eXAT: an Experimental Tool for Programming Multi-Agent Systems in Erlang

Antonella Di Stefano, Corrado Santoro (presenting)
Overview

- Motivations
- Erlang Language
- eXAT: Tool & Case-Studies
- Conclusions
Agent Programming

Two Main Aspects

Agent Behaviour

FSM-Based

(State, Event) → (State’, Action)

Agent Intelligence

Production Systems

e.g. CLIPS, JESS

Di Stefano, Santoro, “eXAT: an Experimental Tool for Programming Multi-Agent …”, WOA 2003, Villasimius, Italy
Motivations (II)

Agent Behaviour
Agent Platform
Agent Intelligence
Production Systems

JADE
JESS, CLIPS

Imperative Programming
Declarative Programming

LACK OF IMPLEMENTATION OMOGENEITY

PERFORMANCE PROBLEMS (?)

Di Stefano, Santoro, “eXAT: an Experimental Tool for Programming Multi-Agent …”, WOA 2003, Villasimius, Italy
Proposal: to employ a programming language able to deal with both of the aspects above

The characteristics of Erlang (functional, matching capabilities, multi-process model, etc.) seem well suited to support both agent behaviour and intelligence.

Di Stefano, Santoro, “eXAT: an Experimental Tool for Programming Multi-Agent …”, WOA 2003, Villasimius, Italy
Erlang (I)

Functional language developed at Ericsson (designed initially in 1988)

- **Functional Notation**: functions with “guards”
- **Portability**: bytecode + runtime
- **Good Performances**: optimization, fast execution, real-time style garbage collector, etc.
- **Concurrency**: concurrent communicating processes (CSP, π-calculus)
- **Fault-Tolerant**: native mechanisms to design replication-based fault-tolerant systems
- **Completeness**: a powerful library (e.g., HTTP handling, socket communication, HTML/XML parser, GUI, CORBA, etc.)

http://www.erlang.org

Di Stefano, Santoro, “eXAT: an Experimental Tool for Programming Multi-Agent ...”, WOA 2003, Villasimius, Italy
### Erlang (II)

<table>
<thead>
<tr>
<th>Function call</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fact (0) -&gt; 1;</td>
<td>fact (N) when N == 0 -&gt; 1;</td>
</tr>
<tr>
<td>fact (N) -&gt; N * fact (N-1).</td>
<td>fact (N) -&gt; N * fact (N-1).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function call</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>foo (double, N) -&gt; N*2;</td>
<td>foo (double, 10) 20</td>
</tr>
<tr>
<td>foo (triple, N) -&gt; N*3;</td>
<td>foo (triple, 10) 30</td>
</tr>
<tr>
<td>foo (_, N) -&gt; N.</td>
<td>foo (quad, 10) 10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Expression</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sum ([10,24,53,32])</td>
<td>sum ([]) -&gt; 0;</td>
</tr>
<tr>
<td></td>
<td>sum ([H</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>List</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[H1, H2, H3</td>
<td>T] = “Erlang Language”.</td>
</tr>
<tr>
<td>H1 = 69, H2 = 114, H3 = 108, T = “ang Language”</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>List</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>{SA, S, R, C} = {inform, buyer, seller, ‘(buy)’}.</td>
<td></td>
</tr>
<tr>
<td>SA = inform, S = buyer, R = seller, C = ‘(buy)’</td>
<td></td>
</tr>
</tbody>
</table>

---

*Di Stefano, Santoro, “eXAT: an Experimental Tool for Programming Multi-Agent ...”, WOA 2003, Villasimius, Italy*
do_call (ServerName, Service, Param) ->
    ServerName ! { self (), Service, Param },
    receive
        RetValue -> RetValue
    end.

proc1 (P) -> ...
proc2 (P) -> ...
server () ->
    receive
        {From, service1, P} ->
            From ! proc1 (P), server ();
        {From, service2, P} ->
            From ! proc2 (P), server ();
    ...
end.
Di Stefano, Santoro, “eXAT: an Experimental Tool for Programming Multi-Agent …”, WOA 2003, Villasimius, Italy
∀X, \text{is} \text{_} \text{a} \text{ (human, X)} \Rightarrow \text{is} \text{_} \text{a} \text{ (animal, X)} \land \text{legs} \text{ (X, 2)}.

-module (test_module).
-export ([rule/2, start/0]).

rule (Engine, \{is\_a, human, X\}) ->
  eres:assert (Engine, \{is\_a, animal, X\}),
  eres:assert (Engine, \{legs, X, 2\}),
  true.

start () ->
  eres:new (test, \{test\_module, rule\}),
  eres:assert (test, \{is\_a, human, corrado\}).

