

Agent Dimension

2024-11-11

Multi-Agent Oriented Programming Programming Multi-Agent Systems Using JaCaMo

Agent Dimension

PósAutomação — UFSC

© 0



2024-11-11

∟Agent

-prespine agent actor

The goal of this part is to introduce agent oriented programming So it is about programming and about agent What is an agent?

- it is not like a conventional program (that starts and ends)

- it is continuously running (like a server)

- continuously perceiving ("inputs") the environment (sensors, messages, user commands, ...)

- the "output" is continuous acting

- output is not data (as a procedural program) neither knowledge (as an inference engine)

- it changes a lot!

- it is about how to program to act instead of programming to change data, or to infer something

- it is about programming an agent and not a computer or a "mind"!

œ Û



deliberation

2024-11-11

# └─AI Agent

- considering an AI context, we take a symbolic approach:

- the agent has knowledge

- its behaviour is based on knowledge "The Knowledge Level" [?]
- the developer defines that K (it can be learnt, but not the focus today) what is K? information, rules, plans, goals, ...

- in our case (agents), the focus is on K elements directed to actions

AI Agent

© 0



Reasoning cycle

while true do  $K \leftarrow K \pm perception()$  $G \leftarrow G \pm deliberation(K)$  $A \leftarrow means-end(G)$ do(A)

2024-11-1

#### - what is the engine?

- it is a continuous process that

LAI Agent

- perceives
- decides actions to achieve a goal
- does the actions
- the agent has autonomy the choose actions - an agent decides what to do!
- part of the task that usually a programmer does (ordering the actions) is done by the agent.
- to program an agent is to define K (and not to write an algorithm)

Let's move to a more practical perspective to consolidate the basic concepts (we latter return to the conceptual background)

AI Arent



Reasoning cycle  $K \leftarrow K \pm perception()$  $G \leftarrow G \pm deliberation(K)$  $A \leftarrow means-end(G)$  $\_ do(A)$ 

2024-11-1

- what is the engine?
- it is a continuous process that

LAI Agent

- perceives
- decides actions to achieve a goal
- does the actions
- the agent has autonomy the choose actions
  an agent decides what to do!
- part of the task that usually a programmer does (ordering the actions) is done by the agent.
- to program an agent is to define K (and not to write an algorithm)

Let's move to a more practical perspective to consolidate the basic concepts (we latter return to the conceptual background)







while true do  $K \leftarrow K \pm perception()$ Reasoning  $G \leftarrow G \pm deliberation(K)$ cycle  $A \leftarrow means-end(G)$ do(A)

2024-11-1

- what is the engine?
- it is a continuous process that

LAI Agent

- perceives
- decides actions to achieve a goal
- does the actions
- the agent has autonomy the choose actions - an agent decides what to do! - part of the task that usually a programmer does (ordering the actions) is done
- by the agent.
- to program an agent is to define K (and not to write an algorithm)

Let's move to a more practical perspective to consolidate the basic concepts (we latter return to the conceptual background)

© 0

© 0



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

### deliberation ~> autonomy

2024-11-1

# └─AI Agent

- what is the engine?
- it is a continuous process that
- perceives
- decides actions to achieve a goal
- does the actions
- the agent has autonomy the choose actions
   an agent decides what to do!
   part of the task that usually a programmer do
- part of the task that usually a programmer does (ordering the actions) is done by the agent.
- to program an agent is to define K (and not to write an algorithm)

Let's move to a more practical perspective to consolidate the basic concepts (we latter return to the conceptual background)

2024-11-11

#### └─Agent Knowledge (in Jason)

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

#### temperature(20) happy(bob).

Arent Knowledre (in tasor)

Goals : the agent objectives | temperature (20 | happy(bob).

JaCaMo is a framework with languages that allows us to implement systems based on K agents and ...

so elements of agent knowledge in JaCaMo (beliefs quite usual, novelty are goals and plans, and how they are "interpreted")

- Syntax inspired by Prolog: predicate(arguments)
- Plans = know how
- informal semantics: <if this happens> <- <do this>

- event oriented

temperature(20).
happy(bob).

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

Goals : the agent objectives !temperature(20] !happy(bob).

