Multi-Agent Oriented Programming

The JaCaMo Platform

O. Boissier[1] R.H. Bordini[2] J.F. Hübner[3] A. Ricci[4]

1. Mines Saint-Etienne (ENSMSE), Saint Etienne, France

2 Pontificia Universidade Catolica do Rio Grande do Sul (PUCRS), Porto Alegre, Brazil

3. Federal University of Santa Catarina (UFSC), Florianópolis, Brazil

4. University of Bologna (UNIBO), Bologna, Italy

September 2016

Tutorial Organisation

- Introduction to Multi-Agent Oriented Programming
- **Programming Agents within JaCaMo**
- Programming Agents' Environment within JaCaMo
- Programming Agents' Interaction within JaCaMo
- Programming Agents' Organisations within JaCaMo
- Conclusion & Perspectives

Multi-Agent Oriented Programming

Programming Agents

Outline

Literature

Books: [Bordini et al., 2005], [Bordini et al., 2009]

Proceedings: ProMAS, DALT, LADS, EMAS, AGERE, ...

Surveys: [Bordini et al., 2006], [Fisher et al., 2007] ...

Languages of historical importance: Agent0 [Shoham, 1993], AgentSpeak(L) [Rao, 1996], MetateM [Fisher, 2005], 3APL [Hindriks et al., 1997], Golog [Giacomo et al., 2000]

Other prominent languages:
Jason [Bordini et al., 2007], Jadex [Pokahr et al., 2005], 2APL [Dastani, 2008], GOAL [Hindriks, 2009], JACK [Winikoff, 2005], JIAC, AgentFactory

But many others languages and platforms...

Some Languages and Platforms

Jason (Hübner, Bordini, ...); 3APL and 2APL (Dastani, van Riemsdijk, Meyer, Hindriks, ...); Jadex (Braubach, Pokahr); MetateM (Fisher, Guidini, Hirsch, ...); ConGoLog (Lesperance, Levesque, ... / Boutilier – DTGolog); Teamcore/ MTDP (Milind Tambe, ...); IMPACT (Subrahmanian, Kraus, Dix, Eiter); CLAIM (Amal El Fallah-Seghrouchni, ...); GOAL (Hindriks); BRAHMS (Sierhuis, ...); SemantiCore (Blois, ...); STAPLE (Kumar, Cohen, Huber); Go! (Clark, McCabe); Bach (John Lloyd, ...); MINERVA (Leite, ...); SOCS (Torroni, Stathis, Toni, ...); FLUX (Thielscher); JIAC (Hirsch, ...); JADE (Agostino Poggi, ...); JACK (AOS); Agentis (Agentis Software); Jackdaw (Calico Jack); *simpAL*, *ALOO* (Ricci, ...);

...

Motivation

- Agents are used to solve problems.
- The characteristics of the problem influence the way the agents used to solve it are built (their architecture).
- It may be the case that some architectures are designed using general principles $\rightsquigarrow$ we then talk about agent models.
- Some of these models have a theory associated with them that allows the verification of some properties.

Agent Models Analysis Grid

External Factors Dimension

Scope of the reasoning on external factors (the environment, other agents, the organization)

- Situated Agents
  - agents that reason about themselves and about their environment
- Social Agents
  - agents that reason about themselves, about their environment and about the interactions with others
- Organized Agents
  - agents that reason about themselves, about their environment and about the interactions with others and about the social structures enforcing these interactions

Agent Models Analysis Grid

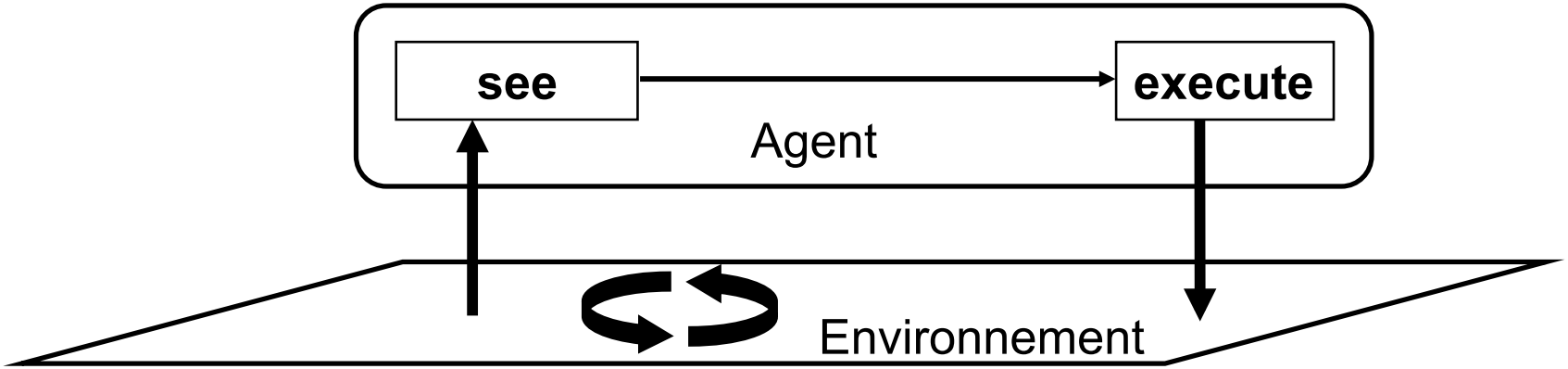
Coupling Dimension

Strength of the coupling with external factors (the environment, other agents, the organization)

