MODULE $AJupiter$

Specification of the *Jupiter* protocol presented by *Attiya* and others.

EXTENDS OT, $Jupiter$

Messages between the *Server* and the Clients. There are two kinds of messages according to their destinations.

$Msg \triangleq [c : Client,\ ack : Int,\ op : Op \cup \{Nop\}] \cup$ messages sent to the *Server* from a client $c \in Client$

$\quad [ack : Int,\ op : Op \cup \{Nop\}]$ messages broadcast to Clients from the *Server*

VARIABLES

For the client replicas:

$cbuf$, $cbuf[c]$: buffer (of operations) at the client $c \in Client$

$crec$, $crec[c]$: the number of new messages have been received by the client $c \in Client$ since the last time a message was sent

For the server replica:

$sbuf$, $sbuf[c]$: buffer (of operations) at the *Server*, one per client $c \in Client$

$srec$, $srec[c]$: the number of new messages have been ..., one per client $c \in Client$

For model checking:

$chins$ a set of chars to insert

$comm \triangleq$ INSTANCE $CSComm$ WITH $Msg \leftarrow Msg$

$eVars \triangleq \langle chins \rangle$ variables for the environment

$cVars \triangleq \langle cbuf,\ crec \rangle$ variables for the clients

$ecVars \triangleq \langle eVars,\ cVars \rangle$ variables for the clients and the environment

$sVars \triangleq \langle sbuf,\ srec \rangle$ variables for the server

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

$Vars \triangleq \langle eVars,\ cVars,\ sVars,\ commVars,\ state \rangle$ all variables

$TypeOK \triangleq$

For the client replicas:

$\quad \land\ cbuf \in [Client \to Seq(Op \cup \{Nop\})]$

$\quad \land\ crec \in [Client \to Int]$

For the server replica:

$\quad \land\ sbuf \in [Client \to Seq(Op \cup \{Nop\})]$

$\quad \land\ srec \in [Client \to Int]$

For all replicas.

$\quad \land\ state \in [Replica \to List]$

For communication between the server and the clients:

$\quad \land\ comm\,!\,TypeOK$

For model checking:

```
63      ∧ chins ∈ SUBSET Char
64  ----------------------------------------------------------------
    The Init predicate.
68  Init ≜
69      ∧ chins = Char
        For the client replicas:
73      ∧ cbuf = [c ∈ Client ↦ ⟨⟩]
74      ∧ crec = [c ∈ Client ↦ 0]
        For the server replica:
78      ∧ sbuf = [c ∈ Client ↦ ⟨⟩]
79      ∧ srec = [c ∈ Client ↦ 0]
        For all replicas.
83      ∧ state = [r ∈ Replica ↦ InitState]
        For communication between the server and the clients:
87      ∧ comm!Init
88  ----------------------------------------------------------------
    Client c ∈ Client issues an operation op.
92  DoOp(c, op) ≜
93          ∧ state' = [state EXCEPT ![c] = Apply(op, @)]
94          ∧ cbuf' = [cbuf EXCEPT ![c] = Append(@, op)]
95          ∧ crec' = [crec EXCEPT ![c] = 0]
96          ∧ comm!CSend([c ↦ c, ack ↦ crec[c], op ↦ op])

98  DoIns(c) ≜
99      ∃ ins ∈ Ins :
100         ∧ ins.pos ∈ 1 .. (Len(state[c]) + 1)
101         ∧ ins.ch ∈ chins
102         ∧ ins.pr = Priority[c]
103         ∧ chins' = chins \ {ins.ch}  We assume that all inserted elements are unique.
104         ∧ DoOp(c, ins)
105         ∧ UNCHANGED sVars

107 DoDel(c) ≜
108     ∃ del ∈ Del :
109         ∧ del.pos ∈ 1 .. Len(state[c])
110         ∧ DoOp(c, del)
111         ∧ UNCHANGED ⟨sVars, eVars⟩

113 Do(c) ≜
114         ∨ DoIns(c)
115         ∨ DoDel(c)

    Client c ∈ Client receives a message from the Server.
```

```
Rev(c) ≜
    ∧ comm!CRev(c)
    ∧ crec' = [crec EXCEPT ![c] = @ + 1]
    ∧ LET m ≜ Head(cincoming[c])
          cBuf ≜ cbuf[c]   the buffer at client c ∈ Client
          cShiftedBuf ≜ SubSeq(cBuf, m.ack + 1, Len(cBuf))   buffer shifted
          xop ≜ XformOpOps(m.op, cShiftedBuf)   transform op vs. shifted buffer
          xcBuf ≜ XformOpsOp(cShiftedBuf, m.op)   transform shifted buffer vs. op
      IN  ∧ cbuf' = [cbuf EXCEPT ![c] = xcBuf]
          ∧ state' = [state EXCEPT ![c] = Apply(xop, @)]   apply the transformed operation xop
    ∧ UNCHANGED ⟨sVars, eVars⟩
```

The *Server* receives a message.

```
SRev ≜
    ∧ comm!SRev
    ∧ LET m ≜ Head(sincoming)   the message to handle with
          c ≜ m.c   the client c ∈ Client that sends this message
          cBuf ≜ sbuf[c]   the buffer at the Server for client c ∈ Client
          cShiftedBuf ≜ SubSeq(cBuf, m.ack + 1, Len(cBuf))   buffer shifted
          xop ≜ XformOpOps(m.op, cShiftedBuf)   transform op vs. shifted buffer
          xcBuf ≜ XformOpsOp(cShiftedBuf, m.op)   transform shifted buffer vs. op
      IN  ∧ srec' = [cl ∈ Client ↦
                      IF cl = c
                      THEN srec[cl] + 1   receive one more operation from client c ∈ Client
                      ELSE 0]   reset srec for other clients than c ∈ Client
          ∧ sbuf' = [cl ∈ Client ↦
                      IF cl = c
                      THEN xcBuf   transformed buffer for client c ∈ Client
                      ELSE Append(sbuf[cl], xop)]   store transformed xop into other clients' bufs
          ∧ state' = [state EXCEPT ![Server] = Apply(xop, @)]   apply the transformed operation
          ∧ comm!SSend(c, [cl ∈ Client ↦ [ack ↦ srec[cl], op ↦ xop]])
    ∧ UNCHANGED ecVars
```

The safety properties to check: Eventual Convergence (*EC*), Quiescent Consistency (*QC*), Strong Eventual Convergence (*SEC*), Weak *List* Specification, (*WLSpec*), and Strong *List* Specification, (*SLSpec*).

Eventual Consistency (*EC*)

Quiescent Consistency (*QC*)

```
QC ≜ comm!EmptyChannel ⇒ Cardinality(Range(state)) = 1

INSTANCE JupiterH
```

Strong Eventual Consistency (*SEC*)

\ * Modification History
\ * *Last* modified *Tue Sep* 11 21:06:16 *CST* 2018 by *hengxin*
\ * Created Sat *Jun* 23 17:14:18 *CST* 2018 by *hengxin*