This module provides the model values and safety and liveness properties for the model of µ*ONOS Config* controllers.

MODULE *Config*

INSTANCE *Naturals*

INSTANCE *FiniteSets*

INSTANCE *Sequences*

INSTANCE *TLC*

GenerateTestCases $\triangleq$ TRUE

Nil $\triangleq$ "<nil>"

Change $\triangleq$ "Change"
Rollback $\triangleq$ "Rollback"

Commit $\triangleq$ "Commit"
Apply $\triangleq$ "Apply"

Pending $\triangleq$ "Pending"
InProgress $\triangleq$ "InProgress"
Complete $\triangleq$ "Complete"
Aborted $\triangleq$ "Aborted"
Canceled $\triangleq$ "Canceled"
Failed $\triangleq$ "Failed"

Node $\triangleq$ {"node1"}

NumTransactions $\triangleq$ 3
NumTerms $\triangleq$ 1
NumConns $\triangleq$ 1
NumStarts $\triangleq$ 1

Path $\triangleq$ {"path1"}
Value $\triangleq$ {"value1", "value2"}

A transaction log.
VARIABLE *transactions*

A record of per-target configurations
VARIABLE *configuration*

A record of target masterships

VARIABLE *mastership*

A record of node connections to the target
VARIABLE *conns*

The target state
VARIABLE *target*

A sequence of state changes used for model checking.
VARIABLE *history*

$vars \triangleq \langle transactions,\ configuration,\ mastership,\ conns,\ target,\ history \rangle$

µ*ONOS Config* coordinates configuration changes using a collection of four controllers, each of which is responsible for managing one type of data. The four controllers are specified in a separate module for each, and they're imported here for use in the model.

LOCAL $Transaction \triangleq$ INSTANCE *Transaction*

LOCAL $Configuration \triangleq$ INSTANCE *Configuration*

LOCAL $Mastership \triangleq$ INSTANCE *Mastership*

LOCAL $Target \triangleq$ INSTANCE *Target*

This section defines the state changes that can occur across all the µ*ONOS Config* controllers.

$AppendChange(i) \triangleq$
$\quad \land\ Transaction!AppendChange(i)$

$RollbackChange(i) \triangleq$
$\quad \land\ Transaction!RollbackChange(i)$

$ReconcileTransaction(n,\ i) \triangleq$
$\quad \land\ i \in$ DOMAIN $transactions$
$\quad \land\ \lor\ \land\ Transaction!ReconcileTransaction(n,\ i)$
$\qquad\qquad \land\ GenerateTestCases \Rightarrow Transaction!Test!Log([node \mapsto n,\ index \mapsto i])$
$\qquad \lor\ \land\ GenerateTestCases$
$\qquad\qquad \land\ \neg$ENABLED $Transaction!ReconcileTransaction(n,\ i)$
$\qquad\qquad \land$ UNCHANGED $vars$
$\qquad\qquad \land\ Transaction!Test!Log([node \mapsto n,\ index \mapsto i])$

$ReconcileConfiguration(n) \triangleq$
$\quad \lor\ \land\ Configuration!ReconcileConfiguration(n)$
$\qquad \land$ UNCHANGED $\langle transactions,\ history \rangle$
$\qquad \land\ GenerateTestCases \Rightarrow Configuration!Test!Log([node \mapsto n])$
$\quad \lor\ \land\ GenerateTestCases$
$\qquad \land\ \neg$ENABLED $Configuration!ReconcileConfiguration(n)$

$\land$ UNCHANGED $vars$
$\land$ $Configuration!Test!Log([node \mapsto n])$

$ReconcileMastership(n) \triangleq$
$\lor \land Mastership!ReconcileMastership(n)$
$\land$ UNCHANGED $\langle transactions, configuration, target, history\rangle$
$\land GenerateTestCases \Rightarrow Mastership!Test!Log([node \mapsto n])$
$\lor \land GenerateTestCases$
$\land \neg$ENABLED $Mastership!ReconcileMastership(n)$
$\land$ UNCHANGED $vars$
$\land Mastership!Test!Log([node \mapsto n])$

$ConnectNode(n) \triangleq$
$\land Target!Connect(n)$
$\land$ UNCHANGED $\langle transactions, configuration, mastership, history\rangle$

$DisconnectNode(n) \triangleq$
$\land Target!Disconnect(n)$
$\land$ UNCHANGED $\langle transactions, configuration, mastership, history\rangle$

$StartTarget \triangleq$
$\land Target!Start$
$\land$ UNCHANGED $\langle transactions, configuration, mastership, history\rangle$

$StopTarget \triangleq$
$\land Target!Stop$
$\land$ UNCHANGED $\langle transactions, configuration, mastership, history\rangle$

Formal specification, constraints, and theorems.

