MODULE *Transactions*

EXTENDS *Proposals*

INSTANCE *Naturals*

INSTANCE *FiniteSets*

INSTANCE *Sequences*

LOCAL INSTANCE *TLC*

Transaction type constants

CONSTANTS
TransactionChange,
TransactionRollback

Transaction isolation constants

CONSTANTS
ReadCommitted,
Serializable

Phase constants

CONSTANTS
TransactionInitialize,
TransactionValidate,
TransactionAbort,
TransactionCommit,
TransactionApply

Status constants

CONSTANTS
TransactionInProgress,
TransactionComplete,
TransactionFailed

State constants

CONSTANTS
TransactionPending,
TransactionValidated,
TransactionCommitted,
TransactionApplied,
TransactionAborted

A transaction log. Transactions may either request a set of changes to a set of targets or rollback a prior change.

VARIABLE *transaction*

LOCAL $InitState \triangleq$
$[transactions \mapsto transaction,$
$proposals \mapsto proposal]$

LOCAL $NextState \triangleq$
$[transactions \mapsto transaction',$
$proposals \mapsto proposal']$

LOCAL $Trace \triangleq$ INSTANCE $Trace$ WITH
$Module \leftarrow$ "Transactions",
$InitState \leftarrow InitState,$
$NextState \leftarrow NextState$

This section models the *Transaction* log reconciler.

Transactions come in two flavors: - *Change* transactions contain a set of changes to be applied to a set of targets - *Rollback* transactions reference a prior change transaction to be reverted to the previous state

Transacations proceed through a series of phases:
* *Initialize* - create and link *Proposals*
* *Validate* - validate changes and rollbacks
* *Commit* - commit changes to *Configurations*
* *Apply* - commit changes to Targets

Reconcile a transaction
$ReconcileTransaction(i) \triangleq$

Initialize is the only transaction phase that's globally serialized. While in the Initializing phase, the reconciler checks whether the prior transaction has been Initialized before creating *Proposals* in the *Initialize* phase. Once all of the transaction's proposals have been Initialized, the transaction will be marked Initialized. If any proposal is *Failed*, the transaction will be marked *Failed* as well.

$\land \lor \land transaction[i].phase = TransactionInitialize$
$\land \lor \land transaction[i].state = TransactionInProgress$

All prior transaction must be initialized before proceeding to initialize this transaction.

$\land \neg \exists j \in$ DOMAIN $transaction :$
$\land j < i$
$\land transaction[j].phase = TransactionInitialize$
$\land transaction[j].state = TransactionInProgress$

If the transaction's targets are not yet set, create proposals and add targets to the transaction state.

$\land \lor \land$ DOMAIN $transaction[i].targets = \{\}$

If the transaction is a change, the targets are taken

```
    from the change values.
∧ ∨ ∧ transaction[i].type = TransactionChange
    ∧ LET targets ≜ DOMAIN transaction[i].change
      IN
        ∧ proposal′ = [t ∈ DOMAIN proposal \ targets ↦ proposal[t]]
                      @@ [t ∈ targets ↦
                            LET p ≜ IF t ∈ DOMAIN proposal THEN proposal[t] ELSE ⟨⟩
                            IN  Append(p, [type      ↦ ProposalChange,
                                           index     ↦ i,
                                           change    ↦
                                             [index  ↦ i,
                                              values ↦ transaction[i].change[t]],
                                           rollback  ↦
                                             [index  ↦ 0],
                                           dependency ↦ [index ↦ 0],
                                           phase     ↦ ProposalInitialize,
                                           state     ↦ ProposalInProgress])]
        ∧ transaction′ = [transaction EXCEPT ![i].targets =
                            [t ∈ targets ↦ Len(proposal′[t])]]
    If the transaction is a rollback, the targets affected are
    the targets of the change transaction being rolled back.
∨ ∧ transaction[i].type = TransactionRollback
      If the rollback index is a valid Change transaction,
      initialize proposals for all of the Change targets.
    ∧ ∨ ∧ transaction[i].rollback ∈ DOMAIN transaction
        ∧ transaction[transaction[i].rollback].type = TransactionChange
        ∧ LET targets ≜ DOMAIN transaction[transaction[i].rollback].change
          IN
            ∧ proposal′ = [t ∈ DOMAIN proposal \ targets ↦ proposal[t]]
                          @@ [t ∈ targets ↦
                                LET p ≜ IF t ∈ DOMAIN proposal THEN proposal[t] EL
                                IN  Append(p, [type      ↦ ProposalRollback,
                                               index     ↦ i,
                                               change    ↦
                                                 [index  ↦ 0],
                                               rollback  ↦
                                                 [index  ↦ transaction[i].rollback],
                                               dependency ↦ [index ↦ 0],
                                               phase     ↦ ProposalInitialize,
                                               state     ↦ ProposalInProgress])]
            ∧ transaction′ = [transaction EXCEPT ![i].targets =
                                [t ∈ targets ↦ Len(proposal′[t])]]
      If the rollback index is not a valid Change transaction
      fail the Rollback transaction.
    ∨ ∧ ∨ ∧ transaction[i].rollback ∈ DOMAIN transaction
```

