On the use of Erlang as a Promising Language to Develop Agent Systems

Antonella Di Stefano, Corrado Santoro

University of Catania, Italy

*AI*IA TABOO WOA 2004 – Torino, 30 Novembre 2004*

Overview

- Motivations
- Our Agent Model
- Agent Language Requirements
- Erlang for Agents
- Conclusions

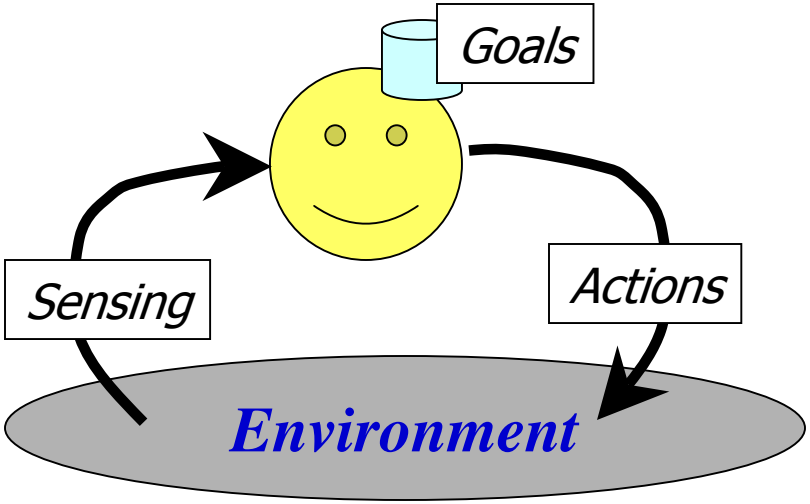
Motivations

- About 70% of agent platforms are written in Java
- Many of them need to integrate other (ad-hoc) languages or tools to implement the "intelligence" (BDI model, rule pruduction system, etc.)
- So, is Java really suited for implementing agents?
- Are there other languages that fit the "agent model" (better than Java)?
- Java is a very good general-purpose language, but not a "silver bullet"

Back to Agent Definition

"An autonomous agent is a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future."

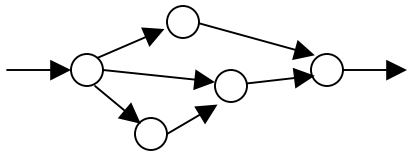
Franklin and Graesser, "Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents", ATAL-96



An agent ***senses*** the environment and ***acts*** onto it on the basis of the inputs and its internal ***state***.

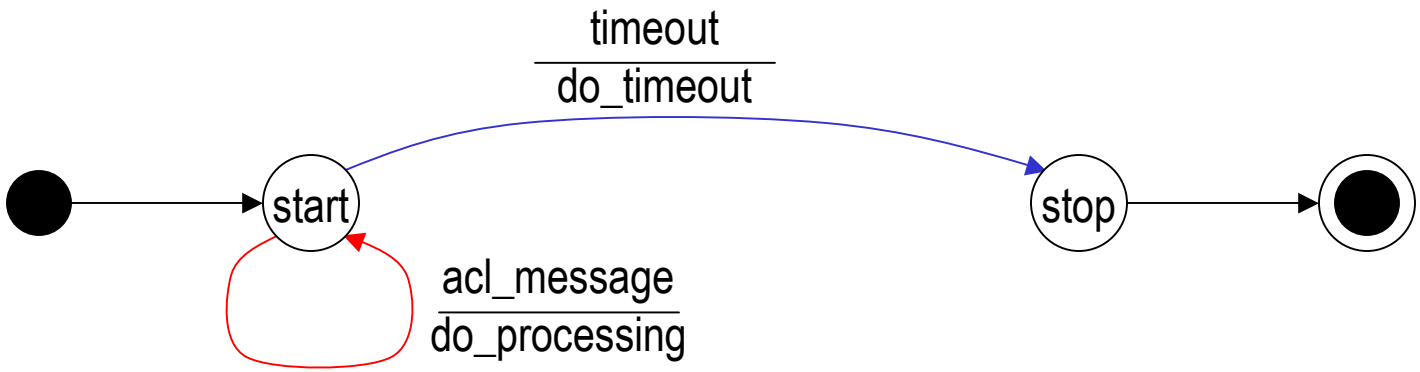
Agent Behavior → FSM

(Action, NewState) = f (Sense, CurrentState)