$Init \triangleq$
$\land transactions = [$
$i \in \{\} \mapsto [$
$phase \mapsto Nil,$
$values \mapsto [$
$p \in \{\} \mapsto Nil],$
$change \mapsto [$
$commit \mapsto Nil,$
$apply \mapsto Nil],$
$rollback \mapsto [$
$commit \mapsto Nil,$
$apply \mapsto Nil]]]$
$\land configuration = [$
$state \mapsto Pending,$
$term \mapsto 0,$
$committed \mapsto [$

```
                index     ↦ 0,
                change    ↦ 0,
                target    ↦ 0,
                ordinal   ↦ 0,
                revision  ↦ 0,
                values    ↦ [
                    p ∈ {} ↦ Nil]],
            applied ↦ [
                index     ↦ 0,
                target    ↦ 0,
                ordinal   ↦ 0,
                revision  ↦ 0,
                values    ↦ [
                    p ∈ {} ↦ Nil]]]
    ∧ target = [
            id         ↦ 1,
            running ↦ TRUE,
            values    ↦ [
                p ∈ {}  ↦ [
                    index ↦ 0,
                    value ↦ Nil]]]
    ∧ mastership = [
            master ↦ CHOOSE n ∈ Node : TRUE,
            term    ↦ 1,
            conn    ↦ 1]
    ∧ conns = [
            n ∈ Node ↦ [
                id            ↦ 1,
                connected ↦ TRUE]]
    ∧ history = ⟨⟩

Next ≜
    ∨ ∃ i ∈ 1 .. NumTransactions :
        ∨ AppendChange(i)
        ∨ RollbackChange(i)
    ∨ ∃ n ∈ Node, i ∈ 1 .. NumTransactions :
        ReconcileTransaction(n, i)
    ∨ ∃ n ∈ Node :
        ReconcileConfiguration(n)
    ∨ ∃ n ∈ Node :
        ReconcileMastership(n)
    ∨ ∃ n ∈ Node :
        ∨ ConnectNode(n)
        ∨ DisconnectNode(n)
    ∨ StartTarget
```

$\lor$ *StopTarget*

Spec $\triangleq$
$\land$ *Init*
$\land \Box[Next]_{vars}$
$\land \forall\, i \in 1 \,..\, NumTransactions$:
WF$_{\langle transactions \rangle}$(*Transaction*!*RollbackChange*(i))
$\land \forall\, n \in Node,\ i \in 1 \,..\, NumTransactions$:
WF$_{\langle transactions,\ configuration,\ mastership,\ conns,\ target,\ history \rangle}$(*Transaction*!*ReconcileTransaction*($n,\ i$))
$\land \forall\, n \in Node$:
WF$_{\langle configuration,\ mastership,\ conns,\ target \rangle}$(*Configuration*!*ReconcileConfiguration*(n))
$\land \forall\, n \in Node$:
WF$_{\langle mastership,\ conns \rangle}$(*Mastership*!*ReconcileMastership*(n))
$\land \forall\, n \in Node$:
WF$_{\langle conns,\ target \rangle}$(*Target*!*Connect*($n$) $\lor$ *Target*!*Disconnect*(n))
$\land$ WF$_{\langle conns,\ target \rangle}$(*Target*!*Start* $\lor$ *Target*!*Stop*)

This section contains state constraints used for model checking.

LimitTerms $\triangleq$
$\lor$ *mastership.term* $<$ *NumTerms*
$\lor \land$ *mastership.term* $=$ *NumTerms*
$\land$ *mastership.master* $\neq$ *Nil*

LimitConns $\triangleq$
$\forall\, n \in$ DOMAIN *conns* :
$\lor$ *conns*[n].*id* $<$ *NumConns*
$\lor \land$ *conns*[n].*id* $=$ *NumConns*
$\land$ *conns*[n].*connected*

LimitStarts $\triangleq$
$\lor$ *target.id* < 2
$\lor \land$ *target.id* $= 2$
$\land$ *target.running*

TypeOK $\triangleq$
$\land$ *Transaction*!*TypeOK*
$\land$ *Configuration*!*TypeOK*
$\land$ *Mastership*!*TypeOK*

This section contains invariants, action properties, and liveness properties used the model to verify the spec preserves order and consistency guarantees, always eventually terminates, and is deadlock free.

