

```

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  $f \in [Client \rightarrow 1 \dots ClientNum] : Injective(f)$ 

24  direction flags
25  Local  $\triangleq$  0
26  Remote  $\triangleq$  1
27  |-----|
28  ASSUME
29   $\wedge Range(InitState) \cap Char = \{\}$   due to the uniqueness requirement
30   $\wedge Priority \in [Client \rightarrow 1 \dots ClientNum]$ 
31  |-----|
   | The set of all operations. Note: The positions are indexed from 1. |
36  Rd  $\triangleq$  [type : {"Rd"}]
37  Del  $\triangleq$  [type : {"Del"}, pos : 1 .. MaxLen]
38  Ins  $\triangleq$  [type : {"Ins"}, pos : 1 .. (MaxLen + 1), ch : Char, pr : 1 .. ClientNum]  pr: priority
40  Op  $\triangleq$  Ins  $\cup$  Del
41  |-----|
   | Cop: operation of type Op with context |
45  Oid  $\triangleq$  [c : Client, seq : Nat]  operation identifier
46  Cop  $\triangleq$  [op : Op  $\cup$  {Nop}, oid : Oid, ctx : SUBSET Oid]

   | OT of two operations of type Cop. |
51  COT(lcp, rcp)  $\triangleq$  [lcp EXCEPT !.op = Xform(lcp.op, rcp.op), !.ctx = @  $\cup$  {rcp.oid}]
52  |-----|
53  VARIABLES
   | For the client replicas: |
57  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.

63 *css*, *css*[*c*]: the 2D state space at client $c \in Client$

64 *ccur*, *cur*[*c*]: the current node of *css*[*c*]

65 *sss*, *sss*[*c*]: the 2D state space maintained by the *Server* for client $c \in Client$

66 *scur*, *scur*[*c*]: the current node of *sss*[*c*]

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 *cssVars* \triangleq $\langle css, ccur \rangle$ variables for 2D state spaces at clients

86 *sssVars* \triangleq $\langle sss, scur \rangle$ variables for 2D state spaces at the *Server*

87 *commVars* \triangleq $\langle cincoming, sincoming \rangle$ variables for communication

88 *vars* \triangleq $\langle eVars, cVars, commVars, cssVars, sssVars, state \rangle$ all variables

89 |

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.

98 *IsSS*(*G*) \triangleq

99 $\wedge G = [node \mapsto G.node, edge \mapsto G.edge]$

100 $\wedge G.node \subseteq (SUBSET \ Oid)$

101 $\wedge G.edge \subseteq [from : G.node, to : G.node, cop : Cop, lr : \{Local, Remote\}]$

103 *TypeOK* \triangleq

For the client replicas:

107 $\wedge cseq \in [Client \rightarrow Nat]$

For the 2D state spaces:

111 $\wedge \forall c \in Client : IsSS(css[c]) \wedge IsSS(sss[c])$

112 $\wedge ccur \in [Client \rightarrow SUBSET \ Oid]$

113 $\wedge scur \in [Client \rightarrow SUBSET \ Oid]$

114 $\wedge state \in [Replica \rightarrow List]$

For communication between the server and the clients:

118 $\wedge comm!TypeOK$

For model checking:

```

122       $\wedge chins \subseteq Char$ 
123  |-----|
124  | The Init predicate.
125  |-----|
127  Init  $\triangleq$ 
128  | For the client replicas:
129  |-----|
131       $\wedge cseq = [c \in Client \mapsto 0]$ 
132  | For the 2D state spaces:
133  |-----|
135       $\wedge css = [c \in Client \mapsto [node \mapsto \{\{\}\}, edge \mapsto \{\}]]$ 
136       $\wedge ccur = [c \in Client \mapsto \{\}]$ 
137       $\wedge sss = [c \in Client \mapsto [node \mapsto \{\{\}\}, edge \mapsto \{\}]]$ 
138       $\wedge scur = [c \in Client \mapsto \{\}]$ 
139  | For all replicas:
140  |-----|
142       $\wedge state = [r \in Replica \mapsto InitState]$ 
143  | For communication between the server and the clients:
144  |-----|
146       $\wedge comm!Init$ 
147  | For model checking:
148  |-----|
150       $\wedge chins = Char$ 
151  |-----|
152  | Locate the node in the 2D state space ss which matches the context ctx of cop.
153  |-----|
155  Locate(cop, ss)  $\triangleq$  CHOOSE  $n \in (ss.node) : n = cop.ctx$ 
156  |
157  | xForm: iteratively transform cop with a path through the 2D state space ss at some client,
158  | following the edges with the direction flag d.
159  |-----|
162  xForm(cop, ss, cur, d)  $\triangleq$ 
163      LET  $u \triangleq Locate(cop, ss)$ 
164       $v \triangleq u \cup \{cop.oid\}$ 
165      RECURSIVE xFormHelper( $-, -, -, -$ )
166      | 'h' stands for "helper"; xss: eXtra ss created during transformation
167      xFormHelper(uh, vh, coph, xss)  $\triangleq$ 
168          IF  $uh = cur$ 
169          THEN xss
170          ELSE LET  $e \triangleq$  CHOOSE  $e \in ss.edge : e.from = uh \wedge e.lr = d$ 
171               $uprime \triangleq e.to$ 
172               $copprime \triangleq e.cop$ 
173               $coph2copprime \triangleq COT(coph, copprime)$ 
174               $copprime2coph \triangleq COT(copprime, coph)$ 
175               $vprime \triangleq vh \cup \{copprime.oid\}$ 
176              IN xFormHelper(uprime, vprime, coph2copprime,
177                  [xss EXCEPT  $!.node = @ \circ \langle vprime \rangle$ ,
178                      | the order of recording edges here is important
179                      | so that the last one is labeled with the final transformed operation
180                      |  $!.edge = @ \circ \langle [from \mapsto vh, to \mapsto vprime, cop \mapsto copprime2coph, lr \mapsto$ 
181                      |  $[from \mapsto uprime, to \mapsto vprime, cop \mapsto coph2copprime,$ 

```

```

182     IN    $xFormHelper(u, v, cop, [node \mapsto \langle v \rangle,$ 
183            $edge \mapsto \langle [from \mapsto u, to \mapsto v, cop \mapsto cop, lr \mapsto 1 - d] \rangle])$ 
184 |-----|
      Client  $c \in Client$  perform operation  $cop$  guided by the direction flag  $d$ .
188    $ClientPerform(cop, c, d) \triangleq$ 
189     LET  $xss \triangleq xForm(cop, css[c], ccur[c], d)$ 
190        $xn \triangleq xss.node$ 
191        $xe \triangleq xss.edge$ 
192        $xcur \triangleq Last(xn)$ 
193        $xcop \triangleq Last(xe).cop$ 
194     IN    $\wedge css' = [css \text{ EXCEPT } ![c].node = @ \cup Range(xn),$ 
195            $![c].edge = @ \cup Range(xe)]$ 
196            $\wedge ccur' = [ccur \text{ EXCEPT } ![c] = xcur]$ 
197            $\wedge state' = [state \text{ EXCEPT } ![c] = Apply(xcop.op, @)]$ 
      Client  $c \in Client$  issues an operation  $op$ .
201    $DoOp(c, op) \triangleq$   $op$ : the raw operation generated by the client  $c \in Client$ 
202      $\wedge cseq' = [cseq \text{ EXCEPT } ![c] = @ + 1]$ 
203      $\wedge$  LET  $cop \triangleq [op \mapsto op, oid \mapsto [c \mapsto c, seq \mapsto cseq'[c]], ctx \mapsto ccur[c]]$ 
204       IN    $\wedge ClientPerform(cop, c, Remote)$ 
205        $\wedge comm! CSend(cop)$ 