$\land\ transaction[transaction[i].rollback].type = TransactionRollback$
$\lor\ transaction[i].rollback \notin \text{DOMAIN}\ transaction$
$\land\ transaction' = [transaction\ \text{EXCEPT}\ ![i].state = TransactionFailed]$
$\land\ \text{UNCHANGED}\ \langle proposal \rangle$

If the transaction's proposals have been initialized, check proposals for completion or failures.

$\lor\ \land\ \text{DOMAIN}\ transaction[i].targets \neq \{\}$

If all proposals have been *Complete*, mark the transaction *Complete*.

$\land\ \lor\ \land\ \forall\, t \in \text{DOMAIN}\ transaction[i].targets :$
$\text{LET}\ p \triangleq transaction[i].targets[t]$
IN
$\land\ proposal[t][p].phase = ProposalInitialize$
$\land\ proposal[t][p].state\ = ProposalComplete$
$\land\ transaction' = [transaction\ \text{EXCEPT}\ ![i].state = TransactionComplete]$
$\land\ \text{UNCHANGED}\ \langle proposal \rangle$

If any proposal has been *Failed*, mark the transaction *Failed*.

$\lor\ \land\ \exists\, t \in \text{DOMAIN}\ transaction[i].targets :$
$\text{LET}\ p \triangleq transaction[i].targets[t]$
IN
$\land\ proposal[t][p].phase = ProposalInitialize$
$\land\ proposal[t][p].state\ = ProposalFailed$
$\land\ transaction' = [transaction\ \text{EXCEPT}\ ![i].state = TransactionFailed]$
$\land\ \text{UNCHANGED}\ \langle proposal \rangle$

Once the transaction has been Initialized, proceed to the *Validate* phase. If any of the transaction's proposals depend on a *Serializable* transaction, verify the dependency has been *Validated* to preserve serializability before moving the transaction to the *Validate* phase.

$\lor\ \land\ transaction[i].state = TransactionComplete$
$\land\ \forall\, t \in \text{DOMAIN}\ transaction[i].targets :$
$\text{LET}\ p \triangleq transaction[i].targets[t]$
IN
$\land\ proposal[t][p].dependency.index \in \text{DOMAIN}\ transaction$
$\land\ transaction[proposal[t][p].dependency.index].isolation = Serializable$
$\Rightarrow\ transaction[proposal[t][p].dependency.index].status$
$\in \{TransactionValidated,\ TransactionCommitted,\ TransactionApplied,\ Transactio$
$\land\ transaction' = [transaction\ \text{EXCEPT}\ ![i].phase = TransactionValidate,$
$![i].state\ = TransactionInProgress]$
$\land\ \text{UNCHANGED}\ \langle proposal \rangle$

If the transaction failed initialization, proceed to the *Abort* phase to ensure indexes are still updated for the target configurations.

$\lor\ \land\ transaction[i].state = TransactionFailed$
$\land\ transaction' = [transaction\ \text{EXCEPT}\ ![i].phase = TransactionAbort,$
$![i].state\ = TransactionInProgress]$
$\land\ \text{UNCHANGED}\ \langle proposal \rangle$