An Agent Model

From a directed graph to a functional representation



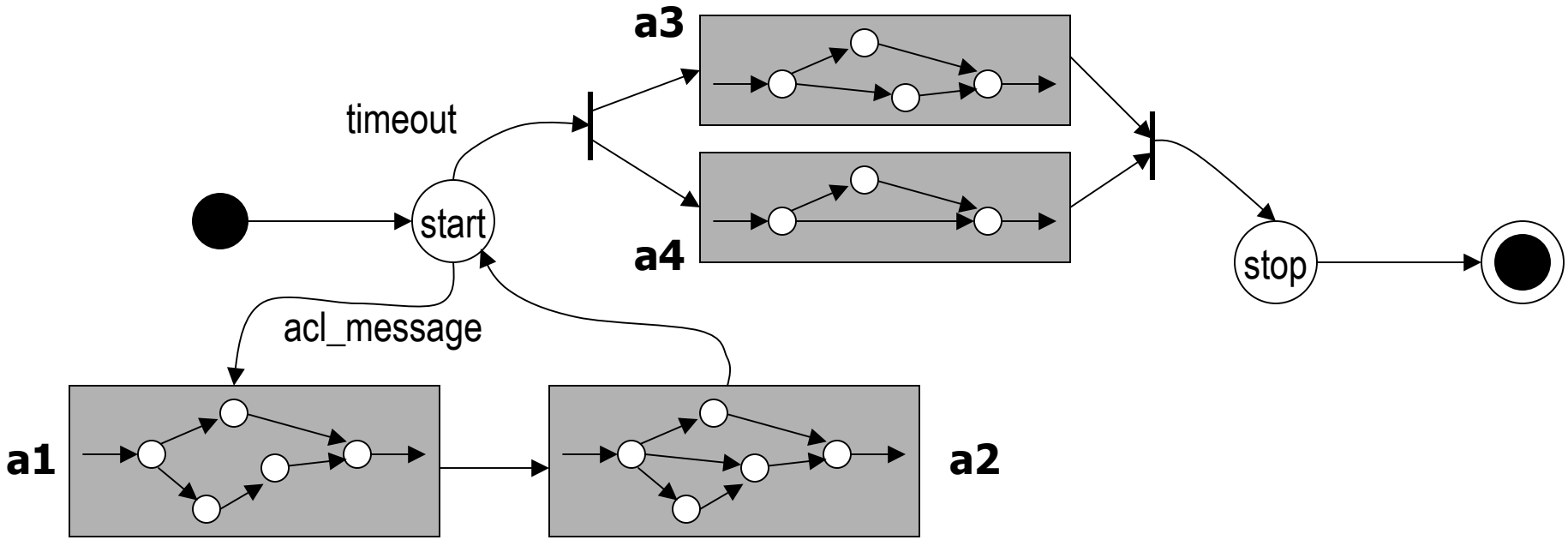
Each clause represent an edge (transition)

Behaviour Function b

```
b (acl_message, start)   =     (do_processing, start)
b (timeout, start)       =     (do_timeout, stop)
```