207    $DoIns(c) \triangleq$ 
208      $\exists ins \in Ins :$ 
209        $\wedge ins.pos \in 1 \dots (Len(state[c]) + 1)$ 
210        $\wedge ins.ch \in chins$ 
211        $\wedge ins.pr = Priority[c]$ 
212        $\wedge chins' = chins \setminus \{ins.ch\}$   $\text{We assume that all inserted elements are unique.}$ 
213        $\wedge DoOp(c, ins)$ 
214        $\wedge \text{UNCHANGED } \langle sssVars \rangle$ 

216    $DoDel(c) \triangleq$ 
217      $\exists del \in Del :$ 
218        $\wedge del.pos \in 1 \dots Len(state[c])$ 
219        $\wedge DoOp(c, del)$ 
220        $\wedge \text{UNCHANGED } \langle sssVars, eVars \rangle$ 

222    $Do(c) \triangleq$ 
223      $\vee DoIns(c)$ 
224      $\vee DoDel(c)$ 
      Client  $c \in Client$  receives a message from the Server.
228    $Rev(c) \triangleq$ 
229      $\wedge comm! CRev(c)$ 
230      $\wedge$  LET  $cop \triangleq Head(cincoming[c])$   $\text{the received (transformed) operation}$ 
231       IN    $ClientPerform(cop, c, Local)$ 
232        $\wedge \text{UNCHANGED } \langle eVars, cVars, sssVars \rangle$ 

```

233 |

The *Server* performs operation *cop*.

237 $ServerPerform(cop) \triangleq$

238 LET $c \triangleq cop.oid.c$

239 $xss \triangleq xForm(cop, sss[c], scur[c], Remote)$

240 $xn \triangleq xss.node$

241 $xe \triangleq xss.edge$

242 $xcur \triangleq Last(xn)$

243 $xcop \triangleq Last(xe).cop$

244 IN $\wedge sss' = [cl \in Client \mapsto$

245 IF $cl = c$

246 THEN $[sss[cl] \text{ EXCEPT } !.node = @ \cup Range(xn),$

247 $!.edge = @ \cup Range(xe)]$

248 ELSE LET $scurcl \triangleq scur[cl]$

249 $scurclprime \triangleq scurcl \cup \{cop.oid\}$

250 IN $[sss[cl] \text{ EXCEPT } !.node = @ \cup \{scurclprime\},$

251 $!.edge = @ \cup \{[from \mapsto scurcl, to \mapsto scurclprime,$

252 $cop \mapsto xcop, lr \mapsto Remote]\}]$

253]

254 $\wedge scur' = [cl \in Client \mapsto$

255 IF $cl = c$ THEN $xcur$ ELSE $scur[cl] \cup \{cop.oid\}]$

256 $\wedge state' = [state \text{ EXCEPT } ![Server] = Apply(xcop.op, @)]$

257 $\wedge comm!SSendSame(c, xcop)$ broadcast the transformed operation

The *Server* receives a message.

261 $SRev \triangleq$

262 $\wedge comm!SRev$

263 $\wedge \text{LET } cop \triangleq Head(sincoming)$

264 IN $ServerPerform(cop)$

265 $\wedge \text{UNCHANGED } \langle eVars, cVars, cssVars \rangle$

266 |

The next-state relation.

270 $Next \triangleq$

271 $\vee \exists c \in Client : Do(c) \vee Rev(c)$

272 $\vee SRev$

The *Spec.* (*TODO*: Check the fairness condition.)

276 $Spec \triangleq Init \wedge \Box[Next]_{vars} \wedge WF_{vars}(Next)$

277 |

In *Jupiter* (not limited to *XJupiter*), each client synchronizes with the server. In *XJupiter*, this is expressed as the following *CSSync* property.

282 ASSUME ($TLCSet(1, \langle \text{"SameOids"}, 0 \rangle)$)

283 $CSSync \triangleq$

284 $\forall c \in Client :$

285 $(ccur[c] = scur[c] \wedge TLCCnt(1, 100)) \Rightarrow css[c] = sss[c]$

286 |

* Modification History
* *Last modified Thu Oct 11 20:13:07 CST 2018 by hengxin*
* Created *Tue Oct 09 16:33:18 CST 2018 by hengxin*