- Reactive Agent
  - agents that are coupling perception of the external factors and action
- Deliberative Agent
  - agents that are deliberating on the actions to execute from their perception of the external factors and from their goals
- Hybrid Agent
  - agents that are mixing reactivity and deliberation

Reactive Agent model

- The process cycle of an agent is a closed loop between "execute" and "see" (Stimulus/Response)
- reaction to the evolution of the environment
- No explicit representation of the environment, of the other agents, of its skills,
- Decisions are done without reference to the past (no history), to the futur (no planning)



Reactive Agent model

Reactive approach arises in opposition to the symbolic reasoning model (AI). Several approaches that are based on :

- behaviours
  - (Brooks 86), (Steels 89), (robotic)
  - (Drogoul 93) (ethology)
- interactions
  - (Demazeau 93) (image analysis, cartography, âĂę)
  - (Bura 91) (games)
- situations
  - (Agre 87) (games)
  - (Wavish 90) (design, manufacturing)

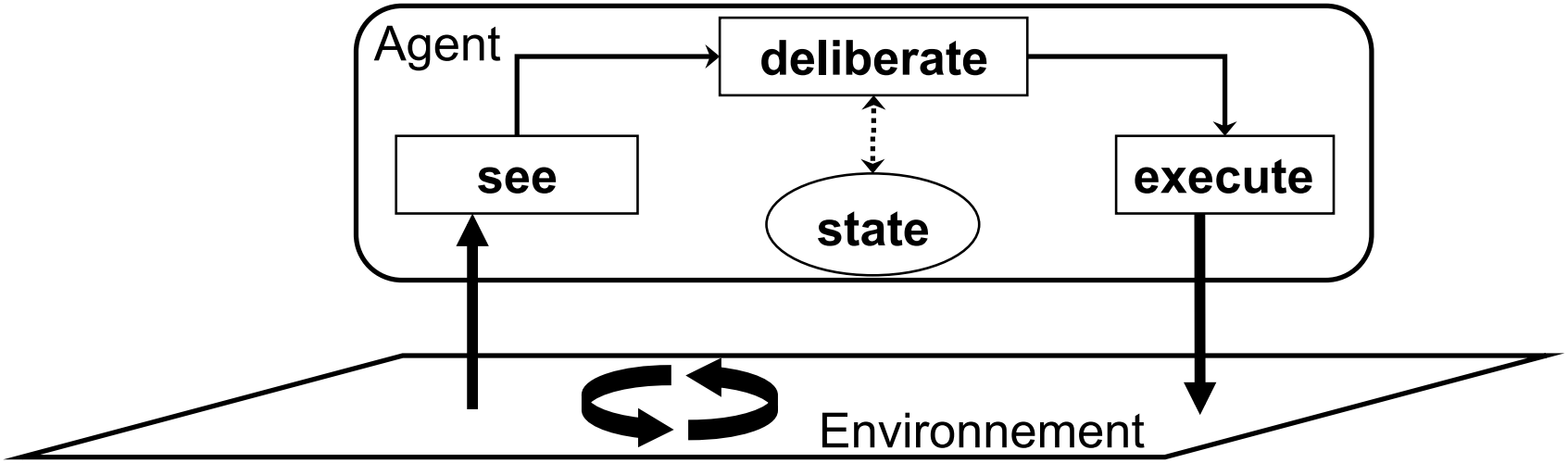
Reactive Agent model

- Example of control cycle of a reactive agent (implemented as a set of condition/action rules):

```
condition-action rules
set of percepts
do {
    percepts := see();
    state := interpret-inputs(percepts);
    rule := match(state,rules);
    execute(rule[action]);
} while (true);
```

Deliberative Agent model

- The process cycle of an agent introduces a "deliberate" function between "see" and "execute" in order to choose the "right" action
- Explicit Representation of the environment, of the other agents, of its skills, ...
- History management, ...



Hybrid Agent Model

Hybrid Agent's Model: Reactive and Deliberative Agent

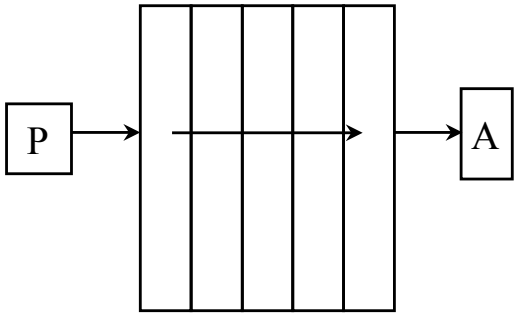
- Reactive agents are too simple - they work well in some scenarios, but they fail to solve complex problems
- Deliberative agents are too complex - they need too much time to deliberate, they fail in very dynamic environments
- The reactive and deliberative behaviors are organized in layers
- Examples: Touring Machines [Ferguson 94], InterRaP [Muller 95],

Agent Architectures

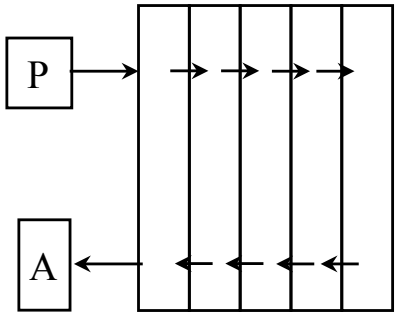
- Modules Organisation:

P : perception, A : action

a) horizontal architecture



b) modular vertical architecture one path



c) layered vertical architecture two paths

- Control flow: one / several
- Data flow: broadcast, translation
- Control structure: inhibition, hierarchy, ...