An Agent Model

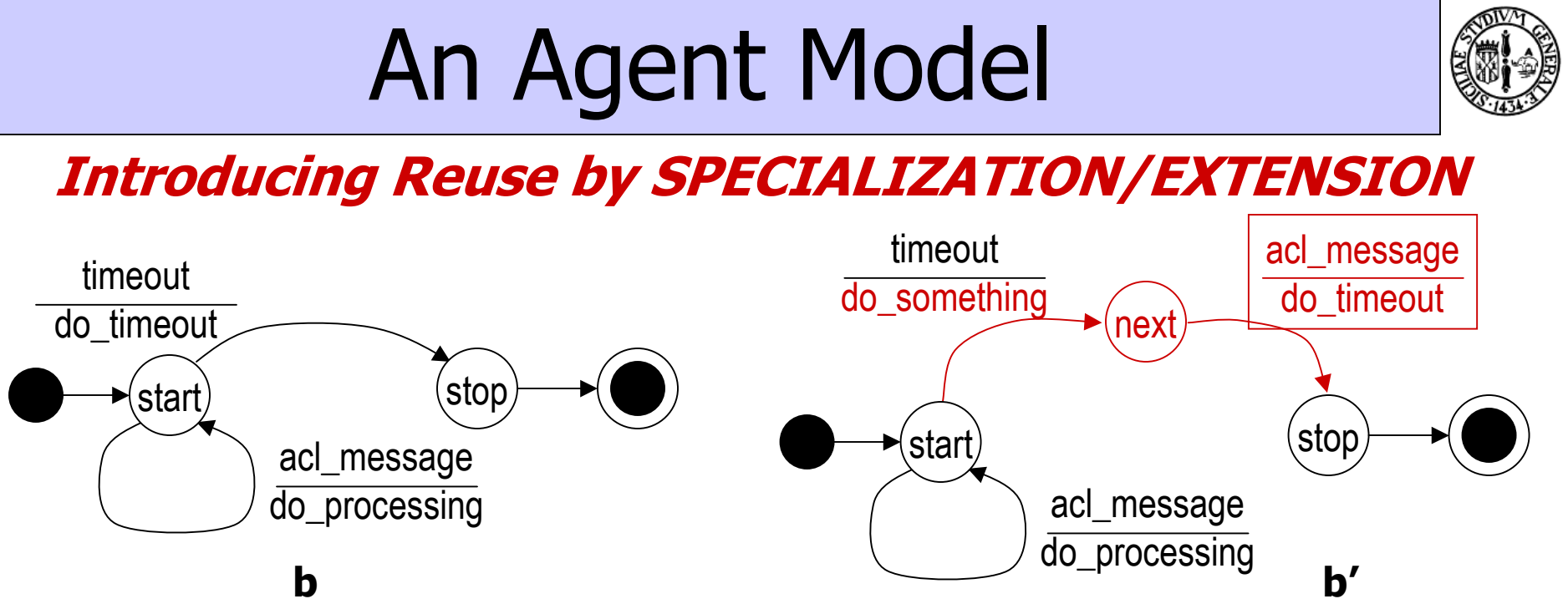
Introducing Modularization and Parallel Processing by COMPOSITION



Behaviour Function b1

```
b1 (acl_message, start) =      (behave(a1,a2), start)
b1 (timeout, start)     =      (behave(a3|a4), stop)
```

An Agent Model



Behaviour Function b

```
1. b (acl_message, start)=    (do_processing, start)
2. b (timeout, start)     =    (do_timeout, stop)
```

Behaviour Function b' extends b

```
b.2  b'(-,-)                  =    (do_something, next)
3.   b'(acl_message, next)=    (do_timeout, stop)
```

We need …

… a language, a platform, a tool to

- ***Express*** a FSM-based behaviour
- ***Compose*** FSMs
- ***Extend*** FSMs

- … and obviously ***execute*** all.

