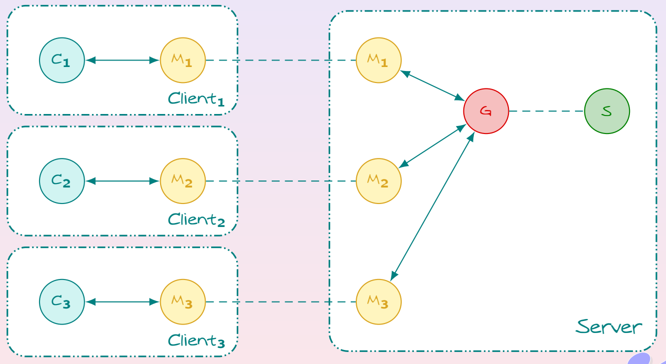
Erlang in Action
IRC lite

Walter Cazzola

Dipartimento di Informatica
Università degli Studi di Milano
e-mail: cazzola@di.unimi.it
twitter: @w_cazzola

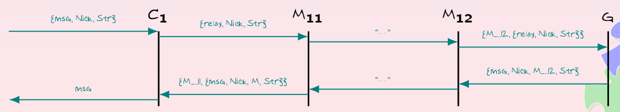
IRC lite
The Architecture



IRC lite
The Architecture (Cont'd)

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; and
- the middle-men take care of the transport of data (they hide the sockets).



IRC lite
The Client Implementation

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

start(Nick) -> connect("localhost", 2223, "AsDT67aQ", "general", Nick).
```

```erlang
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).
```

- it makes itself into a system process;
- it then spawns a connection process (which tries to connect to the server);
- it waits for a connection event in disconnected.

```erlang
disconnected(Group, Nick) ->
  receive
    {connected, MM} ->                    % from the connection process
       io:format("connected to server\nsending 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.
```

IRC lite
The Client Implementation (Cont'd).

```
start_connector(Host, Port, Pwd) ->
   S = self(), spawn_link(fun() -> try_to_connect(S, Host, Port, Pwd) end).
```

Note that

```
S=self(), spawn_link(fun() -> try_to_connect(S, ...) end)
```

is different than

```
spawn_link(fun() -> try_to_connect(self(), ...) end)
```

```
try_to_connect(Parent, Host, Port, Pwd) ->
  %% Parent is the Pid of the process that spawned this process
  case lib_chan:connect(Host, Port, chat, Pwd, []) of
    {error, _Why} ->
       Parent ! {status, {cannot, connect, Host, Port}},
       sleep(2000),
       try_to_connect(Parent, Host, Port, Pwd);
    {ok, MM} ->
       lib_chan_mm:controller(MM, Parent),
       Parent ! {connected, MM}, %% to disconnected
       exit(connectorFinished)
  end.
sleep(T) -> receive after T -> true end.
```

IRC lite
The Client Implementation (Cont'd).

```
wait_login_response(MM) ->
  receive
    {chan, MM, ack} -> active(MM);
    {'EXIT', _Pid, connectorFinished} -> wait_login_response(MM);
    Other ->
       io:format("chat_client login unexpected:~p~n",[Other]),
       wait_login_response(MM)
  end.
```

```
active(MM) ->
  receive
    {msg, Nick, Str} ->
        lib_chan_mm:send(MM, {relay, Nick, Str}),
        active(MM);
    {chan, MM, {msg, From, Pid, Str}} ->
        io:format("~p@~p: ~p~n", [From,Pid,Str]),
        active(MM);
    {close, MM} -> exit(serverDied);
    Other ->
        io:format("chat_client active unexpected:~p~n",[Other]),
        active(MM)
   end.
```

active

- sends messages to the group and vice versa and
- monitors the connection with the group

IRC lite
The Server Implementation: The Chat Controller.

```
{port, 2223}.
{service, chat, password,"AsDT67aQ",mfa,chat_controller,start,[]}.
```

- it uses lib_chan.

```
-module(chat_controller).
-export([start/3]).
-import(lib_chan_mm, [send/2]).

start(MM, _, _) ->
  process_flag(trap_exit, true),
  io:format("chat_controller off we go ...~p~n",[MM]),
  loop(MM).

loop(MM) ->
  receive
    {chan, MM, Msg} ->                                   %% when a client connects
        chat_server ! {mm, MM, Msg},
        loop(MM);
    {'EXIT', MM, _Why} ->                           %% when the session terminates
        chat_server ! {mm_closed, MM};
    Other ->
        io:format("chat_controller unexpected message =~p (MM=~p)~n", [Other, MM]),
        loop(MM)
  end.
```

IRC lite
The Server Implementation: The Chat Server.

```
-module(chat_server).

