MODULE *ZeusOwnership*

EXTENDS *ZeusOwnershipMeta*

This Module specifies the full slow-path of the *Zeus* ownership protocol
as appears in the according paper of *Eurosys'*21 without faults.
It model checks its properties in the face of concurrent conflicting
requests of changing ownerships and emulated reliable commits.

Faults are added on top with the *ZeusOwnershipFaults.tla* spec

---

WARNING: We need to make sure that requester *REQs* are executed at most once; this requires:
an *APP* node to be sticky to its *LB* driver for a Key (unless failure) and send *REQ* msgs via a
FIFO REQ channel so that driver does not re-issues *REQs* that have been already completed in the past!

We emulate executing only once via *committedRTS* (*committedREQs* is used to check INVARIANTS)

$commit\_REQ(o\_ts,\ r\_ts) \triangleq$
$\quad \land\ committedRTS' = committedRTS \cup \{r\_ts\}$
$\quad \land\ committedREQs' = committedREQs \cup \{o\_ts\}$

$upd\_t\_meta(n,\ version,\ state,\ t\_acks) \triangleq$
$\quad \land\ tState' = [tState \text{ EXCEPT } ![n] = state]$
$\quad \land\ tVersion' = [tVersion \text{ EXCEPT } ![n] = version]$
$\quad \land\ tRcvACKs' = [tRcvACKs \text{ EXCEPT } ![n] = t\_acks]$

$upd\_r\_meta(n,\ ver,\ tb,\ id,\ type) \triangleq$
$\quad \land\ rID' = [rID \text{ EXCEPT } ![n] = id]$
$\quad \land\ rEID' = [rEID \text{ EXCEPT } ![n] = mEID]$ always update to latest *mEID*
$\quad \land\ rType' = [rType \text{ EXCEPT } ![n] = type]$
$\quad \land\ rTS' = [rTS \text{ EXCEPT } ![n].ver = ver,\ ![n].tb = tb]$

to update the epoch *id* of last message issue

$upd\_rEID(n) \triangleq upd\_r\_meta(n,\ rTS[n].ver,\ rTS[n].tb,\ rID[n],\ rType[n])$

$upd\_o\_meta(n,\ ver,\ tb,\ state,\ driver,\ vec,\ ACKs) \triangleq$
$\quad \land\ oVector' = [oVector \text{ EXCEPT } ![n] = vec]$
$\quad \land\ oRcvACKs' = [oRcvACKs \text{ EXCEPT } ![n] = ACKs]$
$\quad \land\ oState' = [oState \text{ EXCEPT } ![n] = state]$
$\quad \land\ oDriver' = [oDriver \text{ EXCEPT } ![n] = driver]$
$\quad \land\ oTS' = [oTS \text{ EXCEPT } ![n].ver = ver,\ ![n].tb = tb]$

$upd\_o\_meta\_driver(n,\ ver,\ tb) \triangleq upd\_o\_meta(n,\ ver,\ tb,\ \text{“drive”},\ n,\ oVector[n],\ \{\})$
$upd\_o\_meta\_add\_ack(n,\ sender) \triangleq$
$\quad upd\_o\_meta(n,\ oTS[n].ver,\ oTS[n].tb,\ oState[n],\ oDriver[n],\ oVector[n],\ oRcvACKs[n] \cup \{sender\})$

$upd\_o\_meta\_apply\_val(n,\ m) \triangleq$
$\quad \land\ \text{IF } rTS[n].tb \notin mAliveNodes$
$\qquad \text{THEN } upd\_o\_meta(n,\ oTS[n].ver,\ oTS[n].tb,\ \text{“valid”},\ 0,\ post\_oVec(n,\ 0,\ oVector[n]),\ \{\})$
$\qquad \text{ELSE } upd\_o\_meta(n,\ oTS[n].ver,\ oTS[n].tb,\ \text{“valid”},\ 0,\ post\_oVec(n,\ rTS[n].tb,\ oVector[n]),\ \{\})$

$upd\_o\_meta\_apply\_val\_n\_reset\_o\_state(n) \triangleq$
$upd\_o\_meta(n,\ 0,\ 0,\ \text{“valid”},\ 0,\ [readers \mapsto \{\},\ owner \mapsto 0],\ \{\})$

REQUESTER Helper operators

$choose\_req(n) \triangleq$
LET $choice \triangleq$ CHOOSE $x \in \{0, 1\}$: TRUEIN
IF $is\_reader(n)$
THEN $\land$ IF $choice = 0$
THEN “change-owner”
ELSE “remove-reader”
ELSE $\land$ IF $choice = 0$
THEN “add-owner”
ELSE “add-reader”

