MODULE *XJupiter*

Specification of the *Jupiter* protocol described in *CSCW*′2014 by *Yi Xu*, *Chengzheng* Sun, and *Mo Li*. We call it *XJupiter*, with 'X' for "*Xu*".

EXTENDS *JupiterInterface*

Direction flags for edges in $2D$ state spaces and OT.

$Local \triangleq 0$

$Remote \triangleq 1$

Cop: operation of type *Op* with context

$Oid \triangleq [c : Client,\ seq : Nat]$ operation identifier

$Cop \triangleq [op : Op \cup \{Nop\},\ oid : Oid,\ ctx : \text{SUBSET } Oid]$

OT of two operations of type *Cop*.

$COT(lcop, rcop) \triangleq [lcop \text{ EXCEPT } !.op = Xform(lcop.op, rcop.op),\ !.ctx = @ \cup \{rcop.oid\}]$

VARIABLES

For the client replicas:

$cseq,$ $cseq[c]$: local sequence number at client $c \in Client$

The $2D$ state spaces (ss, for short). Each client maintains one $2D$ state space. The server maintains n $2D$ state spaces, one for each client.

$c2ss,$ $c2ss[c]$: the $2D$ state space at client $c \in Client$

$s2ss,$ $s2ss[c]$: the $2D$ state space maintained by the *Server* for client $c \in Client$

$cur,$ $cur[r]$: the current node of the $2D$ state space at replica $r \in Replica$

For all replicas

$state,$ $state[r]$: state (the list content) of replica $r \in Replica$

For communication between the *Server* and the Clients:

$cincoming,$ $cincoming[c]$: incoming channel at the client $c \in Client$

$sincoming,$ incoming channel at the *Server*

For model checking:

$chins$ a set of chars to insert

$vars \triangleq \langle chins, cseq, cur, cincoming, sincoming, c2ss, s2ss, state\rangle$

$comm \triangleq \text{INSTANCE } CSComm \text{ WITH } Msg \leftarrow Cop$

A $2D$ state space is a directed graph with labeled edges. It is represented by a record with node field and edge field. Each node is characterized by its context, a set of operations. Each edge is labeled with an operation and a direction flag indicating whether this edge is LOCAL or REMOTE. For clarity, we denote edges by records instead of tuples.

$$
\begin{aligned}
&IsSS(G) \triangleq \\
&\quad \land\ G = [node \mapsto G.node,\ edge \mapsto G.edge] \\
&\quad \land\ G.node \subseteq (\text{SUBSET } Oid) \\
&\quad \land\ G.edge \subseteq [from : G.node,\ to : G.node,\ cop : Cop,\ lr : \{Local, Remote\}]
\end{aligned}
$$

```
EmptySS ≜ [node ↦ {{}}, edge ↦ {}]
Take union of two state spaces ss1 and ss2.
ss1 ⊕ ss2 ≜ [node ↦ ss1.node ∪ ss2.node, edge ↦ ss1.edge ∪ ss2.edge]

TypeOK ≜
    For the client replicas:
    ∧ cseq ∈ [Client → Nat]
    For the 2D state spaces:
    ∧ ∀ c ∈ Client : IsSS(c2ss[c]) ∧ IsSS(s2ss[c])
    ∧ cur ∈ [Replica → SUBSET Oid]
    ∧ state ∈ [Replica → List]
    For communication between the server and the clients:
    ∧ comm!TypeOK
    For model checking:
    ∧ chins ⊆ Char
────────────────────────────────────────────────────────────
Init ≜
    For the client replicas:
    ∧ cseq = [c ∈ Client ↦ 0]
    For the 2D state spaces:
    ∧ c2ss = [c ∈ Client ↦ EmptySS]
    ∧ s2ss = [c ∈ Client ↦ EmptySS]
    ∧ cur  = [r ∈ Replica ↦ {}]
    For all replicas:
    ∧ state = [r ∈ Replica ↦ InitState]
    For communication between the server and the clients:
    ∧ comm!Init
    For model checking:
    ∧ chins = Char
────────────────────────────────────────────────────────────
Locate the node in the 2D state space ss which matches the context ctx of cop.
Locate(cop, ss) ≜ CHOOSE n ∈ ss.node : n = cop.ctx
xForm: iteratively transform cop with a path through the 2D state space ss at some client,
following the edges with the direction flag d.
xForm(cop, ss, current, d) ≜
    LET u ≜ Locate(cop, ss)
        v ≜ u ∪ {cop.oid}
        RECURSIVE xFormHelper(_, _, _, _, _, _)
         'h' stands for "helper"; xss: eXtra ss created during transformation
        xFormHelper(uh, vh, coph, xss, xcoph, xcurh) ≜
            IF uh = current
             THEN ⟨xss, xcoph, xcurh⟩
```

```
                ELSE LET e ≜ CHOOSE e ∈ ss.edge : e.from = uh ∧ e.lr = d
                         uprime ≜ e.to
                         copprime ≜ e.cop
                         coph2copprime ≜ COT(coph, copprime)
                         copprime2coph ≜ COT(copprime, coph)
                         vprime ≜ vh ∪ {copprime.oid}
                     IN  xFormHelper(uprime, vprime, coph2copprime,
                           [node ↦ xss.node ∪ {vprime},
                            edge ↦ xss.edge ∪ {[from ↦ vh, to ↦ vprime, cop ↦ copprime2coph, lr ↦ d],
                                   [from ↦ uprime, to ↦ vprime, cop ↦ coph2copprime, lr ↦ 1 − d]}],
                           coph2copprime, vprime)
    IN  xFormHelper(u, v, cop, [node ↦ {v}, edge ↦ {[from ↦ u, to ↦ v, cop ↦ cop, lr ↦ 1 − d]}], cop, v)
```

