MODULE *TwoPhase*

This specification describes the Two-Phase *Commit* protocol, in which a transaction manager (TM) coordinates the resource managers (RMs) to implement the Transaction *Commit* specification of module *TCommit*. In this specification, RMs spontaneously issue *Prepared* messages. We ignore the *Prepare* messages that the TM can send to the RMs.

For simplicity, we also eliminate *Abort* messages sent by an RM when it decides to abort. Such a message would cause the TM to abort the transaction, an event represented here by the TM spontaneously deciding to abort.

This specification describes only the safety properties of the protocol–that is, what is allowed to happen. What must happen would be described by liveness properties, which we do not specify.

CONSTANT RM The set of resource managers

VARIABLES

$rmState$, $rmState[r]$ is the state of resource manager r.

$tmState$, The state of the transaction manager.

$tmPrepared$, The set of RMs from which the TM has received "Prepared" messages.

$msgs$

In the protocol, processes communicate with one another by sending messages. Since we are specifying only safety, a process is not required to receive a message, so there is no need to model message loss. (There's no difference between a process not being able to receive a message because the message was lost and a process simply ignoring the message.) We therefore represent message passing with a variable $msgs$ whose value is the set of all messages that have been sent. Messages are never removed from $msgs$. An action that, in an implementation, would be enabled by the receipt of a certain message is here enabled by the existence of that message in $msgs$. (Receipt of the same message twice is therefore allowed; but in this particular protocol, receiving a message for the second time has no effect.)

$Messages \triangleq$

The set of all possible messages. Messages of type "Prepared" are sent from the RM indicated by the message's rm field to the TM. Messages of type "Commit" and "Abort" are broadcast by the TM, to be received by all RMs. The set $msgs$ contains just a single copy of such a message.

$[type : \{\text{"Prepared"}\},\ rm : RM] \cup [type : \{\text{"Commit"}, \text{"Abort"}\}]$

$TPTypeOK \triangleq$

The type-correctness invariant

$\land\ rmState \in [RM \to \{\text{"working"}, \text{"prepared"}, \text{"committed"}, \text{"aborted"}\}]$
$\land\ tmState \in \{\text{"init"}, \text{"committed"}, \text{"aborted"}\}$
$\land\ tmPrepared \subseteq RM$
$\land\ msgs \subseteq Messages$

$TPInit \triangleq$

The initial predicate.

$\land\ rmState = [r \in RM \mapsto \text{"working"}]$
$\land\ tmState = \text{"init"}$
$\land\ tmPrepared = \{\}$
$\land\ msgs = \{\}$

We now define the actions that may be performed by the processes, first the TM's actions, then the RMs' actions.

$TMRcvPrepared(r) \triangleq$

The TM receives a "Prepared" message from resource manager r.

$\land\ tmState = \text{"init"}$
$\land\ [type \mapsto \text{"Prepared"},\ rm \mapsto r] \in msgs$
$\land\ tmPrepared' = tmPrepared \cup \{r\}$
$\land$ UNCHANGED $\langle rmState,\ tmState,\ msgs\rangle$

$TMCommit \triangleq$

The TM commits the transaction; enabled iff the TM is in its initial state and every RM has sent a "Prepared" message.

$\land\ tmState = \text{"init"}$
$\land\ tmPrepared = RM$
$\land\ tmState' = \text{"committed"}$
$\land\ msgs' = msgs \cup \{[type \mapsto \text{"Commit"}]\}$
$\land$ UNCHANGED $\langle rmState,\ tmPrepared\rangle$

$TMAbort \triangleq$

The TM spontaneously aborts the transaction.

$\land\ tmState = \text{"init"}$
$\land\ tmState' = \text{"aborted"}$
$\land\ msgs' = msgs \cup \{[type \mapsto \text{"Abort"}]\}$
$\land$ UNCHANGED $\langle rmState,\ tmPrepared\rangle$

$RMPrepare(r) \triangleq$

Resource manager r prepares.

$\land\ rmState[r] = \text{"working"}$
$\land\ rmState' = [rmState \text{ EXCEPT } ![r] = \text{"prepared"}]$
$\land\ msgs' = msgs \cup \{[type \mapsto \text{"Prepared"},\ rm \mapsto r]\}$
$\land$ UNCHANGED $\langle tmState,\ tmPrepared\rangle$

$RMChooseToAbort(r) \triangleq$

Resource manager r spontaneously decides to abort. As noted above, r does not send any message in our simplified spec.

$\land\ rmState[r] = \text{"working"}$
$\land\ rmState' = [rmState \text{ EXCEPT } ![r] = \text{"aborted"}]$
$\land$ UNCHANGED $\langle tmState,\ tmPrepared,\ msgs\rangle$

$RMRcvCommitMsg(r) \triangleq$

Resource manager r is told by the TM to commit.

$\land\ [type \mapsto \text{"Commit"}] \in msgs$
$\land\ rmState' = [rmState \text{ EXCEPT } ![r] = \text{"committed"}]$
$\land$ UNCHANGED $\langle tmState,\ tmPrepared,\ msgs\rangle$

$RMRcvAbortMsg(r) \triangleq$

Resource manager r is told by the TM to abort.

$\land\ [type \mapsto \text{“Abort”}] \in msgs$
$\land\ rmState' = [rmState \text{ EXCEPT } ![r] = \text{“aborted”}]$
$\land\ \text{UNCHANGED } \langle tmState,\ tmPrepared,\ msgs\rangle$

$TPNext \triangleq$
$\lor\ TMCommit \lor TMAbort$
$\lor\ \exists\, r \in RM :$
$TMRcvPrepared(r) \lor RMPrepare(r) \lor RMChooseToAbort(r)$
$\lor\ RMRcvCommitMsg(r) \lor RMRcvAbortMsg(r)$

$TPSpec \triangleq TPInit \land \Box[TPNext]_{\langle rmState,\ tmState,\ tmPrepared,\ msgs\rangle}$

The complete spec of the Two-Phase *Commit* protocol.

THEOREM $TPSpec \Rightarrow \Box TPTypeOK$

This theorem asserts that the type-correctness predicate $TPTypeOK$ is an invariant of the specification.

We now assert that the Two-Phase *Commit* protocol implements the Transaction *Commit* protocol of module *TCommit*. The following statement defines $TC!TCSpec$ to be formula *TCSpec* of module *TCommit*. (The TLA+ INSTANCE statement imports all the definitions from module *TCommit* renamed in this way, thus avoiding any name conflicts that might exist between defined operators in the two modules. The constant *RM* and and variable *rmState* are the same in both modules.)

$TC \triangleq \text{INSTANCE } TCommit$

THEOREM $TPSpec \Rightarrow TC!TCSpec$

This theorem asserts that the specification *TPSpec* of the Two-Phase Commit protocol implements the specification *TCSpec* of the Transaction *Commit* protocol.

The two theorems in this module have been checked with *TLC* for six *RMs*, a configuration with 50816 reachable states, in a little over a minute on a 1 *GHz PC*.