Agent Oriented Programming

Features

- Reacting to events $\times$ long-term goals
- Course of actions depends on circumstance
- Plan failure (dynamic environments)
- Social ability
- Combination of theoretical and practical reasoning

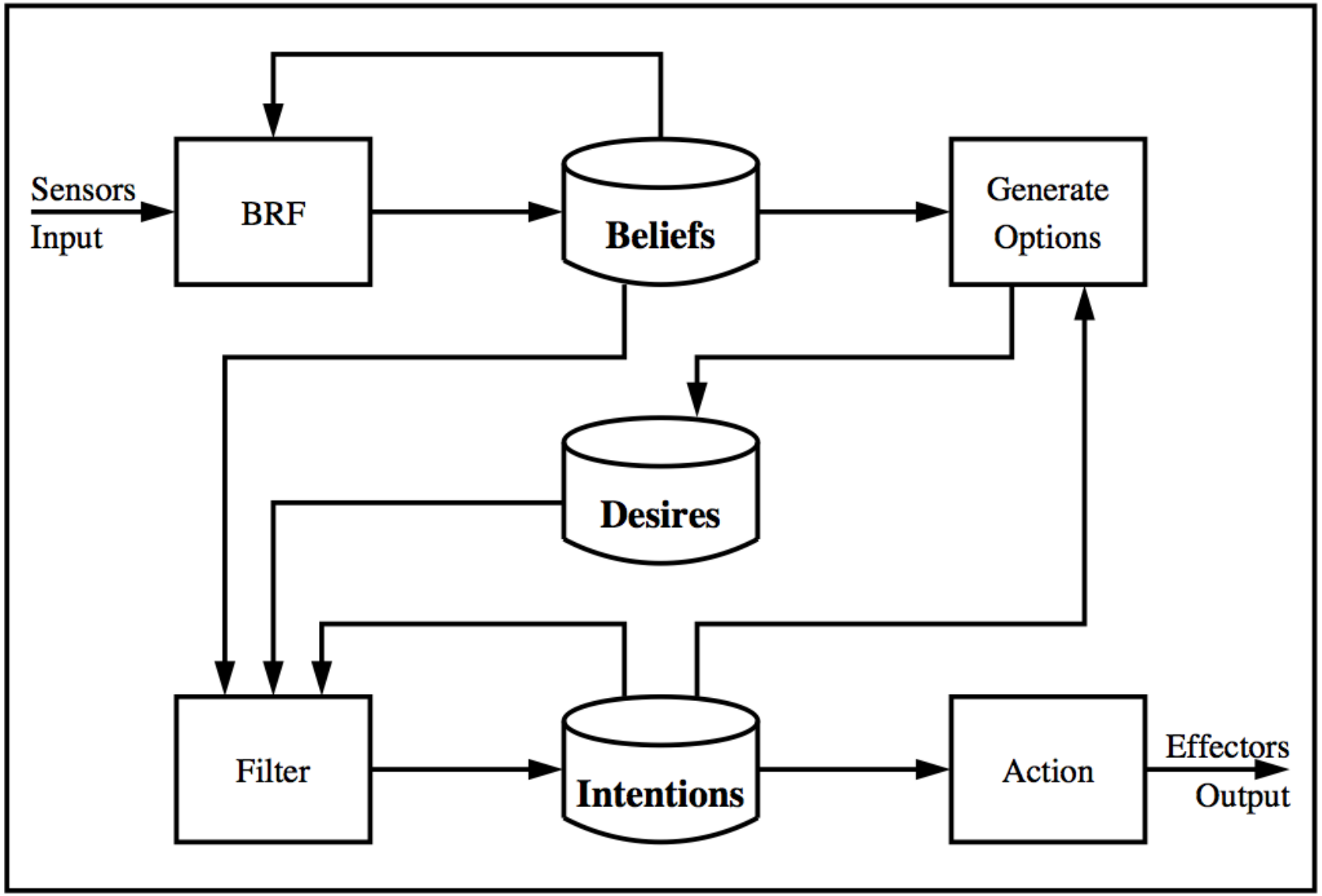
Agent Oriented Programming

Fundamentals

- Use of mentalistic notions and a societal view of computation [Shoham, 1993]
- Heavily influence by the BDI architecture and reactive planning systems [Bratman et al., 1988]

BDI architecture

(the mentalistic view)



BDI architecture [Wooldridge, 2009]

```
while true do
    B ← brf(B, perception()) ;                    // belief revision
    D ← options(B, I) ;                           // desire revision
    I ← filter(B, D, I) ;                         // deliberation
    π ← plan(B, I, A) ;                           // means-end
    while π ≠ ∅ do
        execute( head(π) )
        π ← tail(π)
```

BDI architecture [Wooldridge, 2009]

```
while true do
    B ← brf(B, perception()) ;          // belief revision
    D ← options(B, I) ;                 // desire revision
    I ← filter(B, D, I) ;               // deliberation
    π ← plan(B, I, A) ;                 // means-end
    while π ≠ ∅ do
        execute( head(π) )
        π ← tail(π)
```

fine for pro-activity, but not for reactivity (over commitment)