Agent Language Requirements

- General Requirements
  - Safety
    - No dangling pointers, out-of-bound check, exceptions
  - Completeness
    - Libraries for GUI, network, data handling, collections
  - Portability
    - Cross-platform execution of programs

Agent Language Requirements

- Agent-Specific Requirements
  - Agent Model Compliance
    - FSM expression, composition, extension, execution
  - Support for Rationality
    - Language built-in support of BDI constructs, production rules, etc.
  - Support for Distribution
    - MAS are distributed so a support for writing (low or high level) protocols is mandatory (e.g. socket, HTTP, RPC, RMI, CORBA)

Yet Another Language: Erlang

```
-module(samples). % Module declaration
-export([fact/1, foo /2, sum /1, match_inform/1, execute /2]).
% List of function callable by another module

fact (N) when N == 0 - > 1;
fact (N) -> N * fact (N - 1).

foo (hello, X) -> io:format ("Say 'Hello ~w '\n" , [X]);
foo (goodbye, X) -> io:format ("Say 'Goodbye ~w '\n" , [X]);
foo (_, X) -> io:format ("woops !\n").

sum ([]) -> 0;
sum ([H | T]) -> H + sum (T).

match_inform ([$(,$i,$n,$f,$o,$r,$m ,$ | T]) -> true;
match_inform (X) when islist (X) -> false .

execute ({X, 'wants-to-do', Y}, {Y, 'is-feasible'}) ->
  % .. do something
execute (_,_) -> false.
```

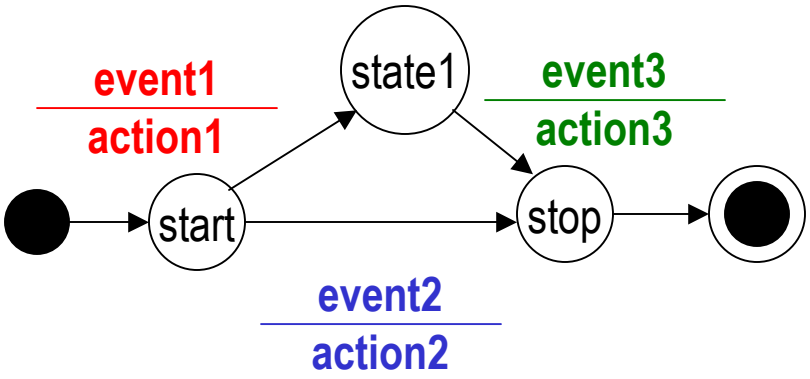
Erlang: a functional and symbolic language with concurrency, distribution and fault-tolerance capabilities

Erlang vs. Agent Language Requirements

- General Requirements
  - ✓Safety: YES
    - No pointers, exceptions, etc.
  - ✓Completeness: YES
    - Set of libraries comparable to that of Java, Python, etc.
    - See http://www.erlang.org
  - ✓Portability: YES
    - (Open Source) Erlang Virtual Machine for many platforms (Windows, MacOS X, FreeBSD, Linux, Solaris, VxWorks)
    - Bytecoded + ahead-of-time compiler (HiPE)

Erlang vs. Agent Language Requirements

- Agent-Specific Requirements
  - ❖ Agent Model Compliance: PARTIALLY
    - Functions with multiple clauses fit well the FSM model, but no support for
      - **Composition**
      - **Specialization**
      - **Execution**




```
-module(sample_fsm).
f(event1, start) ->
  % ... implementation of 'action1'
  state1;
f(event2, start) ->
  % ... implementation of 'action2'
  stop;
f(event3, state1) ->
  % ... implementation of 'action3'
  stop.
```

Erlang vs. Agent Language Requirements

- Agent-Specific Requirements
  - Support for Rationality: NO... BUT
    - Even if syntactically similar to Prolog, Erlang is functional not logic, but
      - **Erlang is symbolic**
      - **Erlang function declaration seems a "rule precondition"**

```
purchasing (['has-goal', Agent, [purchasing, Item, Price ]],
            ['balance-of', Agent, Balance])
            when (Balance - Price) > 3000 ->
  % ... buy the item!
```

→ **If an agent has the goal of purchasing an item and its remaining balance is greater than 3000, then buy the item.**

But rule processing is not present in Erlang, only function clause matching → A rule processing engine is needed

Erlang vs. Agent Language Requirements

- Agent-Specific Requirements
  - ✓ Support for Distribution: YES
    - **Erlang is intrinsically distributed**
    - An Erlang is a collection of Erlang processes that share nothing and (transparently) communicate via message passing
    - Also libraries for sockets, HTTP, SOAP/XMLRPC, CORBA-IIOP, etc.

