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 is being applied to the network
CONSTANT *Applying*

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

Indicates that a configuration change was successful
CONSTANT *Succeeded*

Indicates that a configuration change failed
CONSTANT *Failed*

The set of all nodes
CONSTANT *Node*

The set of all devices
CONSTANT *Device*

The set of all possible configuration changes
CONSTANT *Change*

An empty constant
CONSTANT *Nil*

Per-node election state
VARIABLE *nodeState*

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

Store of network-wide configuration changes
VARIABLE *networkChange*

Store of device configuration changes
VARIABLE *deviceChange*

Leader election

Sets the current leader for the node
$$SetNodeLeader(n,\ l) \triangleq$$
$$\land\ nodeState' = [nodeState \text{ EXCEPT } ![n] = n = l]$$
$$\land\ \text{UNCHANGED } \langle\rangle$$

```
Sets the current leader for a device
SetDeviceLeader(n, d, l) ≜
    ∧ deviceState' = [deviceState EXCEPT ![n] = [deviceState[n] EXCEPT ![d] = n = l]]
    ∧ UNCHANGED ⟨nodeState, networkChange, deviceChange⟩
```

```
Northbound API
Configure(c) ≜
    ∧ networkChange' = Append(networkChange, [changes ↦ c, status ↦ Pending])
    ∧ UNCHANGED ⟨nodeState, deviceState, deviceChange⟩
```

```
Network configuration change scheduler
 Node 'n' handles a network configuration change event 'c'
NetworkSchedulerNetworkChange(n, c) ≜
    ∧ nodeState[n] = TRUE  Verify this node is the leader
    ∧ LET change ≜ networkChange[c] IN
          If the change does not intersect with the set of all pending/applied changes
          prior to the change then set the change status to Applying
        LET changeDevices ≜ DOMAIN change.changes
            priorDevices ≜ {d ∈ deviceChange : {i ∈ DOMAIN deviceChange[d] :
                                  i < c ∧ deviceChange[d][i].status ∈ {Pending, Applying}}}
        IN
            IF Cardinality(changeDevices ∩ priorDevices) = 0 THEN
                ∧ networkChange' = [networkChange EXCEPT ![c].status = Applying]
            ELSE
                ∧ UNCHANGED ⟨networkChange⟩
    ∧ UNCHANGED ⟨nodeState, deviceState, deviceChange⟩
```

```
Network configuration controller
 Adds or updates a device change
SetDeviceChange(d, c, s) ≜
    LET change ≜ networkChange[c] IN
        IF d ∈ DOMAIN change.changes THEN
            IF Cardinality({x ∈ DOMAIN deviceChange[d] : deviceChange[d][x].id = c}) = 0 THEN
                Append(deviceChange[d], [change.changes[d] EXCEPT !.id = c, !.network = c, !.status = s])
            ELSE
                [deviceChange EXCEPT ![CHOOSE x ∈ DOMAIN deviceChange[d] :
                        deviceChange[d][x].id = c].status = s]
        ELSE
            deviceChange[d]

 Node 'n' handles a network configuration change 'c'
NetworkControllerNetworkChange(n, c) ≜
    ∧ nodeState[n] = TRUE
```

```
    ∧ LET change ≜ networkChange[c]IN
          ∨ ∧ change.status = Pending
            ∧ deviceChange′ = [d ∈ Device ↦ SetDeviceChange(d, c, Pending)]
          ∨ ∧ change.status = Applying
            ∧ deviceChange′ = [d ∈ Device ↦ SetDeviceChange(d, c, Applying)]
            ∧ Cardinality(DOMAIN change.changes ∩ {d ∈ deviceChange
                                  : {i ∈ DOMAIN deviceChange[d]
                                      : deviceChange[d][i].network = c
                                      ∧ deviceChange[d][i].status  = Applying}}) < Cardinality(change.change
            ∧ deviceChange′ = [d ∈ Device ↦ IF d ∈ DOMAIN change.changes THEN
                                  Append(deviceChange[d], [change.changes[d] EXCEPT
                                                              !.network = c,
                                                              !.status = Applying])
                                                          ELSE deviceChange[d]]
          ∨ ∧ change.status = Complete
            ∧ UNCHANGED ⟨deviceChange⟩
    ∧ UNCHANGED ⟨nodeState, deviceState, networkChange⟩

 Node 'n' handles a device configuration change 'c'
NetworkControllerDeviceChange(n, d, c) ≜
    ∧ nodeState[n] = TRUE
    ∧ LET change ≜ deviceChange[d][c]
      IN
          ∨ ∧ deviceChange.status = Complete
            ∧ LET netChange ≜ networkChange[change.network]
                  completed ≜ {x ∈ DOMAIN netChange.changes :
                                  deviceChange[x][CHOOSE i ∈ DOMAIN deviceChange[x] :
                                      deviceChange[x][i].network = change.network].status = Complete}
                  succeeded ≜ {x ∈ DOMAIN netChange.changes :
                                  deviceChange[x][CHOOSE i ∈ DOMAIN deviceChange[x] :
                                      deviceChange[x][i].network = change.network].result = Succeeded}
              IN
                  ∨ ∧ Cardinality(completed) = Cardinality(netChange.changes)
                    ∧ IF Cardinality(succeeded) = Cardinality(completed) THEN
                          networkChange′ = [networkChange EXCEPT ![change.network] = [
                                              networkChange[change.network] EXCEPT
                                                  !.status = Complete, !.result = Succeeded]]
                      ELSE
                          networkChange′ = [networkChange EXCEPT ![change.network] = [
                                              networkChange[change.network] EXCEPT
                                                  !.status = Complete, !.result = Failed]]
          ∨ ∧ change.status ≠ Complete
            ∧ UNCHANGED ⟨networkChange⟩
    ∧ UNCHANGED ⟨nodeState, deviceState, deviceChange⟩
```

