PROMELA FILE TRANSFER PROTOCOL F

Here is a complete listing of the set of file transfer protocol validation models that were developed in Chapter 7, with the modifications discussed in Chapter 14. It is an error to retrieve fewer parameters in a message input from a channel than defined in the corresponding channel declaration. Unused parameter fields are therefore set to zero in sends and receives.

```
 1 /*
 2  * PROMELA Validation Model - startup script
 3  */
 4
 5 #include "defines"
 6 #include "user"
 7 #include "present"
 8 #include "session"
 9 #include "fserver"
10 #include "flow_cl"
11 #include "datalink"
12
13 init
14 {   atomic {
15       run userprc(0); run userprc(1);
16       run present(0); run present(1);
17       run session(0); run session(1);
18       run fserver(0); run fserver(1);
19       run fc(0);      run fc(1);
20       run data_link()
21     }
22 }
23
24 /*
25  * Global Definitions
26  */
27
28 #define LOSS               0       /* message loss   */
29 #define DUPS               0       /* duplicate msgs */
30 #define QSZ         2      /* queue size     */
31
```

```
32 mtype = {
33     red, white, blue,
34     abort, accept, ack, sync_ack, close, connect,
35     create, data, eof, open, reject, sync, transfer,
36     FATAL, NON_FATAL, COMPLETE
37     }
38
39 chan use_to_pres[2] = [QSZ] of { byte };
40 chan pres_to_use[2] = [QSZ] of { byte };
41 chan pres_to_ses[2] = [QSZ] of { byte };
42 chan ses_to_pres[2] = [QSZ] of { byte, byte };
43 chan ses_to_flow[2] = [QSZ] of { byte, byte };
44 chan flow_to_ses[2] = [QSZ] of { byte, byte };
45 chan dll_to_flow[2] = [QSZ] of { byte, byte };
46 chan flow_to_dll[2] = [QSZ] of { byte, byte };
47 chan ses_to_fsrv[2] = [0] of { byte };
48 chan fsrv_to_ses[2] = [0] of { byte };
49
50 /*
51  * User Layer Validation Model
52  */
53
54 proctype userprc(bit n)
55 {
56     use_to_pres[n]!transfer;
57     if
58     :: pres_to_use[n]?accept -> goto Done
59     :: pres_to_use[n]?reject -> goto Done
60     :: use_to_pres[n]!abort  -> goto Aborted
61     fi;
62 Aborted:
63     if
64     :: pres_to_use[n]?accept -> goto Done
65     :: pres_to_use[n]?reject -> goto Done
66     fi;
67 Done:
68     skip
69 }
70
71 /*
72  * Presentation Layer Validation Model
73  */
74
75 proctype present(bit n)
76 {   byte status, uabort;
77
78 endIDLE:
79     do
80     :: use_to_pres[n]?transfer ->
81             uabort = 0;
82             break
83     :: use_to_pres[n]?abort ->
84             skip
85     od;
```

```
 86
 87 TRANSFER:
 88     pres_to_ses[n]!transfer;
 89     do
 90     :: use_to_pres[n]?abort ->
 91             if
 92             :: (!uabort) ->
 93                     uabort = 1;
 94                     pres_to_ses[n]!abort
 95             :: (uabort) ->
 96                     assert(1+1!=2)
 97             fi
 98     :: ses_to_pres[n]?accept,0 ->
 99             goto DONE
100     :: ses_to_pres[n]?reject(status) ->
101             if
102             :: (status == FATAL || uabort) ->
103                     goto FAIL
104             :: (status == NON_FATAL && !uabort) ->
105 progress:           goto TRANSFER
106             fi
107     od;
108 DONE:
109     pres_to_use[n]!accept;
110     goto endIDLE;
111 FAIL:
112     pres_to_use[n]!reject;
113     goto endIDLE
114 }
115
116 /*
117  * Session Layer Validation Model
118  */
119
120 proctype session(bit n)
121 {   bit toggle;
122     byte type, status;
123
124 endIDLE:
125     do
126     :: pres_to_ses[n]?type ->
127             if
128             :: (type == transfer) ->
129                     goto DATA_OUT
130             :: (type != transfer)   /* ignore */
131             fi
132     :: flow_to_ses[n]?type,0 ->
133             if
134             :: (type == connect) ->
135                     goto DATA_IN
136             :: (type != connect)    /* ignore */
137             fi
138     od;
139
```

```
140 DATA_IN:            /* 1. prepare local file fsrver */
141     ses_to_fsrv[n]!create;
142     do
143     :: fsrv_to_ses[n]?reject ->
144             ses_to_flow[n]!reject,0;
145             goto endIDLE
146     :: fsrv_to_ses[n]?accept ->
147             ses_to_flow[n]!accept,0;
148             break
149     od;
150                     /* 2. Receive the data, upto eof */
151     do
152     :: flow_to_ses[n]?data,0 ->
153             ses_to_fsrv[n]!data
154     :: flow_to_ses[n]?eof,0 ->
