**IRC lite**

The IRC-lite system is composed of:
- 3 client nodes running on different machines and
- a single server node on another machine.

Such components perform the following functions:
- the chat clients send/receive messages to/from the group control;
- the group controller manages a single chat group
  - a message sent to the controller is broadcast to all the group members
- the chat server tracks the group controllers and manages the joining operation;
- the middle-men take care of the transport of data (they hide the sockets).

**IRC lite**

- module(chat_client).
- export([start/1, connect/5]).

```erlang
start(Nick) ->
    connect("localhost", 2223, "AsDT67aQ", "general", Nick).
connect(Host, Port, HostPsw, Group, Nick) ->
    spawn(fun() ->
        handler(Host, Port, HostPsw, Group, Nick)
    end).

handler(Host, Port, HostPsw, Group, Nick) ->
    process_flag(trap_exit, true),
    start_connector(Host, Port, HostPsw),
    disconnected(Group, Nick).

disconnected(Group, Nick) ->
    receive
        {connected, MM} ->
            io:format("connected to server
                        sending data\n"),
            lib_chan_mm:send(MM, {login, Group, Nick}),
            wait_login_response(MM);
        {status, S} ->
            io:format("~p\n", [S]),
            disconnected(Group, Nick);
        Other ->
            io:format("chat_client disconnected unexpected:-p-n", [Other]),
            disconnected(Group, Nick)
    end.
```

---

**References**

Erlang in Action
Walter Cazzola
Dipartimento di Informatica
Università degli Studi di Milano
e-mail: cazzola@di.unimi.it
The Server Implementation: The Chat Controller.

```
-module(chat_controller).
:start() -> start_server(), lib:chan:start_server("chat.conf").
:start_server() ->
  register(chat_server,
    spawn(fun() ->
      process_flag(trap_exit, true),
      Val = (catch server_loop([])),
      io:format("Server terminated with:~p~n", [Val])
    end)),
  server_loop([])
  receive
    {ok, MM} ->
      io:format("Server login from MM:~p~n", [MM]),
      Server = server_login(MM),
      Server
    | Error ->
      io:format("server error:~p~n", [Error])
  end).
server_loop(L) ->
  receive
    {ok, MM, Ack} ->
      active(MM),
      Server = server_login(MM),
      Server
    | Error ->
      io:format("server error:~p~n", [Error])
  end.
server_login(MM) ->
  receive
    server_loop(L);
    other ->
      remove_group(MM, L)
  end.
```

Active
- sends messages to the group and vice versa and
- monitors the connection with the group
IRC lite
The Server Implementation: The Group Manager.

```erlang
-module(chat_group).
-export([start/2]).

start(C, Nick) ->
  process_flag(trap_exit, true),
  lib_chan_mm:controller(C, self()),
  lib_chan_mm:send(Pid, ack),
  self() ! chan(C, relay(Nick, "I'm starting the group"),
     group_controller([C,Nick])).

delete(Pid, [Pid,Nick]|T) ->
  (Nick, lists:reverse(T, L)),
  delete(Pid, T, [L|]);
delete(_, [], L) ->
  ("????", L).

group_controller([]) ->
  exit(allGone);

receive
  (chan, C, relay(Nick, Str)) ->
    lists:foreach(fun(Pid,_) ->
                     lib_chan_mm:controller(C, self()),
                     lib_chan_mm:send(Pid, {msg,Nick,C,Str})
                   end, L),
    group_controller(L);
  (login, C, Nick) ->
    lib_chan_mm:controller(C, self()),
    lib_chan_mm:send(C, ack),
    self() ! chan(C, relay(Nick, "I'm joining the group"),
      group_controller([C,Nick]));
  (chan_closed, C, Nick) ->
    delete(C, L, []),
    self() ! chan(C, relay(Nick, "I'm leaving the group"),
      group_controller(L));
  Any ->
    io:format('group controller received Msg=p-n, [Any]),
    group_controller(L).
  end.
```

References

- Gul Agha.
  Actors: A Model of Concurrent Computation in Distributed Systems.

- Joe Armstrong.

- Francesco Cesarini and Simon Thompson.
  O'Reilly, June 2009.