

```

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 Integers, OT, TLCUtils, AdditionalFunctionOperators, AdditionalSequenceOperators |
8  |-----|
9  CONSTANTS
10   Client,      the set of client replicas
11   Server,      the (unique) server replica
12   Char,        set of characters allowed
13   InitState    the initial state of each replica

15   Replica  $\triangleq$  Client  $\cup$  {Server}

17   List  $\triangleq$  Seq(Char  $\cup$  Range(InitState))      all possible lists/strings
18   MaxLen  $\triangleq$  Cardinality(Char) + Len(InitState)  the max length of lists in any states;
19   We assume that all inserted elements are unique.

21   ClientNum  $\triangleq$  Cardinality(Client)
22   Priority  $\triangleq$  CHOOSE  $f \in [Client \rightarrow 1 \dots ClientNum] : Injective(f)$ 
  | Direction flags for edges in 2D state spaces and OT. |
26   Local  $\triangleq$  0
27   Remote  $\triangleq$  1
28 |-----|
29 ASSUME
30    $\wedge Range(InitState) \cap Char = \{\}$  due to the uniqueness requirement
31    $\wedge Priority \in [Client \rightarrow 1 \dots ClientNum]$ 
32 |-----|
  | The set of all operations. Note: The positions are indexed from 1. |
37   Rd  $\triangleq$  [type : { "Rd" }]
38   Del  $\triangleq$  [type : { "Del" }, pos : 1 .. MaxLen]
39   Ins  $\triangleq$  [type : { "Ins" }, pos : 1 .. (MaxLen + 1), ch : Char, pr : 1 .. ClientNum] pr: priority
41   Op  $\triangleq$  Ins  $\cup$  Del
42 |-----|
  | Cop: operation of type Op with context |
46   Oid  $\triangleq$  [c : Client, seq : Nat] operation identifier
47   Cop  $\triangleq$  [op : Op  $\cup$  {Nop}, oid : Oid, ctx : SUBSET Oid]

  | OT of two operations of type Cop. |
52   COT(lcop, rcop)  $\triangleq$  [lcop EXCEPT !.op = Xform(lcop.op, rcop.op), !.ctx = @  $\cup$  {rcop.oid}]
53 |-----|
54 VARIABLES
  | For the client replicas: |
58   cseq,      cseq[c]: local sequence number at client c  $\in$  Client

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

```

```

64    $c2ss$ ,       $c2ss[c]$ : the 2D state space at client  $c \in Client$ 
65    $s2ss$ ,       $s2ss[c]$ : the 2D state space maintained by the Server for client  $c \in Client$ 
66    $cur$ ,        $cur[r]$ : the current node of the 2D state space at replica  $r \in Replica$ 
      For all replicas
70    $state$ ,      $state[r]$ : state (the list content) of replica  $r \in Replica$ 
      For communication between the Server and the Clients:
74    $cincoming$ ,  $cincoming[c]$ : incoming channel at the client  $c \in Client$ 
75    $sincoming$ , incoming channel at the Server
      For model checking:
79    $chins$      a set of chars to insert
80 |-----|
81    $comm \triangleq$  INSTANCE CSComm WITH  $Msg \leftarrow Cop$ 
82 |-----|
83    $eVars \triangleq \langle chins \rangle$  variables for the environment
84    $cVars \triangleq \langle cseq \rangle$  variables for the clients
85    $commVars \triangleq \langle cincoming, sincoming \rangle$  variables for communication
86    $vars \triangleq \langle eVars, cVars, cur, commVars, c2ss, s2ss, state \rangle$  all variables
87 |-----|
      A 2D 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.
96    $IsSS(G) \triangleq$ 
97      $\wedge G = [node \mapsto G.node, edge \mapsto G.edge]$ 
98      $\wedge G.node \subseteq (SUBSET\ Oid)$ 
99      $\wedge G.edge \subseteq [from : G.node, to : G.node, cop : Cop, lr : \{Local, Remote\}]$ 
101   $TypeOK \triangleq$ 
      For the client replicas:
105    $\wedge cseq \in [Client \rightarrow Nat]$ 
      For the 2D state spaces:
109    $\wedge \forall c \in Client : IsSS(c2ss[c]) \wedge IsSS(s2ss[c])$ 
110    $\wedge cur \in [Replica \rightarrow SUBSET\ Oid]$ 
111    $\wedge state \in [Replica \rightarrow List]$ 
      For communication between the server and the clients:
115    $\wedge comm!TypeOK$ 
      For model checking:
119    $\wedge chins \subseteq Char$ 
120 |-----|
121   $Init \triangleq$ 
      For the client replicas:
125    $\wedge cseq = [c \in Client \mapsto 0]$ 
      For the 2D state spaces:

```

```

129     $\wedge c2ss = [c \in Client \mapsto [node \mapsto \{\{\}\}, edge \mapsto \{\}]]$ 
130     $\wedge s2ss = [c \in Client \mapsto [node \mapsto \{\{\}\}, edge \mapsto \{\}]]$ 
131     $\wedge cur = [r \in Replica \mapsto \{\}]$ 
    For all replicas:
135     $\wedge state = [r \in Replica \mapsto InitState]$ 
    For communication between the server and the clients:
139     $\wedge comm!Init$ 
    For model checking:
143     $\wedge chins = Char$ 
144 |-----|
    Locate the node in the 2D state space  $ss$  which matches the context  $ctx$  of  $cop$ .
148  $Locate(cop, ss) \triangleq \text{CHOOSE } n \in (ss.node) : n = cop.ctx$ 
     $xForm$ : iteratively transform  $cop$  with a path through the 2D state space  $ss$  at some client,
    following the edges with the direction flag  $d$ .
154  $xForm(cop, ss, current, d) \triangleq$ 
155   LET  $u \triangleq Locate(cop, ss)$ 
156    $v \triangleq u \cup \{cop.oid\}$ 
157   RECURSIVE  $xFormHelper(-, -, -, -)$ 
158   'h' stands for "helper";  $xss$ :  $eXtra$   $ss$  created during transformation
159    $xFormHelper(uh, vh, coph, xss) \triangleq$ 
160     IF  $uh = current$ 
161       THEN  $xss$ 
162     ELSE LET  $e \triangleq \text{CHOOSE } e \in ss.edge : e.from = uh \wedge e.lr = d$ 
163        $uprime \triangleq e.to$ 
164        $copprime \triangleq e.cop$ 
165        $coph2copprime \triangleq COT(coph, copprime)$ 
166        $copprime2coph \triangleq COT(copprime, coph)$ 
167        $vprime \triangleq vh \cup \{copprime.oid\}$ 
168       IN  $xFormHelper(uprime, vprime, coph2copprime,$ 
169          $[xss \text{ EXCEPT } !.node = @ \circ \langle vprime \rangle,$ 
170           the order of recording edges here is important
171           so that the last one is labeled with the final transformed operation
172            $!.edge = @ \circ \langle [from \mapsto vh, to \mapsto vprime, cop \mapsto copprime2coph, lr \mapsto$ 
173              $[from \mapsto uprime, to \mapsto vprime, cop \mapsto coph2copprime,$ 
174             IN  $xFormHelper(u, v, cop, [node \mapsto \langle v \rangle,$ 
175                $edge \mapsto \langle [from \mapsto u, to \mapsto v, cop \mapsto cop, lr \mapsto 1 - d] \rangle])$ 
176 |-----|
    Client  $c \in Client$  perform operation  $cop$  guided by the direction flag  $d$ .
180  $ClientPerform(cop, c, d) \triangleq$ 
181   LET  $xss \triangleq xForm(cop, c2ss[c], cur[c], d)$ 
182    $xn \triangleq xss.node$ 
183    $xe \triangleq xss.edge$ 
184    $xcur \triangleq Last(xn)$ 
185    $xcop \triangleq Last(xe).cop$ 