Device configuration controller

Node 'n' handles a device configuration change event 'c'

$$
\begin{array}{l}
DeviceControllerDeviceChange(n,\ d,\ c) \triangleq \\
\quad \land\ deviceState[n][d] = \text{TRUE} \\
\quad \land\ \text{LET}\ change \triangleq deviceChange[d][c] \\
\quad\ \ \text{IN} \\
\qquad \lor\ \land\ change.status = Applying \\
\qquad\quad \land\ deviceChange' = [deviceChange\ \text{EXCEPT}\ ![d] = [deviceChange[d]\ \text{EXCEPT}\ ![c] = [\\
\qquad\qquad\qquad\qquad deviceChange[d][c]\ \text{EXCEPT}\ !.status = Complete,\ !.result = Succeeded]]] \\
\qquad \lor\ \land\ change.status \neq Applying \\
\qquad\quad \land\ \text{UNCHANGED}\ \langle deviceChange \rangle \\
\quad \land\ \text{UNCHANGED}\ \langle nodeState,\ deviceState,\ networkChange \rangle
\end{array}
$$

Init and next state predicates

$$
\begin{array}{l}
Init \triangleq \\
\quad \land\ nodeState = [n \in Node \mapsto \text{FALSE}] \\
\quad \land\ deviceState = [n \in Node \mapsto [d \in Device \mapsto \text{FALSE}]] \\
\quad \land\ networkChange = \langle\rangle \\
\quad \land\ deviceChange = [n \in Device \mapsto \langle\rangle]
\end{array}
$$

$$
\begin{array}{l}
Next \triangleq \\
\quad \lor\ \exists\, d \in \text{SUBSET}\ Device : \exists\, c \in Change : Configure([d \to c]) \\
\quad \lor\ \exists\, n \in Node : \\
\qquad \exists\, l \in Node : \\
\qquad\ \ SetNodeLeader(n,\ l) \\
\quad \lor\ \exists\, n \in Node : \\
\qquad \exists\, d \in Device : \\
\qquad\ \ \exists\, l \in Node : \\
\qquad\quad SetDeviceLeader(n,\ d,\ l) \\
\quad \lor\ \exists\, n \in Node : \\
\qquad \exists\, c \in \text{DOMAIN}\ networkChange : \\
\qquad\ \ NetworkSchedulerNetworkChange(n,\ c) \\
\quad \lor\ \exists\, n \in Node : \\
\qquad \exists\, c \in \text{DOMAIN}\ networkChange : \\
\qquad\ \ NetworkControllerNetworkChange(n,\ c) \\
\quad \lor\ \exists\, n \in Node : \\
\qquad \exists\, d \in Device : \\
\qquad\ \ \exists\, c \in \text{DOMAIN}\ deviceChange[d] : \\
\qquad\quad NetworkControllerDeviceChange(n,\ d,\ c) \\
\quad \lor\ \exists\, n \in Node : \\
\qquad \exists\, d \in Device : \\
\qquad\ \ \exists\, c \in \text{DOMAIN}\ deviceChange[d] : \\
\qquad\ \ DeviceControllerDeviceChange(n,\ d,\ c)
\end{array}
$$

\ * Modification History
\ * Last modified Sat *Sep* 28 02:27:30 *PDT* 2019 by *jordanhalterman*
\ * Created *Fri Sep* 27 13:14:24 *PDT* 2019 by *jordanhalterman*