BDI architecture [Wooldridge, 2009]

```
while true do
    B ← brf(B, perception()) ;                    // belief revision
    D ← options(B, I) ;                           // desire revision
    I ← filter(B, D, I) ;                         // deliberation
    π ← plan(B, I, A) ;                           // means-end
    while π ≠ ∅ do
        execute( head(π) )
        π ← tail(π)
        B ← brf(B, perception())
        if ¬sound(π, I, B) then
            π ← plan(B, I, A)
```

revise commitment to plan – re-planning for context adaptation

BDI architecture [Wooldridge, 2009]

```
while true do
    B ← brf(B, perception()) ;                          // belief revision
    D ← options(B, I) ;                                 // desire revision
    I ← filter(B, D, I) ;                               // deliberation
    π ← plan(B, I, A) ;                                 // means-end
    while π ≠ ∅ and ¬succeeded(I, B) and ¬impossible(I, B) do
        execute( head(π) )
        π ← tail(π)
        B ← brf(B, perception())
        if ¬sound(π, I, B) then
            π ← plan(B, I, A)
```

revise commitment to intentions – Single-Minded Commitment

BDI architecture [Wooldridge, 2009]

```
while true do
    B ← brf(B, perception()) ;                          // belief revision
    D ← options(B, I) ;                                 // desire revision
    I ← filter(B, D, I) ;                               // deliberation
    π ← plan(B, I, A) ;                                 // means-end
    while π ≠ ∅ and ¬succeeded(I, B) and ¬impossible(I, B) do
        execute( head(π) )
        π ← tail(π)
        B ← brf(B, perception())
        if reconsider(I, B) then
            D ← options(B, I)
            I ← filter(B, D, I)
        if ¬sound(π, I, B) then
            π ← plan(B, I, A)
```

reconsider the intentions (not always!)

Jason

(let's go programming those nice concepts)

Outline

(BDI) Hello World – agent bob

```
happy(bob).                                          // B
!say(hello).                                         // D

+!say(X) : happy(bob) <- .print(X).                  // I
```

Desires in Hello World

```
+happy(bob) <- !say(hello).

+!say(X) : not today(monday) <- .print(X).
```

Hello World

source of beliefs

```
+happy(bob)[source(A)]
    :  someone_who_knows_me_very_well(A)
    <- !say(hello).

+!say(X) : not today(monday) <- .print(X).
```

Hello World

plan selection

```
+happy(H)[source(A)]
    :  sincere(A) & .my_name(H)
    <- !say(hello).

+happy(H)
    :  not .my_name(H)
    <- !say(i_envy(H)).

+!say(X) : not today(monday) <- .print(X).
```

Hello World

intention revision

```
+happy(H)[source(A)]
    :  sincere(A) & .my_name(H)
    <- !say(hello).

+happy(H)
    :  not .my_name(H)
    <- !say(i_envy(H)).

+!say(X) : not today(monday) <- .print(X); !say(X).

-happy(H)
    :  .my_name(H)
    <- .drop_intention(say(hello)).
```

Hello World

intention revision

```
+happy(H)[source(A)]
    :  sincere(A) & .my_name(H)
    <- !say(hello).

+happy(H)
    :  not .my_name(H)
    <- !say(i_envy(H)).

+!say(X) : not today(monday) <- .print(X); !say(X).

-happy(H)
    :  .my_name(H)
    <- .drop_intention(say(hello)).
```

Outline

AgentSpeak

The foundational language for *Jason*

- Originally proposed by Rao [Rao, 1996]
- Programming language for BDI agents
- Elegant notation, based on logic programming
- Inspired by PRS (Georgeff & Lansky), dMARS (Kinny), and BDI Logics (Rao & Georgeff)
- Abstract programming language aimed at theoretical results

Jason

A practical implementation of a variant of AgentSpeak

- *Jason* implements the operational semantics of a variant of AgentSpeak
- Has various extensions aimed at a more practical programming language (e.g. definition of the MAS, communication, ...)
- Highly customised to simplify extension and experimentation
- Developed by Jomi F. Hübner, Rafael H. Bordini, and others

Main Language Constructs

Beliefs: represent the information available to an agent (e.g. about the environment or other agents)

Goals: represent states of affairs the agent wants to bring about

Plans: are recipes for action, representing the agent's know-how

Events: happen as consequence to changes in the agent's beliefs or goals

Intentions: plans instantiated to achieve some goal

Main Language Constructs and Runtime Structures

Beliefs: represent the information available to an agent (e.g. about the environment or other agents)