155             ses_to_fsrv[n]!eof;
156             break
157     :: pres_to_ses[n]?transfer ->
158             ses_to_pres[n]!reject(NON_FATAL)
159     :: flow_to_ses[n]?close,0 ->    /* remote user aborted */
160             ses_to_fsrv[n]!close;
161             break
162     :: timeout ->           /* got disconnected */
163             ses_to_fsrv[n]!close;
164             goto endIDLE
165     od;
166                     /* 3. Close the connection */
167     ses_to_flow[n]!close,0;
168     goto endIDLE;
169
170 DATA_OUT:           /* 1. prepare local file fsrver */
171     ses_to_fsrv[n]!open;
172     if
173     :: fsrv_to_ses[n]?reject ->
174             ses_to_pres[n]!reject(FATAL);
175             goto endIDLE
176     :: fsrv_to_ses[n]?accept ->
177             skip
178     fi;
179                     /* 2. initialize flow control */
180     ses_to_flow[n]!sync,toggle;
181     do
182     :: atomic {
183             flow_to_ses[n]?sync_ack,type ->
184             if
185             :: (type != toggle)
186             :: (type == toggle) -> break
187             fi
188        }
189     :: timeout ->
190             ses_to_fsrv[n]!close;
191             ses_to_pres[n]!reject(FATAL);
192             goto endIDLE
193     od;
```

```
194     toggle = 1 - toggle;
195                     /* 3. prepare remote file fsrver */
196     ses_to_flow[n]!connect,0;
197     if
198     :: flow_to_ses[n]?reject,0 ->
199             ses_to_fsrv[n]!close;
200             ses_to_pres[n]!reject(FATAL);
201             goto endIDLE
202     :: flow_to_ses[n]?connect,0 ->
203             ses_to_fsrv[n]!close;
204             ses_to_pres[n]!reject(NON_FATAL);
205             goto endIDLE
206     :: flow_to_ses[n]?accept,0 ->
207             skip
208     :: timeout ->
209             ses_to_fsrv[n]!close;
210             ses_to_pres[n]!reject(FATAL);
211             goto endIDLE
212     fi;
213                     /* 4. Transmit the data, upto eof */
214     do
215     :: fsrv_to_ses[n]?data ->
216             ses_to_flow[n]!data,0
217     :: fsrv_to_ses[n]?eof ->
218             ses_to_flow[n]!eof,0;
219             status = COMPLETE;
220             break
221     :: pres_to_ses[n]?abort ->      /* local user aborted */
222             ses_to_fsrv[n]!close;
223             ses_to_flow[n]!close,0;
224             status = FATAL;
225             break
226     od;
227                     /* 5. Close the connection */
228     do
229     :: pres_to_ses[n]?abort         /* ignore */
230     :: flow_to_ses[n]?close,0 ->
231             if
232             :: (status == COMPLETE) ->
233                     ses_to_pres[n]!accept,0
234             :: (status != COMPLETE) ->
235                     ses_to_pres[n]!reject(status)
236             fi;
237             break
238     :: timeout ->
239             ses_to_pres[n]!reject(FATAL);
240             break
241     od;
242     goto endIDLE
243 }
244
245 /*
246  * File Server Validation Model
247  */
```

```
248
249 proctype fserver(bit n)
250 {
251 end:        do
252     :: ses_to_fsrv[n]?create ->     /* incoming */
253             if
254             :: fsrv_to_ses[n]!reject
255             :: fsrv_to_ses[n]!accept ->
256                     do
257                     :: ses_to_fsrv[n]?data
258                     :: ses_to_fsrv[n]?eof -> break
259                     :: ses_to_fsrv[n]?close -> break
260                     od
261             fi
262     :: ses_to_fsrv[n]?open ->               /* outgoing */
263             if
264             :: fsrv_to_ses[n]!reject
265             :: fsrv_to_ses[n]!accept ->
266                     do
267                     :: fsrv_to_ses[n]!data -> progress: skip
268                     :: ses_to_fsrv[n]?close -> break
269                     :: fsrv_to_ses[n]!eof -> break
270                     od
271             fi
272     od
273 }
274
275 /*
276  * Flow Control Layer Validation Model
277  */
278
279 #define true        1
280 #define false       0
281
282 #define M   4       /* range sequence numbers   */
283 #define W   2       /* window size: M/2         */
284
285 proctype fc(bit n)
286 {   bool    busy[M];        /* outstanding messages    */
287     byte    q;              /* seq# oldest unacked msg */
288     byte    m;              /* seq# last msg received  */
289     byte    s;              /* seq# next msg to send   */