Plans :

2024-11-11

#### └─Agent Knowledge (in Jason)

Arent Knowledre (in tasor)

JaCaMo is a framework with languages that allows us to implement systems based on K agents and ...

so elements of agent knowledge in JaCaMo (beliefs quite usual, novelty are goals and plans, and how they are "interpreted")

- Syntax inspired by Prolog: predicate(arguments)
- Plans = know how
- informal semantics: <if this happens> <- <do this>

- event oriented

 ${\bf Beliefs}\,$  : information about the environment, other agents, itself,

temperature(20).
happy(bob).

Goals : the agent objectives
 !temperature(20).
 !happy(bob).

Plans



© 0

2024-11-11

#### └─Agent Knowledge (in Jason)

Arent Knowledge (in tasos)

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

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).</pre>

JaCaMo is a framework with languages that allows us to implement systems based on K agents and ...

so elements of agent knowledge in JaCaMo (beliefs quite usual, novelty are goals and plans, and how they are "interpreted")

- Syntax inspired by Prolog: predicate(arguments)
- Plans = know how
- informal semantics: <if this happens> <- <do this>
   event oriented

. . .

2024-11-11

#### $\square$ Agent Knowledge (in Jason) — K = B + G + P

Refef information about the environment, other agont, ind - topper the second second

Arent Knowledge (in tasos)

JaCaMo is a framework with languages that allows us to implement systems based on K agents and ...

so elements of agent knowledge in JaCaMo (beliefs quite usual, novelty are goals and plans, and how they are "interpreted")

- Syntax inspired by Prolog: predicate(arguments)
- Plans = know how
- informal semantics: <if this happens> <- <do this>
   event oriented

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).</pre>

**Beliefs** : information about the environment, other agents, itself,



# Agent Knowledge (in Jason) — K = B + G + P

. . .

2024-11-11

 $\square$  Agent Knowledge (in Jason) — K = B + G + P

+!happy(bob) <- kiss(bob)

appy(bob) <- thappy(bob)

Arent Knowledge (in tasos) - K = B + G + P

JaCaMo is a framework with languages that allows us to implement systems based on K agents and ...

so elements of agent knowledge in JaCaMo (beliefs quite usual, novelty are goals and plans, and how they are "interpreted")

- Syntax inspired by Prolog: predicate(arguments)
- Plans = know how
- informal semantics: <if this happens> <- <do this>
   event oriented

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

**Beliefs** : information about the environment, other agents, itself,

+temperature(10) <- !temperature(20).

-happy(bob) <- !happy(bob).</pre>

© 0

2024-11-11

-Knowledge Sources

Knowledge Sources

Beliefs, goals, and plans are provided by

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

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



2024-11-11

### Smart Room Scenario — initial implementation



Smart Room Scenario — initial implementation

- HVAC provides perception of its state and the current temperature - exposes 3 actions for the agent

(details of how to program this artifact will be presented later)

© 0



## └─Agent Programming (in JaCaMo)



Agent Programming (in JaCaMo)



Let's focus on programming that agent

- the perception of the current temperature is mapped to a belief like temperature(30)

- the objective to maintain some temperature is mapped to a belief like !temperature(30)

- the agent has plans to react to changes in the current temperature and the creation of new goals to maintain some temperature

© 0

# 2024-11-11

0

### Agent Programming (in JaCaMo)

(agents are programmed in JaCaMo using the Jason language)
- these 2 lines are a complete Jason program, a program with 2 plans
- beliefs are added by perception
- (read the plans): in the event of a new belief temperature(30), react to it
creating the goal !temperature(20)
- in the event of a new a goal !temperature(20), react to it by doing
startCooling
- the program has no begin/end, declarative approach (K is declared)
- set of reactive "rules" (implemented by the plans)

- which are the problems of this implementation? (implement, run, and see!)

+temperature(30) <- !temperature(20).
+!temperature(20) <- startCooling.</pre>



# 2024-11-11

### └─Agent Programming (in JaCaMo)

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

!temperature(20) <- startCooling.</pre>

+temperature(30) <- !temperature(20).
+temperature(20) <- stopAirConditioner.</pre>

+!temperature(20) <- startCooling.

(improved version with stopAirConditioner, that stops)(image the agent behaviour) - which are the problems of this implementation?