Goals: represent states of affairs the agent wants to bring about

Plans: are recipes for action, representing the agent's know-how

Events: happen as consequence to changes in the agent's beliefs or goals

Intentions: plans instantiated to achieve some goal

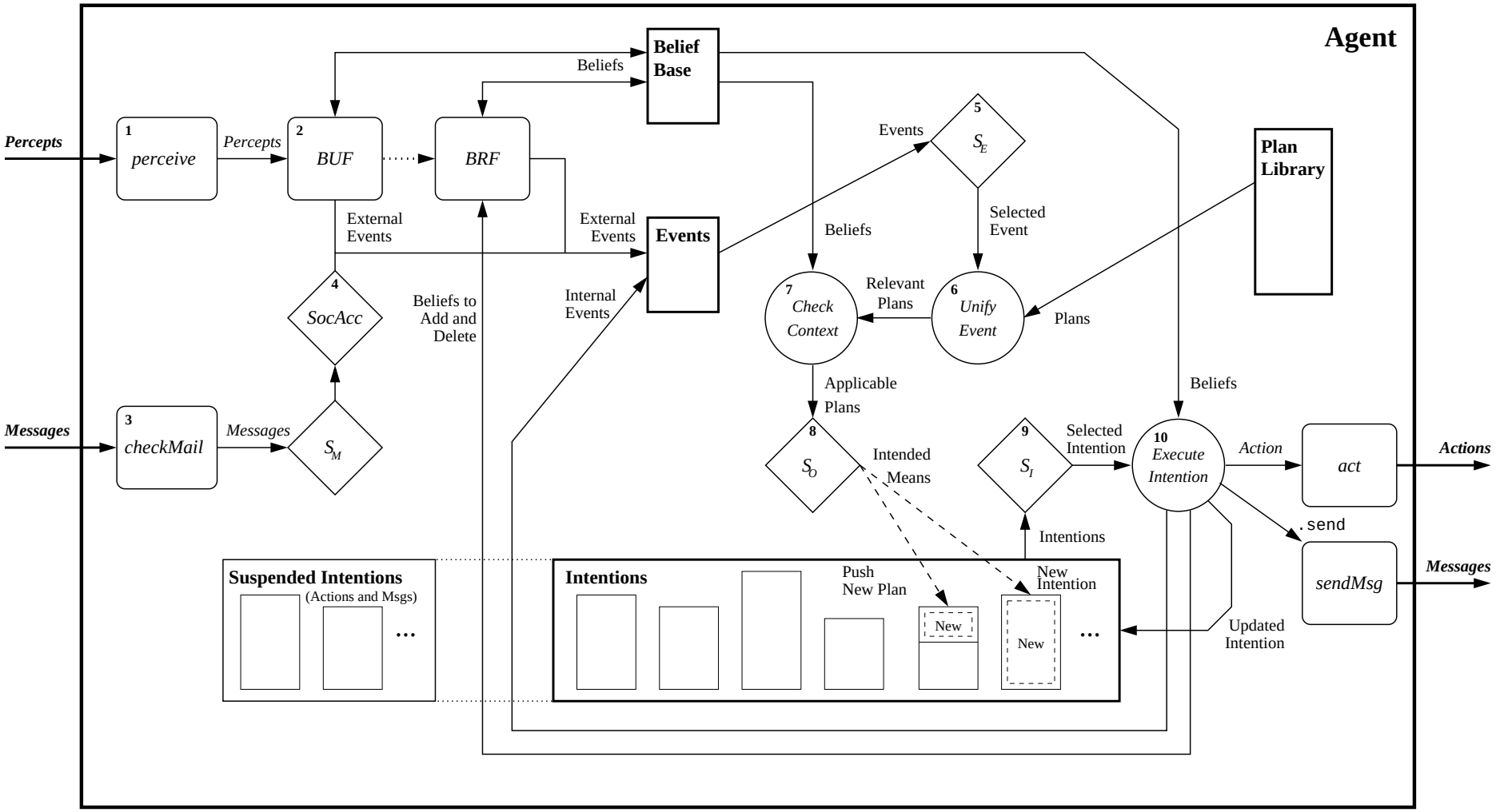
Outline

Basic Reasoning cycle

runtime interpreter

- perceive the environment and update belief base
- process new messages
- select event
- select relevant plans
- select applicable plans
- create/update intention
- select intention to execute
- execute one step of the selected intention

Jason Reasoning Cycle



Outline

Beliefs — Representation

Syntax

Beliefs are represented by annotated literals of first order logic

`functor(`$term_1$`, ..., `$term_n$`)[`$annot_1$`, ..., `$annot_m$`]`

Example (belief base of agent Tom)

```
red(box1)[source(percept)].
friend(bob,alice)[source(bob)].
lier(alice)[source(self),source(bob)].
~lier(bob)[source(self)].
```

Beliefs — Dynamics I

by perception

beliefs annotated with source(percept) are automatically updated accordingly to the perception of the agent

by intention

the plan operators + and - can be used to add and remove beliefs annotated with source(self) (mental notes)

```
+lier(alice); // adds lier(alice)[source(self)]
-lier(john); // removes lier(john)[source(self)]
```

Beliefs — Dynamics II

by communication

when an agent receives a tell message, the content is a new belief annotated with the sender of the message

```
.send(tom,tell,lier(alice)); // sent by bob
// adds lier(alice)[source(bob)] in Tom's BB
...
.send(tom,untell,lier(alice)); // sent by bob
// removes lier(alice)[source(bob)] from Tom's BB
```

Goals — Representation

Types of goals

- Achievement goal: goal to do
- Test goal: goal to know

Syntax

Goals have the same syntax as beliefs, but are prefixed by
! (achievement goal) or
? (test goal)

Example (Initial goal of agent Tom)

```
!write(book).
```

Goals — Dynamics I

by intention

the plan operators **!** and **?** can be used to add a new goal annotated with source(self)

```
...
// adds new achievement goal !write(book)[source(self)]
!write(book);

// adds new test goal ?publisher(P)[source(self)]
?publisher(P);
...
```