[ \{is\_a, human, corrado\},
  \{is\_a, animal, corrado\}, \{legs, corrado, 2\} ]
An ERES engine without rules is a *Linda tuple space*

\[
\text{eres:assert} \begin{array}{ll}
\text{(Engine, Fact)} & \text{out (Space, Tuple)} \\
\text{eres:retract} \begin{array}{ll}
\text{(Engine, FactTemplate)} & \text{in (Space, TupleTemplate)}
\end{array}
\end{array}
\]

A Linda-based (mutually-exclusive) queue:

\[
\text{enqueue} (Q, X) \rightarrow \\
\begin{array}{ll}
\text{\{tail, PTR\} = eres:in} & \begin{array}{ll}
\text{(Q, \{tail, nil\}),} \\
\text{eres:out} & \begin{array}{ll}
\text{(Q, \{tail, PTR + 1\}), eres:out} & \begin{array}{ll}
\text{(Q, \{data, PTR, X\})},} \\
& \text{true.}
\end{array}
\end{array}
\end{array}
\end{array}
\]

\[
\text{dequeue} (Q) \rightarrow \\
\begin{array}{ll}
\text{\{head, PTR\} = eres:in} & \begin{array}{ll}
\text{(Q, \{head, nil\}),} \\
\text{eres:out} & \begin{array}{ll}
\text{(Q, \{head, PTR + 1\}),} \\
\text{\{data, P, X\} = eres:in} & \begin{array}{ll}
\text{(Q, \{data, PTR, nil\})},} \\
& X.
\end{array}
\end{array}
\end{array}
\end{array}
\]

Used in eXAT to store incoming ACL messages

*Di Stefano, Santoro, “eXAT: an Experimental Tool for Programming Multi-Agent …”, WOA 2003, Villasimius, Italy*
• ACL message composition
• ACL parsing / SL0 parsing (generated by the “yecc” utility)
• Message Transport (erlang-native MTP)
• Specification of Ontologies… is a future work

acl:spectact_name (Sender, Receiver, Ontology, Language, Content, Other-Params)

acl:inform (buyer, seller, 'e-commerce', sl0, "(purchase computer)", []).

acl:reply (Request, SpeechAct_Name, Content)

<<list of tuples {name,value}>> = acl:sl0_parsecontent (Content)
• **Template Registration**: events to trap (eres, acl, silent) for each FSM state, name of callback function

• **Action Registration**: name of the FSM-template, name of callback function

Action Callback:

```
action-function ( Event, State, Agent )
```
-module (agentbob).
-export ([action/3, start/0]).

action ([inform, alice, Receiver, Ontology, lisp,
         Content, Slots], State, Agent) ->
    % 'inform' from Alice ... do the action

start () ->
    agent:new (bob, receiver, {agentbob, action}).

public class ReceiverAgent extends Agent {
    public void setup () {
        AID aid[] = new AID[1];
        aid [0] = new AID ("Alice", false);
        addBehaviour (new ReceiverBehaviour (this, -1, MessageTemplate.and (MessageTemplate.MatchPerformative (ACLMessage.INFORM), MessageTemplate.and ( MessageTemplate.MatchReceiver (aid), MessageTemplate.MatchLanguage ("lisp")) )) {
            public void action () {
                //.. do something
            }
        });
    }
}
-module (agentbob).
-export ([action/3, start/0]).

action ([inform, alice, Receiver, Ontology, lisp, Content, Slots], State, Agent) ->
    % 'inform' from alice: ... do the action

action ([inform, Sender, Receiver, Ontology, lisp, Content, Slots], State, Agent) ->
    % 'inform' from another agent ... do the action

action (M, State, Agent) ->
    acl:reply (M, notunderstood, acl:getcontent (M)).

start () ->
    agent:new (bob, receiver, {agentbob, action}).
\textbf{Example: User-Defined FSM-Template}

\begin{verbatim}
-module (myfsm).
-export ([template/4, events/2]).

template (Fact, init, Agent, AgentFun) ->
  apply ( AgentFun, [Fact, init, Agent] ),
  s1;

template ([inform | T] = M, s1, Agent, AgentFun) ->
  apply ( AgentFun, [M, s1, Agent] ),
  stop.

events (Agent, AgentFun) ->
  [ {init,
      [ {eres, goods, \{price, computer, fun (X) -> X < 1000 end \} ] },
    {s1, [ {acl} ] } } ].
\end{verbatim}

\textit{Di Stefano, Santoro, “eXAT: an Experimental Tool for Programming Multi-Agent …”, WOA 2003, Villasimius, Italy}
**Performances:** We said nothing!

- measures will be done when a standard MTP will be available

**Functional Programming:** Not so easy as imperative

- a training phase is required for a “traditional” C++ / Java programmer

**“Real” Agents:** We did only some simple tests

- To validate the eXAT approach, some real-case implementations are needed
Thank You