# 2024-11-11

#### └─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
< ttemperature(P).
+temperature(T) : preference(T)
 <- stopkirConditioner.</pre>

// 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)</pre>
```

```
<- stopAirConditioner.
```

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

What is new:

- new belief, not perceived, but defined in the inicial code of the agent
- variable with upper case first letter
- plans have context, used for the agent to select the most appropriated
  the evaluation of the context is like a query to the belief base, and it may assign values to variables

#### Agent behaviour:

any change in temperature produces actions to start cooling, is it ok? what if the preference changes?

© 0

# 2024-11-11

#### Agent Programming (in JaCaMo)

// initial belief, given by the developer
preference(20).
// initial goal, given by the developer
these temerature.

- // maintenance the goal pattern
  +!keep\_temperature
   : temperature(T) & preference(P) & T > P
- <- startCooling; !keep\_temperature.
- :temperature
  : temperature
  : temperature(T) & preference(P) & T <= P</pre>
- <- stopAirConditioner; [keep temperature.

// 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.

maintenance goal — long term goals agent is not reacting to changes in beliefs anymore, it has a "forever" goal that, based on the circumstances, select a proper plan of actions

- does it reacts to changes in the preference? pro-activity

#### Main Features

# 2024-11-11

# –Main Features

 practicity over florg seriel gold and the created exactivity rever they priming over gold examinery trends (gold) means to achieve gold examinery series of prime reversited based on the circumants emergy arraying and are reflected based on the circumants emergy reverse of the reversited on the circumants emergy reverse and networks (Reg) emerging classical degreends for gar architectures emerging classical degreends for gar architectures emerging classical degreends for gar architectures emerging classical degreends (Reg) emerging classical

Main Feature

- pro-activity: new (long-term) goals can be created
- reactivity: even when pursuing some goals
- autonomy: to find (good) means 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]
  - ...

because:

- agents have a reasoning cycle
- based on knowledge
- reasoning about what to do (practical reasoning) (detailed later in the course)

Are usual languages (Java, Python, Prolog, ...) appropriate to implement agents? Can we use them? Sure we can. But they will give us a lot of work to code agents. Wrap-up

• Reaching Level - Wrap-up

Wrap-up

agents know

agents act

# • Knowledge Level

# agents **know**

Practical Reasoning

agents **act** 



2024-11-11

2024-11-11

Agent Interaction (communication)

# Agent Interaction (communication)

# Agent-Agent Communication



#### └─Agent–Agent Communication

#### agent communication innysage

Arent-Arent Communication

Agent Communication Language

- the language to communicate is not the same as the language to program agents, since they have different purposes

- works at the Knowledge level (again!)

- when sending a message, the sender intends to change the mind of receiver (mentalistic view)

- K is transmitted (thing I know that, know how to, I wish, ...) so send beliefs, desires, plans, ...

- used to build negotiation, coordination, information share





### └─ Semantic of messages



A message has: • an intention (tell, ask, achieve, ...) • a content (belief, roal, plan)



A message has:

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



#### └─ Semantic of messages



A message has:

Semantic of messages

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



A message has:

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









# A message has:

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



#### └─ Semantic of messages



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



- 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

2024-11-11

#### └─JaCaMo implementation

Jac Monie implementation Konker, essell(she, shell, shappy (sl.ise)) - nordern span using mane - ordern span using mane - ordern stations, askNer, askNer, askNer, - ordern stations - ordern station