Goals — Dynamics II

by communication – achievement goal

when an agent receives an achieve message, the content is a new achievement goal annotated with the sender of the message

```
.send(tom,achieve,write(book)); // sent by Bob
// adds new goal write(book)[source(bob)] for Tom
...
.send(tom,unachieve,write(book)); // sent by Bob
// removes goal write(book)[source(bob)] for Tom
```

Goals — Dynamics III

by communication – test goal

when an agent receives an askOne or askAll message, the content is a new test goal annotated with the sender of the message

```
.send(tom,askOne,published(P),Answer); // sent by Bob
// adds new goal ?publisher(P)[source(bob)] for Tom
// the response of Tom will unify with Answer
```

Triggering Events — Representation

- Events happen as consequence to changes in the agent's beliefs or goals
- An agent reacts to events by executing plans
- Types of plan triggering events

+b	(belief addition)
-b	(belief deletion)
+!g	(achievement-goal addition)
-!g	(achievement-goal deletion)
+?g	(test-goal addition)
-?g	(test-goal deletion)

Plans — Representation

An AgentSpeak plan has the following general structure:

triggering_event : context <- body.

where:

- the triggering event denotes the events that the plan is meant to handle
- the context represent the circumstances in which the plan can be used
- the body is the course of action to be used to handle the event if the context is believed true at the time a plan is being chosen to handle the event

Plans — Operators for Plan **Context**

Boolean operators

&	(and)
|	(or)
not	(not)
=	(unification)
>, >=	(relational)
<, <=	(relational)
==	(equals)
\==	(different)

Arithmetic operators

+	(sum)
-	(subtraction)
*****	(multiply)
/	(divide)
div	(divide – integer)
mod	(remainder)
******	(power)

Plans — Operators for Plan **Body**

```
+rain :  time_to_leave(T) & clock.now(H) & H >= T
   <- !g1;           // new sub-goal
      !!g2;          // new goal
      ?b(X);         // new test goal
      +b1(T-H);      // add mental note
      -b2(T-H);      // remove mental note
      -+b3(T*H);     // update mental note
      jia.get(X);    // internal action
      X > 10;        // constraint to carry on
      close(door);   // external action
      !g3[hard_deadline(3000)].   // goal with deadline
```

Plans — Example

```
+green_patch(Rock)[source(percept)]
    :  not battery_charge(low)
    <- ?location(Rock,Coordinates);
       !at(Coordinates);
       !examine(Rock).

+!at(Coords)
    :  not at(Coords) & safe_path(Coords)
    <- move_towards(Coords);
       !at(Coords).
+!at(Coords)
    :  not at(Coords) & not safe_path(Coords)
    <- ...
+!at(Coords) :  at(Coords).
```

Plans — Dynamics

The plans that form the plan library of the agent come from

- initial plans defined by the programmer
- plans added dynamically and intentionally by
  - .add_plan
  - .remove_plan
- plans received from
  - tellHow messages
  - untellHow

A note about “Control”

Agents can control (manipulate) their own (and influence the others)

- beliefs
- goals
- plan

By doing so they control their behaviour

The developer provides initial values of these elements and thus also influence the behaviour of the agent

Outline

Failure Handling: Contingency Plans

Example (an agent blindly committed to g)

```
+!g :  g.

+!g :  ...  <- ...  ?g.

-!g :  true <- !g.
```

Meta Programming

Example (an agent that asks for plans *on demand*)

```
-!G[error(no_relevant)] :  teacher(T)
   <- .send(T, askHow, { +!G }, Plans);
      .add_plan(Plans);
      !G.
```

in the event of a failure to achieve **any** *goal G due to no relevant plan, asks a teacher for plans to achieve G and then try G again*

- The failure event is annotated with the error type, line, source, ... error(no_relevant) means no plan in the agent's plan library to achieve G
- { +!G } is the syntax to enclose triggers/plans as terms

Other Language Features

Strong Negation

```
+!leave(home)
    :  ~raining
    <- open(curtains); ...

+!leave(home)
    :  not raining & not ~raining
    <- .send(mum,askOne,raining,Answer,3000); ...
```

Prolog-like Rules in the Belief Base

```
tall(X) :-
   woman(X) & height(X, H) & H > 1.70
   |
   man(X) & height(X, H) & H > 1.80.

likely_color(Obj,C) :-
   colour(Obj,C)[degOfCert(D1)] &
   not (colour(Obj,_)[degOfCert(D2)] & D2 > D1) &
   not ~colour(C,B).
```

Plan Annotations

- Like beliefs, plans can also have annotations, which go in the plan label
- Annotations contain meta-level information for the plan, which selection functions can take into consideration
- The annotations in an intended plan instance can be changed dynamically (e.g. to change intention priorities)
- There are some pre-defined plan annotations, e.g. to force a breakpoint at that plan or to make the whole plan execute atomically

Example (an annotated plan)

```
@myPlan[chance_of_success(0.3), usual_payoff(0.9),
        any_other_property]
+!g(X) : c(t) <- a(X).
```

Internal Actions

- Unlike actions, internal actions do not change the environment
- Code to be executed as part of the agent reasoning cycle
- AgentSpeak is meant as a high-level language for the agent's practical reasoning and internal actions can be used for invoking legacy code elegantly

- Internal actions can be defined by the user in Java

```
libname.action_name(...)
```

Standard Internal Actions

- Standard (pre-defined) internal actions have an empty library name
  - `.print(`$term_1$`,`$term_2$`,...)`
  - `.union(`$list_1$`, `$list_2$`, `$list_3$`)`
  - `.my_name(`*var*`)`
  - `.send(`*ag*`,`*perf*`,`*literal*`)`
  - `.intend(`*literal*`)`
  - `.drop_intention(`*literal*`)`
- Many others available for: printing, sorting, list/string operations, manipulating the beliefs/annotations/plan library, creating agents, waiting/generating events, etc.

