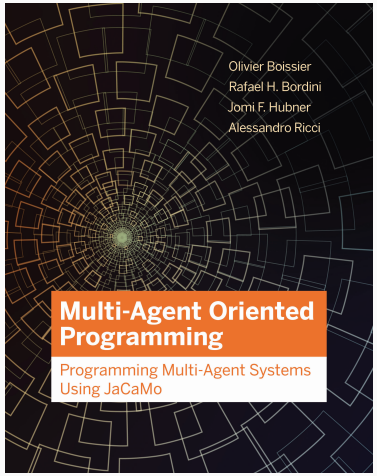
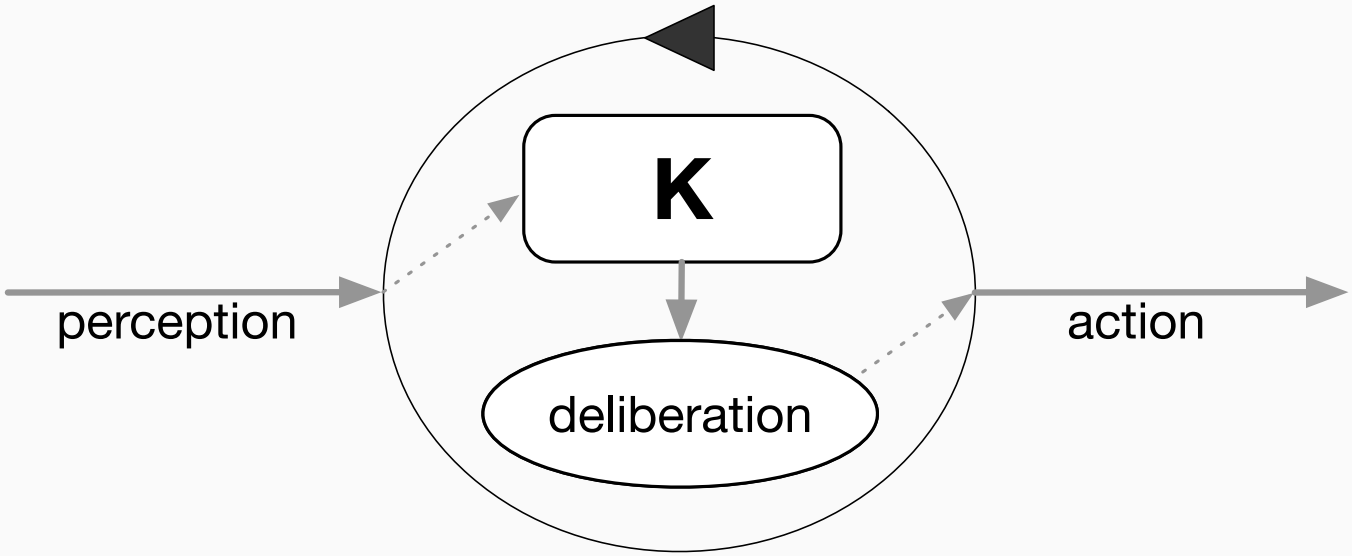


Agent Dimension

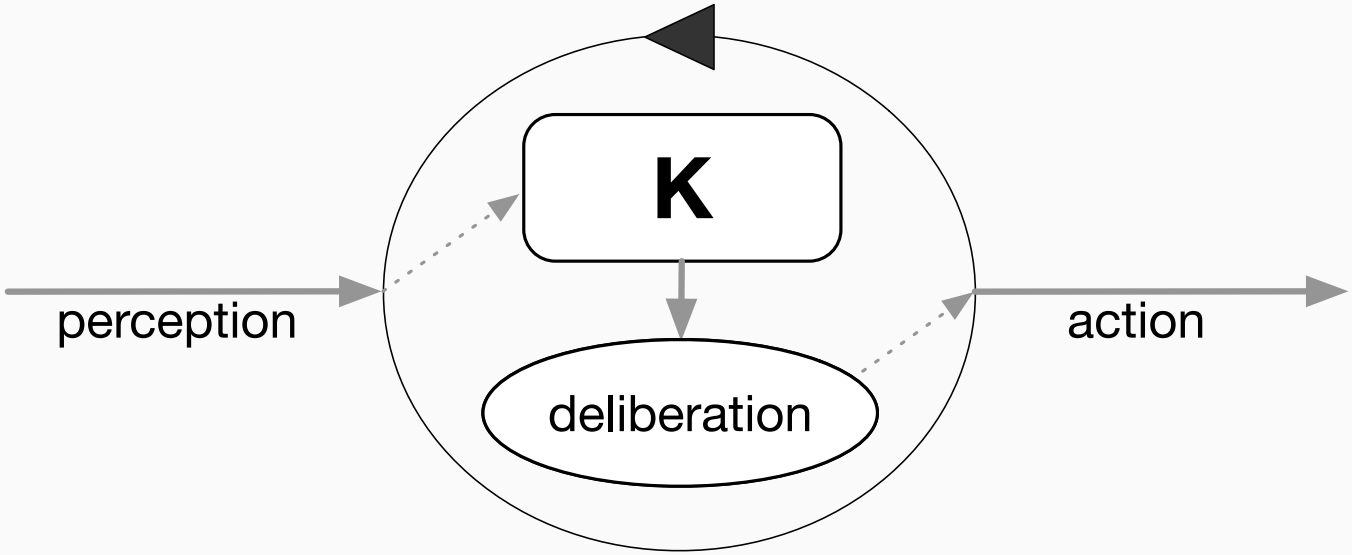
PósAutomação — UFSC

Agent

Agent



Agent



[reasoning cycle]

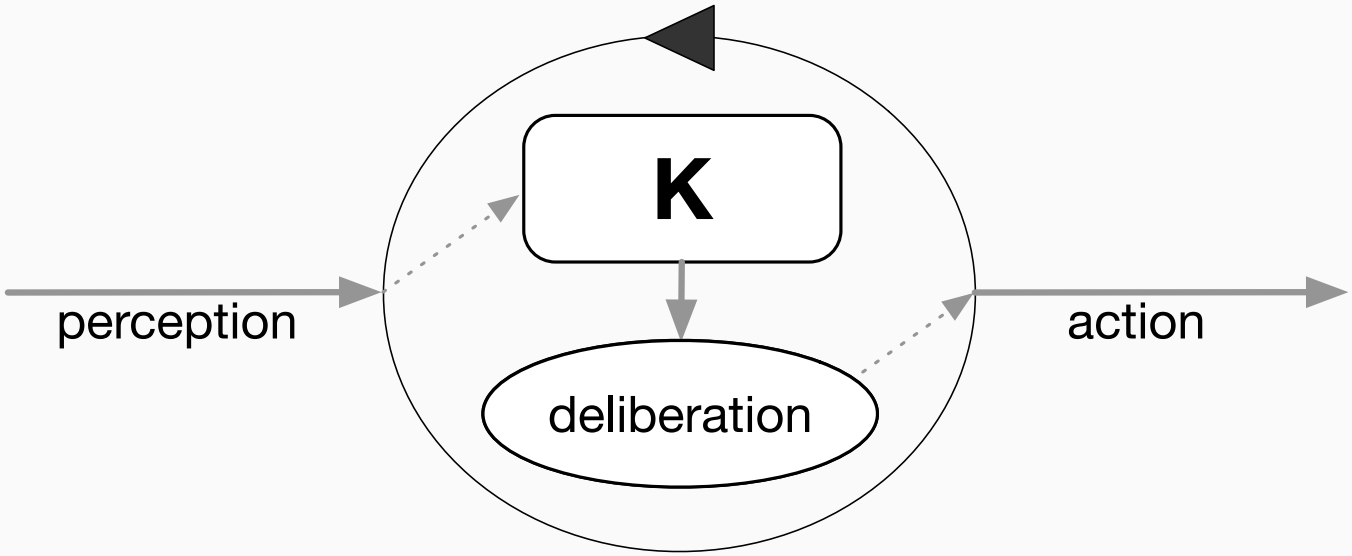
while true do

$K \leftarrow K \pm perception()$

$A \leftarrow deliberation(K)$

$act(A)$

Agent



to **program** an agent is to define K

deliberation ⇝ **autonomy**

Agent Knowledge (in JaCaMo) —

Beliefs : information about the environment, other agents, itself, application, ...

```
temperature(20).
happy(bob).
```

Goals : the agent objectives

```
!temperature(20).
!happy(bob).
```

Plans :

Agent Knowledge (in JaCaMo) —

Beliefs : information about the environment, other agents, itself, application, ...

```
temperature(20).
happy(bob).
```

Goals : the agent objectives

```
!temperature(20).
!happy(bob).
```

Plans :

Agent Knowledge (in JaCaMo) —

Beliefs : information about the environment, other agents, itself, application, ...

```
temperature(20).
happy(bob).
```

Goals : the agent objectives

```
!temperature(20).
!happy(bob).
```

Plans : specifies how goals can be **achieved** by **actions**

```
+!temperature(20) <- startCooling.
+!happy(bob) <- kiss(bob).
```

Agent Knowledge (in JaCaMo) —

Beliefs : information about the environment, other agents, itself, application, ...

```
temperature(20).
happy(bob).
```

Goals : the agent objectives

```
!temperature(20).
!happy(bob).
```

Plans : specifies how goals can be **achieved** by **actions**

```
+!temperature(20) <- startCooling.
+!happy(bob) <- kiss(bob).
```

specifies **reactions** to mental state changes

```
+temperature(10) <- !temperature(20).
-happy(bob) <- !happy(bob).
```

Agent Knowledge (in JaCaMo) — $K = B + G + P$

Beliefs : information about the environment, other agents, itself, application, ...

```
temperature(20).
happy(bob).
```