We often write programs that model the world or interact with the world. Writing such a program in a Erlang is easy. First we perform an analysis, which is a three-step process:

1. We identify all the truly concurrent activities in our real-world activity;
2. We identify all the message channels between the concurrent activities;
3. We write down all the message which can flow on the different message channels.

Now we write the program. The structure of the program should exactly follow the structure of the problem. ***[Joe Armstrong, Erlang inventor]***

Erlang vs. Agent Language Requirements

- **General Requirements**
  - ✓ Safety
  - ✓ Completeness
  - ✓ Portability
- **Agent-Specific Requirements**
  - ❖ *Agent Model Compliance*
  - – *Support for Rationality*
  - ✓ Support for Distirbution

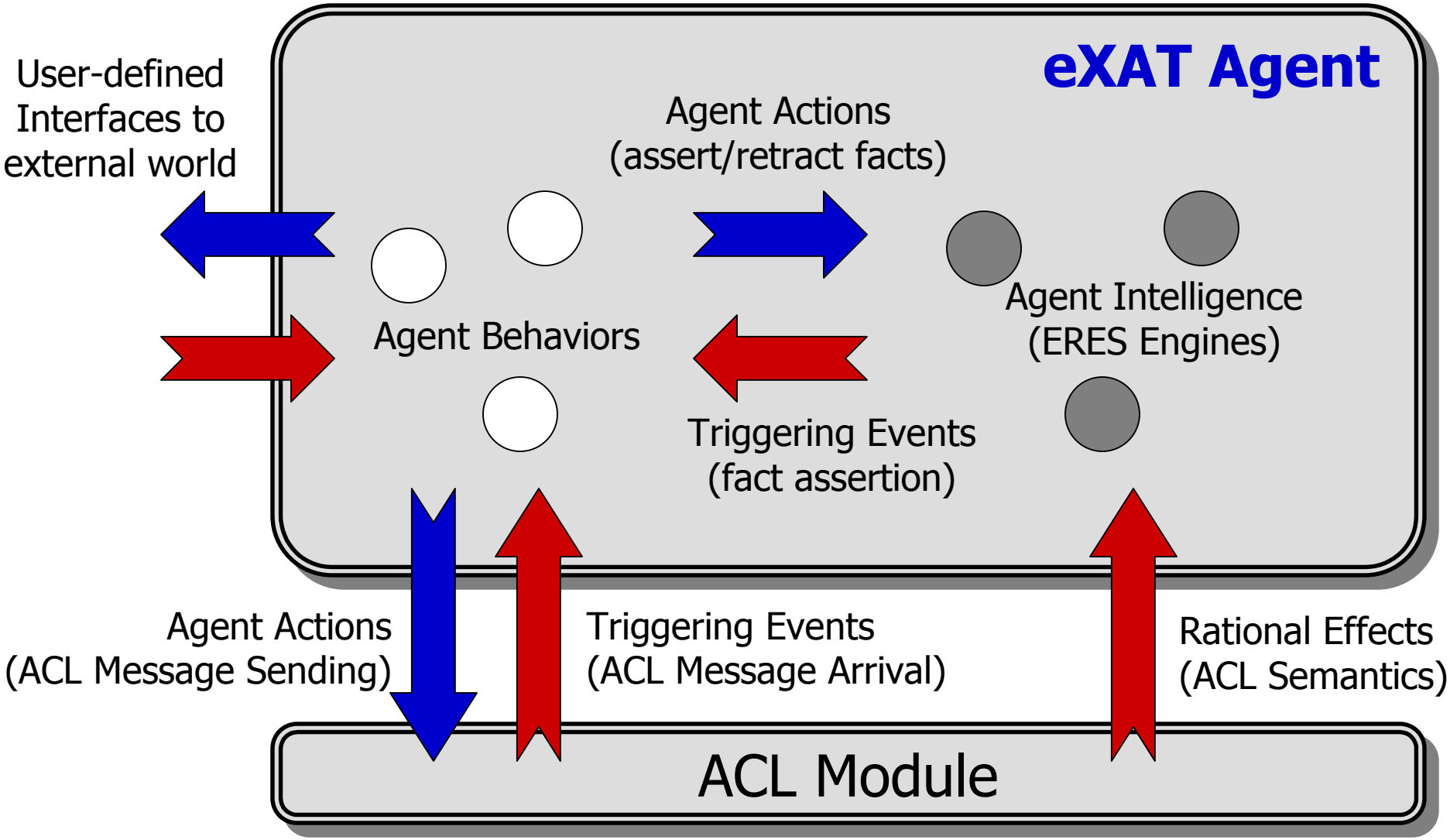
Provided by eXAT → **e**rlang e**X**perimental **A**gent **T**ool