$\lor\ \land\ transaction[i].phase = TransactionValidate$

```
∧ ∨ ∧ transaction[i].state = TransactionInProgress
      Move the transaction's proposals to the Validating state
    ∧ ∨ ∧ ∃ t ∈ DOMAIN transaction[i].targets :
          LET p ≜ transaction[i].targets[t]
          IN
              ∧ proposal[t][p].phase ≠ ProposalValidate
              ∧ proposal' = [proposal EXCEPT ![t] =
                                 [proposal[t] EXCEPT ![p].phase = ProposalValidate,
                                                     ![p].state  = ProposalInProgress]]
        ∧ UNCHANGED ⟨transaction⟩
      If all proposals have been Complete, mark the transaction Complete.
      ∨ ∧ ∀ t ∈ DOMAIN transaction[i].targets :
          LET p ≜ transaction[i].targets[t]
          IN
              ∧ proposal[t][p].phase = ProposalValidate
              ∧ proposal[t][p].state  = ProposalComplete
        ∧ transaction' = [transaction EXCEPT ![i].state  = TransactionComplete,
                                             ![i].status = TransactionValidated]
        ∧ UNCHANGED ⟨proposal⟩
      If any proposal has been Failed, mark the transaction Failed.
      ∨ ∧ ∃ t ∈ DOMAIN transaction[i].targets :
          LET p ≜ transaction[i].targets[t]
          IN
              ∧ proposal[t][p].phase = ProposalValidate
              ∧ proposal[t][p].state  = ProposalFailed
        ∧ transaction' = [transaction EXCEPT ![i].state = TransactionFailed]
        ∧ UNCHANGED ⟨proposal⟩
  Once the transaction has been Validated, proceed to the Commit phase.
  If any of the transaction's proposals depend on a Serializable transaction,
  verify the dependency has been Committed to preserve serializability before
  moving the transaction to the Commit phase.
  ∨ ∧ transaction[i].state = TransactionComplete
    ∧ ∀ t ∈ DOMAIN transaction[i].targets :
        LET p ≜ transaction[i].targets[t]
        IN
            ∧ proposal[t][p].dependency.index ∈ DOMAIN transaction
            ∧ transaction[proposal[t][p].dependency.index].isolation = Serializable
            ⇒ transaction[proposal[t][p].dependency.index].status
                  ∈ {TransactionCommitted, TransactionApplied, TransactionAborted}
    ∧ transaction' = [transaction EXCEPT ![i].phase = TransactionCommit,
                                         ![i].state  = TransactionInProgress]
    ∧ UNCHANGED ⟨proposal⟩
  If the transaction failed validation, proceed to the Abort phase
  to ensure indexes are still updated for the target configurations.
  ∨ ∧ transaction[i].state = TransactionFailed
```

```
        ∧ transaction′ = [transaction EXCEPT ![i].phase = TransactionAbort,
                                             ![i].state  = TransactionInProgress]
        ∧ UNCHANGED ⟨proposal⟩
∨ ∧ transaction[i].phase = TransactionCommit
  ∧ ∨ ∧ transaction[i].state = TransactionInProgress
      Move the transaction's proposals to the Committing state
    ∧ ∨ ∧ ∃ t ∈ DOMAIN transaction[i].targets :
          LET p ≜ transaction[i].targets[t]
          IN
              ∧ proposal[t][p].phase ≠ ProposalCommit
              ∧ proposal′ = [proposal EXCEPT ![t] =
                                [proposal[t] EXCEPT ![p].phase = ProposalCommit,
                                                    ![p].state  = ProposalInProgress]]
        ∧ UNCHANGED ⟨transaction⟩
      If all proposals have been Complete, mark the transaction Complete.
      ∨ ∧ ∀ t ∈ DOMAIN transaction[i].targets :
            LET p ≜ transaction[i].targets[t]
            IN
                ∧ proposal[t][p].phase = ProposalCommit
                ∧ proposal[t][p].state  = ProposalComplete
        ∧ transaction′ = [transaction EXCEPT ![i].state  = TransactionComplete,
                                             ![i].status = TransactionCommitted]
        ∧ UNCHANGED ⟨proposal⟩
    Once the transaction has been Committed, proceed to the Apply phase.
    If any of the transaction's proposals depend on a Serializable transaction,
    verify the dependency has been Applied to preserve serializability before
    moving the transaction to the Apply phase.
    ∨ ∧ transaction[i].state = TransactionComplete
      ∧ ∀ t ∈ DOMAIN transaction[i].targets :
          LET p ≜ transaction[i].targets[t]
          IN
              ∧ proposal[t][p].dependency.index ∈ DOMAIN transaction
              ∧ transaction[proposal[t][p].dependency.index].isolation = Serializable
              ⇒ transaction[proposal[t][p].dependency.index].status
                    ∈ {TransactionApplied, TransactionAborted}
      ∧ transaction′ = [transaction EXCEPT ![i].phase = TransactionApply,
                                           ![i].state  = TransactionInProgress]
      ∧ UNCHANGED ⟨proposal⟩
∨ ∧ transaction[i].phase = TransactionApply
  ∧ transaction[i].state  = TransactionInProgress
    Move the transaction's proposals to the Applying state
  ∧ ∨ ∧ ∃ t ∈ DOMAIN transaction[i].targets :
          LET p ≜ transaction[i].targets[t]
          IN
              ∧ proposal[t][p].phase ≠ ProposalApply
```

```
                    ∧ proposal′ = [proposal EXCEPT ![t] =
                                     [proposal[t] EXCEPT ![p].phase = ProposalApply,
                                                         ![p].state  = ProposalInProgress]]
                  ∧ UNCHANGED ⟨transaction⟩
              If all proposals have been Complete, mark the transaction Complete.
              ∨ ∧ ∀ t ∈ DOMAIN transaction[i].targets :
                    LET p ≜ transaction[i].targets[t]
                    IN
                      ∧ proposal[t][p].phase = ProposalApply
                      ∧ proposal[t][p].state  = ProposalComplete
                  ∧ transaction′ = [transaction EXCEPT ![i].state  = TransactionComplete,
                                                       ![i].status = TransactionApplied]
                  ∧ UNCHANGED ⟨proposal⟩
              If any proposal has been Failed, mark the transaction Failed.
              ∨ ∧ ∃ t ∈ DOMAIN transaction[i].targets :
                    LET p ≜ transaction[i].targets[t]
                    IN
                      ∧ proposal[t][p].phase = ProposalApply
                      ∧ proposal[t][p].state  = ProposalFailed
                  ∧ transaction′ = [transaction EXCEPT ![i].state = TransactionFailed]
                  ∧ UNCHANGED ⟨proposal⟩
          The Aborting state is used to clean up transactions that have failed during
          the Initializing or Validating phases.
          ∨ ∧ transaction[i].phase = TransactionAbort
            ∧ transaction[i].state  = TransactionInProgress
                Move the transaction's proposals to the Aborting state
            ∧ ∨ ∧ ∃ t ∈ DOMAIN transaction[i].targets :
                    LET p ≜ transaction[i].targets[t]
                    IN
                      ∧ proposal[t][p].phase ≠ ProposalAbort
                      ∧ proposal′ = [proposal EXCEPT ![t] =
                                       [proposal[t] EXCEPT ![p].phase = ProposalAbort,
                                                           ![p].state  = ProposalInProgress]]
                  ∧ UNCHANGED ⟨transaction⟩
              If all proposals have been Complete, mark the transaction Complete.
              ∨ ∧ ∀ t ∈ DOMAIN transaction[i].targets :
                    LET p ≜ transaction[i].targets[t]
                    IN
                      ∧ proposal[t][p].phase  = ProposalAbort
                      ∧ proposal[t][p].state = ProposalComplete
                  ∧ transaction′ = [transaction EXCEPT ![i].state  = TransactionComplete,
                                                       ![i].status = TransactionAborted]
                  ∧ UNCHANGED ⟨proposal⟩
```

