

balancer_no_balance). If the loads are not balanced, the load balancer takes a pending request of the most loaded server and redirects it to the least loaded server (transition balancer_balance). The load balancer has to maintain for each server the number of requests sent to this server.

Listing 1: Helena file of the load balancing system (file examples/load_balancer.lna)

```

1  /*****
2  *
3  *  Example file of the Helena distribution
4  *
5  *  File   : load_balancer.lna
6  *  Author: Sami Evangelista
7  *  Date  : 27 oct. 2004
8  *
9  *  This file contains the description of a load balancing system.
10 *
11 *****/
12
13
14 load_balancer (C := 6,    /* number of clients */
15               S := 2) { /* number of servers */
16
17     /* clients */
18     type client_id : range 1 .. C;
19     type clients_no : range 0 .. client_id'last;
20
21     /* servers */
22     type server_id : range 1 .. S;
23
24     /* load */
25     type servers_load : vector [server_id] of clients_no;
26     constant servers_load empty_load := [0];
27
28
29     /* return the least loaded server */
30     function least (servers_load load) -> server_id {
31         server_id result := server_id'first;
32         for(i in server_id)
33             if(load[i] < load[result])
34                 result := i;
35         return result;
36     }
37
38     /* return the most loaded server */
39     function most (servers_load load) -> server_id {
40         server_id result := server_id'first;
41         for(i in server_id)
42             if(load[i] > load[result])
43                 result := i;
44         return result;
45     }
46
47     /* check if load is balanced */
48     function is_balanced (servers_load load) -> bool {
49         clients_no max_no := 0;
50         clients_no min_no := clients_no'last;
51         for(i in server_id)
52         {
53             if(load[i] > max_no) max_no := load[i];
54             if(load[i] < min_no) min_no := load[i];
55         }
56         return (max_no - min_no) <= 1;

```

```

57 }
58
59 /* increment the load of server i */
60 function incr (servers_load l, server_id i) -> servers_load
61   return l :: ([i] := l[i] + 1);
62
63 /* decrement the load of server i */
64 function decr (servers_load l, server_id i) -> servers_load
65   return l :: ([i] := l[i] - 1);
66
67 /* return the difference between the two loads */
68 function diff (clients_no c1, clients_no c2) -> clients_no
69   return (c1 > c2) ? (c1 - c2) : (c2 - c1);
70
71
72 /*
73  * clients
74  */
75 place client_idle {
76   dom : client_id;
77   init : for(c in client_id) <( c )>;
78   capacity : 1;
79 }
80 place client_waiting {
81   dom : client_id;
82   capacity : 1;
83 }
84 place client_request {
85   dom : client_id;
86   capacity : 1;
87 }
88 place client_ack {
89   dom : client_id;
90   capacity : 1;
91 }
92 transition client_send {
93   in { client_idle : <( c )>; }
94   out { client_waiting : <( c )>;
95         client_request : <( c )>; }
96   description : "client_%d:_send_request", c;
97 }
98 transition client_receive {
99   in { client_waiting : <( c )>;
100       client_ack : <( c )>; }
101   out { client_idle : <( c )>; }
102   description : "client_%d:_receives_response", c;
103 }
104
105
106 /*
107  * servers
108  */
109 place server_idle {
110   dom : server_id;
111   init : for(s in server_id) <( s )>;
112   capacity : 1;
113 }
114 place server_waiting {
115   dom : server_id * client_id;
116   capacity : 1;
117 }
118 place server_processing {

```

```

119     dom : server_id * client_id;
120     capacity : 1;
121 }
122 place server_notification {
123     dom : server_id;
124     capacity : 1;
125 }
126 place server_notification_ack {
127     dom : server_id;
128     capacity : 1;
129 }
130 place server_request {
131     dom : client_id * server_id;
132     capacity : 1;
133 }
134 transition server_notify {
135     in { server_idle      : <( s )>;
136         server_request    : <( c, s )>; }
137     out { server_waiting  : <( s, c )>;
138         server_notification : <( s )>; }
139     description: "server_%d:_lb_process_notification", s;
140 }
141 transition server_receive {
142     in { server_waiting      : <( s, c )>;
143         server_notification_ack : <( s )>; }
144     out { server_processing  : <( s, c )>; }
145     description: "server_%d:_reception_of_request_from_client_%d", s, c;
146 }
147 transition server_send {
148     in { server_processing : <( s, c )>; }
149     out { server_idle      : <( s )>;
150         client_ack        : <( c )>; }
151     description: "server_%d:_send_response_to_client_%d", s, c;
152 }
153
154
155 /*
156  * load balancer process
157  */
158 place balancer_idle {
159     dom : servers_load;
160     init : <( empty_load )>;
161     capacity : 1;
162 }
163 place balancer_routing {
164     dom : servers_load * client_id;
165     capacity : 1;
166 }
167 place balancer_balancing {
168     dom : servers_load;
169     capacity : 1;
170 }
171 transition balancer_receive_client {
172     in { balancer_idle      : <( l )>;
173         client_request    : <( c )>; }
174     out { balancer_routing : <( l, c )>; }
175     description: "lb:_receive_request_of_client_%d", c;
176 }
177 transition balancer_route {
178     in { balancer_routing : <( l, c )>; }
179     out { balancer_idle   : <( incr(l, ll) )>;
180         server_request    : <( c, ll )>; }

```

```

181     let { server_id l1 := least(1); }
182     description: "lb:_route_request_of_client_%d_to_server_%d", c, l1;
183 }
184 transition balancer_receive_notification {
185     in { balancer_idle      : <( 1 )>;
186         server_notification : <( s )>; }
187     out { server_notification_ack : <( s )>;
188         balancer_balancing      : <( decr(1, s) )>; }
189     description: "lb:_receive_notification_of_server_%d", s;
190 }
191 transition balancer_balance {
192     in { balancer_balancing : <( 1 )>;
193         server_request      : <( c, most(1) )>; }
194     out { balancer_idle      : <( decr(incr(1, l1), ml) )>;
195          server_request      : <( c, l1 )>; }
196     let { server_id l1 := least(1);
197          server_id ml := most(1); }
198     guard: not is_balanced(1);
199     description: "lb:_redirect_request_of_client_%d_from_server_%d\
200 to_server_%d", c, ml, l1;
201 }
202 transition balancer_no_balance {
203     in { balancer_balancing : <( 1 )>; }
204     out { balancer_idle      : <( 1 )>; }
205     guard: is_balanced(1);
206     description: "lb:_no_rebalance";
207 }
208
209
210 /*
211  * state propositions
212  *
213  * load_not_balanced: for each couple of servers (s1,s2) with s1 != s2,
214  * the difference between the number of requests pending or accepted by
215  * s1 and the number of requests pending or accepted by s2 is at most 1.
216  */
217 proposition load_not_balanced:
218     not forall (s1 in server_id, s2 in server_id | s1 != s2 :
219         diff (card (sr in server_request | sr->2 = s1) +
220             card (sn in server_notification | sn->1 = s1),
221             card (sr in server_request | sr->2 = s2) +
222             card (sn in server_notification | sn->1 = s2)) <= 1);
223 proposition balancing:
224     balancer_balancing' card = 1;
225 }

```

Listing 2: Helena file of the load balancing system properties (file examples/load_balancer.prop.lna)

```

1  /*
2  * reject any deadlock state
3  */
4  state property not_dead:
5      reject deadlock;
6
7  /*
8  * the loads are balanced or are being rebalanced
9  */
10 state property balance_ok:
11     reject load_not_balanced;
12     accept balancing;

```