$max\_commited\_ver(S,\ n) \triangleq$ IF $\forall\, i \in S : i.tb \neq n$ THEN $[ver \mapsto 0,\ tb \mapsto 0]$
ELSE CHOOSE $i \in S : \land\, i.tb = n$
$\land\, \forall\, j \in S : \lor\, j.tb \neq n$
$\lor\, j.ver \leq i.ver$

$next\_rTS\_ver(n) \triangleq max\_commited\_ver(committedRTS,\ n).ver + 1$

$upd\_rs\_meta\_n\_send\_req(n,\ r\_type) \triangleq$
$\land\ upd\_r\_meta(n,\ next\_rTS\_ver(n),\ n,\ 0,\ r\_type)$
$\land\ upd\_o\_meta(n,\ 0,\ 0,\ \text{“request”},\ 0,\ [readers \mapsto \{\},\ owner \mapsto 0],\ \{\})$
$\land\ o\_send\_req([ver \mapsto next\_rTS\_ver(n),\ tb \mapsto n],\ 0,\ r\_type)$

REQUESTER ACTIONS

$ORequesterREQ(n) \triangleq$ Requester issues a *REQ*
$\land\ is\_valid\_requester(n)$
$\land\ is\_reader(n)$
$\land\ next\_rTS\_ver(n) \leq O\_MAX\_VERSION$ bound execution – > Bound this in reachable states
$\land\ upd\_rs\_meta\_n\_send\_req(n,\ \text{“change-owner”})$ to limit the state space only choose change ownership
$\land\ upd\_rs\_meta\_n\_send\_req(n,\ choose\_req(n))$
$\land\ unchanged\_mtc$

Requester receives *NACK* and replays *REQ* w/ higher *rID*
$ORequesterNACK(n) \triangleq$
$\land\ is\_in\_progress\_requester(n)$
$\land\ rID[n] < O\_MAX\_VERSION$ *TODO*: may Bound *rID* to number of *APP_NODES* instead
$\land\, \exists\, m \in oMsgs : o\_rcv\_nack(m,\ n)$
$\land\ upd\_r\_meta(n,\ rTS[n].ver,\ n,\ rID[n] + 1,\ rType[n])$
$\land\ o\_send\_req([ver \mapsto rTS[n].ver,\ tb \mapsto n],\ rID[n] + 1,\ rType[n])$
$\land\ unchanged\_mtco$

$ORequesterRESP(n) \triangleq$ Requester receives a *RESP* and sends a *VAL* to arbiters
$\exists\, m \in oMsgs :$
$\quad \wedge\ o\_rcv\_resp(m,\ n)$
$\quad \wedge\ is\_in\_progress\_requester(n)$
$\quad \wedge\ commit\_REQ(m.oTS,\ rTS[n])$
$\quad \wedge\ upd\_t\_meta(n,\ m.tVersion,\ \text{“valid”},\ tRcvACKs[n])$ todo this is optional
$\quad \wedge\ upd\_o\_meta(n,\ m.oTS.ver,\ m.oTS.tb,\ \text{“valid”},\ 0,\ post\_oVec(n,\ n,\ m.oVector),\ \{\})$
$\quad \wedge\ o\_send\_val(m.oTS)$
$\quad \wedge\ unchanged\_mtr$

$ORequesterActions \triangleq$
$\quad \exists\, n \in APP\_LIVE\_NODES :$
$\quad\quad \vee\ ORequesterREQ(n)$
$\quad\quad \vee\ ORequesterNACK(n)$
$\quad\quad \vee\ ORequesterRESP(n)$

DRIVER ACTIONS
$ODriverINV(n,\ m) \triangleq$
$\quad \wedge\ o\_rcv\_req(m)$
$\quad \wedge\ oState[n] = \text{“valid”}$
$\quad \wedge\ oTS[n].ver < O\_MAX\_VERSION$ bound execution – > Bound this in reachable states
$\quad \wedge\ upd\_t\_meta(n,\ 0,\ tState[n],\ tRcvACKs[n])$
$\quad \wedge\ upd\_r\_meta(n,\ m.rTS.ver,\ m.rTS.tb,\ m.rID,\ m.rType)$
$\quad \wedge\ upd\_o\_meta\_driver(n,\ oTS[n].ver + 1,\ n)$
$\quad \wedge\ o\_send\_inv(n,\ n,\ [ver \mapsto oTS[n].ver + 1,\ tb \mapsto n],\ oVector[n],\ m.rTS,\ m.rID,\ m.rType)$
$\quad \wedge\ unchanged\_mc$

