)
```

$\land\ networkChange' = [networkChange \text{ EXCEPT } ![c].state = Complete]$
$\land\ \text{UNCHANGED } \langle deviceChange \rangle$

Reconcile a rollback

$\lor\ \land\ networkChange[c].phase = Rollback$
$\land\ networkChange' = [networkChange \text{ EXCEPT } ![c].state = Complete]$

$\land\ \text{UNCHANGED } \langle nodeVars,\ deviceVars,\ constraintVars \rangle$

This section models the *DeviceChange* reconciler.

$ReconcileDeviceChange(n,\ d,\ c) \triangleq$

$\land\ deviceChange[d][c].state = Pending$
$\land\ deviceChange[d][c].attempt > 0$
$\land\ \lor\ \land\ deviceState[d] = Connected$
$\land\ deviceChange' = [deviceChange \text{ EXCEPT } ![d] = [deviceChange[d] \text{ EXCEPT } ![c].state = Complete]]$
$\lor\ \land\ deviceState[d] = Disconnected$
$\land\ deviceChange' = [deviceChange \text{ EXCEPT } ![d] = [deviceChange[d] \text{ EXCEPT } ![c].state = Failed]]$
$\land\ \text{UNCHANGED } \langle nodeVars,\ networkChange,\ deviceVars,\ constraintVars \rangle$

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*

$ConnectDevice(d) \triangleq$

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

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

$Init \triangleq$

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

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

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

* Modification History
* Last modified *Thu Dec* 12 15:15:54 *PST* 2019 by *jordanhalterman*
* Created *Fri Sep* 27 13:14:24 *PDT* 2019 by *jordanhalterman*