StatusCommitted(i) $\triangleq$
$\land$ *Len*(*history*) $=$ *Len*(*history*$'$)

$\land \lor \land transactions'[i].change.commit \notin \{Pending,\ Canceled\}$
$\quad \land transactions[i].change.commit \neq transactions'[i].change.commit$
$\quad \lor \land transactions'[i].rollback.commit \notin \{Pending,\ Canceled\}$
$\quad \land transactions[i].rollback.commit \neq transactions'[i].rollback.commit$

$StatusApplied(i) \triangleq$
$\land Len(history) = Len(history')$
$\land \lor \land transactions'[i].change.apply \notin \{Pending,\ Canceled,\ Aborted\}$
$\quad \land transactions[i].change.apply \neq transactions'[i].change.apply$
$\quad \lor \land transactions'[i].rollback.apply \notin \{Pending,\ Canceled,\ Aborted\}$
$\quad \land transactions[i].rollback.apply \neq transactions'[i].rollback.apply$

$ValidStatus(t,\ i,\ j) \triangleq$
$\land j \in \text{DOMAIN } history$
$\land history[j].index = i$
$\land \lor \land history[j].phase = Change$
$\quad \land history[j].event = Commit$
$\quad \land t[i].change.commit = history[j].status$
$\quad \lor \land history[j].phase = Change$
$\quad \land history[j].event = Apply$
$\quad \land t[i].change.apply = history[j].status$
$\quad \lor \land history[j].phase = Rollback$
$\quad \land history[j].event = Commit$
$\quad \land t[i].rollback.commit = history[j].status$
$\quad \lor \land history[j].phase = Rollback$
$\quad \land history[j].event = Apply$
$\quad \land t[i].rollback.apply = history[j].status$

$ValidCommit(t,\ i) \triangleq$
$\text{LET } j \triangleq \text{CHOOSE } j \in \text{DOMAIN } history :$
$\qquad \land history[j].event = Commit$
$\qquad \land \neg\exists\, k \in \text{DOMAIN } history :$
$\qquad\quad \land history[k].event = Commit$
$\qquad\quad \land k > j$
$\text{IN} \quad ValidStatus(t,\ i,\ j)$

$ValidApply(t,\ i) \triangleq$
$\text{LET } j \triangleq \text{CHOOSE } j \in \text{DOMAIN } history :$
$\qquad \land history[j].event = Apply$
$\qquad \land \neg\exists\, k \in \text{DOMAIN } history :$
$\qquad\quad \land history[k].event = Apply$
$\qquad\quad \land k > j$
$\text{IN} \quad ValidStatus(t,\ i,\ j)$

$AtomicStatusChange \triangleq$
$\forall\, i \in 1\,..\,NumTransactions :$

$\land\ i \in \text{DOMAIN } transactions \Rightarrow$
$\quad \land\ StatusCommitted(i) \Rightarrow ValidCommit(transactions', i)$
$\quad \land\ StatusApplied(i) \Rightarrow ValidApply(transactions', i)$

$Transition \triangleq \Box[AtomicStatusChange]_{\langle transactions, history \rangle}$

LOCAL $IsOrderedChange(p, i) \triangleq$
$\land\ history[i].phase = Change$
$\land\ history[i].event = p$
$\land\ history[i].status = Complete$
$\land\ \neg\exists\, j \in \text{DOMAIN } history :$
$\quad \land\ j < i$
$\quad \land\ history[j].phase = Change$
$\quad \land\ history[j].event = p$
$\quad \land\ history[j].status = Complete$
$\quad \land\ history[j].index \geq history[i].index$

LOCAL $IsOrderedRollback(p, i) \triangleq$
$\land\ history[i].phase = Rollback$
$\land\ history[i].event = p$
$\land\ history[i].status = Complete$
$\land\ \exists\, j \in \text{DOMAIN } history :$
$\quad \land\ j < i$
$\quad \land\ history[j].phase = Change$
$\quad \land\ history[j].status = Complete$
$\quad \land\ history[j].index = history[i].index$
$\land\ \neg\exists\, j \in \text{DOMAIN } history :$
$\quad \land\ j < i$
$\quad \land\ history[j].phase = Change$
$\quad \land\ history[j].event = p$
$\quad \land\ history[j].status = Complete$
$\quad \land\ history[j].index > history[i].index$
$\quad \land\ \neg\exists\, k \in \text{DOMAIN } history :$
$\quad\quad \land\ k > j$
$\quad\quad \land\ k < i$
$\quad\quad \land\ history[k].phase = Rollback$
$\quad\quad \land\ history[k].event = p$
$\quad\quad \land\ history[j].status = Complete$
$\quad\quad \land\ history[k].index = history[j].index$

The *Order* invariant checks the recorded 'history' to ensure that changes and rollbacks are being committed and applied in consistent sequential order. Additionally, it enforces the invariant that if a rollback fails the apply phase, it and all subsequent pending transactions must be aborted and rolled back before a new transaction can be applied.