no code in the receiver, the semantics of the ACL is implemented on the interpreter! distributed means several machines decentralised means no central control KQML and FIPA-ACL are initiatives to standardise ACL KQML was the standard when Jason was first developed

# 2024-11-11

#### └─JaCaMo Performatives

tell and untell: change beliefs of receiver
 achieve and unachieve: change goals of receiver
 ackOne and ackAll: ask for beliefs of the receiver
 ackHow, cellifore, and untellifore: exchange plans with other agent
 isguals add as cover in the receiver

Theoretical background is speech acts [?, ?]: to say is to act; to power of word.

synchronous cases: .send(a,askOne,v(X),A) it blocks the intention until an answer is received, the answer is assigned to A

signal is quite recent in JaCaMo (Jason) e.g. .send(bob,signal,hello)

- 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



2024-11-11

#### └─Smart Room Scenario

#### decentralised solution

we will solve it using agent communication



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

tell open\_soting(Convid, Options, Timeou

personal\_ assistant

controller

# 2024-11-11

#### $\square$ Interaction Protocols $\rightsquigarrow$ coordination

room_		persona
controller /		assistar
T		
	tell open_voting(ConvId, Options, Timeout)	
<	tell ballot(Convld, Vote)	
	tell close_voting(ConvId, Result)	

decentralised solution requires coordination (of actions)

coordination of actions, order actions here, order of comunicative actions

© 0

# Protocol Implementation

# 2024-11-11

# └─Protocol Implementation

live coding

Wrap-up: Agent Model

inheritance

⊳



2024-11-11

### └─Wrap-up: Agent Model



# Wrap-up: Agent Programming

# 2024-11-11

### Wrap-up: Agent Programming

Agendysche
 Lege: PEE
 Lege: PEE
 Agenet programming language
 Anour
 Agenet pairs intersperser
 Implemente on operational amanetics of Agenetysche
 Specific and communication
 Highly communication
 Lofid such
 Copen summe

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



œ Û

# Fundamentals

# 2024-11-11

Fundamentals

I do not plan to present all the following slides, I will select them regarding the interests of the audience
#### 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...

2024-11-11

└─Fundamentals └─Literature

Hard work have being done on this approach already.

#### 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, ...);

└─Fundamentals └─Some Languages and Platforms

#### Some Languages and Platforms

June Olithen, Rodell, J., VIP, and ZMP (Dours), con Resembly, New TheoRe, J., Jack Menda, N. Hadd Martell White, Caracteristic Methods, N. Hadd Method, M. Hadd Methods, J. C. 2010, Phys. Rev. Lett. 1997, 1997 (2010), 2010, 2011, 2011, 2011, 2011, 2011, 2011, 2011, Falls-September, J., COAM, Blandtah, JRAMBS Rieshne, J., Parkensen, K. B., Karl, K. M. Bart, M. Stern, M. Ja-Bart, M. Stern, M. Stern, M. Stern, M. Stern, M. Ja-Stern, J. Theory, W. Stern, C. K. M. Martell, 2011, 2011, Statistic Tan, J. THXT Wireless (2011), 2011, 2011, 2011, 2014, 2011, 2011, 2011, 2011, 2011, 2011, 2011, 2011, 2011, 2014, 20

some proposals are libraries/packages for existing languages others are new languages many agent languages have efficient and stable interpreters



. . .

2024-11-11

#### Agent Oriented Programming — Inspiration

2024-11-11

└─Fundamentals └─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 [Beatman et al., 1988]

I do recommend to read foundational papers like these from a Philosopher (trying to solve the problem for humans)

- 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



2024-11-11

#### └─Fundamentals └─BDI architecture



BDI architectury

mentalistic view (the behavior of the agent is explained in terms of its mental state: B, D, and I):

- B: beliefs (information) the agent has about its environment (updated by perception) - D: what the agent wishes - I: desires the agent has \*committed\* to (based on the current beliefs and other intentions)

Two main processes: deliberate: Desire -> Intention means-ends: Intention -> Actions

BDI explains the actions of the agent! (because the agent intents to, desires to, and believes it is feasible)

So next slides will highlight properties of the commitment (serious commitment but not too much)

#### while true do

