MODULE *Config*

INSTANCE *Naturals*

INSTANCE *FiniteSets*

INSTANCE *Sequences*

INSTANCE *TLC*

CONSTANT *TraceEnabled*, *TraceDepth*

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*

An empty constant
CONSTANT *Nil*

The set of all nodes
CONSTANT *Node*

The set of all devices
CONSTANT *Device*

ASSUME *Pending* $\in$ STRING
ASSUME *Complete* $\in$ STRING
ASSUME *Failed* $\in$ STRING
ASSUME *Rollback* $\in$ STRING

ASSUME $Connected \in \text{STRING}$
ASSUME $Disconnected \in \text{STRING}$
ASSUME $Error \in \text{STRING}$
ASSUME $Nil \in \text{STRING}$

ASSUME $\land\ IsFiniteSet(Node)$
$\land\ \forall\, n \in Node :$
$\land\ n \notin Device$
$\land\ n \in \text{STRING}$
ASSUME $\land\ IsFiniteSet(Device)$
$\land\ \forall\, d \in Device :$
$\land\ d \notin Node$
$\land\ d \in \text{STRING}$

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 *networkChanges*

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 *deviceChanges*

A record of device states - either *Connected* or *Disconnected*
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

$configVars \triangleq \langle networkChanges,\ deviceChanges \rangle$

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

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

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

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

$$PriorNetworkChanges(c) \triangleq \{n \in \text{DOMAIN } networkChanges : n < c\}$$

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

$$NetworkCompletedChanges(c) \triangleq \{d \in \text{DOMAIN } networkChanges[c].changes : \\ \land c \in \text{DOMAIN } deviceChanges[d] \\ \land deviceChanges[d][c].state = Complete\}$$

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

$$NetworkChangesComplete(c) \triangleq Cardinality(NetworkCompletedChanges(c)) = Cardinality(\text{DOMAIN } networkChanges[c].changes)$$

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

$$PriorIncompleteDevices(c) \triangleq \text{UNION } \{\text{DOMAIN } networkChanges[n].changes : \\ n \in \{n \in PriorNetworkChanges(c) : \neg NetworkChangesComplete(n)\}\}$$

Return the set of all devices configured by network change c

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

Return a boolean indicating whether network change 'c' is complete

$$IsCompleteNetworkChange(c) \triangleq \\ \land networkChanges[c].phase = Change \\ \land networkChanges[c].state = Complete$$

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

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

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

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

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 deviceChanges[d]
    ∧ deviceChanges[d][c].incarnation = networkChanges[c].incarnation

 Return a boolean indicating whether device changes have been propagated
 for the given network change
HasDeviceChanges(c) ≜
    Cardinality({d ∈ DOMAIN networkChanges[c].changes : HasDeviceChange(d, c)}) =
        Cardinality(DOMAIN networkChanges[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 d ∈ DOMAIN networkChanges[c].changes THEN
        IF c ∈ DOMAIN deviceChanges[d] THEN
            IF deviceChanges[d][c].state = Complete THEN
                deviceChanges[d]
            ELSE
                [deviceChanges[d] EXCEPT ![c].incarnation = networkChanges[c].incarnation,
                                         ![c].state = Pending]
        ELSE
            [x ∈ {c} ↦ [
                phase       ↦ networkChanges[c].phase,
                state       ↦ Pending,
                value       ↦ networkChanges[c].value,
                incarnation ↦ networkChanges[c].incarnation]] @@ deviceChanges[d]
    ELSE
        deviceChanges[d]

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

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

 Roll back device changes
```

```
RollbackDeviceChanges(c) ≜
    deviceChanges' = [d ∈ DOMAIN deviceChanges ↦ RollbackDeviceChange(d, c)]

 Return a boolean indicating whether the given device change is Failed
IsFailedDeviceChange(d, c) ≜
    ∧ c ∈ DOMAIN deviceChanges[d]
    ∧ deviceChanges[d][c].incarnation = networkChanges[c].incarnation
    ∧ deviceChanges[d][c].state = Failed

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

 Return a boolean indicating whether any device change is Failed for the given network change
HasFailedDeviceChanges(c) ≜
    Cardinality({d ∈ DOMAIN networkChanges[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 networkChanges[c].changes :
        IsCompleteDeviceChange(d, c)}) =
            Cardinality(DOMAIN networkChanges[c].changes)

