```
1  ---------- 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".
7  EXTENDS StateSpace
8  ----------
9  VARIABLES
       The 2D state spaces (2ss, for short). Each client maintains one 2D state space. The server
       maintains n 2D state spaces, one for each client.
15     c2ss,     c2ss[c]: the 2D state space at client c ∈ Client
16     s2ss      s2ss[c]: the 2D state space maintained by the Server for client c ∈ Client
18 vars ≜ ⟨intVars, ctxVars, c2ss, s2ss⟩
19 ----------
20 TypeOK ≜
21     ∧  TypeOKInt
22     ∧  TypeOKCtx
23     ∧  Comm(Cop)!TypeOK
24     ∧  ∀ c ∈ Client : IsSS(c2ss[c]) ∧ IsSS(s2ss[c])
25 ----------
26 Init ≜
27     ∧ InitInt
28     ∧ InitCtx
29     ∧ Comm(Cop)!Init
30     ∧ c2ss = [c ∈ Client ↦ EmptySS]
31     ∧ s2ss = [c ∈ Client ↦ EmptySS]
32 ----------
   xForm: iteratively transform cop with a path through the 2D state space ss at some client.
37 xForm(cop, ss, current) ≜
38     LET u ≜ Locate(cop, ss)
39         v ≜ u ∪ {cop.oid}
40         RECURSIVE xFormHelper(_, _, _, _)
41          'h' stands for "helper"; xss: eXtra ss created during transformation
42         xFormHelper(uh, vh, coph, xss) ≜
43             IF uh = current
44              THEN [xss ↦ xss, xcop ↦ coph]
45              ELSE LET e ≜ CHOOSE e ∈ ss.edge : e.from = uh ∧ ClientOf(e.cop) ≠ ClientOf(cop)
46                       uprime ≜ e.to
47                       copprime ≜ e.cop
48                       coph2copprime ≜ COT(coph, copprime)
49                       copprime2coph ≜ COT(copprime, coph)
50                        vprime ≜ vh ∪ {copprime.oid}
51                   IN   xFormHelper(uprime, vprime, coph2copprime,
52                          [node ↦ xss.node ∪ {vprime},
53                           edge ↦ xss.edge ∪ {[from ↦ vh, to ↦ vprime, cop ↦ copprime2coph],
54                                              [from ↦ uprime, to ↦ vprime, cop ↦ coph2copprime]}])
```

```
55      IN   xFormHelper(u, v, cop, [node ↦ {v}, edge ↦ {[from ↦ u, to ↦ v, cop ↦ cop]}])
56  ----------------------------------------------------------------------------
    Client c ∈ Client perform operation cop.
60  ClientPerform(cop, c) ≜
61      LET xform ≜ xForm(cop, c2ss[c], ds[c])  xform: [xss, xcop]
62      IN   ∧ c2ss' = [c2ss EXCEPT ![c] = @ ⊕ xform.xss]
63           ∧ state' = [state EXCEPT ![c] = Apply(xform.xcop.op, @)]
    Client c ∈ Client generates an operation op.
67  DoOp(c, op)  ≜
68        LET cop ≜ [op ↦ op, oid ↦ [c ↦ c, seq ↦ cseq'[c]], ctx ↦ ds[c]]
69            IN   ∧ ClientPerform(cop, c)
70                 ∧ UpdateDS(c, cop)
71                 ∧ Comm(Cop)!CSend(cop)

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

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

83  Do(c) ≜
84        ∧ DoCtx(c)
85        ∧ ∨ DoIns(c)
86          ∨ DoDel(c)
87        ∧ UNCHANGED s2ss
    Client c ∈ Client receives a message from the Server.
91  Rev(c) ≜
92        ∧ Comm(Cop)!CRev(c)
93        ∧ LET cop ≜ Head(cincoming[c])  the received (transformed) operation
94            IN   ClientPerform(cop, c)
95        ∧ RevCtx(c)
96        ∧ UNCHANGED ⟨chins, s2ss⟩
97  ----------------------------------------------------------------------------
    The Server performs operation cop.
101 ServerPerform(cop) ≜
102     LET c ≜ ClientOf(cop)
103      scur ≜ ds[Server]
104     xform ≜ xForm(cop, s2ss[c], scur)  xform: [xss, xcop]
105      xcop ≜ xform.xcop
106      xcur ≜ scur ∪ {cop.oid}
107     IN   ∧ s2ss' = [cl ∈ Client ↦
```

```
                        IF cl = c
                         THEN s2ss[cl] ⊕ xform.xss
                         ELSE s2ss[cl] ⊕ [node ↦ {xcur},
                                 edge ↦ {[from ↦ scur, to ↦ xcur, cop ↦ xcop]}]
                      ]
        ∧ 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)
    ∧ SRevCtx
    ∧ UNCHANGED ⟨chins, 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 : (ds[c] = ds[Server]) ⇒ c2ss[c] = s2ss[c]
```

---

* Modification History
* Last modified *Mon Dec* 24 10:38:54 *CST* 2018 by *hengxin*
* Created *Tue Oct* 09 16:33:18 *CST* 2018 by *hengxin*