```
B \leftarrow brf(B, perception())D \leftarrow options(B, I)I \leftarrow deliberate(B, D, I)\pi \leftarrow meansend(B, I, A)while \pi \neq \emptyset do
execute( head(\pi) )
\pi \leftarrow tail(\pi)
```

// belief revision
// desire revision
// get intentions
 // gets a plan

└─Fundamentals └─BDI reasoning cycle [Wooldridge, 2009]

#### 

BDI reasoning cycle [Wooldridge, 2019]

intentions are desire + commitment.

types of commitments: over commitment, Singel-Minded, .... there are good bibliography on that.



2024-11-11

#### while true do

// belief revision
// desire revision
// get intentions
 // gets a plan

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

└─Fundamentals └─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
$\pi \leftarrow meansend(B, I, A)$ while $\pi \neq 0$ do curcure(head( $\pi$ ))	// gets a plas
$\pi \leftarrow tail(\pi)$	

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

intentions are desire + commitment.

types of commitments: over commitment, Singel-Minded, .... there are good bibliography on that.



2024-11-11

#### while true do

```
B \leftarrow brf(B, perception())
D \leftarrow options(B, I)
I \leftarrow deliberate(B, D, I)
\pi \leftarrow meansend(B, I, A)
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

2024-11-11

// belief revision

// desire revision

// get intentions

// gets a plan

└─Fundamentals └─BDI reasoning cycle [Wooldridge, 2009]

ile 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 plas
while $\pi \neq 0$ do	
execute(head(x))	
$\pi \leftarrow tail(\pi)$	
$B \leftarrow br f(B, perception())$	
if ¬sound(x, I, B) then	
$\pi \leftarrow meansend(B, L, A)$	
E C	

#### intentions are desire + commitment.

types of commitments: over commitment, Singel-Minded, .... there are good bibliography on that.



#### 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 \negsucceeded(I, B) and \negimpossible(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

└─Fundamentals └─BDI reasoning cycle [Wooldridge, 2009] BDI reasoning cycle [Wooldridge, 2009]

revise commitment to intentions - Single-Minded Commitmen

#### intentions are desire + commitment.

types of commitments: over commitment, Singel-Minded, .... there are good bibliography on that.