Goals : the agent objectives

```
!temperature(20).
!happy(bob).
```

Plans : specifies how goals can be **achieved** by **actions**

```
+!temperature(20) <- startCooling.
+!happy(bob) <- kiss(bob).
```

specifies **reactions** to mental state changes

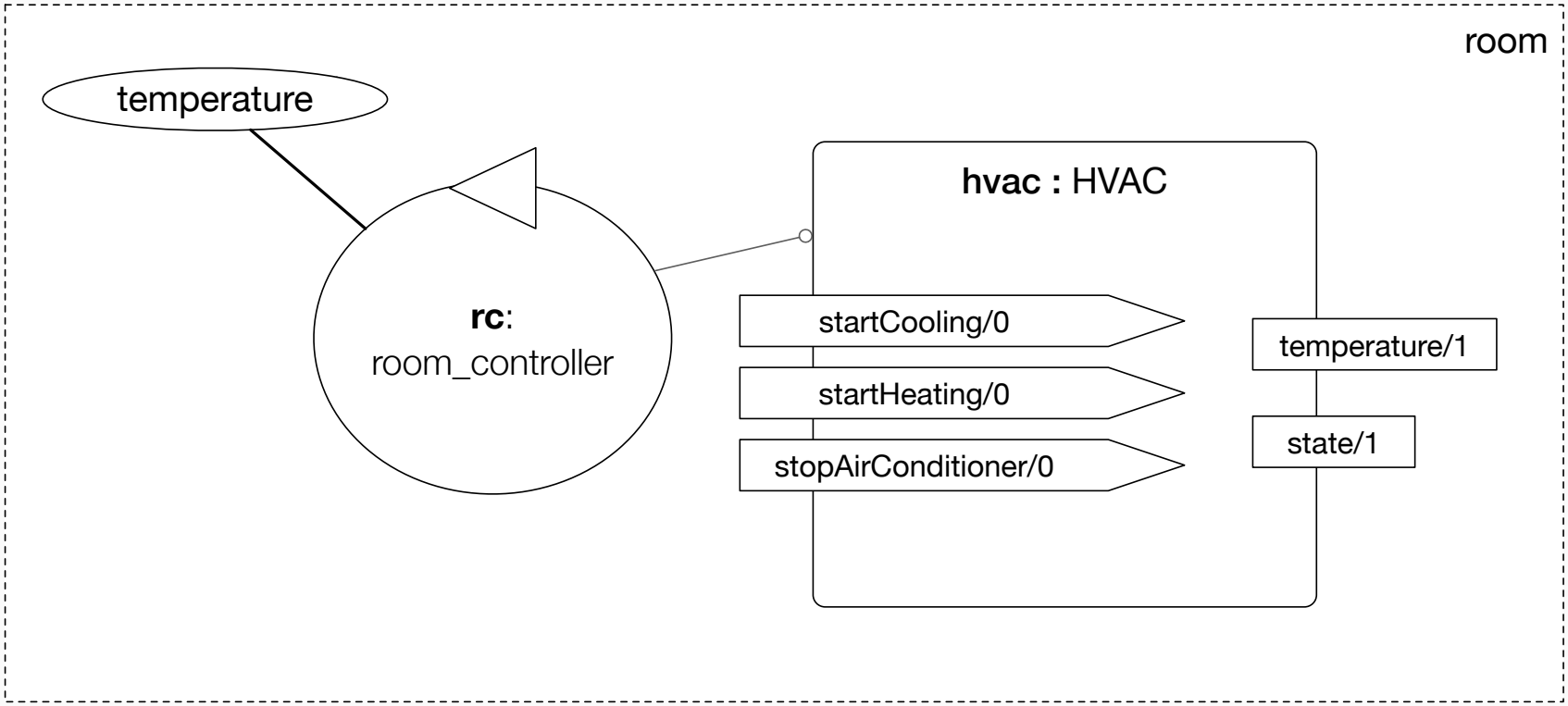
```
+temperature(10) <- !temperature(20).
-happy(bob) <- !happy(bob).
```

Knowledge Sources

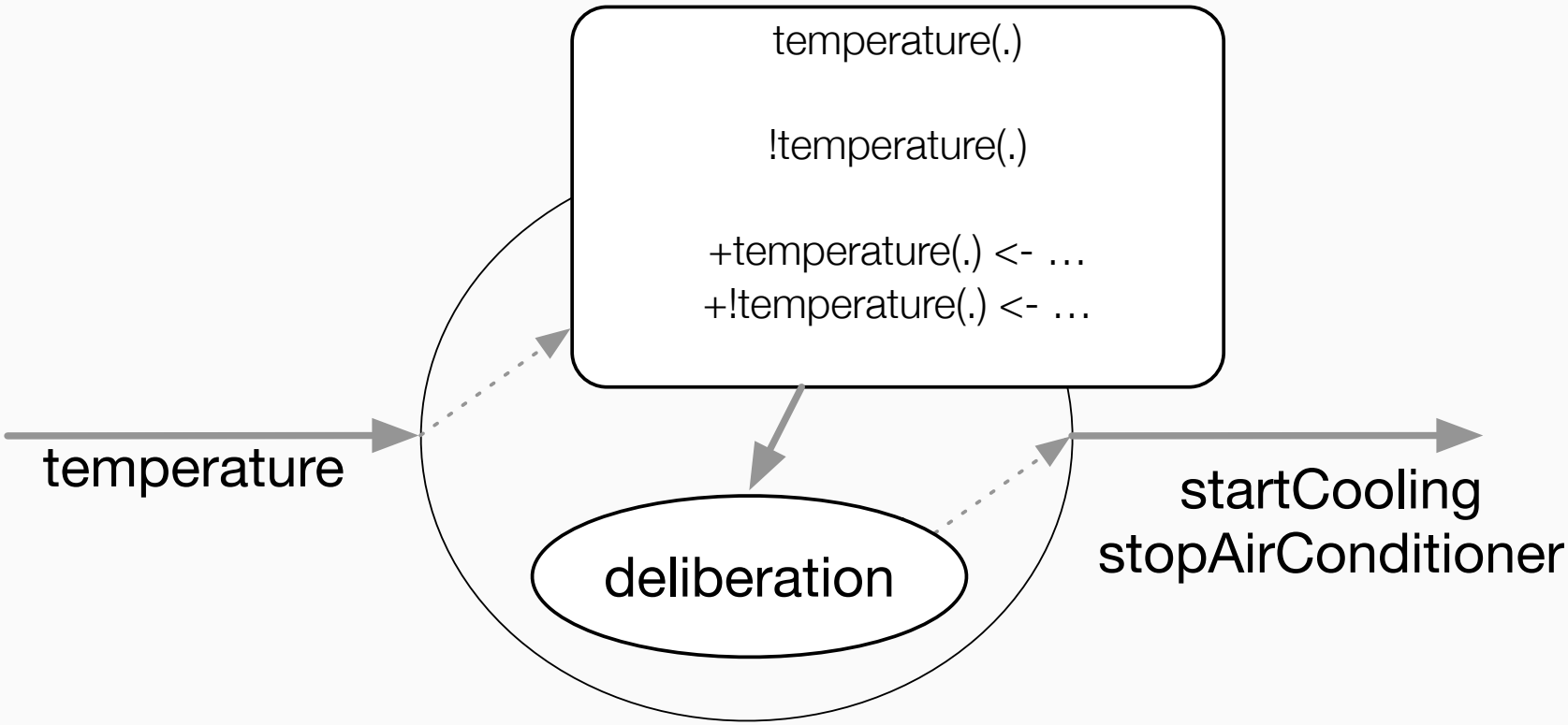
Beliefs, goals, and plans are provided by

- perception: in the case of beliefs
- developers: initial mental state of the agent
- other agents: by communication
- the agent itself: by reasoning or learning

Smart Room Scenario — initial implementation



Agent Programming (in JaCaMo)



Agent Programming (in JaCaMo)

```
+temperature(30)  <- !temperature(20).
+!temperature(20) <- startCooling.
```

Agent Programming (in JaCaMo)

```
+temperature(30)  <- !temperature(20).
+temperature(20)  <- stopAirConditioner.

+!temperature(20) <- startCooling.
```

Agent Programming (in JaCaMo)

```
// initial belief, given by the developer
preference(20).

// reaction to changes in the temperature
+temperature(T)  : preference(P) & math.abs(P-T) > 2
   <- !temperature(P).
+temperature(T)  : preference(T)
   <- stopAirConditioner.

// plans to achieve some temperature
+!temperature(P) : temperature(T) & T >  P
   <- startCooling.
```

Agent Programming (in JaCaMo)

```
// initial belief, given by the developer
preference(20).

// initial goal, given by the developer
!keep_temperature.

// maintenance the goal pattern
+!keep_temperature
    : temperature(T) & preference(P) & T >  P
   <- startCooling;
      !keep_temperature.
+!keep_temperature
    : temperature(T) & preference(P) & T <= P
   <- stopAirConditioner;
      !keep_temperature.
```

Main Features

- **reactivity**: even when achieving some goals
- **pro-activity**: new goals can be created
- **long-term goals**: agents are committed to achieve goals
- **context awareness**: plans are selected based on the circumstances
- **transparency**: we can trace back the reasons for an action
- sound **theoretical background** for agent architectures:
  - practical reasoning [Bratman, 1987]
  - intentions [Cohen and Levesque, 1987]
  - BDI [Rao and Georgeff, 1995]
  - ...

Main Features

- **reactivity**: even when achieving some goals
- **pro-activity**: new goals can be created
- **long-term goals**: agents are committed to achieve goals
- **context awareness**: plans are selected based on the circumstances
- **transparency**: we can trace back the reasons for an action
- sound **theoretical background** for agent architectures:
  - practical reasoning [Bratman, 1987]
  - intentions [Cohen and Levesque, 1987]
  - BDI [Rao and Georgeff, 1995]
  - ...

Main Features

- **reactivity**: even when achieving some goals
- **pro-activity**: new goals can be created
- **long-term goals**: agents are committed to achieve goals
- **context awareness**: plans are selected based on the circumstances
- **transparency**: we can trace back the reasons for an action
- sound **theoretical background** for agent architectures:
  - practical reasoning [Bratman, 1987]
  - intentions [Cohen and Levesque, 1987]
  - BDI [Rao and Georgeff, 1995]
  - ...

Main Features

- **reactivity**: even when achieving some goals
- **pro-activity**: new goals can be created
- **long-term goals**: agents are committed to achieve goals
- **context awareness**: plans are selected based on the circumstances
- **transparency**: we can trace back the reasons for an action
- sound **theoretical background** for agent architectures:
  - practical reasoning [Bratman, 1987]
  - intentions [Cohen and Levesque, 1987]
  - BDI [Rao and Georgeff, 1995]
  - ...

