Helena 3.0
Example 2 - The load balancing system

Sami Evangelista - (Sami [dot] Evangelista [at] lipn.univ-paris13 [dot] fr)

November 29, 2017

We propose to specify and verify a simple load balancing system with Helena. The full net is illustrated by Figure 1. Initial markings and transition guards have been omitted to clarify the figure.

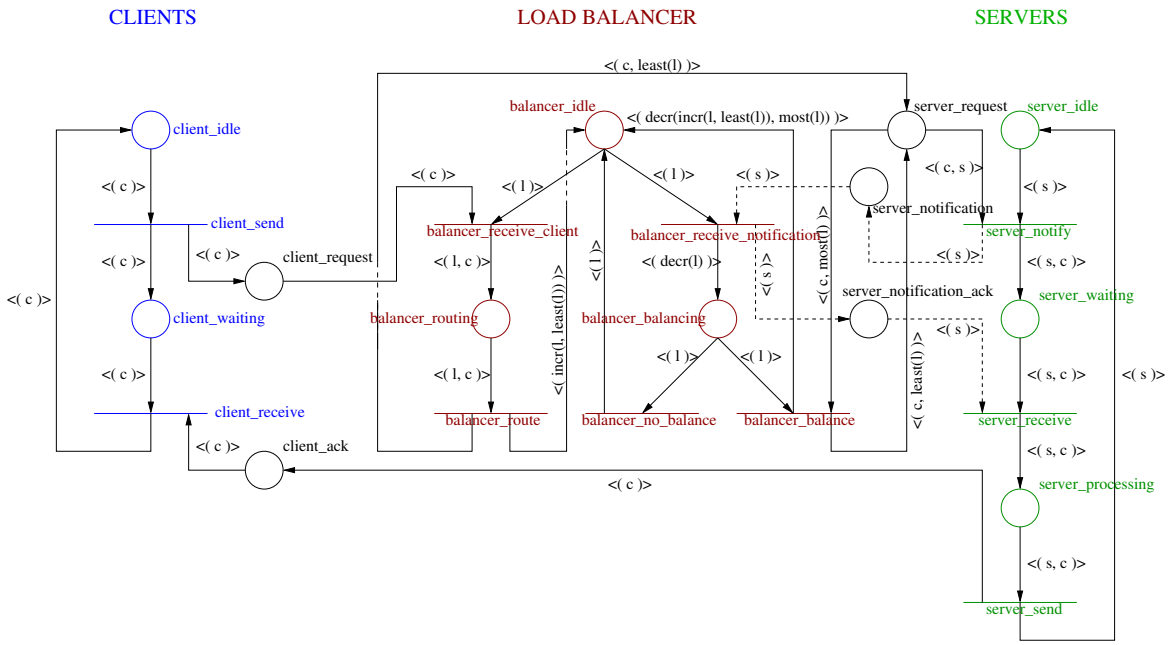

Figure 1: The whole load balancing system

In this system, we have two kinds of process: a set of clients and a set of servers. An additional process called the load balancer distribute requests of clients to servers. Its task is also to redistribute pending requests when servers accept requests in order to maintain the loads of servers balanced.

The clients We note C the number of clients considered. Clients are numbered from 1 to C. The behavior of the clients is quite simple. A client may want to send a request to a set of servers. Instead of asking a server directly, he sends the request to the load balancer which will route the request to the adequate server, i.e., the least loaded server. Once the request sent, the client waits for the answer. When this one arrives, the client comes back to the idle state.

The servers The number of servers is noted S. Servers are numbered from 1 to S. Servers receive requests from clients via the load balancer process. When a server accepts a request, he first has to notify this to the load balancer process, in order that this one rebalances the pending requests. Then he has to wait for an acknowledgment from the load balancer to start treating the request. Once the request treated, he directly sends the answer to the concerned client and goes back to the idle state.

The load balancer The load balancer can perform two kinds of task. The first one is to redirect each client request to the least loaded server. Secondly, when a server accepts a request from a client the load balancer has to rebalance the pending requests. If these are already balanced, the load balancer has nothing to perform and can come back to its idle state (transition

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`)

```
/*****************************************************************************
*
*  Example file of the Helena distribution
*
*  File  : load_balancer.lna
*  Author: Sami Evangelista
*  Date  : 27 oct. 2004
*
*  This file contains the description of a load balancing system.
*
*****************************************************************************/


load_balancer (C := 6,    /*  number of clients  */
               S := 2) {  /*  number of servers  */

   /*  clients  */
   type client_id  : range 1 .. C;
   type clients_no : range 0 .. client_id'last;

   /*  servers  */
   type server_id : range 1 .. S;

   /*  load  */
   type servers_load : vector [server_id] of clients_no;
   constant servers_load empty_load := [0];


   /*  return the least loaded server  */
   function least (servers_load load) -> server_id {
      server_id result := server_id'first;
      for(i in server_id)
         if(load[i] < load[result])
            result := i;
      return result;
   }

   /*  return the most loaded server  */
   function most (servers_load load) -> server_id {
      server_id result := server_id'first;
      for(i in server_id)
         if(load[i] > load[result])
            result := i;
      return result;
   }

   /*  check if load is balanced  */
   function is_balanced (servers_load load) -> bool {
      clients_no max_no := 0;
      clients_no min_no := clients_no'last;
      for(i in server_id)
      {
         if(load[i] > max_no) max_no := load[i];
         if(load[i] < min_no) min_no := load[i];
      }
      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 )>; }
```

```
      let { server_id ll := least(l); }
      description: "lb: route request of client %d to server %d", c, ll;
   }
   transition balancer_receive_notification {
      in  { balancer_idle           : <( 1 )>;
            server_notification     : <( s )>; }
      out { server_notification_ack : <( s )>;
            balancer_balancing      : <( decr(l, s) )>; }
      description: "lb: receive notification of server %d", s;
   }
   transition balancer_balance {
      in  { balancer_balancing : <( l )>;
            server_request     : <( c, most(l) )>; }
      out { balancer_idle      : <( decr(incr(l, ll), ml) )>;
            server_request     : <( c, ll )>; }
      let { server_id ll := least(l);
            server_id ml := most(l); }
      guard: not is_balanced(l);
      description: "lb: redirect request of client %d from server %d \
to server %d", c, ml, ll;
   }
   transition balancer_no_balance {
      in  { balancer_balancing : <( l )>; }
      out { balancer_idle      : <( l )>; }
      guard: is_balanced(l);
      description: "lb: no rebalance";
   }


   /*
    *  state propositions
    *
    *  load_not_balanced: for each couple of servers (s1,s2) with s1 != s2,
    *  the difference between the number of requests pending or accepted by
    *  s1 and the number of requests pending or accepted by s2 is at most 1.
    */
   proposition load_not_balanced:
      not forall (s1 in server_id, s2 in server_id | s1 != s2 :
            diff (card (sr in server_request | sr->2 = s1) +
                  card (sn in server_notification | sn->1 = s1),
                  card (sr in server_request | sr->2 = s2) +
                  card (sn in server_notification | sn->1 = s2)) <= 1);
   proposition balancing:
      balancer_balancing'card = 1;
}
```

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

```
/*
 *  reject any deadlock state
 */
state property not_dead:
   reject deadlock;

/*
 *  the loads are balanced or are being rebalanced
 */
state property balance_ok:
   reject load_not_balanced;
   accept balancing;
```