$Order \triangleq$
$\land\ \forall\, i \in \text{DOMAIN } history :$

```
        history[i].status = Complete ⇒
            ∨ IsOrderedChange(Commit, i)
            ∨ IsOrderedChange(Apply, i)
            ∨ IsOrderedRollback(Commit, i)
            ∨ IsOrderedRollback(Apply, i)
    ∧ ∀ i ∈ DOMAIN transactions :
        ∧ transactions[i].change.apply = Failed
        ∧ transactions[i].rollback.apply ≠ Complete
        ⇒ ¬∃ j ∈ DOMAIN transactions :
                ∧ j > i
                ∧ transactions[i].change.apply ∈ {InProgress, Complete}

LOCAL IsChangeCommitted(i) ≜
    ∧   configuration.committed.revision = i

LOCAL IsChangeApplied(i) ≜
    ∧   configuration.applied.revision = i
```

The *Consistency* invariant verifies that the changes and rollbacks are properly propagated to the configuration when committed and to targets once applied. Additionally, it checks that target configuration is restored following restarts.

```
Consistency ≜
    ∧ ∀ i ∈ DOMAIN transactions :
        ∧ IsChangeCommitted(i)
        ∧ ¬∃ j ∈ DOMAIN transactions :
                ∧ j > i
                ∧ IsChangeCommitted(j)
        ⇒ ∀ p ∈ DOMAIN transactions[i].change.values :
                ∧ configuration.committed.values[p] = transactions[i].change.values[p]
    ∧ ∀ i ∈ DOMAIN transactions :
        ∧ IsChangeApplied(i)
        ∧ ¬∃ j ∈ DOMAIN transactions :
                ∧ j > i
                ∧ IsChangeApplied(j)
        ⇒ ∀ p ∈ DOMAIN transactions[i].change.values :
                ∧ configuration.applied.values[p] = transactions[i].change.values[p]
                ∧ ∧ target.running
                  ∧ configuration.applied.target = target.id
                  ∧ configuration.state = Complete
                  ⇒ target.values[p] = transactions[i].change.values[p]
```

The *Safety* property holds with both *Order* and *Consistency*.

```
Safety ≜ □(Order ∧ Consistency)

THEOREM Spec ⇒ Safety

LOCAL IsChanging(i) ≜
```

$\land \quad i \in \text{DOMAIN } transactions$
$\land \quad transactions[i].phase = Change$

LOCAL $IsChanged(i) \triangleq$
$\land \quad i \in \text{DOMAIN } transactions$
$\land \quad transactions[i].change.commit \in \{Complete, Failed\}$
$\land \quad transactions[i].change.apply \in \{Complete, Aborted, Failed\}$

LOCAL $IsRollingBack(i) \triangleq$
$\land \quad i \in \text{DOMAIN } transactions$
$\land \quad transactions[i].phase = Rollback$

LOCAL $IsRolledBack(i) \triangleq$
$\land \quad i \in \text{DOMAIN } transactions$
$\land \quad transactions[i].rollback.commit \in \{Complete, Failed\}$
$\land \quad transactions[i].rollback.apply \in \{Complete, Aborted, Failed\}$

$Terminates(i) \triangleq$
$\land\, IsChanging(i) \leadsto IsChanged(i)$
$\land\, IsRollingBack(i) \leadsto IsRolledBack(i)$

The *Termination* property is a liveness check to verify no deadlocks or livelocks exist in the system. So long as the system can make progress, every change and every rollback is guaranteed to have a path to complete both phases at all times.

$Termination \triangleq$
$\forall\, i \in 1 \,..\, NumTransactions : Terminates(i)$

The *Liveness* property holds when *Termination* holds.

$Liveness \triangleq Termination$

THEOREM $Spec \Rightarrow Liveness$

Copyright 2023 *Intel* Corporation