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(L) ->
  receive
    {mm, Channel, {login, Group, Nick}} ->
       case lookup(Group, L) of
         {ok, Pid} -> Pid ! {login, Channel, Nick}, server_loop(L);
         error ->
            Pid = spawn_link(fun() -> chat_group:start(Channel, Nick) end),
            server_loop([{Group,Pid}|L])
       end;
    {mm_closed, _} -> server_loop(L);
    {'EXIT', Pid, allGone} -> L1 = remove_group(Pid, L), server_loop(L1);
    Msg -> io:format("Server received Msg=~p~n", [Msg]), server_loop(L)
  end.

lookup(G, [{G,Pid}|_]) -> {ok, Pid};
lookup(G, [_|T])       -> lookup(G, T);
lookup(_,[])           -> error.

remove_group(Pid, [{G,Pid}|T]) -> io:format("~p removed~n",[G]), T;
remove_group(Pid, [H|T])       -> [H|remove_group(Pid, T)];
remove_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(C, ack),
  self() ! {chan, C, {relay, Nick, "I'm starting the group"}},
  group_controller([{C,Nick}]).

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

group_controller([]) -> exit(allGone);
group_controller(L) ->
  receive
    {chan, C, {relay, Nick, Str}} ->
      lists:foreach(fun({Pid,_}) -> 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}|L]);
    {chan_closed, C} ->
      {Nick, L1} = delete(C, L, []),
      self() ! {chan, C, {relay, Nick, "I'm leaving the group"}},
      group_controller(L1);
    Any ->
      io:format("group controller received Msg=~p~n", [Any]),
      group_controller(L)
  end.
```

IRC lite
Chatting around ...

```
1> chat_server:start().
lib_chan starting:"chat.conf"
ConfigData=[{port,2223}, {service,chat,password,"AsDT67aQ",mfa,chat_controller,start,[]}]
chat_controller off we go ...<0.39.0>
chat_controller off we go ...<0.41.0>
chat_controller off we go ...<0.43.0>
server error should die with exit(normal) was:{mm_closed,<0.39.0>}
chat_controller off we go ...<0.46.0>
server error should die with exit(normal) was:mm_closed,<0.46.0>}
server error should die with exit(normal) was:mm_closed,<0.41.0>}
server error should die with exit(normal) was:mm_closed,<0.43.0>}
```

```
1> ChatDaemon = chat_client:start(walter).
walter@<0.41.0>: "I'm joining the group"
'walter cazzola'@<0.43.0>: "I'm joining the group"
2> ChatDaemon ! {msg, walter, "Hello World!!!"}.
{msg,walter,"Hello World!!!"}
walter@<0.41.0>: "Hello World!!!"
'walter cazzola'@<0.43.0>: "Hello Walter!!!"
cazzola@<0.39.0>: "Hello Walter!!!"
cazzola@<0.39.0>: "I'm leaving the group"
cazzola@<0.46.0>: "I'm joining the group"
cazzola@<0.46.0>: "I'm leaving the group"
```

```
1> ChatDaemon = chat_client:start('walter cazzola').
'walter cazzola'@<0.43.0>: "I'm joining the group"
walter@<0.41.0>: "Hello World!!!"
2> ChatDaemon!{msg,'walter cazzola',"Hello Walter!!!"}.
{msg,'walter cazzola',"Hello Walter!!!"}
'walter cazzola'@<0.43.0>: "Hello Walter!!!"
cazzola@<0.39.0>: "Hello Walter!!!"
cazzola@<0.39.0>: "I'm leaving the group"
cazzola@<0.46.0>: "I'm joining the group"
cazzola@<0.46.0>: "I'm leaving the group"
walter@<0.41.0>: "I'm leaving the group"
```

```
1> ChatDaemon = chat_client:start(cazzola).
cazzola@<0.39.0>: "I'm starting the group"
walter@<0.41.0>: "I'm joining the group"
'walter cazzola'@<0.43.0>: "I'm joining the group"
walter@<0.41.0>: "Hello World!!!"
'walter cazzola'@<0.43.0>: "Hello Walter!!!"
2> ChatDaemon ! {msg, cazzola, "Hello Walter!!!"}.
{msg,cazzola,"Hello Walter!!!"}
cazzola@<0.39.0>: "Hello Walter!!!"
3> ^C [21:35]cazzola@surtur:~/lp/erlang/chat>erl
1> ChatDaemon = chat_client:start(cazzola).
cazzola@<0.46.0>: "I'm joining the group"
```

References

- Gul Agha.
  Actors: A Model of Concurrent Computation in Distributed Systems.
  MIT Press, Cambridge, 1986.
- Joe Armstrong.
  Programming Erlang: Software for a Concurrent World.
  The Pragmatic Bookshelf, fifth edition, 2007.
- Francesco Cesarini and Simon Thompson.
  Erlang Programming: A Concurrent Approach to Software Development.
  O'Reilly, June 2009.