Client $c \in$ *Client* perform operation cop guided by the direction flag d.

```
ClientPerform(cop, c, d) ≜
    LET xform ≜ xForm(cop, c2ss[c], cur[c], d)   xform: ⟨xss, xcop, xcur⟩
        xss   ≜ xform[1]
        xcop  ≜ xform[2]
        xcur  ≜ xform[3]
    IN  ∧ c2ss' = [c2ss EXCEPT ![c] = @ ⊕ xss]
        ∧ cur' = [cur EXCEPT ![c] = xcur]
        ∧ state' = [state EXCEPT ![c] = Apply(xcop.op, @)]
```

Client $c \in$ *Client* generates an operation *op*.

```
DoOp(c, op) ≜
    ∧ cseq' = [cseq EXCEPT ![c] = @ + 1]
    ∧ LET cop ≜ [op ↦ op, oid ↦ [c ↦ c, seq ↦ cseq'[c]], ctx ↦ cur[c]]
      IN  ∧ ClientPerform(cop, c, Remote)
          ∧ comm!CSend(cop)

DoIns(c) ≜
    ∃ ins ∈ {op ∈ Ins : op.pos ∈ 1 . . (Len(state[c]) + 1) ∧ op.ch ∈ chins ∧ op.pr = Priority[c]} :
        ∧ DoOp(c, ins)
        ∧ chins' = chins \ {ins.ch}   We assume that all inserted elements are unique.

DoDel(c) ≜
    ∃ del ∈ {op ∈ Del : op.pos ∈ 1 . . Len(state[c])} :
        ∧ DoOp(c, del)
        ∧ UNCHANGED ⟨chins⟩

Do(c) ≜
    ∧ ∨ DoIns(c)
      ∨ DoDel(c)
    ∧ UNCHANGED ⟨s2ss⟩
```

Client $c \in$ *Client* receives a message from the *Server*.

```
Rev(c) ≜
    ∧ comm!CRev(c)
    ∧ LET cop ≜ Head(cincoming[c])  \* the received (transformed) operation
       IN  ClientPerform(cop, c, Local)
    ∧ UNCHANGED ⟨chins, cseq, s2ss⟩
----------------------------------------------------------------------------
```

The *Server* performs operation cop.

```
ServerPerform(cop) ≜
    LET c ≜ cop.oid.c
     scur ≜ cur[Server]
    xform ≜ xForm(cop, s2ss[c], scur, Remote)  \* xform: ⟨xss, xcop, xcur⟩
      xss ≜ xform[1]
     xcop ≜ xform[2]
     xcur ≜ xform[3]
    IN  ∧ s2ss' = [cl ∈ Client ↦
                     IF cl = c
                      THEN s2ss[cl] ⊕ xss
                      ELSE s2ss[cl] ⊕ [node ↦ {xcur},
                                       edge ↦ {[from ↦ scur, to ↦ xcur,
                                                cop ↦ xcop, lr ↦ Remote]}]
                  ]
        ∧ cur' = [cur EXCEPT ![Server] = xcur]
        ∧ state' = [state EXCEPT ![Server] = Apply(xcop.op, @)]
        ∧ comm!SSendSame(c, xcop)  \* broadcast the transformed operation
```

The *Server* receives a message.

```
SRev ≜
    ∧ comm!SRev
    ∧ LET cop ≜ Head(sincoming)
       IN  ServerPerform(cop)
    ∧ UNCHANGED ⟨chins, cseq, c2ss⟩
----------------------------------------------------------------------------
Next ≜
    ∨ ∃ c ∈ Client : Do(c) ∨ Rev(c)
    ∨ SRev

Fairness ≜
    WF_vars(SRev ∨ ∃ c ∈ Client : Rev(c))

Spec ≜ Init ∧ □[Next]_vars  ∧ Fairness
----------------------------------------------------------------------------
```

In *Jupiter* (not limited to *XJupiter*), each client synchronizes with the server. In *XJupiter*, this is expressed as the following *CSSync* property.

```
CSSync ≜
    ∀ c ∈ Client : (cur[c] = cur[Server]) ⇒ c2ss[c] = s2ss[c]
============================================================================
```

\ * Modification History
\ * *Last* modified *Tue Dec* 04 19:34:37 *CST* 2018 by *hengxin*
\ * Created *Tue Oct* 09 16:33:18 *CST* 2018 by *hengxin*