Formal specification, constraints, and theorems.

$$
\begin{array}{l}
\mathit{InitTransaction} \triangleq \\
\quad \land\ \mathit{transaction} = [i \in \{\} \mapsto \\
\qquad\qquad [\mathit{type} \ \mapsto \mathit{TransactionChange}, \\
\qquad\qquad\ \mathit{phase} \mapsto \mathit{TransactionInitialize}, \\
\qquad\qquad\ \mathit{state} \ \mapsto \mathit{TransactionInProgress}, \\
\qquad\qquad\ \mathit{status} \mapsto \mathit{TransactionPending}]] \\
\quad \land\ \mathit{Trace}\,!\,\mathit{Init}
\end{array}
$$

$$
\begin{array}{l}
\mathit{NextTransaction} \triangleq \\
\quad \lor \exists\, i \in \text{DOMAIN}\ \mathit{transaction} : \\
\qquad \mathit{Trace}\,!\,\mathit{Step}(\text{“Reconcile”},\ \mathit{ReconcileTransaction}(i),\ [\mathit{index} \mapsto i])
\end{array}
$$

* Modification History
* Last modified *Mon Feb* 21 01:40:59 *PST* 2022 by *jordanhalterman*
* Created Sun *Feb* 20 10:07:06 *PST* 2022 by *jordanhalterman*