

```

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  $\triangleq$   $\langle \textit{intVars}, \textit{ctxVars}, \textit{c2ss}, \textit{s2ss} \rangle$ 
14 └──────────────────┐
15 TypeOK  $\triangleq$ 
16      $\wedge$  TypeOKInt
17      $\wedge$  TypeOKCtx
18      $\wedge$  Comm(Cop)! TypeOK
19      $\wedge$   $\forall c \in \textit{Client} : \textit{IsSS}(\textit{c2ss}[c]) \wedge \textit{IsSS}(\textit{s2ss}[c])$ 
20 └──────────────────┐
21 Init  $\triangleq$ 
22      $\wedge$  InitInt
23      $\wedge$  InitCtx
24      $\wedge$  Comm(Cop)! Init
25      $\wedge$  c2ss = [c ∈ Client  $\mapsto$  EmptySS]
26      $\wedge$  s2ss = [c ∈ Client  $\mapsto$  EmptySS]
27 └──────────────────┐
  xForm: iteratively transform cop with a path through the 2D state space ss at some client.
32 xForm(cop, ss, cur)  $\triangleq$ 
33     LET u  $\triangleq$  Locate(cop, ss)
34     v  $\triangleq$  u  $\cup$  {cop.oid}
35     RECURSIVE xFormHelper(u, u, u, u)
36     xFormHelper(uh, vh, coph, xss)  $\triangleq$  xss: eXtra ss created during transformation
37     IF uh = cur THEN [xss  $\mapsto$  xss, xcop  $\mapsto$  coph]
38     ELSE LET e  $\triangleq$  CHOOSE e ∈ ss.edge : e.from = uh  $\wedge$  ClientOf(e.cop)  $\neq$  ClientOf(cop)
39         copprime  $\triangleq$  e.cop
40         uprime  $\triangleq$  e.to
41         vprime  $\triangleq$  vh  $\cup$  {copprime.oid}
42         coph2copprime  $\triangleq$  COT(coph, copprime)
43         copprime2coph  $\triangleq$  COT(copprime, coph)
44         IN xFormHelper(uprime, vprime, coph2copprime,
45             xss  $\oplus$  [node  $\mapsto$  {vprime},
46                 edge  $\mapsto$  {[from  $\mapsto$  vh, to  $\mapsto$  vprime, cop  $\mapsto$  copprime2coph],
47                     [from  $\mapsto$  uprime, to  $\mapsto$  vprime, cop  $\mapsto$  coph2copprime]}])
48         IN xFormHelper(u, v, cop, [node  $\mapsto$  {v}, edge  $\mapsto$  {[from  $\mapsto$  u, to  $\mapsto$  v, cop  $\mapsto$  cop]}])
49 └──────────────────┐
  Client c ∈ Client perform operation cop.

```

```

53  $ClientPerform(cop, c) \triangleq$ 
54   LET  $xform \triangleq xForm(cop, c2ss[c], ds[c])$   $xform: [xss, xcop]$ 
55   IN    $\wedge c2ss' = [c2ss \text{ EXCEPT } ![c] = @ \oplus xform.xss]$ 
56        $\wedge state' = [state \text{ EXCEPT } ![c] = Apply(xform.xcop.op, @)]$ 
    Client  $c \in Client$  generates an operation  $op$ .
60  $DoOp(c, op) \triangleq$ 
61   LET  $cop \triangleq [op \mapsto op, oid \mapsto [c \mapsto c, seq \mapsto cseq'[c]], ctx \mapsto ds[c]]$ 
62   IN    $\wedge ClientPerform(cop, c)$ 
63        $\wedge Comm(Cop)!CSend(cop)$ 
65  $DoIns(c) \triangleq$ 
66    $\exists ins \in \{op \in Ins : op.pos \in 1 \dots (Len(state[c]) + 1) \wedge op.ch \in chins \wedge op.pr = Priority[c]\} :$ 
67      $\wedge DoOp(c, ins)$ 
68      $\wedge chins' = chins \setminus \{ins.ch\}$ 
70  $DoDel(c) \triangleq$ 
71    $\exists del \in \{op \in Del : op.pos \in 1 \dots Len(state[c])\} :$ 
72      $\wedge DoOp(c, del)$ 
73      $\wedge \text{UNCHANGED } chins$ 
75  $Do(c) \triangleq$ 
76    $\wedge DoCtx(c)$ 
77    $\wedge \vee DoIns(c)$ 
78    $\vee DoDel(c)$ 
79    $\wedge \text{UNCHANGED } s2ss$ 
    Client  $c \in Client$  receives a message from the Server.
83  $Rev(c) \triangleq$ 
84    $\wedge Comm(Cop)!CRev(c)$ 
85    $\wedge \text{LET } cop \triangleq Head(cincoming[c])$ 
86   IN    $ClientPerform(cop, c)$ 
87    $\wedge RevCtx(c)$ 
88    $\wedge \text{UNCHANGED } \langle chins, s2ss \rangle$ 
89 |-----|
    The Server performs operation  $cop$ .
93  $ServerPerform(cop) \triangleq$ 
94   LET  $c \triangleq ClientOf(cop)$ 
95    $scur \triangleq ds[Server]$ 
96    $xform \triangleq xForm(cop, s2ss[c], scur)$   $xform: [xss, xcop]$ 
97    $xcop \triangleq xform.xcop$ 
98    $xcur \triangleq scur \cup \{cop.oid\}$ 
99   IN    $\wedge s2ss' = [cl \in Client \mapsto$ 
100        IF  $cl = c$ 
101        THEN  $s2ss[cl] \oplus xform.xss$ 
102        ELSE  $s2ss[cl] \oplus [node \mapsto \{xcur\},$ 
103             $edge \mapsto \{[from \mapsto scur, to \mapsto xcur, cop \mapsto xcop]\}]$ 

```

```

104         ]
105          $\wedge state' = [state \text{ EXCEPT } ![Server] = Apply(xcop.op, @)]$ 
106          $\wedge Comm(Cop)!SSendSame(c, xcop)$ 
107         The Server receives a message.
108
109      $SRev \triangleq$ 
110          $\wedge Comm(Cop)!SRev$ 
111          $\wedge \text{LET } cop \triangleq Head(sincoming)$ 
112          $\text{IN } ServerPerform(cop)$ 
113          $\wedge SRevCtx$ 
114          $\wedge \text{UNCHANGED } \langle chins, c2ss \rangle$ 
115
116 |-----|
117      $Next \triangleq$ 
118          $\vee \exists c \in Client : Do(c) \vee Rev(c)$ 
119          $\vee SRev$ 
120
121      $Fairness \triangleq$  There is no requirement that the clients ever generate operations.
122      $WF_{vars}(SRev \vee \exists c \in Client : Rev(c))$ 
123
124      $Spec \triangleq Init \wedge \Box[Next]_{vars} \wedge Fairness$ 
125 |-----|
126      $CSSync \triangleq$  Each client  $c \in Client$  is synchronized with the Server.
127      $\forall c \in Client : (ds[c] = ds[Server]) \Rightarrow c2ss[c] = s2ss[c]$ 
128
129 THEOREM  $Spec \Rightarrow \Box CSSync$ 
130 |-----|
131
132 \ * Modification History
133 \ * Last modified Mon Dec 31 11:05:08 CST 2018 by hengxin
134 \ * Created Tue Oct 09 16:33:18 CST 2018 by hengxin

```