Jason Customisations

- Agent class customisation:
  selectMessage, selectEvent, selectOption, selectIntention, buf, brf, ...
- Agent architecture customisation:
  perceive, act, sendMsg, checkMail, ...
- Belief base customisation:
  add, remove, contains, ...
  - Example available with *Jason*: persistent belief base (in text files, in data bases, ...)

Outline

Jason × Java

Consider a very simple robot with two goals:

- when a piece of gold is seen, go to it
- when battery is low, go charge it

Java code – go to gold

```
public class Robot extends Thread {
   boolean seeGold, lowBattery;
   public void run() {
      while (true) {
          while (!  seeGold) {
              a = randomDirection();
              doAction(go(a));

          }
          while (seeGold) {
              a = selectDirection();

              doAction(go(a));

} } }   }
```

Java code – charge battery

```
public class Robot extends Thread {
   boolean seeGold, lowBattery;
   public void run() {
      while (true) {
         while (!  seeGold) {
            a = randomDirection();
            doAction(go(a));
            if (lowBattery) charge();
         }
         while (seeGold) {
            a = selectDirection ();
            if (lowBattery) charge();
            doAction(go(a));
            if (lowBattery) charge();
} } }   }
```

Jason code

```
direction(gold)   :- see(gold).
direction(random) :- not see(gold).

+!find(gold)                        // long term goal
   <- ?direction(A);
      go(A);
      !find(gold).
+battery(low)                       // reactivity
   <- !charge.

^!charge[state(started)]            // goal meta-events
   <- .suspend(find(gold)).
^!charge[state(finished)]
   <- .resume(find(gold)).
```

Jason × Prolog

- With the *Jason* extensions, nice separation of theoretical and practical reasoning
- BDI architecture allows
  - long-term goals (goal-based behaviour)
  - reacting to changes in a dynamic environment
  - handling multiple foci of attention (concurrency)
- Acting on an environment and a higher-level conception of a distributed system

Outline

Some Shortfalls

- IDEs and programming tools are still not anywhere near the level of OO languages
- Debugging is a serious issue — much more than "mind tracing" is needed
- Combination with organisational models is very recent — much work still needed
- Principles for using declarative goals in practical programming problems still not "textbook"
- Large applications and real-world experience much needed!

Some Trends

- Modularity and encapsulation
- Debugging MAS is hard: problems of concurrency, simulated environments, emergent behaviour, mental attitudes
- Logics for Agent Programming languages
- Further work on combining with interaction, environments, and organisations
- We need to put everything together: rational agents, environments, organisations, normative systems, reputation systems, economically inspired techniques, etc.

$\rightsquigarrow$ Multi-Agent Programming

Some Related Projects I

- Speech-act based communication
  Joint work with Renata Vieira, Álvaro Moreira, and Mike Wooldridge
- Cooperative plan exchange
  Joint work with Viviana Mascardi, Davide Ancona
- Plan Patterns for Declarative Goals
  Joint work with M.Wooldridge
- Planning (Felipe Meneguzzi and Colleagues)
- Web and Mobile Applications (Alessandro Ricci and Colleagues)
- Belief Revision
  Joint work with Natasha Alechina, Brian Logan, Mark Jago

Some Related Projects II

- Ontological Reasoning
  - Joint work with Renata Vieira, Álvaro Moreira
  - JASDL: joint work with Tom Klapiscak
- Goal-Plan Tree Problem (Thangarajah et al.)
  Joint work with Tricia Shaw
- Trust reasoning (ForTrust project)
- Agent verification and model checking
  Joint project with M.Fisher, M.Wooldridge, W.Visser, L.Dennis, B.Farwer

Some Related Projects III

- Environments, Organisation and Norms
  - Normative environments
    Join work with A.C.Rocha Costa and F.Okuyama
  - MADeM integration (Francisco Grimaldo Moreno)
  - Normative integration (Felipe Meneguzzi)
- More on `jason.sourceforge.net`, related projects