$ODriverNACK(n,\ m) \triangleq$
$\quad \wedge\ \ o\_rcv\_req(m)$
$\quad \wedge\ \ rTS[n] \neq m.rTS$
$\quad \wedge\ \ oState[n] \neq \text{“valid”}$
$\quad \wedge\ \ msg\_not\_exists(o\_rcv\_nack,\ m.rTS.tb)$ *NACK* does not exist (bound state space)
$\quad \wedge\ \ o\_send\_nack(m.rTS,\ m.rID)$
$\quad \wedge\ \ unchanged\_mtrco$

$ODriverACK(n,\ m) \triangleq$
$\quad \wedge\ o\_rcv\_ack(m,\ n)$
$\quad \wedge\ upd\_o\_meta\_add\_ack(n,\ m.sender)$
$\quad \wedge$ IF $m.tVersion \neq 0$
$\quad\quad$ THEN $upd\_t\_meta(n,\ m.tVersion,\ tState[n],\ tRcvACKs[n])$
$\quad\quad$ ELSE $unchanged\_t$
$\quad \wedge\ unchanged\_Mmrc$

$ODriverRESP(n) \triangleq$
$\quad \wedge\ oState[n] = \text{“drive”}$

$\land\ has\_rcved\_all\_ACKs(n)$
$\land\ requester\_is\_alive(n)$
$\land\ msg\_not\_exists(o\_rcv\_resp,\ rTS[n].tb)$ *RESP* does not exist (bound state space)
$\land\ o\_send\_resp(rTS[n],\ oTS[n],\ post\_oVec(n,\ rTS[n].tb,\ oVector[n]),\ tVersion[n])$
$\land\ unchanged\_mtrco$

$ODriverActions\ \triangleq$
$\exists\, n \in LB\_LIVE\_NODES :$
$\lor\ ODriverRESP(n)$
$\lor \exists\, m \in oMsgs :$
$\lor\ ODriverINV(n,\ m)$
$\lor\ ODriverNACK(n,\ m)$
$\lor\ ODriverACK(n,\ m)$

LB ARBITER ACTIONS
$inv\_to\_be\_applied(n,\ m)\ \triangleq$
$\lor\ o\_rcv\_inv\_greater\_ts(m,\ n)$
$\lor\ (o\_rcv\_inv\_equal\_ts(m,\ n) \land oState[n] = \text{“invalid”} \land m.epochID > rEID[n])$

$check\_n\_apply\_inv(n,\ m)\ \triangleq$
$\land\ inv\_to\_be\_applied(n,\ m)$
$\land\ upd\_r\_meta(n,\ m.rTS.ver,\ m.rTS.tb,\ m.rID,\ m.rType)$
$\land\ upd\_o\_meta(n,\ m.oTS.ver,\ m.oTS.tb,\ \text{“invalid”},\ m.driver,\ m.oVector,\ \{\})$

We do not model lost messages thus arbiter need not respond w/ *INV* when ts is smaller
$OLBArbiterINV(n,\ m)\ \triangleq$
$\land\ check\_n\_apply\_inv(n,\ m)$
$\land\ \lor\ oState[n] \neq \text{“drive”}$
$\lor\ o\_send\_nack(rTS[n],\ rID[n])$
$\land\ o\_send\_ack(n,\ m.oTS,\ 0)$
$\land\ unchanged\_mtc$

$OLBArbiterVAL(n,\ m)\ \triangleq$
$\land\ o\_rcv\_val(m,\ n)$
$\land\ upd\_o\_meta\_apply\_val(n,\ m)$
$\land\ unchanged\_Mmtrc$

$OLBArbiterActions\ \triangleq$
$\exists\, n \in LB\_LIVE\_NODES : \exists\, m \in oMsgs :$
$\lor\ OLBArbiterINV(n,\ m)$
$\lor\ OLBArbiterVAL(n,\ m)$

(O)wner or (R)eader ARBITER ACTIONS

reader doesn't apply an *INV* but always responds with an *ACK*

(and data if non-sharing *rType* and in *tValid* state)

$ORArbiterINV(n,\ m) \triangleq$
$\quad \land\ is\_reader(n)$
$\quad \land\ tState[n] =$ "valid"
$\quad \land\ o\_rcv\_inv(m,\ n)$
$\quad \land\ o\_send\_ack(n,\ m.oTS,\ tVersion[n])$
$\quad \land\ unchanged\_mtrco$

$OOArbiterINV(n,\ m) \triangleq$
$\quad \land\ is\_owner(n)$
$\quad \land\ m.type =$ "S_INV"
$\quad \land\ m.oVector.owner = n$ otherwise owner lost a $VAL -> SFMOArbiterINVLostVAL$
$\quad \land\ tState[n] =$ "valid"
$\quad \land\ check\_n\_apply\_inv(n,\ m)$
$\quad \land\ o\_send\_ack(n,\ m.oTS,\ tVersion[n])$
$\quad \land\ unchanged\_mtc$