Main Features

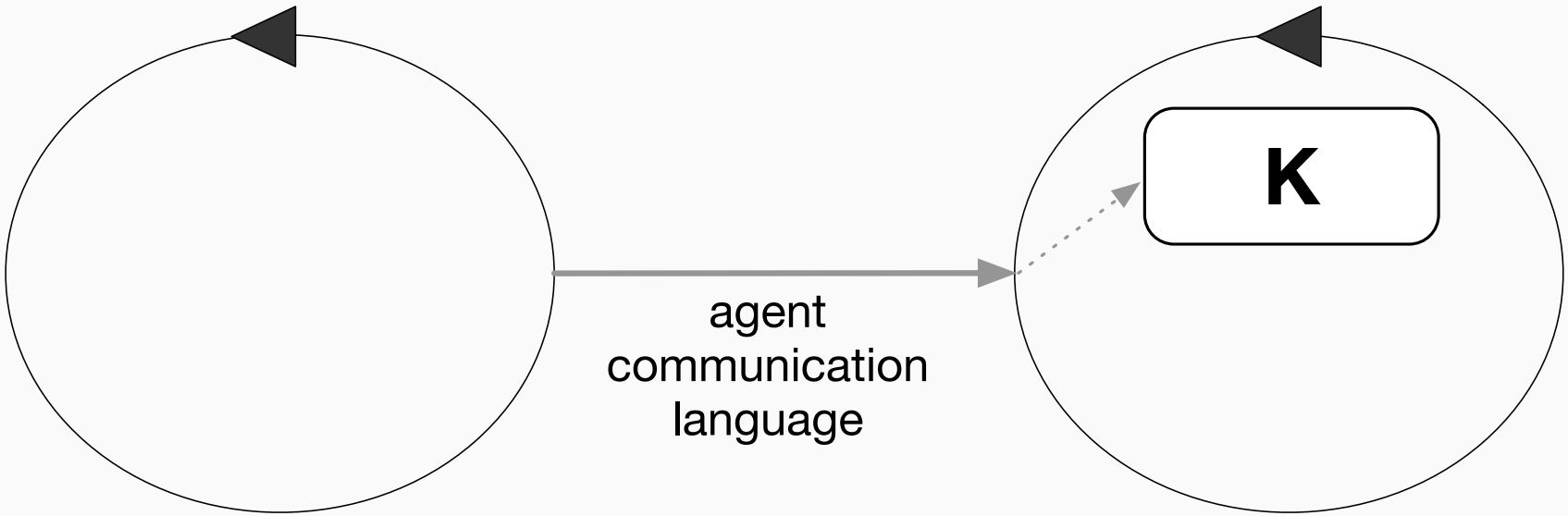
- **reactivity**: even when achieving some goals
- **pro-activity**: new goals can be created
- **long-term goals**: agents are committed to achieve goals
- **context awareness**: plans are selected based on the circumstances
- **transparency**: we can trace back the reasons for an action
- sound **theoretical background** for agent architectures:
  - practical reasoning [Bratman, 1987]
  - intentions [Cohen and Levesque, 1987]
  - BDI [Rao and Georgeff, 1995]
  - ...

Main Features

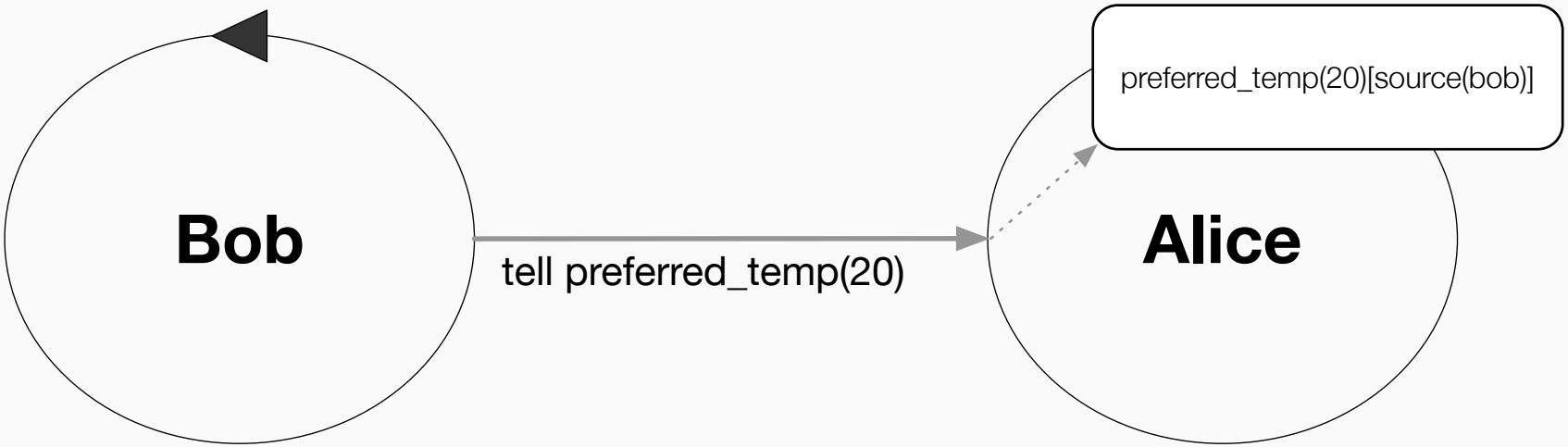
- **reactivity**: even when achieving some goals
- **pro-activity**: new goals can be created
- **long-term goals**: agents are committed to achieve goals
- **context awareness**: plans are selected based on the circumstances
- **transparency**: we can trace back the reasons for an action
- sound **theoretical background** for agent architectures:
  - practical reasoning [Bratman, 1987]
  - intentions [Cohen and Levesque, 1987]
  - BDI [Rao and Georgeff, 1995]
  - ...

Agent Interaction (communication)

Agent–Agent Communication



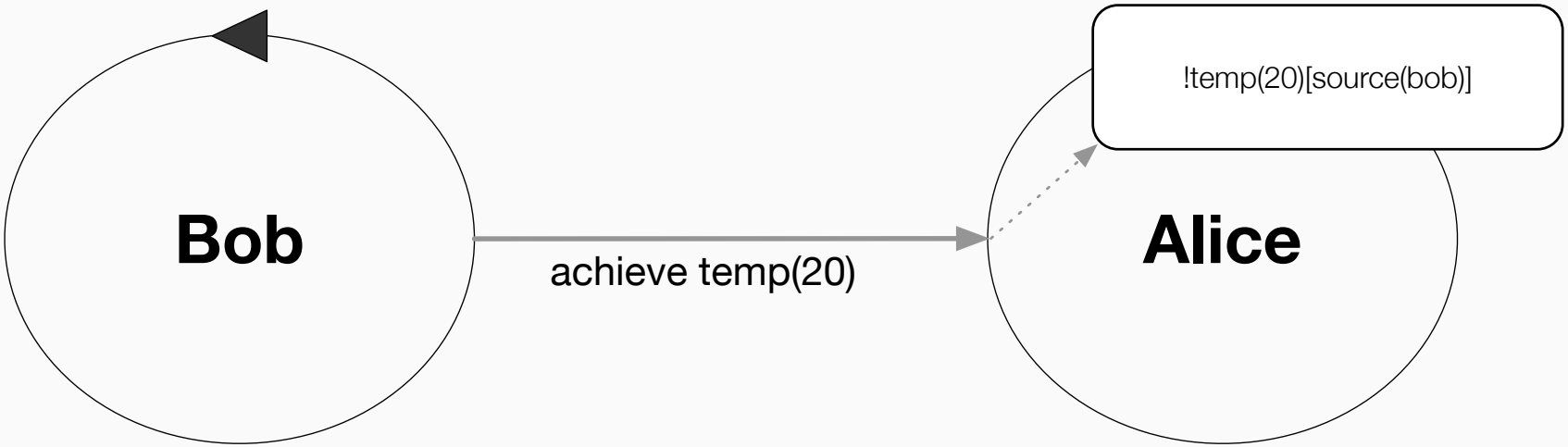
Semantic of messages



A message has:

- an intention (tell, ask, achieve, ...)
- a content (belief, goal, plan)

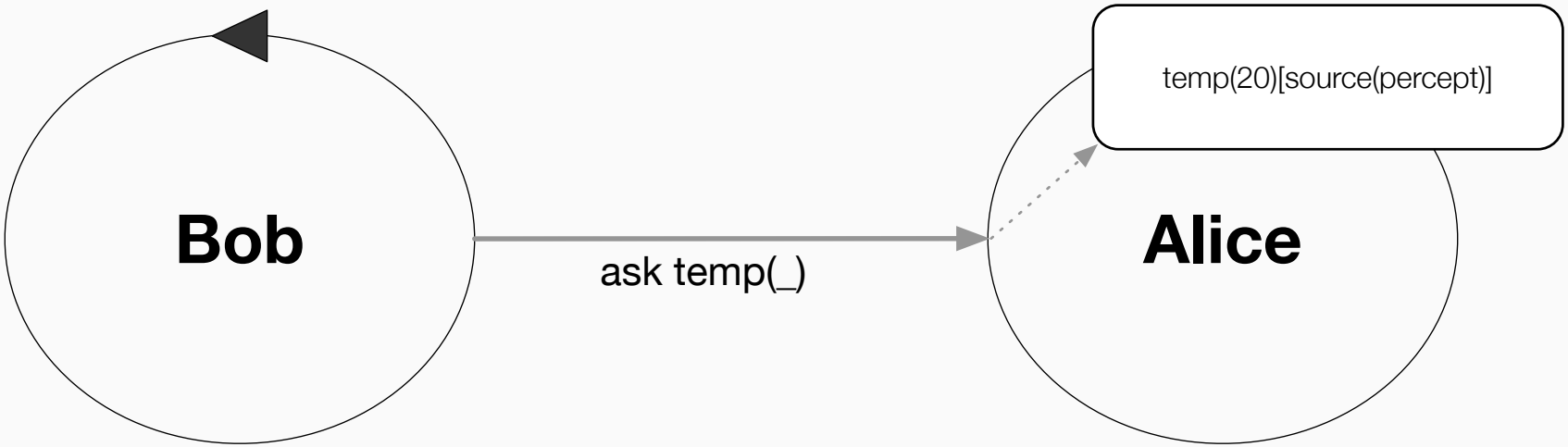
Semantic of messages



A message has:

- an intention (tell, ask, achieve, ...)
- a content (belief, goal, plan)