 Reconcile a network change state
ReconcileNetworkChange(n, c) ≜
    ∧ leader[n]
    ∧ networkChanges[c].state = Pending
    ∧ ∨ ∧ ¬HasDeviceChanges(c)
        ∧ CreateDeviceChanges(c)
        ∧ UNCHANGED ⟨networkChanges⟩
      ∨ ∧ HasDeviceChanges(c)
        ∧ ∨ ∧ networkChanges[c].phase = Change
            ∧ ∨ ∧ CanApplyNetworkChange(c)
                ∧ networkChanges' = [networkChanges EXCEPT
                      ![c].incarnation = networkChanges[c].incarnation + 1]
                ∧ UNCHANGED ⟨deviceChanges⟩
              ∨ ∧ DeviceChangesComplete(c)
                ∧ networkChanges' = [networkChanges EXCEPT
                      ![c].state = Complete]
                ∧ UNCHANGED ⟨deviceChanges⟩
              ∨ ∧ HasFailedDeviceChanges(c)
                ∧ RollbackDeviceChanges(c)
```

$\land$ UNCHANGED $\langle networkChanges \rangle$

TODO

$\lor \land networkChanges[c].phase = Rollback$
$\land networkChanges' = [networkChanges$ EXCEPT
$![c].state = Complete]$
$\land$ UNCHANGED $\langle deviceChanges \rangle$

$\land$ UNCHANGED $\langle nodeVars, deviceVars, constraintVars \rangle$

This section models the *DeviceChange* reconciler.

$ReconcileDeviceChange(n, d, c) \triangleq$
$\land master[n][d]$
$\land deviceChanges[d][c].state = Pending$
$\land deviceChanges[d][c].incarnation > 0$
$\land \lor \land deviceState[d] = Connected$
$\land deviceChanges' = [deviceChanges$ EXCEPT
$![d] = [deviceChanges[d]$ EXCEPT $![c].state = Complete]]$
$\lor \land deviceState[d] = Disconnected$
$\land deviceChanges' = [deviceChanges$ EXCEPT
$![d] = [deviceChanges[d]$ EXCEPT $![c].state = Failed]]$
$\land$ UNCHANGED $\langle nodeVars, networkChanges, deviceVars, constraintVars \rangle$

This section models device states. Devices begin in the *Disconnected* state and can only be configured while in the *Connected* state.

Set device d state to *Connected*

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

Set device d state to *Disconnected*

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

Init and next state predicates

$Init \triangleq$
$\land leader = [n \in Node \mapsto$ FALSE$]$
$\land master = [n \in Node \mapsto [d \in Device \mapsto$ FALSE$]]$
$\land networkChanges = \langle \rangle$
$\land deviceChanges = [d \in Device \mapsto [x \in \{\} \mapsto [phase \mapsto Change, state \mapsto Pending]]]$
$\land deviceState = [d \in Device \mapsto Disconnected]$

$\land\ electionCount = 0$
$\land\ configCount = 0$
$\land\ connectionCount = 0$

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

$TraceNext\ \triangleq$
$\lor\ Next$
$\lor\ \land\ \neg\text{ENABLED}\ Next$
$\quad\ \land\ \text{UNCHANGED}\ \langle vars \rangle$

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

$Inv\ \triangleq\ \forall\, c1 \in \text{DOMAIN}\ networkChanges :$
$\quad \forall\, c2 \in \text{DOMAIN}\ networkChanges :$
$\quad\ \text{LET}\ d1 \triangleq \text{DOMAIN}\ networkChanges[c1].changes$
$\quad\quad\ s1 \triangleq networkChanges[c1].state$
$\quad\quad\ d2 \triangleq \text{DOMAIN}\ networkChanges[c2].changes$
$\quad\quad\ s2 \triangleq networkChanges[c2].state$
$\quad\ \text{IN}$

$$(c1 > c2 \land d1 \in \text{SUBSET } d2 \land s1 = Complete) \Rightarrow s2 = Complete$$

$$Term \triangleq \Diamond(\forall\, c \in \text{DOMAIN } networkChanges : IsCompleteNetworkChange(c))$$

* Modification History
* Last modified *Mon Jan* 03 01:43:02 *PST* 2022 by *jordanhalterman*
* Created *Wed Sep* 22 13:22:32 *PDT* 2021 by *jordanhalterman*