Summary

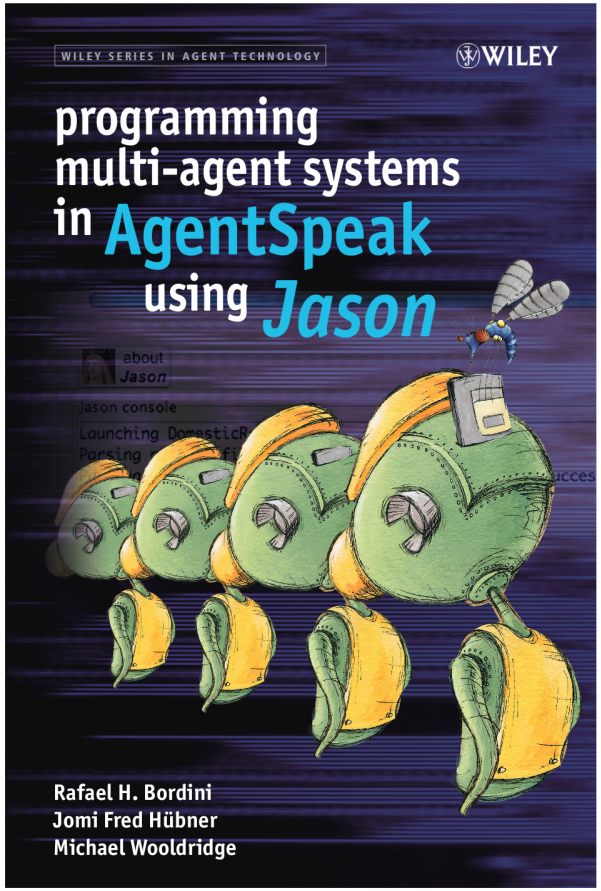
- AgentSpeak
  - Logic + BDI
  - Agent programming language
- *Jason*
  - AgentSpeak interpreter
  - Implements the operational semantics of AgentSpeak
  - Speech-act based communicaiton
  - Highly customisable
  - Useful tools
  - Open source
  - Open issues

Acknowledgements

- Many thanks to the
  - Various colleagues acknowledged/referenced throughout these slides
  - *Jason* users for helpful feedback
  - CNPq for supporting some of our current researh

Further Resources

- http://jason.sourceforge.net
- R.H. Bordini, J.F. Hübner, and M. Wooldrige
  Programming Multi-Agent Systems in AgentSpeak using *Jason*
  John Wiley & Sons, 2007.



Multi-Agent Oriented Programming

The JaCaMo Platform

O. Boissier[1] R.H. Bordini[2] J.F. Hübner[3] A. Ricci[4]

1. Mines Saint-Etienne (ENSMSE), Saint Etienne, France

2 Pontificia Universidade Catolica do Rio Grande do Sul (PUCRS), Porto Alegre, Brazil

3. Federal University of Santa Catarina (UFSC), Florianópolis, Brazil

4. University of Bologna (UNIBO), Bologna, Italy

September 2016

Bibliography I

Bordini, R. H., Braubach, L., Dastani, M., Fallah-Seghrouchni, A. E., Gómez-Sanz, J. J., Leite, J., O'Hare, G. M. P., Pokahr, A., and Ricci, A. (2006).

A survey of programming languages and platforms for multi-agent systems.

Informatica (Slovenia), 30(1):33–44.

Bordini, R. H., Dastani, M., Dix, J., and Fallah-Seghrouchni, A. E., editors (2005).

Multi-Agent Programming: Languages, Platforms and Applications, volume 15 of *Multiagent Systems, Artificial Societies, and Simulated Organizations*.

Springer.

Bordini, R. H., Dastani, M., Dix, J., and Fallah-Seghrouchni, A. E., editors (2009).

Multi-Agent Programming: Languages, Tools and Applications.

Springer.

Bordini, R. H., Hübner, J. F., and Wooldridge, M. (2007).

Programming Multi-Agent Systems in AgentSpeak Using **Jason**.

Wiley Series in Agent Technology. John Wiley & Sons.

Bibliography II

Bratman, M. E., Israel, D. J., and Pollack, M. E. (1988).
Plans and resource-bounded practical reasoning.
Computational Intelligence, 4:349–355.

Dastani, M. (2008).
2apl: a practical agent programming language.
Autonomous Agents and Multi-Agent Systems, 16(3):214–248.

Fisher, M. (2005).
Metatem: The story so far.
In *PROMAS*, pages 3–22.

Fisher, M., Bordini, R. H., Hirsch, B., and Torroni, P. (2007).
Computational logics and agents: A road map of current technologies and future trends.
Computational Intelligence, 23(1):61–91.

Giacomo, G. D., Lespérance, Y., and Levesque, H. J. (2000).
Congolog, a concurrent programming language based on the situation calculus.
Artif. Intell., 121(1-2):109–169.

Bibliography III

Hindriks, K. V. (2009).
Programming rational agents in GOAL.
In [Bordini et al., 2009], pages 119–157.

Hindriks, K. V., de Boer, F. S., van der Hoek, W., and Meyer, J.-J. C. (1997).
Formal semantics for an abstract agent programming language.
In Singh, M. P., Rao, A. S., and Wooldridge, M., editors, *ATAL*, volume 1365 of *Lecture Notes in Computer Science*, pages 215–229. Springer.

Pokahr, A., Braubach, L., and Lamersdorf, W. (2005).
Jadex: A bdi reasoning engine.
In [Bordini et al., 2005], pages 149–174.

Rao, A. S. (1996).
Agentspeak(l): Bdi agents speak out in a logical computable language.
In de Velde, W. V. and Perram, J. W., editors, *MAAMAW*, volume 1038 of *Lecture Notes in Computer Science*, pages 42–55. Springer.

Shoham, Y. (1993).
Agent-oriented programming.
Artif. Intell., 60(1):51–92.

Bibliography IV

Winikoff, M. (2005).
Jack intelligent agents: An industrial strength platform.
In [Bordini et al., 2005], pages 175–193.

Wooldridge, M. (2009).
An Introduction to MultiAgent Systems.
John Wiley and Sons, 2nd edition.