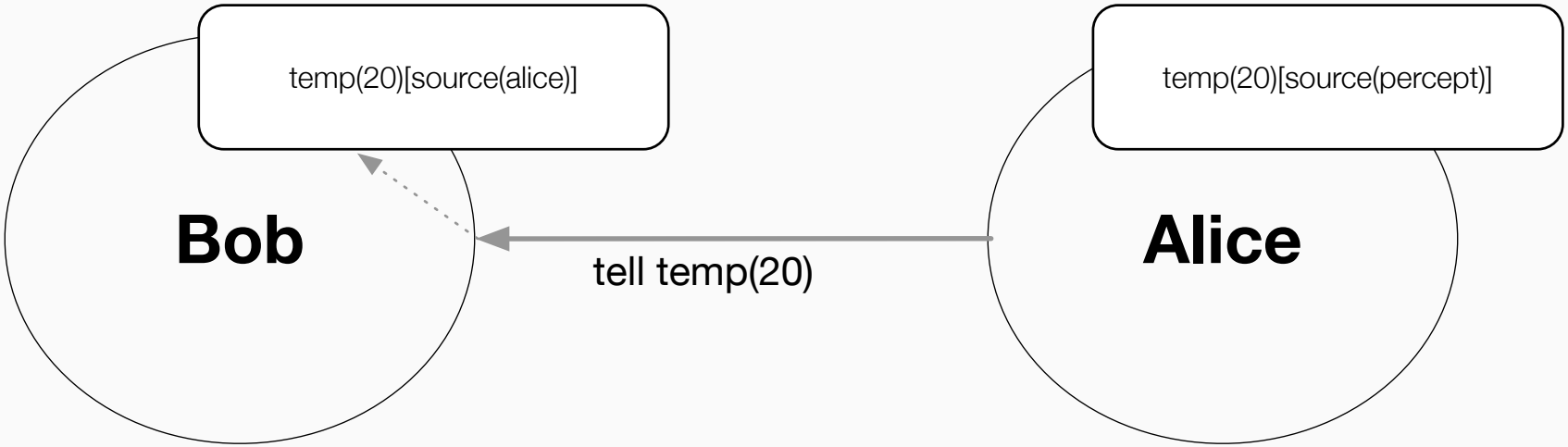
Semantic of messages



A message has:

- an intention (tell, ask, achieve, ...)
- a content (belief, goal, plan)

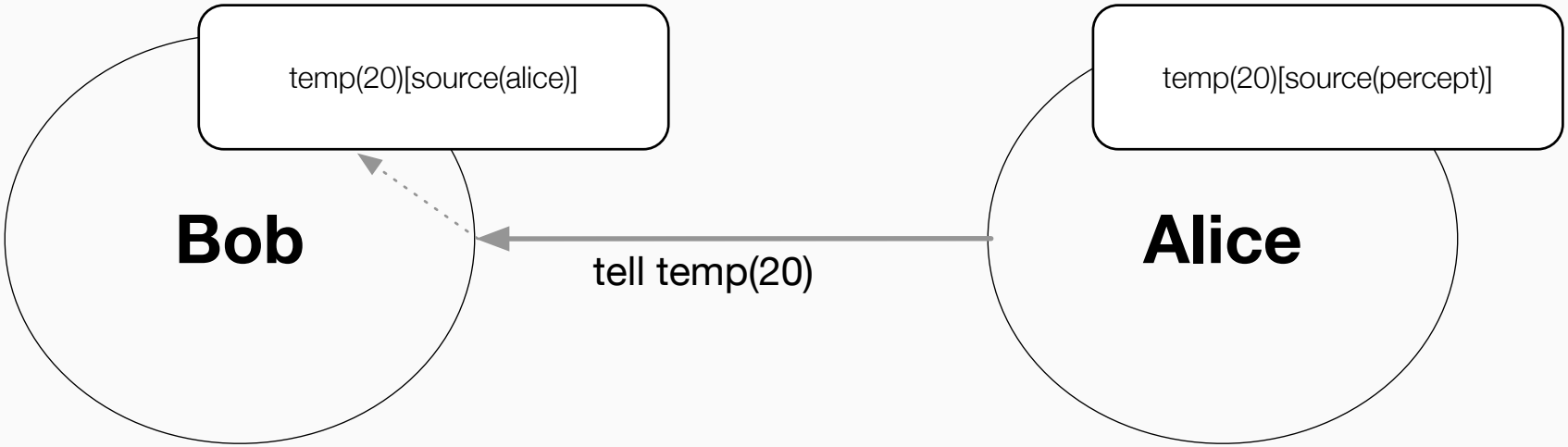
Semantic of messages



A message has:

- an intention (tell, ask, achieve, ...)
- a content (belief, goal, plan)

Semantic of messages



- we are not programming computers,
  we are programming agents, which are based on knowledge
- communication is not about data exchange, but
  knowledge sharing

JaCaMo implementation

Sender: `.send(bob,tell,happy(alice))`

- receiver: agent unique name
- performative: tell, achieve, askOne, askHow, ...
- content: a literal

Receiver

- nothing is needed

Properties

- distributed & support for decentralized
- (usually) asynchronous
- KQML vs FIPA-ACL
- not reduced to method invocation

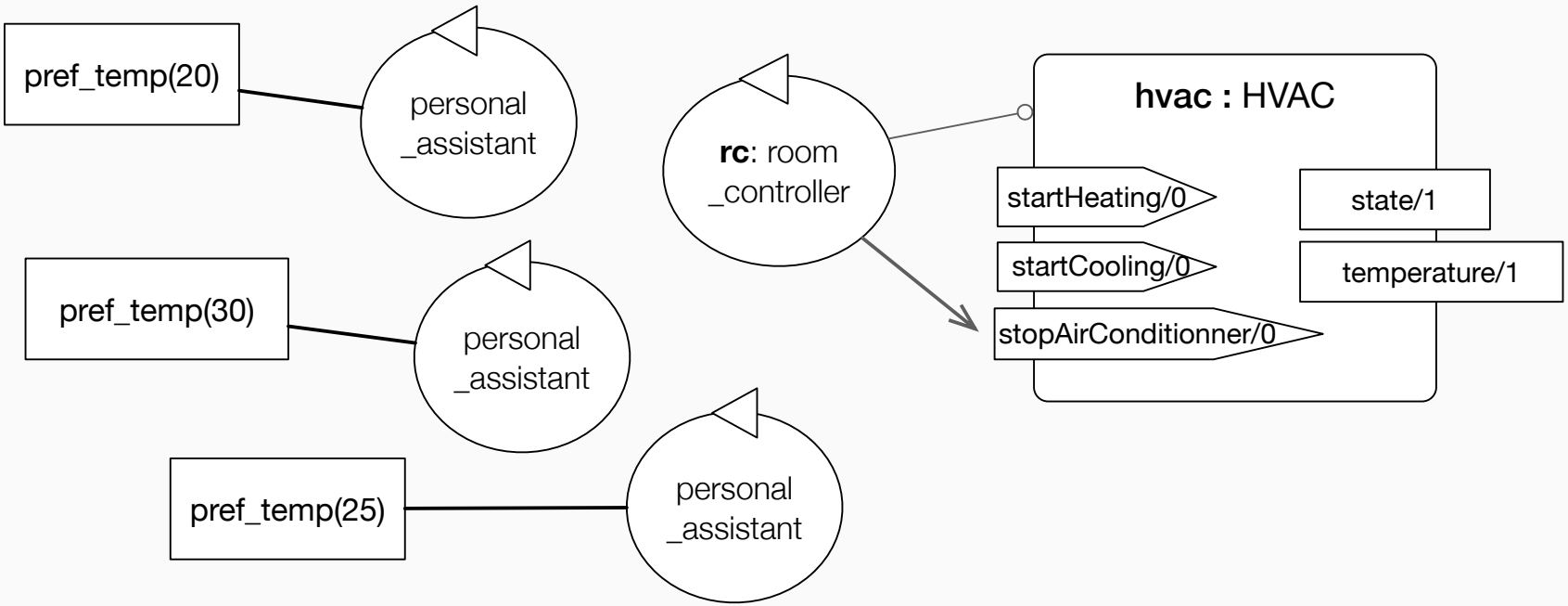
JaCaMo Performatives

- **tell** and untell: change beliefs of receiver
- **achieve** and unachieve: change goals of receiver
- **askOne** and askAll: ask for beliefs of the receiver
- **askHow**, tellHow, and untellHow: exchange plans with other agent
- **signal**: add an event in the receiver

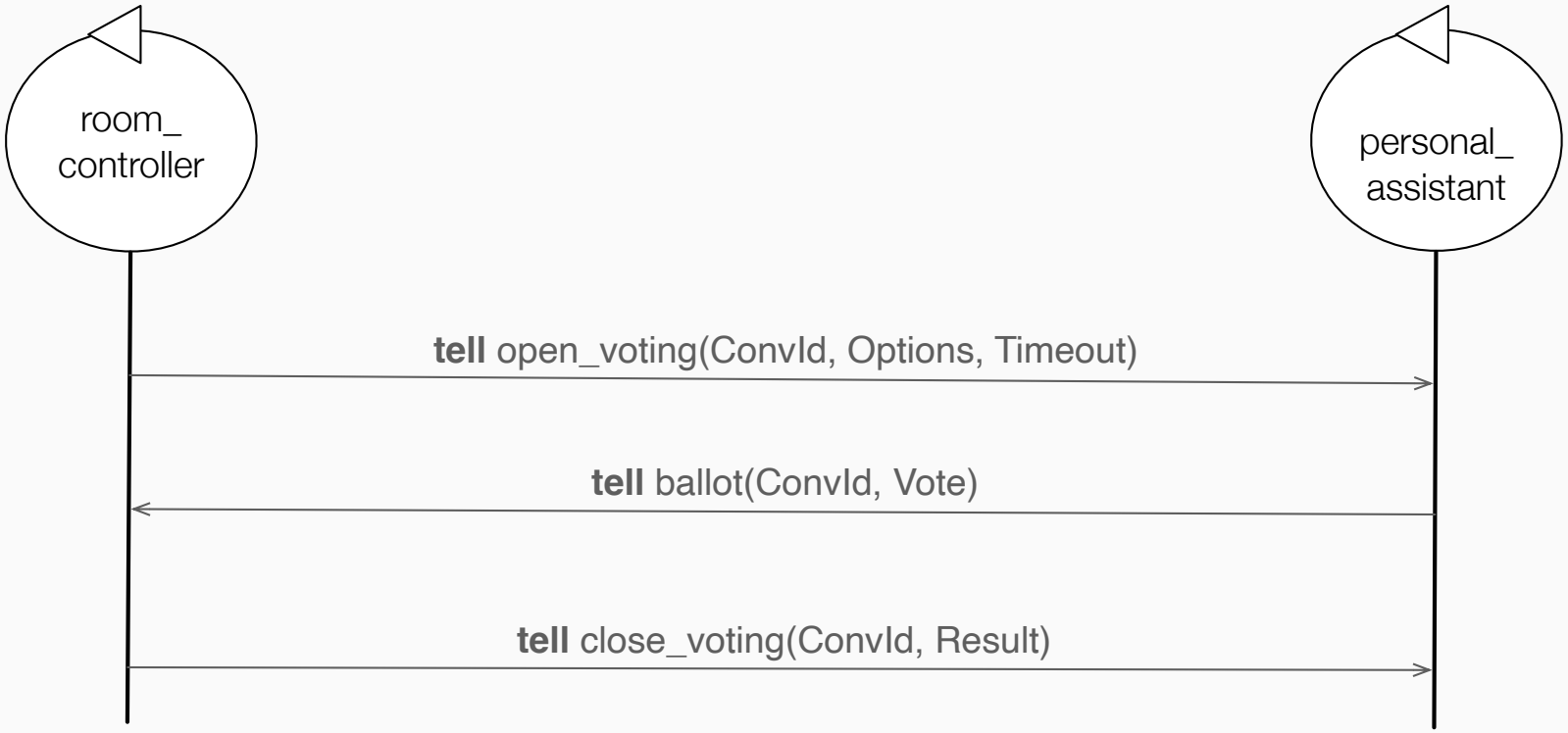
Smart Room Scenario

many users

The system have to consider the preference of temperature of many users and use a voting strategy to define the target temperature