Our agent platform for writing software agents in Erlang

- Totally written in Erlang
- Source code available
- BSD-style license
- About 70 downloads

`http://www.diit.unict.it/users/csanto/exat/`

eXAT Internals: the Agent Model



eXAT Internals: ERES Engines

- ERES module
  - Multiple rule-processing engines
  - Knowledge base and production rules
  - To implement the agent's reasoning process

If X is the child of Y and Y is female, then Y is X's mother; otherwise, if Y is male, then Y is X's father

```
is_parent (true, Engine, Relation, Y, X) ->
  eres:assert (Engine, {Relation, Y, X});
is_parent (_, _, _, _, _) ->
  nil.

rule (Engine, {'child-of', X, Y}) ->
  is_parent (eres:asserted (Engine, {female, Y}), Engine, 'mother-of', Y, X),
  is_parent (eres:asserted (Engine, {male, Y}), Engine, 'father-of', Y, X).
```

eXAT Internals: Behavior Model

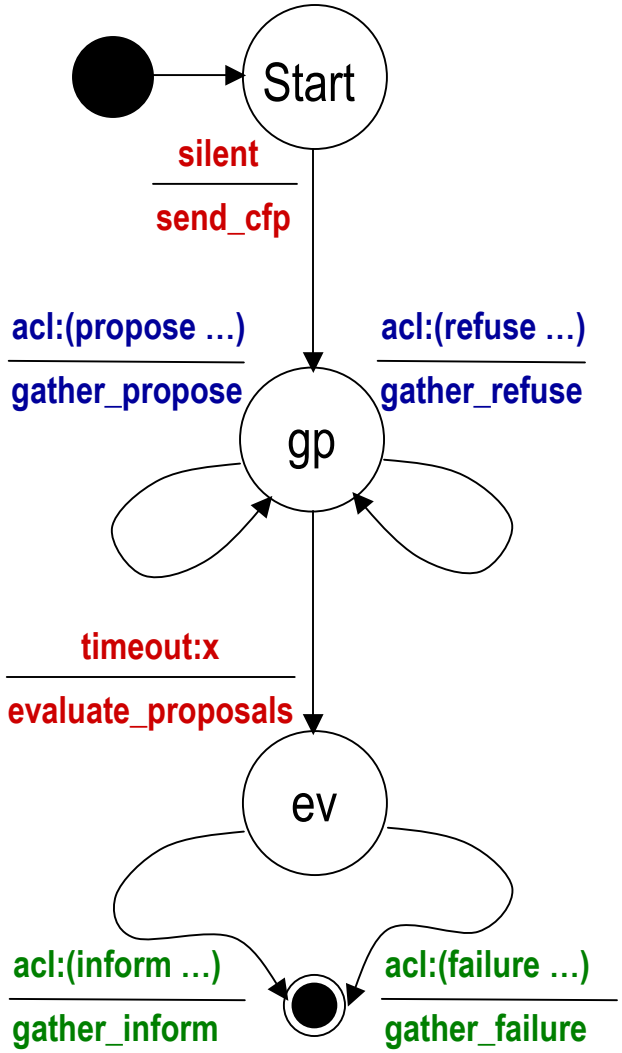
- Agent computations are modeled as FSMs using *behaviors*
- A behaviour is executed *asynchronously* (autonomous agent semantics)

