```
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
10     c2ss,    c2ss[c]: the 2D state space (2ss, for short) at client c ∈ Client
11     s2ss     s2ss[c]: the 2D state space maintained by the Server for client c ∈ Client
13 vars ≜ ⟨intVars, ctxVars, c2ss, s2ss⟩
14 ----------
15 TypeOK ≜
16     ∧  TypeOKInt
17     ∧  TypeOKCtx
18     ∧  Comm(Cop)!TypeOK
19     ∧  ∀ c ∈ Client : IsSS(c2ss[c]) ∧ IsSS(s2ss[c])
20 ----------
21 Init ≜
22     ∧ InitInt
23     ∧ InitCtx
24     ∧ Comm(Cop)!Init
25     ∧ c2ss = [c ∈ Client ↦ EmptySS]
26     ∧ s2ss = [c ∈ Client ↦ EmptySS]
27 ----------
28 xForm(cop, ss, cur) ≜  Transform cop with a path (i.e., operation sequence) through 2D state space ss.
29     LET u ≜ Locate(cop, ss)
30         v ≜ u ∪ {cop.oid}
31         RECURSIVE xFormHelper(_, _, _, _)
32          xFormHelper(uh, vh, coph, xss) ≜   xss: eXtra ss created during transformation
33              IF uh = cur THEN [xss ↦ xss, xcop ↦ coph]
34               ELSE LET e ≜ CHOOSE e ∈ ss.edge : e.from = uh ∧ ClientOf(e.cop) ≠ ClientOf(cop)
35                        copprime ≜ e.cop
36                        uprime ≜ e.to
37                        vprime ≜ vh ∪ {copprime.oid}
38                        coph2copprime ≜ COT(coph, copprime)
39                         copprime2coph ≜ COT(copprime, coph)
40                    IN   xFormHelper(uprime, vprime, coph2copprime,
41                             xss ⊕ [node ↦ {vprime},
42                                    edge ↦ {[from ↦ vh, to ↦ vprime, cop ↦ copprime2coph],
43                                            [from ↦ uprime, to ↦ vprime, cop ↦ coph2copprime]}])
44      IN   xFormHelper(u, v, cop, [node ↦ {v}, edge ↦ {[from ↦ u, to ↦ v, cop ↦ cop]}])
45 ----------
46 ClientPerform(cop, c) ≜  Client c ∈ Client perform operation cop.
47     LET xform ≜ xForm(cop, c2ss[c], ds[c])  xform: [xss, xcop]
48      IN   ∧ c2ss′ = [c2ss EXCEPT ![c] = @ ⊕ xform.xss]
```

```
49              ∧ SetNewAop(c, xform.xcop.op)

51  DoOp(c, op)  ≜
52        LET cop ≜ [op ↦ op, oid ↦ [c ↦ c, seq ↦ cseq′[c]], ctx ↦ ds[c]]
53            IN  ∧ ClientPerform(cop, c)
54                ∧ Comm(Cop)!CSend(cop)

56  Do(c) ≜
57        ∧ DoCtx(c)
58        ∧ DoInt(DoOp, c)
59        ∧ UNCHANGED s2ss

61  Rev(c) ≜
62        ∧ Comm(Cop)!CRev(c)
63        ∧ ClientPerform(Head(cincoming[c]), c)
64        ∧ RevCtx(c)
65        ∧ RevInt(c)
66        ∧ UNCHANGED s2ss

68  ServerPerform(cop) ≜
69      LET c ≜ ClientOf(cop)
70       scur ≜ ds[Server]
71      xform ≜ xForm(cop, s2ss[c], scur)  xform: [xss, xcop]
72       xcop ≜ xform.xcop
73       xcur ≜ scur ∪ {cop.oid}
74       IN  ∧ s2ss′ = [cl ∈ Client ↦
75                       IF cl = c
76                        THEN s2ss[cl] ⊕ xform.xss
77                        ELSE s2ss[cl] ⊕ [node ↦ {xcur},
78                            edge ↦ {[from ↦ scur, to ↦ xcur, cop ↦ xcop]}]
79                      ]
80           ∧ SetNewAop(Server, xcop.op)
81           ∧ Comm(Cop)!SSendSame(c, xcop)

83  SRev ≜
84      ∧ Comm(Cop)!SRev
85      ∧ ServerPerform(Head(sincoming))
86      ∧ SRevCtx
87      ∧ SRevInt
88      ∧ UNCHANGED c2ss
89  ├──────────────────────────────────────────────────────────────────────┤
90  Next ≜
91      ∨ ∃ c ∈ Client : Do(c) ∨ Rev(c)
92      ∨ SRev

94  Fairness ≜  There is no requirement that the clients ever generate operations.
95      WF_vars(SRev ∨ ∃ c ∈ Client : Rev(c))
```

97 $Spec \triangleq Init \land \Box[Next]_{vars}$ $\land Fairness$

98 ----------

99 $CSSync \triangleq$ Each client $c \in Client$ is synchonized with the $Server$.

100 $\quad \forall\, c \in Client : (ds[c] = ds[Server]) \Rightarrow c2ss[c] = s2ss[c]$

102 THEOREM $Spec \Rightarrow \Box CSSync$

103 ==========

* Modification History
* Last modified *Tue Jan* 01 11:41:17 *CST* 2019 by *hengxin*
* Created *Tue Oct* 09 16:33:18 *CST* 2018 by *hengxin*