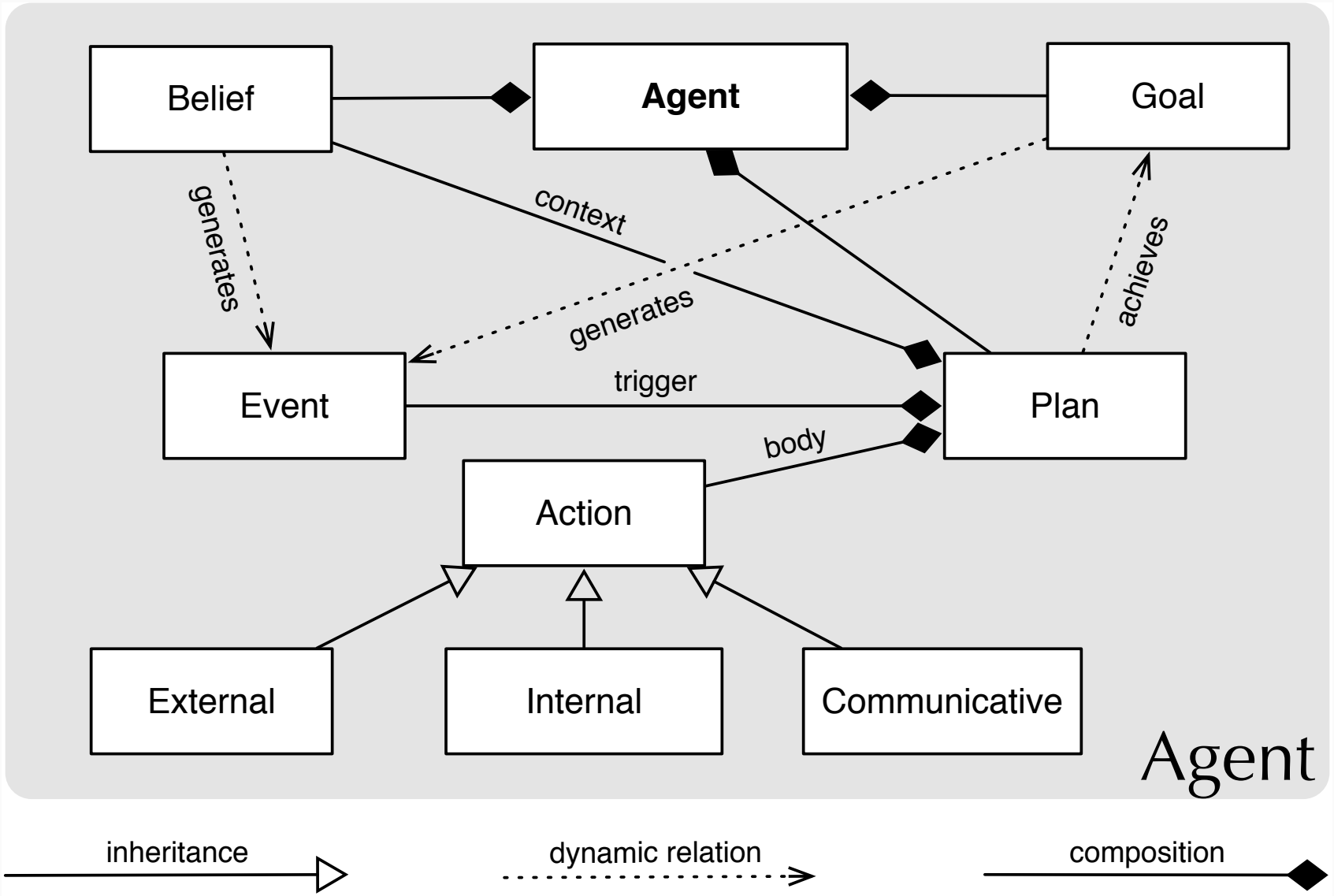


Interaction Protocols ⇝ *coordination*



Protocol Implementation

Wrap-up: Agent Model



Wrap-up: Agent Programming

- **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

Fundamentals

Literature

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

Proceedings: EMAS, ProMAS, DALT, LADS, 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], ASTRA, SARL

But many others languages and platforms...

Some Languages and Platforms

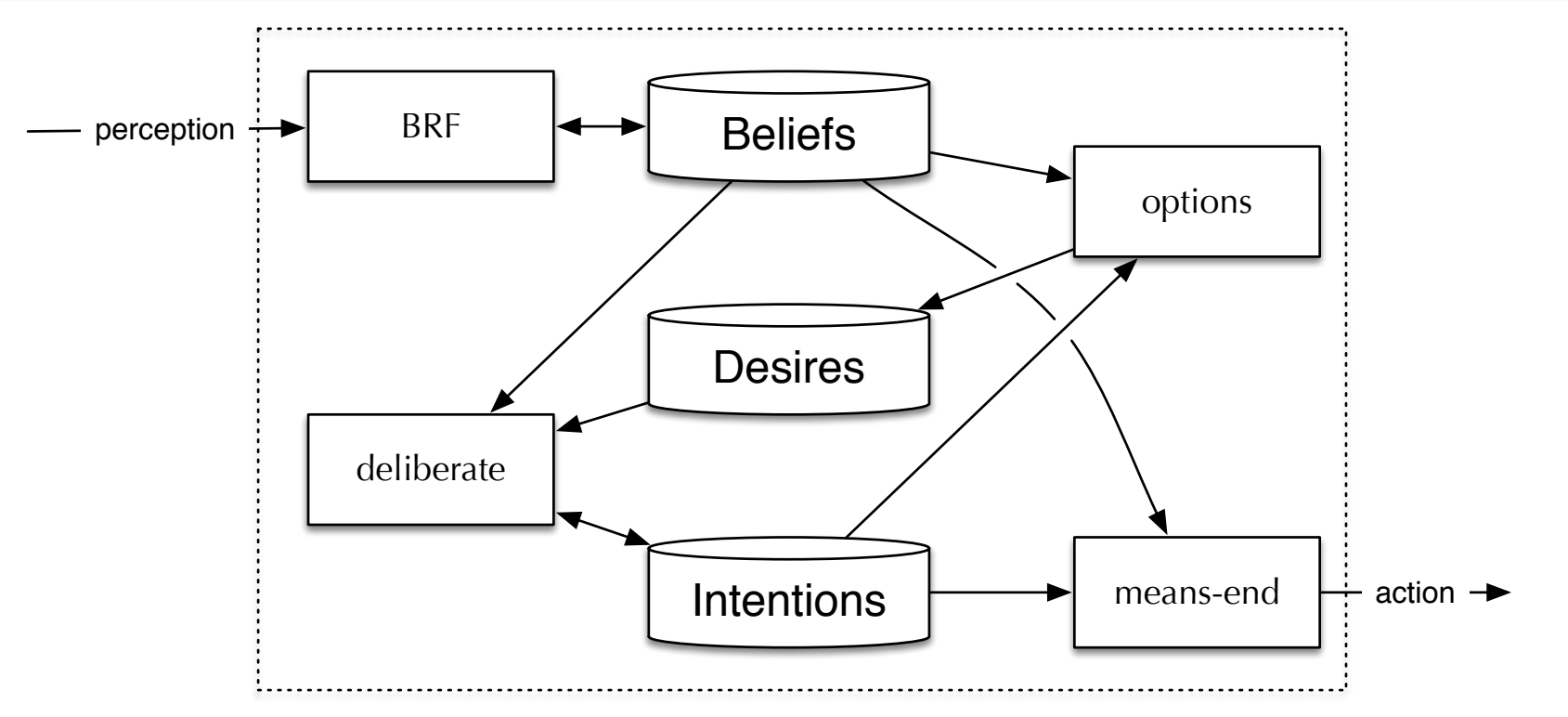
Jason (Hübner, Bordini, …); 3APL and 2APL (Dastani, van Riemsdijk, Meyer, Hindriks, …); Jadex (Braubach, Pokahr); MetateM (Fisher, Guidini, Hirsch, …); ConGoLog (Lesperance, Levesque, … / Boutilier – DTGolog); Teamcore/ MTDP (Milind Tambe, …); IMPACT (Subrahmanian, Kraus, Dix, Eiter); CLAIM (Amal El 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); ASTRA (Rem Collier); SARL (Stephane Galland, Sebastian Rodriguez); *simpAL*, *ALOO* (Ricci, …);

• • •

Agent Oriented Programming — Inspiration

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

BDI architecture



BDI reasoning cycle [Wooldridge, 2009]

while true **do**

$B \leftarrow brf(B, perception())$ // belief revision

$D \leftarrow options(B, I)$ // desire revision

$I \leftarrow deliberate(B, D, I)$ // get intentions

$\pi \leftarrow meansend(B, I, A)$ // gets a plan

while $\pi \neq \emptyset$ **do**

execute($head(\pi)$)

$\pi \leftarrow tail(\pi)$

BDI reasoning cycle [Wooldridge, 2009]

while true **do**

$B \leftarrow brf(B, perception())$ // belief revision

$D \leftarrow options(B, I)$ // desire revision

$I \leftarrow deliberate(B, D, I)$ // get intentions

$\pi \leftarrow meansend(B, I, A)$ // gets a plan

while $\pi \neq \emptyset$ **do**

execute($head(\pi)$)

$\pi \leftarrow tail(\pi)$

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

BDI reasoning cycle [Wooldridge, 2009]

while true **do**

$B \leftarrow brf(B, perception())$ // belief revision

$D \leftarrow options(B, I)$ // desire revision

$I \leftarrow deliberate(B, D, I)$ // get intentions

$\pi \leftarrow meansend(B, I, A)$ // gets a plan

while $\pi \neq \emptyset$ **do**

execute($head(\pi)$)

