```
┌──────────────────────────── 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 StateSpace
├──────────────────────────────────────────────────────────────────────────┤
VARIABLES
    c2ss,    c2ss[c]: the 2D state space (2ss, for short) at client c ∈ Client
    s2ss     s2ss[c]: the 2D state space maintained by the Server for client c ∈ Client

vars ≜ ⟨intVars, ctxVars, c2ss, s2ss⟩
├──────────────────────────────────────────────────────────────────────────┤
TypeOK ≜
    ∧  TypeOKInt
    ∧  TypeOKCtx
    ∧  ∀ c ∈ Client : IsSS(c2ss[c]) ∧ IsSS(s2ss[c])
├──────────────────────────────────────────────────────────────────────────┤
Init ≜
    ∧ InitInt
    ∧ InitCtx
    ∧ c2ss = [c ∈ Client ↦ EmptySS]
    ∧ s2ss = [c ∈ Client ↦ EmptySS]
├──────────────────────────────────────────────────────────────────────────┤
xForm(cop, ss, cur) ≜   Transform cop with an operation sequence in 2D state space ss.
    LET u ≜ Locate(cop, ss)
        v ≜ u ∪ {cop.oid}
        RECURSIVE xFormHelper(_, _, _, _)
         xFormHelper(uh, vh, coph, xss) ≜   xss: eXtra ss created during transformation
            IF uh = cur THEN [xss ↦ xss, xcop ↦ coph]
             ELSE LET e ≜ CHOOSE e ∈ ss.edge : e.from = uh ∧ ClientOf(e.cop) ≠ ClientOf(cop)
                          copprime ≜ e.cop
                          uprime ≜ e.to
                          vprime ≜ vh ∪ {copprime.oid}
                          coph2copprime ≜ COT(coph, copprime)
                           copprime2coph ≜ COT(copprime, coph)
                      IN  xFormHelper(uprime, vprime, coph2copprime,
                            xss ⊕ [node ↦ {vprime},
                                   edge ↦ {[from ↦ vh, to ↦ vprime, cop ↦ copprime2coph],
                                           [from ↦ uprime, to ↦ vprime, cop ↦ coph2copprime]}])
     IN  xFormHelper(u, v, cop, [node ↦ {v}, edge ↦ {[from ↦ u, to ↦ v, cop ↦ cop]}])

ClientPerform(c, cop) ≜
    LET xform ≜ xForm(cop, c2ss[c], ds[c])  xform: [xss, xcop]
     IN  ∧ c2ss' = [c2ss EXCEPT ![c] = @ ⊕ xform.xss]
         ∧ SetNewAop(c, xform.xcop.op)

ServerPerform(cop) ≜
```

```
     LET c ≜ ClientOf(cop)
      scur ≜ ds[Server]
     xform ≜ xForm(cop, s2ss[c], scur)  xform: [xss, xcop]
      xcop ≜ xform.xcop
      xcur ≜ scur ∪ {cop.oid}
      IN    ∧ s2ss′ = [cl ∈ Client ↦
                          IF cl = c
                           THEN s2ss[cl] ⊕ xform.xss
                           ELSE  s2ss[cl] ⊕ [node ↦ {xcur},
                                edge ↦ {[from ↦ scur, to ↦ xcur, cop ↦ xcop]}]]
            ∧ SetNewAop(Server, xcop.op)
            ∧ Comm!SSendSame(c, xcop)
----------------------------------------------------------------------------
DoOp(c, op)  ≜
      LET cop ≜ [op ↦ op, oid ↦ [c ↦ c, seq ↦ cseq[c]], ctx ↦ ds[c]]
       IN    ∧ ClientPerform(c, cop)
             ∧ Comm!CSend(cop)

Do(c) ≜
     ∧ DoInt(DoOp, c)
     ∧ DoCtx(c)
     ∧ UNCHANGED s2ss

Rev(c) ≜
     ∧ RevInt(ClientPerform, c)
     ∧ RevCtx(c)
     ∧ UNCHANGED s2ss

SRev ≜
     ∧ SRevInt(ServerPerform)
     ∧ SRevCtx
     ∧ UNCHANGED c2ss
----------------------------------------------------------------------------
Next ≜
     ∨ ∃ c ∈ Client : Do(c) ∨ Rev(c)
     ∨ SRev

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

Spec ≜ Init ∧ □[Next]_vars  ∧ Fairness
----------------------------------------------------------------------------
CSSync ≜  Each client c ∈ Client is synchonized with the Server.
     ∀ c ∈ Client : (ds[c] = ds[Server]) ⇒ c2ss[c] = s2ss[c]

THEOREM Spec ⇒ □CSSync
============================================================================
```

\ * Modification History

\ * Last modified *Wed Jan* 02 21:11:22 *CST* 2019 by *hengxin*

\ * Created *Tue Oct* 09 16:33:18 *CST* 2018 by *hengxin*