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 2*D* 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

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

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

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

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

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

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

A 2*D* 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.

$IsSS(G) \triangleq$

$\quad \land\ G = [node \mapsto G.node,\ edge \mapsto G.edge]$

$\quad \land\ G.node \subseteq (\text{SUBSET } Oid)$

$\quad \land\ G.edge \subseteq [from : G.node,\ to : G.node,\ cop : Cop,\ lr : \{Local,\ Remote\}]$

$EmptySS \triangleq [node \mapsto \{\{\}\},\ edge \mapsto \{\}]$

Take union of two state spaces *ss*1 and *ss*2.

$ss1 \oplus ss2 \triangleq [node \mapsto ss1.node \cup ss2.node,\ edge \mapsto ss1.edge \cup ss2.edge]$

$TypeOK \triangleq$

$\quad \land\quad TypeOKInt$

$\quad \land\quad cseq \in [Client \to Nat]$

For the 2*D* state spaces:

$\quad \land \forall\, c \in Client : IsSS(c2ss[c]) \land IsSS(s2ss[c])$

$\quad \land\ cur \in [Replica \to \text{SUBSET } Oid]$

```
        For communication between the server and the clients:
69      ∧ Comm(Cop)!TypeOK
70  ----------------------------------------------------------------
71  Init ≜
72      ∧ InitInt
73      ∧ cseq = [c ∈ Client ↦ 0]
        For the 2D state spaces:
77      ∧ c2ss = [c ∈ Client ↦ EmptySS]
78      ∧ s2ss = [c ∈ Client ↦ EmptySS]
79      ∧ cur  = [r ∈ Replica ↦ {}]
        For communication between the server and the clients:
83      ∧ Comm(Cop)!Init
84  ----------------------------------------------------------------
    Locate the node in the 2D state space ss which matches the context ctx of cop.
88  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.
94  xForm(cop, ss, current, d) ≜
95      LET u ≜ Locate(cop, ss)
96          v ≜ u ∪ {cop.oid}
97          RECURSIVE xFormHelper(_, _, _, _, _, _)
98           'h' stands for "helper"; xss: eXtra ss created during transformation
99          xFormHelper(uh, vh, coph, xss, xcoph, xcurh) ≜
100             IF uh = current
101              THEN ⟨xss, xcoph, xcurh⟩
102              ELSE LET e ≜ CHOOSE e ∈ ss.edge : e.from = uh ∧ e.lr = d
103                       uprime ≜ e.to
104                       copprime ≜ e.cop
105                       coph2copprime ≜ COT(coph, copprime)
106                       copprime2coph ≜ COT(copprime, coph)
107                        vprime ≜ vh ∪ {copprime.oid}
108                   IN   xFormHelper(uprime, vprime, coph2copprime,
109                            [node ↦ xss.node ∪ {vprime},
110                             edge ↦ xss.edge ∪ {[from ↦ vh, to ↦ vprime, cop ↦ copprime2coph, lr ↦ d],
111                                                [from ↦ uprime, to ↦ vprime, cop ↦ coph2copprime, lr ↦ 1 − d]}],
112                                    coph2copprime, vprime)
113     IN   xFormHelper(u, v, cop, [node ↦ {v}, edge ↦ {[from ↦ u, to ↦ v, cop ↦ cop, lr ↦ 1 − d]}], cop, v)
114 ----------------------------------------------------------------
    Client c ∈ Client perform operation cop guided by the direction flag d.
118 ClientPerform(cop, c, d) ≜
119     LET xform ≜ xForm(cop, c2ss[c], cur[c], d)  xform: ⟨xss, xcop, xcur⟩
120         xss  ≜ xform[1]
121         xcop ≜ xform[2]
```

```
122             xcur  ≜  xform[3]
123     IN      ∧ c2ss' = [c2ss EXCEPT ![c] = @ ⊕ xss]
124             ∧ cur' = [cur EXCEPT ![c] = xcur]
125             ∧ state' = [state EXCEPT ![c] = Apply(xcop.op, @)]
```

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

```
129 DoOp(c, op)  ≜
130         ∧ cseq' = [cseq EXCEPT ![c] = @ + 1]
131         ∧ LET cop  ≜  [op ↦ op, oid ↦ [c ↦ c, seq ↦ cseq'[c]], ctx ↦ cur[c]]
132             IN    ∧ ClientPerform(cop, c, Remote)
133                   ∧ Comm(Cop)!CSend(cop)

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

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

145 Do(c)  ≜
146       ∧ ∨ DoIns(c)
147         ∨ DoDel(c)
148       ∧ UNCHANGED ⟨s2ss⟩
```

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

```
152 Rev(c)  ≜
153       ∧ Comm(Cop)!CRev(c)
154       ∧ LET cop  ≜  Head(cincoming[c])   the received (transformed) operation
155           IN    ClientPerform(cop, c, Local)
156       ∧ UNCHANGED ⟨chins, cseq, s2ss⟩
157 ├──────────────────────────────────────────────────────────────────────┤
```

The *Server* performs operation cop.

```
161 ServerPerform(cop)  ≜
162     LET c  ≜  cop.oid.c
163       scur  ≜  cur[Server]
164       xform  ≜  xForm(cop, s2ss[c], scur, Remote)   xform: ⟨xss, xcop, xcur⟩
165         xss  ≜  xform[1]
166       xcop  ≜  xform[2]
167       xcur  ≜  xform[3]
168     IN    ∧ s2ss' = [cl ∈ Client ↦
169                       IF cl = c
170                        THEN s2ss[cl] ⊕ xss
171                        ELSE s2ss[cl] ⊕ [node ↦ {xcur},
172                                         edge ↦ {[from ↦ scur, to ↦ xcur,
```

```
                                                cop ↦ xcop, lr ↦ Remote]}]
                    ]
          ∧ cur′ = [cur EXCEPT ![Server] = xcur]
          ∧ state′ = [state EXCEPT ![Server] = Apply(xcop.op, @)]
          ∧ Comm(Cop)!SSendSame(c, xcop)   broadcast the transformed operation
```

The *Server* receives a message.

```
SRev ≜
    ∧ Comm(Cop)!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 21:12:08 *CST* 2018 by *hengxin*
* Created *Tue Oct* 09 16:33:18 *CST* 2018 by *hengxin*