$\pi \leftarrow tail(\pi)$

$B \leftarrow brf(B, perception())$

if $\neg sound(\pi, I, B)$ **then**

$\pi \leftarrow meansend(B, I, A)$

revise commitment to plan – re-planning for context adaptation

BDI reasoning cycle [Wooldridge, 2009]

while true **do**

$B \leftarrow brf(B, perception())$ // belief revision

$D \leftarrow options(B, I)$ // desire revision

$I \leftarrow deliberate(B, D, I)$ // get intentions

$\pi \leftarrow meansend(B, I, A)$ // gets a plan

while $\pi \neq \emptyset$ **and** $\neg succeeded(I, B)$ **and** $\neg impossible(I, B)$ **do**

execute($head(\pi)$)

$\pi \leftarrow tail(\pi)$

$B \leftarrow brf(B, perception())$

if $\neg sound(\pi, I, B)$ **then**

$\pi \leftarrow meansend(B, I, A)$

revise commitment to intentions – Single-Minded Commitment

BDI reasoning cycle [Wooldridge, 2009]

while true **do**

- $B \leftarrow brf(B, perception())$ `// belief revision`
- $D \leftarrow options(B, I)$ `// desire revision`
- $I \leftarrow deliberate(B, D, I)$ `// get intentions`
- $\pi \leftarrow meansend(B, I, A)$ `// gets a plan`
- **while** $\pi \neq \emptyset$ **and** $\neg succeeded(I, B)$ **and** $\neg impossible(I, B)$ **do**
  - execute($head(\pi)$)
  - $\pi \leftarrow tail(\pi)$
  - $B \leftarrow brf(B, perception())$
  - **if** $reconsider(I, B)$ **then**
    - $D \leftarrow options(B, I)$
    - $I \leftarrow deliberation(B, D, I)$
  - **if** $\neg sound(\pi, I, B)$ **then**
    - $\pi \leftarrow meansend(B, I, A)$

reconsider the intentions (not always!)

Intentions

- Intentions pose problems for the agents: they need to determine a way to achieve them
  (planning and acting)
- Intentions provide a “screen of admissibility” for adopting new intentions
- Agents keep tracking their success of attempting to achieve their intentions
- Agents should not spend all their time revising intentions
  (losing pro-activity and reactivity)

(BDI & Jason) Hello World – agent bob

```
happy(bob).                                    // B

!say(hello).                                   // D

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

(BDI & Jason) Hello World – agent bob

beliefs

- prolog like (FOL)

```
happy(bob).                                    // B

!say(hello).                                   // D

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

(BDI & Jason) Hello World – agent bob

desires

- prolog like
- with ! prefix

```
happy(bob).                                    // B

!say(hello).                                   // D

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

(BDI & Jason) Hello World – agent bob

```
happy(bob).

!say(hello).

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

plans

- define when a desire becomes an intention
  ⇝ **deliberate**
- how it is satisfied
- are used for practical reasoning
  ⇝ **means-end**

BDI Hello World — desires from perception (*options*)

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

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

BDI 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).
```

BDI 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).
```

BDI 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).
```

BDI 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)).
```

BDI Hello World — intention revision

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

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

+!say(X) : not today(mond

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

features

- we can have several intentions based on the same plans
  ⇝ running concurrently
- long term goals running
  ⇝ reaction meanwhile
  ⇝ not overcommitted
- plan selection based on circumstance
- sequence of actions (partially) computed by the interpreter
  ⇝ programmer **declares** plans

Jason

AgentSpeak: The foundational language for *Jason*

- Programming language for BDI agents
- Originally proposed by Rao [Rao, 1996]
- 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 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

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 unifies 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 — 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

Reasoning Cycle

Runtime Structures for the Reasoning Cycle

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

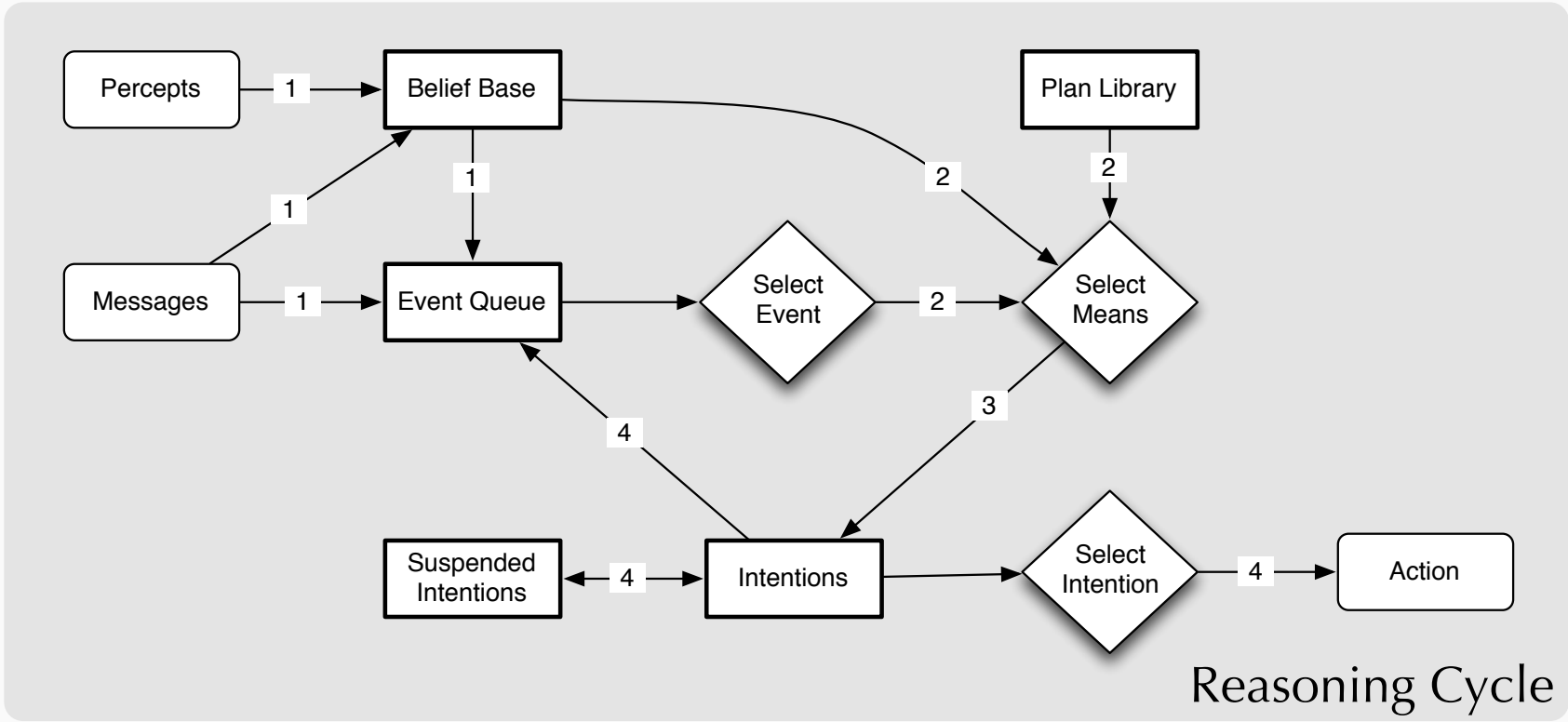
Goals: represent states of affairs the agent wants to bring about

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

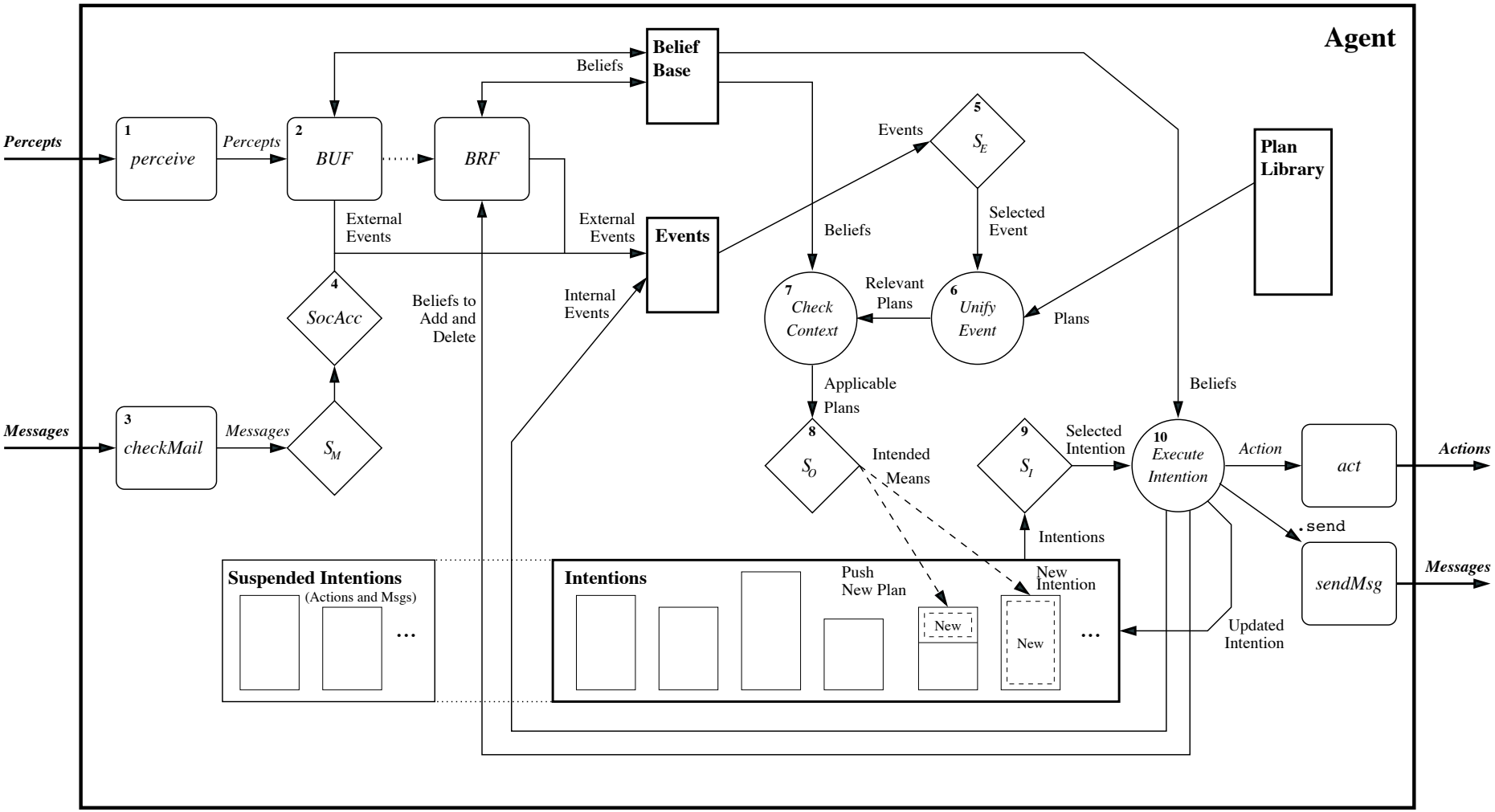
Events: happen as consequence to changes in the agent's beliefs or goals