```

```

186     IN     $\wedge c2ss' = [c2ss \text{ EXCEPT } ![c].node = @ \cup Range(xn),$ 
187            $![c].edge = @ \cup Range(xe)]$ 
188      $\wedge cur' = [cur \text{ EXCEPT } ![c] = xcur]$ 
189      $\wedge state' = [state \text{ EXCEPT } ![c] = Apply(xcop.op, @)]$ 
    Client  $c \in Client$  generates an operation  $op$ .
193    $DoOp(c, op) \triangleq$ 
194      $\wedge cseq' = [cseq \text{ EXCEPT } ![c] = @ + 1]$ 
195      $\wedge \text{LET } cop \triangleq [op \mapsto op, oid \mapsto [c \mapsto c, seq \mapsto cseq'[c]], ctx \mapsto cur[c]]$ 
196     IN     $\wedge ClientPerform(cop, c, Remote)$ 
197      $\wedge comm! CSend(cop)$ 

199    $DoIns(c) \triangleq$ 
200      $\exists ins \in \{op \in Ins : op.pos \in 1 \dots (Len(state[c]) + 1) \wedge op.ch \in chins \wedge op.pr = Priority[c]\} :$ 
201      $\wedge DoOp(c, ins)$ 
202      $\wedge chins' = chins \setminus \{ins.ch\}$  We assume that all inserted elements are unique.

204    $DoDel(c) \triangleq$ 
205      $\exists del \in \{op \in Del : op.pos \in 1 \dots Len(state[c])\} :$ 
206      $\wedge DoOp(c, del)$ 
207      $\wedge \text{UNCHANGED } \langle eVars \rangle$ 

209    $Do(c) \triangleq$ 
210      $\wedge \vee DoIns(c)$ 
211      $\vee DoDel(c)$ 
212      $\wedge \text{UNCHANGED } \langle s2ss \rangle$ 
    Client  $c \in Client$  receives a message from the Server.

216    $Rev(c) \triangleq$ 
217      $\wedge comm! CRev(c)$ 
218      $\wedge \text{LET } cop \triangleq Head(cincomig[c])$  the received (transformed) operation
219     IN     $ClientPerform(cop, c, Local)$ 
220      $\wedge \text{UNCHANGED } \langle eVars, cVars, s2ss \rangle$ 
221 |-----|
    The Server performs operation  $cop$ .

225    $ServerPerform(cop) \triangleq$ 
226      $\text{LET } c \triangleq cop.oid.c$ 
227      $scur \triangleq cur[Server]$ 
228      $xss \triangleq xForm(cop, s2ss[c], scur, Remote)$ 
229      $xn \triangleq xss.node$ 
230      $xe \triangleq xss.edge$ 
231      $xcur \triangleq Last(xn)$ 
232      $xcop \triangleq Last(xe).cop$ 
233     IN     $\wedge s2ss' = [cl \in Client \mapsto$ 
234           IF  $cl = c$ 
235           THEN  $[s2ss[cl] \text{ EXCEPT } !.node = @ \cup Range(xn),$ 
236            $!.edge = @ \cup Range(xe)]$ 

```

```

237             ELSE  $[s2ss[cl]$  EXCEPT  $!.node = @ \cup \{xcur\}$ ,
238                  $!.edge = @ \cup \{[from \mapsto scur, to \mapsto xcur,$ 
239                      $cop \mapsto xcop, lr \mapsto Remote]\}$ 
240             ]
241              $\wedge cur' = [cur$  EXCEPT  $![Server] = xcur]$ 
242              $\wedge state' = [state$  EXCEPT  $![Server] = Apply(xcop.op, @)]$ 
243              $\wedge comm!SSendSame(c, xcop)$  broadcast the transformed operation
244         ]
245         The Server receives a message.
246     SRev  $\triangleq$ 
247          $\wedge comm!SRev$ 
248          $\wedge$  LET  $cop \triangleq Head(sincoming)$ 
249             IN  $ServerPerform(cop)$ 
250          $\wedge$  UNCHANGED  $\langle eVars, cVars, c2ss \rangle$ 
251     ]
252     Next  $\triangleq$ 
253          $\vee \exists c \in Client : Do(c) \vee Rev(c)$ 
254          $\vee SRev$ 
255     ]
256     Spec  $\triangleq Init \wedge \Box[Next]_{vars} \wedge WF_{vars}(SRev \vee \exists c \in Client : Rev(c))$ 
257 ]
258 In Jupiter (not limited to XJupiter), each client synchronizes with the server. In XJupiter, this
is expressed as the following CSSync property.
259 CSSync  $\triangleq$ 
260      $\forall c \in Client : (cur[c] = cur[Server]) \Rightarrow c2ss[c] = s2ss[c]$ 
261 ]
262
263 \* Modification History
264 \* Last modified Wed Nov 07 13:35:57 CST 2018 by hengxin
265 \* Created Tue Oct 09 16:33:18 CST 2018 by hengxin

```