290     byte    window;         /* nr of outstanding msgs  */
291     byte    type;           /* msg type                */
292     bit     received[M];    /* receiver housekeeping   */
293     bit     x;              /* scratch variable        */
294     byte    p;              /* seq# of last msg acked  */
295     byte    I_buf[M], O_buf[M];     /* message buffers */
296
297     /* sender part */
298 end:        do
299     :: atomic {
300        (window < W  && len(ses_to_flow[n]) >  0
301                     && len(flow_to_dll[n]) < QSZ) ->
```

```
302                     ses_to_flow[n]?type,x;
303                     window = window + 1;
304                     busy[s] = true;
305                     O_buf[s] = type;
306                     flow_to_dll[n]!type,s;
307                     if
308                     :: (type != sync) ->
309                             s = (s+1)%M
310                     :: (type == sync) ->
311                             window = 0;
312                             s = M;
313                             do
314                             :: (s > 0) ->
315                                     s = s-1;
316                                     busy[s] = false
317                             :: (s == 0) ->
318                                     break
319                             od
320                     fi
321             }
322     :: atomic {
323             (window > 0 && busy[q] == false) ->
324             window = window - 1;
325             q = (q+1)%M
326        }
327 #if DUPS
328     :: atomic {
329             (len(flow_to_dll[n]) < QSZ
330              && window > 0 && busy[q] == true) ->
331             flow_to_dll[n]! O_buf[q],q
332        }
333 #endif
334     :: atomic {
335             (timeout && len(flow_to_dll[n]) < QSZ
336              && window > 0 && busy[q] == true) ->
337             flow_to_dll[n]! O_buf[q],q
338        }
339
340     /* receiver part */
341 #if LOSS
342     :: dll_to_flow[n]?type,m /* lose any message */
343 #endif
344     :: dll_to_flow[n]?type,m ->
345             if
346             :: atomic {
347                     (type == ack) ->
348                     busy[m] = false
349                }
350             :: atomic {
351                     (type == sync) ->
352                     flow_to_dll[n]!sync_ack,m;
353                     m = 0;
354                     do
355                     :: (m < M) ->
```

```
356                             received[m] = 0;
357                             m = m+1
358                     :: (m == M) ->
359                             break
360                     od
361                }
362             :: (type == sync_ack) ->
363                     flow_to_ses[n]!sync_ack,m
364             :: (type != ack && type != sync && type != sync_ack)->
365                     if
366                     :: atomic {
367                             (received[m] == true) ->
368                                     x = ((0<p-m   && p-m<=W)
369                                     ||   (0<p-m+M && p-m+M<=W)) };
370                                     if
371                                     :: (x) -> flow_to_dll[n]!ack,m
372                                     :: (!x) /* else skip */
373                                     fi
374                     :: atomic {
375                             (received[m] == false) ->
376                                     I_buf[m] = type;
377                                     received[m] = true;
378                                     received[(m-W+M)%M] = false
379                      }
380                     fi
381             fi
382     :: (received[p] == true && len(flow_to_ses[n])<QSZ
383                             && len(flow_to_dll[n])<QSZ) ->
384             flow_to_ses[n]!I_buf[p],0;
385             flow_to_dll[n]!ack,p;
386             p = (p+1)%M
387     od
388 }
389
390 /*
391  * Datalink Layer Validation Model
392  */
393
394 proctype data_link()
395 {   byte type, seq;
396
397 end:    do
398     :: flow_to_dll[0]?type,seq ->
399             if
400             :: dll_to_flow[1]!type,seq
401             :: skip /* lose message */
402             fi
403     :: flow_to_dll[1]?type,seq ->
404             if
405             :: dll_to_flow[0]!type,seq
406             :: skip /* lose message */
407             fi
408     od
409 }
```