Intentions: plans instantiated to achieve some goal

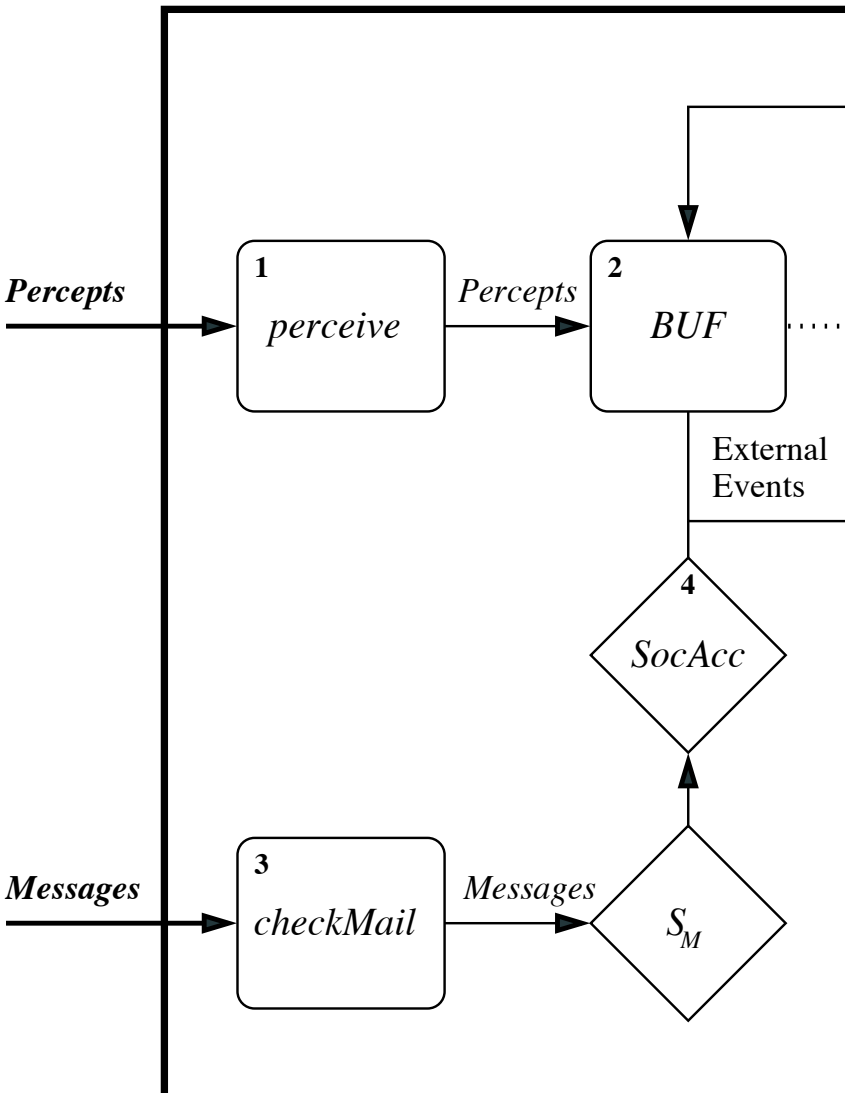
Basic Reasoning Cycle — runtime interpreter



Jason Reasoning Cycle

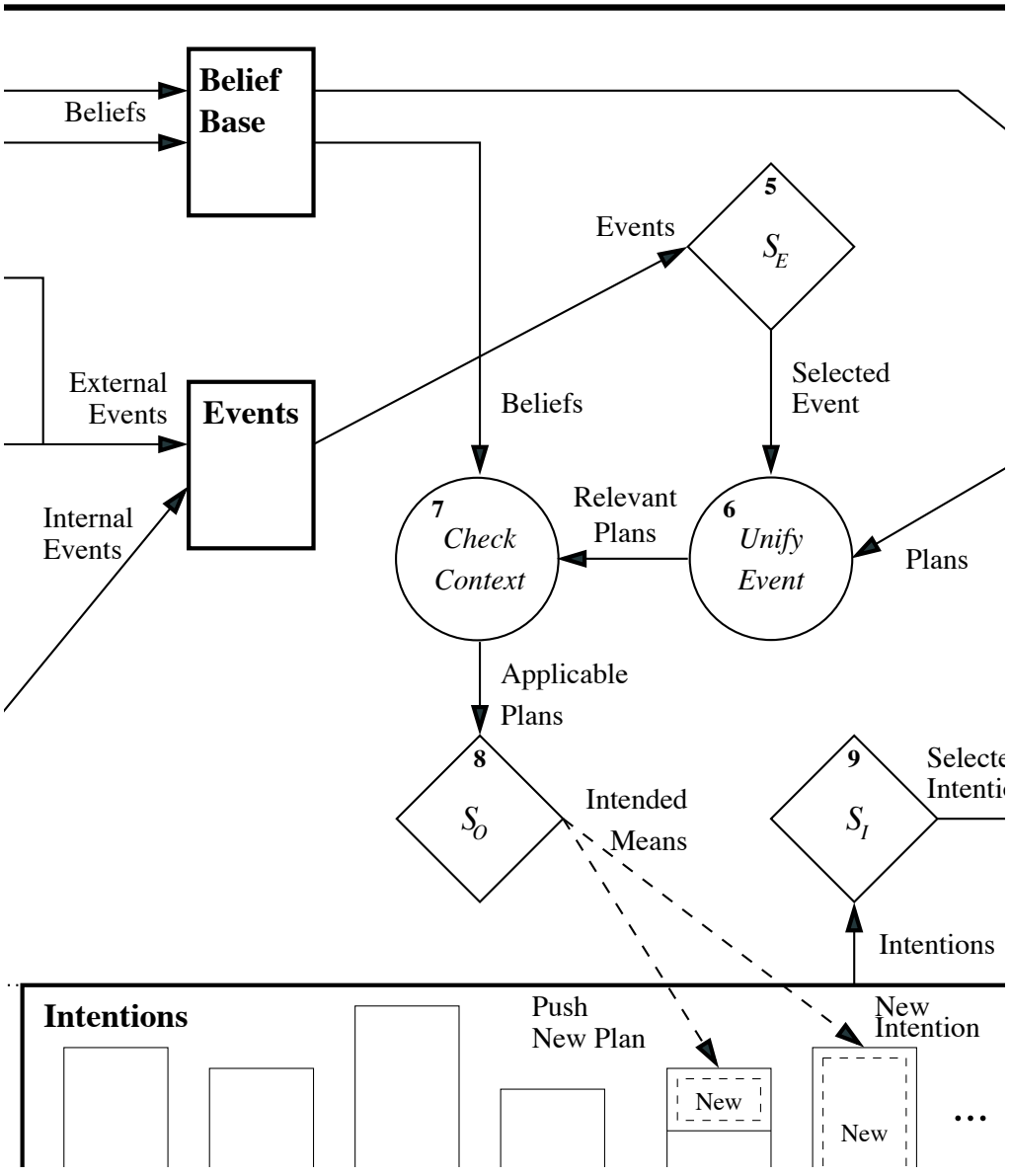


Jason Reasoning Cycle



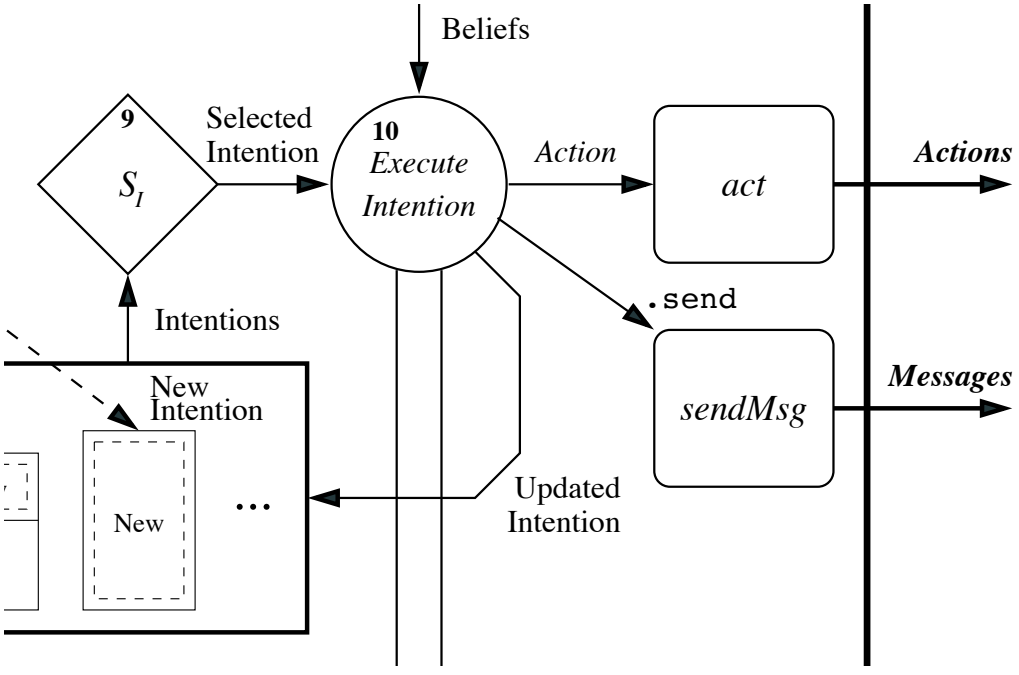
- machine perception
- belief revison
- knowledge representation
- communication, argumentation
- trust
- social power

Jason Reasoning Cycle



- planning
- reasoning
- decision theoretic techniques
- learning (reinforcement)