eXAT Behavior Model

- E, the set of *event types* (acl, timeout, eres, silent)
- P, the set of *data patterns* to be associated to a certain event type
- S, the set of *states* of the FSM
- A, the set *actions* to be done
- $f: S \times E \times P \rightarrow A \times S$, the transition function

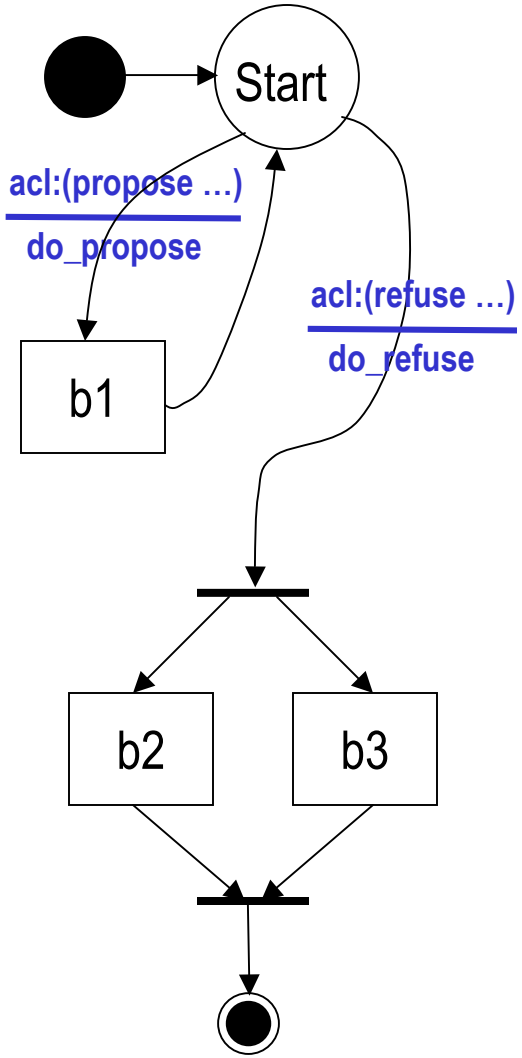
Separation between *event types* and *associated patterns* to allow reuse

eXAT Behaviors: An Example




```
-module (contractnetinitiator).

action (Self, start) -> [{cfp_event, send_cfp}];
action (Self, gp) -> [{propose_event,gather_propose},
                      {refuse_event,gather_refuse},
                   {timeout_event,evaluate_proposals}];
action (Self, ev) -> [{failure_event,gather_failure},
                      {inform_event,gather_inform}].

event (Self, cfp_event) -> {silent, nil};
event (Self, propose_event) ->
  {acl, propose_pattern};
event (Self, refuse_event) ->
  {acl, refuse_pattern};

pattern (Self, propose_pattern) ->
  [#aclmessage {speechact = propose,
                protocol = 'fipa-contractnet'}];
pattern (Self, refuse_pattern) ->
 [#aclmessage {speechact = refuse,
                protocol = 'fipa-contractnet'}];
...
```

