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
$nodeVars \triangleq \langle leader, master \rangle$

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

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

This section models the northbound API for the configuration service.

Enqueue network configuration change c

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

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

This section models 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

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

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) ≜
    ∧ Cardinality(ConnectedDevices(c) ∩ NetworkChangeDevices(c)) ≠ 0
    ∧ Cardinality(NetworkChangeDevices(c) ∩ PriorIncompleteDevices(c)) = 0
    ∧ ∨ networkChange[c].incarnation = 0
      ∨ Cardinality({d ∈ DOMAIN networkChange[c].changes :
            ∧ deviceChange[d][c].incarnation = networkChange[c].incarnation
            ∧ deviceChange[d][c].phase = Rollback
            ∧ deviceChange[d][c].state  = Complete}) =
                Cardinality(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 'incarnation' as the network change.
HasDeviceChange(d, c) ≜
    ∧ c ∈ DOMAIN deviceChange[d]
    ∧ deviceChange[d][c].incarnation = networkChange[c].incarnation

Return a boolean indicating whether device changes have been propagated
for the given network change
HasDeviceChanges(c) ≜
    Cardinality({d ∈ DOMAIN networkChange[c].changes : HasDeviceChange(d, c)}) =
      Cardinality(DOMAIN networkChange[c].changes)

Add or update the given device changes for the given network change.
If a device change already exists, update the 'incarnation' field.
CreateDeviceChange(d, c) ≜
    IF Cardinality(DOMAIN deviceChange[d]) = 0 THEN
        [x ∈ {c} ↦ [
                phase       ↦ networkChange[c].phase,
                state       ↦ Pending,
                value       ↦ networkChange[c].value,
                incarnation ↦ networkChange[c].incarnation]]
     ELSE
        IF d ∈ DOMAIN networkChange[c].changes THEN
            IF c ∈ DOMAIN deviceChange[d] THEN
                IF deviceChange[d][c].state = Complete THEN
                    deviceChange[d][c]
                 ELSE
                    [deviceChange[d] EXCEPT ![c].incarnation = networkChange[c].incarnation,
                                            ![c].state = Pending]
             ELSE
                [x ∈ {c} ↦ [
                    phase       ↦ networkChange[c].phase,
```

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

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

 Rollback device change c for device d
RollbackDeviceChange(d, c) ≜
    IF  ∧ c ∈ DOMAIN deviceChange[d]
        ∧ ∨ deviceChange[d][c].phase = Change
          ∨ ∧ deviceChange[d][c].phase = Rollback
            ∧ deviceChange[d][c].state  = Failed
     THEN
        [deviceChange[d] EXCEPT ![c].phase = Rollback, ![c].state = Pending]
     ELSE
        deviceChange[d]

 Roll back device changes
RollbackDeviceChanges(c) ≜
    deviceChange′ = [d ∈ DOMAIN deviceChange ↦ RollbackDeviceChange(d, c)]

 Return a boolean indicating whether the given device change is Failed
IsFailedDeviceChange(d, c) ≜
    ∧ c ∈ DOMAIN deviceChange[d]
    ∧ deviceChange[d][c].incarnation = networkChange[c].incarnation
    ∧ 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].incarnation = networkChange[c].incarnation
    ∧ deviceChange[d][c].phase = Change
    ∧ 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 networkChange[c].changes :
        IsFailedDeviceChange(d, c)}) ≠ 0

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

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

Reconcile a network change state

$$
\begin{array}{l}
ReconcileNetworkChange(n,\ c) \triangleq \\
\quad \land\ leader[n] \\
\quad \land\ networkChange[c].state = Pending \\
\quad \land\ \lor\ \land\ \neg HasDeviceChanges(c) \\
\qquad\quad \land\ CreateDeviceChanges(c) \\
\qquad\quad \land\ \text{UNCHANGED } \langle networkChange \rangle \\
\qquad \lor\ \land\ HasDeviceChanges(c) \\
\qquad\quad \land\ \lor\ \land\ networkChange[c].phase = Change \\
\qquad\qquad\quad \land\ \lor\ \land\ CanApplyNetworkChange(c) \\
\qquad\qquad\qquad\quad \land\ networkChange' = [networkChange \text{ EXCEPT} \\
\qquad\qquad\qquad\qquad\quad ![c].incarnation = networkChange[c].incarnation + 1] \\
\qquad\qquad\qquad\quad \land\ \text{UNCHANGED } \langle deviceChange \rangle \\
\qquad\qquad\qquad \lor\ \land\ DeviceChangesComplete(c) \\
\qquad\qquad\qquad\quad \land\ networkChange' = [networkChange \text{ EXCEPT} \\
\qquad\qquad\qquad\qquad\quad ![c].state = Complete] \\
\qquad\qquad\qquad\quad \land\ \text{UNCHANGED } \langle deviceChange \rangle \\
\qquad\qquad\qquad \lor\ \land\ HasFailedDeviceChanges(c) \\
\qquad\qquad\qquad\quad \land\ RollbackDeviceChanges(c) \\
\qquad\qquad\qquad\quad \land\ \text{UNCHANGED } \langle networkChange \rangle \\
\qquad\qquad \text{TODO} \\
\qquad\qquad \lor\ \land\ networkChange[c].phase = Rollback \\
\qquad\qquad\quad \land\ networkChange' = [networkChange \text{ EXCEPT} \\
\qquad\qquad\qquad\quad ![c].state\ \ \ = Complete] \\
\qquad\qquad\quad \land\ \text{UNCHANGED } \langle deviceChange \rangle \\
\quad \land\ \text{UNCHANGED } \langle nodeVars,\ deviceVars,\ constraintVars \rangle
\end{array}
$$

This section models the *DeviceChange* reconciler.

$$
\begin{array}{l}
ReconcileDeviceChange(n,\ d,\ c) \triangleq \\
\quad \land\ master[n][d] \\
\quad \land\ deviceChange[d][c].state = Pending \\
\quad \land\ deviceChange[d][c].incarnation > 0 \\
\quad \land\ \lor\ \land\ deviceState[d] = Connected \\
\qquad\quad \land\ deviceChange' = [deviceChange \text{ EXCEPT} \\
\qquad\qquad\quad ![d] = [deviceChange[d] \text{ EXCEPT } ![c].state = Complete]] \\
\qquad \lor\ \land\ deviceState[d] = Disconnected \\
\qquad\quad \land\ deviceChange' = [deviceChange \text{ EXCEPT} \\
\qquad\qquad\quad ![d] = [deviceChange[d] \text{ EXCEPT } ![c].state = Failed]] \\
\quad \land\ \text{UNCHANGED } \langle nodeVars,\ networkChange,\ deviceVars,\ constraintVars \rangle
\end{array}
$$

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

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

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

$\exists\, d \in Device :$
$\exists\, c \in \text{DOMAIN } deviceChange[d] :$
$ReconcileDeviceChange(n,\ d,\ c)$
$\lor \exists\, d \in Device :$
$ConnectDevice(d)$
$\lor \exists\, d \in Device :$
$DisconnectDevice(d)$

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

* Modification History
* Last modified *Wed Sep* 22 13:23:25 *PDT* 2021 by *jordanhalterman*
* Created *Wed Sep* 22 13:22:32 *PDT* 2021 by *jordanhalterman*