Jason Reasoning Cycle



- intention reconsideration
- scheduling
- action theories

Other Features

Failure Handling: Contingency Plans

Example (an agent blindly committed to g)

```
+!g : g.    // g is a declarative goal

+!g : ... <- a1; ?g.
+!g : ... <- a2; ?g.
+!g : ... <- a3; ?g.

+!g <- !g. // keep trying
-!g <- !g. // in case of some failure

+g <-.succeed_goal(g).
```

Failure Handling: Contingency Plans

Example (single minded commitment)

```
+!g : g.    // g is a declarative goal

+!g : ... <- a1; ?g.
+!g : ... <- a2; ?g.
+!g : ... <- a3; ?g.

+!g <- !g. // keep trying
-!g <- !g. // in case of some failure

+g <-.succeed_goal(g).

+f : .super_goal(g,SG) <-.fail_goal(SG).
```

f is the drop condition for goal g

Compiler pre-processing – directives

Example (single minded commitment)

```
{ begin smc(g,f) }
      +!g : ... <- a1.
      +!g : ... <- a2.
      +!g : ... <- a3.
{ end }
```

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

Internal Actions

- Unlike actions, internal actions do not change the environment
- They are 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

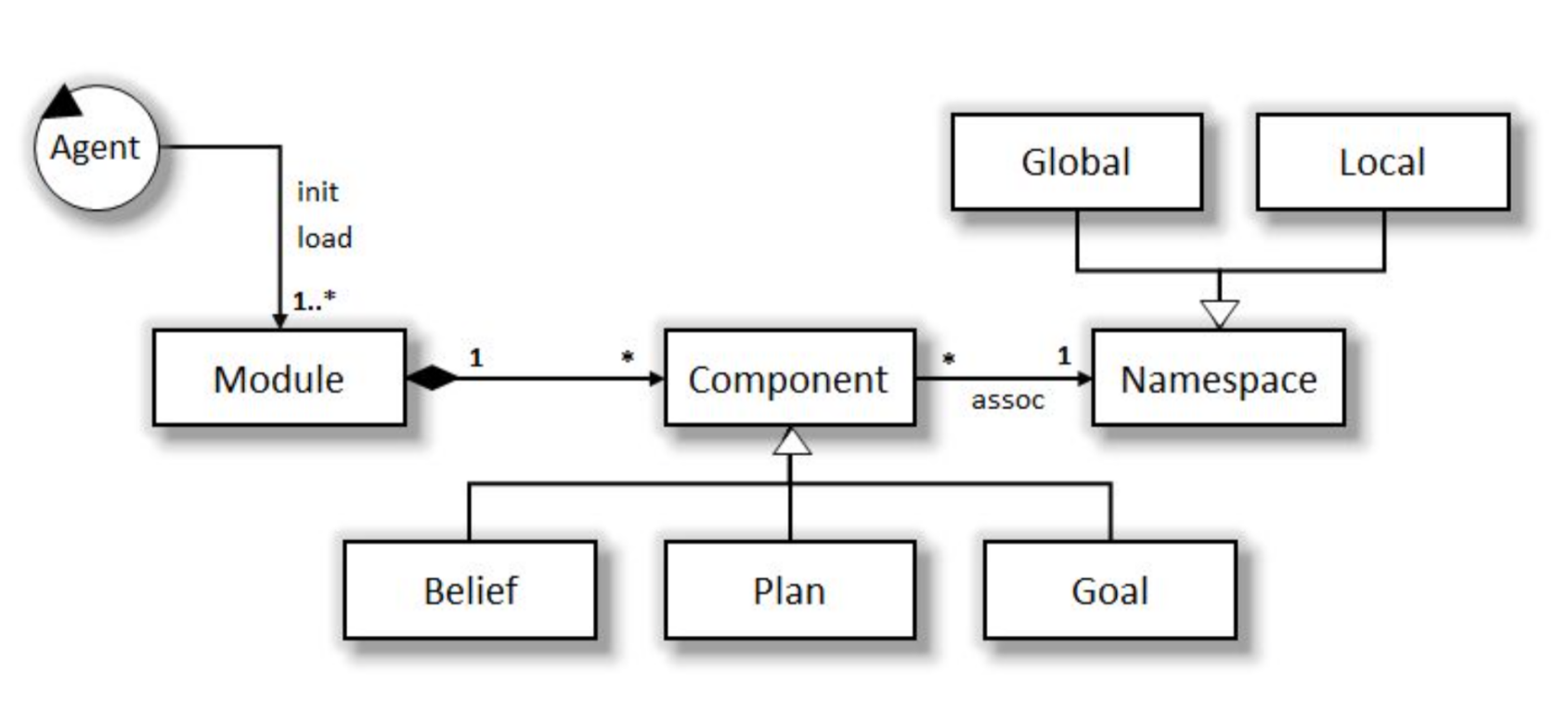
- Internal actions can be defined by the user in Java

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

Standard Internal Actions

- Standard (pre-defined) internal actions have an empty library name
  - `.print(`$term_1, term_2, \ldots$`)`
  - `.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.

Namespaces & Modularity



Namespaces & Modularity

```
{include("initiator.asl", pc)}
{include("initiator.asl", tv)}

!pc::startCNP(fix(pc)).
!tv::startCNP(fix(tv)).

+pc::winner(X)
   <- .print(X).
```

Inspection of agent **alice**

- Beliefs

tv::

introduction(participant)[source(compan

propose(*11.075337225252543*)[sourc

propose(*12.043311087442898*)[sourc

propose(*12.81277904935436*)[source

winner(company_A1)[source(self)].

#8priv::

state(finished)[source(self)].

pc::

introduction(participant)[source(compan

propose(*11.389500048463455*)[sourc

propose(*11.3925536837716 82*)[sourc

propose(*12.348901000262853*)[sourc

winner(company_A2)[source(self)].

Concurrent Plans

```
+!ga <- ...; !gb; ...
+!gb <- ...; !g1 |&| !g2; a1; ...

+!ga <- ...; !gb; ...
+!gb <- ...; !g1 ||| !g2; a1; ...

+!g <- x; (a;b) |&| (c;d) ||| (e;f); y.
```

Jason(ER) — motivation

```
+cfp(Id,Task)[source(A)] // answer to Call For Proposal
    : price(Task,Offer) & not my_offer(Task)
   <- +offered(Task);
      .send(A,tell,propose(Id,Offer)).
+cfp(Id,_)[source(A)]
   <- .send(A,tell,refuse(Id)).

+accept_proposal(Id) : my_offer(Task)
   <- !do(Task);
      -my_offer(Task).

+reject_proposal(Id) : my_offer(Task)
   <- -my_offer(Task).
```

Scope & sub-plans & goal conditions

```
+!g(X) : c <: gc <- a1; !g1.
{

   +e : c1 <- a2(X).
   +!g1 ....
}
```

Example of a participant in a CNP

```
+!participate_cnp <: false. {
   +cfp(Id,Task)[source(A)] // answer to Call For Proposal
       : price(Task,Offer) & not my_offer(Task,_)
      <: false
      <- +my_offer(Task, Offer); .send(A,tell,propose(Id,Of
      {
       +accept_proposal(Id) <- !do(Task); -my_offer(Task,_)

       +reject_proposal(Id) <- -my_offer(Task,_); .done.
      }
   +cfp(Id,_)[source(A)] <- .send(A,tell,refuse(Id)).

   +!do(T) <- ...
}
```

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, ...)

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

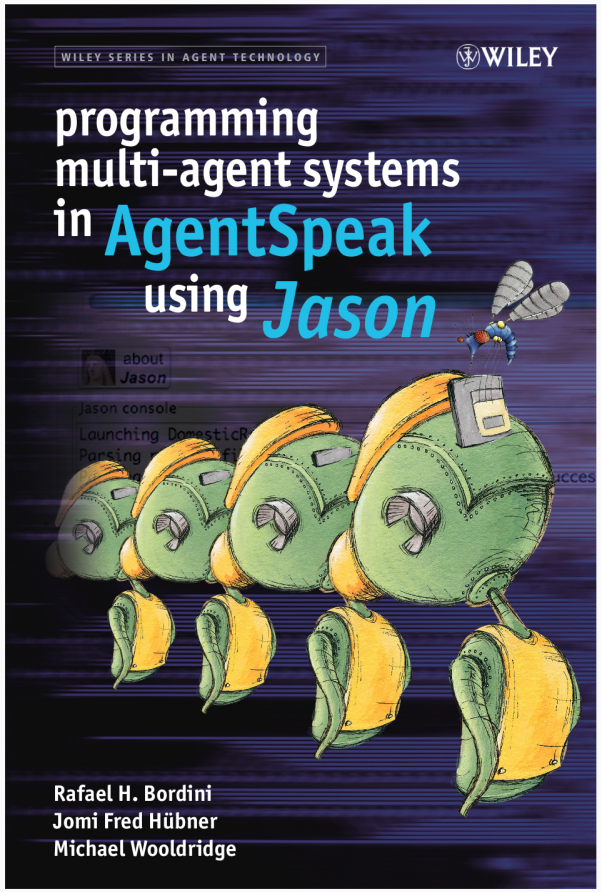
Jason × Prolog

- 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

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.



Bibliography i

Austin, J. L. (1975).

How to do things with Words.

Harvard University Press, Cambridge, 2 edition.

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.

Bibliography ii


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 iii


Bratman, M. E. (1987).
Intention, Plans, and Practical Reason.
Harvard University Press, Cambridge.


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


Cohen, P. R. and Levesque, H. J. (1987).
Intention = choice + commitment.
In *Proceedings of the 6th National Conference on Artificial Intelligence*, pages 410–415. Morgan Kaufmann.

Bibliography iv


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.

Bibliography v


Georgeff, M. P. and Lansky, A. L. (1987).
Reactive reasoning and planning.
In *Proc. of the Sixth National Conference on Artificial Intelligence (AAAI 87)*, pages 677–682.


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.


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

Bibliography vi

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.

Newell, A. (1982).

The knowledge level.

Artificial Intelligence, 18:87–127.

Pokahr, A., Braubach, L., and Lamersdorf, W. (2005).

Jadex: A bdi reasoning engine.

In [Bordini et al., 2005], pages 149–174.

Bibliography vii

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.

Rao, A. S. and Georgeff, M. P. (1995).

BDI agents: from theory to practice.

In Lesser, V., editor, *Proceedings of the First International Conference on MultiAgent Systems (ICMAS'95)*, pages 312–319. AAAI Pess.

Searle, J. R. and Vanderveken, D. (1985).

Foundations of illocutionary logic.

Cambridge University Press, Cambridge.

Bibliography viii


Shoham, Y. (1993).

Agent-oriented programming.

Artif. Intell., 60(1):51–92.


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.