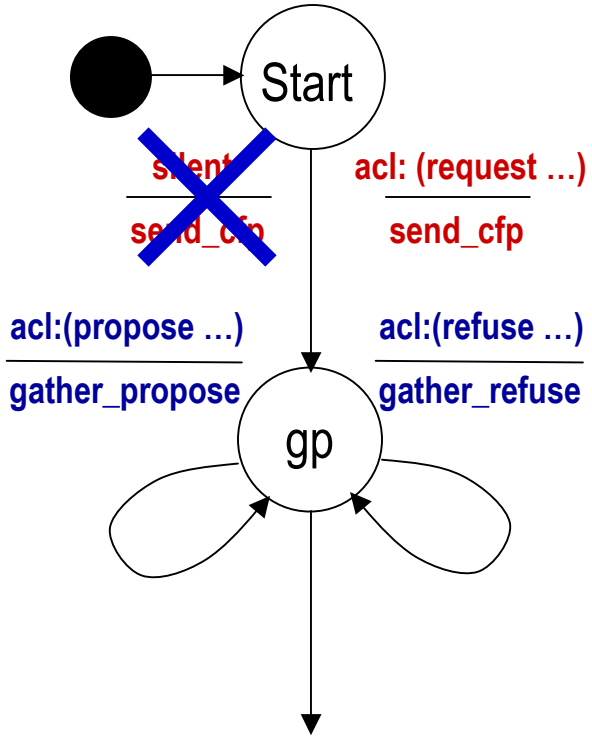
eXAT Behaviors: Composition




```
-module (my_behaviour).

action (Self, start) -> [{propose_event, do_propose},
                         {refuse_event, do_refuse}].

event (Self, propose_event) ->
  {acl, propose_pattern};
event (Self, refuse_event) ->
  {acl, refuse_pattern};

pattern (Self, propose_pattern) ->
  [#aclmessage {speechact = propose}];
pattern (Self, refuse_pattern) ->
 [#aclmessage {speechact = refuse}].

do_propose (Self, Event, Data, State) ->
  agent:behave (Self, b1).

do_refuse (Self, Event, Data, State) ->
  agent:behave (Self, [b2, b3]),
  object:stop (Self).
```

eXAT Behaviors: Extension

An "eXAT Behaviour" is a class that can be extended (reused), according to object-oriented concepts



*E.g., a contract-net initiated by the arrival of a **request** SA:*

→ redefine the **cfp_event**

```
-module (triggeredcontractnetinitiator).

extends () ->
  contractnetinitiator.

event (Self, cfp_event) ->
  {acl, request_pattern}.

pattern (Self, request_pattern) ->
  [#aclmessage {speechact = request}].
```

- **Definition of *methods* with multiple *clauses* and *guards***
- **Redefinition (in a subclass) of even a *single clause of a method* or *add a clause***

Erlang + eXAT vs. Agent Development Requirements

- General Requirements
  - ✓Safety: YES
  - ✓Completeness: YES
  - ✓Portability: YES
- Agent-Specific Requirements
  - ✓Agent Model Compliance: YES
  - ✓Support for Rationality: YES
  - ✓Support for Distribution: YES

Java vs. Agent Development Requirements

- General Requirements
  - ✓Safety: YES
  - ✓Completeness: YES
  - ✓Portability: YES
- Agent-Specific Requirements
  - – Agent Model Compliance: NO
  - – Support for Rationality: NO
  - ✓Support for Distribution: YES
    - Even if not fully transparent (LMI is different than RMI)

Conclusions

- We don't want to denigrate Java!
- But… each problem must be solved using the approach that best fit
- And Erlang seems a good choice for agents
- Erlang is listed in the "Agent software" web page of AgentLink
- In April and Go! [Clark and McCabe], messaging model and message matching constructs are derived from Erlang

- eXAT implementation status:
  - Version 1.0.0 is currently released
  - It does not support standard FIPA MTP. It will be added soon.
  - A version of ERES (2.0) implementing the RETE algorithm is being developed
  - "eXperimental" means a continuous "work-in-progress" to
    - … improve the platform, and
    - … include any suggestion you would give us

Download **eXAT** at:

http://www.diit.unict.it/users/csanto/exat/

Contact Info

Corrado Santoro
University of Catania, Engineering Faculty
Viale Andrea Doria, 6 - 95125 - CATANIA - ITALY
EMail: csanto@diit.unict.it

Download **erlang** at:

http://www.erlang.org