$OOArbiterVAL(n,\ m) \triangleq$
$\quad \land\ o\_rcv\_val(m,\ n)$
$\quad \land$ IF $oVector[n].owner = n$
$\quad\quad$ THEN $\land\ upd\_o\_meta\_apply\_val(n,\ m)$
$\quad\quad$ ELSE $\land\ upd\_o\_meta\_apply\_val\_n\_reset\_o\_state(n)$
$\quad \land\ unchanged\_Mmtrc$

$OAPPArbiterActions \triangleq$
$\exists\, n \in APP\_LIVE\_NODES : \exists\, m \in oMsgs :$
$\quad \lor\ ORArbiterINV(n,\ m)$
$\quad \lor\ OOArbiterINV(n,\ m)$
$\quad \lor\ OOArbiterVAL(n,\ m)$

Owner actions emulating tx updates

$TOwnerINV(n) \triangleq$
$\quad \land\ upd\_t\_meta(n,\ tVersion[n] + 1,$ "write", $\{\})$
$\quad \land\ t\_send(n,$ "T_INV", $tVersion[n] + 1)$
$\quad \land\ unchanged\_mrco$

$TOwnerACK(n) \triangleq$
$\quad \exists\, m \in oMsgs :$
$\quad \land\ t\_rcv\_ack(m,\ n)$
$\quad \land\ upd\_t\_meta(n,\ tVersion[n],\ tState[n],\ tRcvACKs[n] \cup \{m.sender\})$
$\quad \land\ unchanged\_Mmrco$

$TOwnerVAL(n) \triangleq$
$\quad \land\ oVector[n].readers \subseteq tRcvACKs[n]$ has received all acks from readers
$\quad \land\ upd\_t\_meta(n,\ tVersion[n],$ "valid", $\{\})$

$\land\ t\_send(n,\ \text{“T\_VAL”},\ tVersion[n])$
$\land\ unchanged\_mrco$

Reader actions emulating tx updates

$TReaderINV(n) \triangleq$
$\exists\, m \in oMsgs :$
$\land\ t\_rcv\_inv(m,\ n)$
$\land\ m.tVersion > tVersion[n]$
$\land\ upd\_t\_meta(n,\ m.tVersion,\ \text{“invalid”},\ \{\})$
$\land\ t\_send(n,\ \text{“T\_ACK”},\ m.tVersion)$
$\land\ unchanged\_mrco$

$TReaderVAL(n) \triangleq$
$\exists\, m \in oMsgs :$
$\land\ t\_rcv\_val(m,\ n)$
$\land\ m.tVersion = tVersion[n]$
$\land\ upd\_t\_meta(n,\ tVersion[n],\ \text{“valid”},\ \{\})$
$\land\ unchanged\_Mmrco$

$TOwnerReaderActions \triangleq$
$\exists\, n \in APP\_LIVE\_NODES :$
$\lor\ \land\ is\_valid\_owner(n)$
$\land\ \lor\ TOwnerINV(n)$
$\lor\ TOwnerACK(n)$
$\lor\ TOwnerVAL(n)$
$\lor\ \land\ is\_reader(n)$
$\land\ \lor\ TReaderINV(n)$
$\lor\ TReaderVAL(n)$

Modeling *Sharding* protocol (Requester and Arbiter actions)

$ONext \triangleq$
$\lor\ OInit\_min\_owner\_rest\_readers$
$\lor\ ORequesterActions$
$\lor\ ODriverActions$
$\lor\ OLBArbiterActions$
$\lor\ OAPPArbiterActions$
$\lor\ TOwnerReaderActions$

The complete definition of the algorithm

$Spec \triangleq\ OInit \land \Box[ONext]_{vars}$

$Invariants \triangleq\ \land (\Box OTypeOK)$
$\land (\Box CONSISTENT\_DATA) \quad \land (\Box ONLY\_ONE\_CONC\_REQ\_COMMITS)$
$\land (\Box AT\_MOST\_ONE\_OWNER) \land (\Box OWNER\_LATEST\_DATA)$
$\land (\Box CONSISTENT\_SHARERS) \land (\Box CONSISTENT\_OVECTORS)$

THEOREM $Spec \Rightarrow Invariants$

$LSpec \triangleq Spec \wedge \mathrm{WF}\_vars(ONext)$

$LIVENESS \triangleq \exists\, i \in LB\_NODES:\ \Box\Diamond(oState[i] = \text{“valid”} \wedge oTS[i].ver > 3)$

THEOREM $LSpec \Rightarrow LIVENESS$