2024-11-11

#### 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 \negsucceeded(I, B) and \negimpossible(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)
```

Fundamentals → Fundamentals → BDI reasoning cycle [Wooldridge, 2009]

DI reasoning cycle [Wosldridge, 2019]	
while true do	
$B \leftarrow brf(B, perception())$	// belief revisio
$D \leftarrow options(B, I)$	// degire revigio
$I \leftarrow deliberate(B, D, I)$	// get intention
$\pi \leftarrow meansend(B, I, A)$	// gets a pla
while $\pi \neq \emptyset$ and $\neg$ succooded $(I, B)$ and $\neg$	improssible(I, B) do
execute(head(n))	
$\pi \leftarrow tail(\pi)$	
$B \leftarrow brf(B, perception())$	
if $reconsider(I, B)$ then	
$D \leftarrow options(B, I)$	
$I \leftarrow deliberation(B, D, I)$	
if $\neg sound(\pi, I, B)$ then	
$\pi \leftarrow meansend(B, I, A)$	
L -	

#### intentions are desire + commitment.

types of commitments: over commitment, Singel-Minded, .... there are good bibliography on that.

2024-11-11

-Fundamentals

-Intentions

 Intention poor problems for the agonts: they need to determine a option takened ways to akhine the approximation of the (planning and acting)
 Intentions provide 3" accrete of adminishibility" for adopting new intentions
 Agonts heavy tracking their success of antempting to achieve their intentions
 Agonts hould not open all their time treining intentions (boing reso-activity and rescritivity)

- 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)

2024-11-11

└─Fundamentals └─(BDI & Jason) Hello World – agent bob (BDI & Jason) Hello World - agent bob friend(alice). // 8 !may(ballo). // 0 +!may(b) <- .print(b). // I

- how does it look like? (comparing with other languages like C, Java, ....) - jason uses procedural goals (goals to do) and not declarative goals (goals to be), as 2APL. it comes from the orignal PRS inspiration where we are specifying behaviour instead of (env) states [?]

plans are not prolog (theoretical reasoning), they are the for practical reasoning. the language gives constructors to program BDI with the required features shown in the Woodridge algorithm

friend(alice).

!say(hello).

+!say(M) <- .print(M). // I

// B

// D

friend(alice).

!say(hello).

// B beliefs • prolog like (FOL)

+!say(M) <- .print(M). // I

└─Fundamentals └─(BDI & Jason) Hello World – agent bob (RDI & Json) Hells World - agent bab
friend(alics). // 8
!say(ballo). // D
+!say(0) <- .print(N). // I</pre>

how does it look like? (comparing with other languages like C, Java, ....)
jason uses procedural goals (goals to do) and not declarative goals (goals to be), as 2APL. it comes from the orignal PRS inspiration where we are specifying behaviour instead of (env) states [?]
plans are not prolog (theoretical reasoning), they are the for practical reasoning.

- the language gives constructors to program BDI with the required features shown in the Woodridge algorithm



2024-11-11

friend(alice).

!say(hello).

// B
desires
// 1
• prolog like
• with ! prefix

+!say(M) <- .print(M). // I

2024-11-11

└─Fundamentals └─(BDI & Jason) Hello World – agent bob (BDM & Jason) Hello World - agent bob friend(alics). // 8 !say(ballo). // D +!may(bi) <- .print(b). // I

how does it look like? (comparing with other languages like C, Java, ....)
jason uses procedural goals (goals to do) and not declarative goals (goals to be), as 2APL. it comes from the orignal PRS inspiration where we are specifying behaviour instead of (env) states [?]
plans are not prolog (theoretical reasoning), they are the for practical reasoning.

- the language gives constructors to program BDI with the required features shown in the Woodridge algorithm

© 0

2024-11-11

friesd(alios). // # fauy(ballo). // # +fauy(ballo). // #

(BDI & Jason) Hello World – arent bob

friend(alice).

!say(hello).

+!say(M) <- .print(M).

- // B plans
  - define when a desire becomes an intention
     → deliberate
  - how it is satisfied

└─Fundamentals └─(BDI & Jason) Hello World – agent bob

- how does it look like? (comparing with other languages like C, Java, ....)
- jason uses procedural goals (goals to do) and not declarative goals (goals to be), as 2APL. it comes from the orignal PRS inspiration where we are specifying behaviour instead of (env) states [?]

plans are not prolog (theoretical reasoning), they are the for practical reasoning. the language gives constructors to program BDI with the required features shown in the Woodridge algorithm

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

friend(alice).

+happy(A) <- !say(hi(A)).</pre>

+!say(M) <- .print(M).

Fundamentals BDI Hello World — desires from perception (options)

desire via perception, the agent starts believing someone is happy and then creates a new desire

BDI Hello World - desires from perception (options)

+happy(A) <- !say(hi(A))</pre>

+!say(M) <- .print(M).

friend(alice).

+happy(A) : friend(A) <- !say(hi(A)). +happy(A) : not friend(A) <- !say(good\_afternoon(A)).</pre>

+!say(M) <- .print(M).

2024-11-11

−Fundamentals └─BDI Hello World — plan selection BDI Hello World — plan selection

friend(alio

happy(A) : friend(A) <- !say(hi(A)). happy(A) : not friend(A) <- !say(good\_afternoon(A)).</pre>

+!say(M) <- .print(M)

the agent selects the plan that is more suitable for the current circumstance. plan context is used for that. whenever (trigger event) - I start to believe that A is happy and (context) - I belief that A is a friend

then (body) - create a new desire to say hi

whenever I have the desire to say something, commit to that desire and use the body of the plan to fullfil it.



BDI Hello World — intention revision

friend(alice

+happy(Å) : friend(Å) <- !say(hi(Å)).
+happy(Å) : not friend(Å) <- !say(good\_afternoon(Å)).</pre>

fsay(M) <- .print(M); .wait(1000); fsay(M).</pre>

+busy(bob) <- .drop\_intention(say(\_)).

friend(alice).

```
+happy(A) : friend(A) <- !say(hi(A)).
+happy(A) : not friend(A) <- !say(good_afternoon(A)).</pre>
```

```
+!say(M) <- .print(M); .wait(1000); !say(M).
```

```
+busy(bob) <- .drop_intention(say(_)).
```

2024-11-11

└─Fundamentals └─BDI Hello World — intention revision

BDI Hello World — intention revision

#### BDI Hello World — intention revision

friend(alice).

+happy(A) : friend(A)
+happy(A) : not friend(A)

+!say(M) <- .print(M); .wa

+busy(bob) <- .drop\_intent

- we can have several intentions based on the same plans
- → running concurrently

features

- long term goals running
   → reaction meanwhile
  - $\rightsquigarrow$  not overcommitted
- plan selection based on circumstance
- sequence of actions (partially) computed by the interpreter

   → programmer declares plans

└─Fundamentals └─BDI Hello World — intention revision happy(A) : friend(A) <- !say(hi(A)). happy(A) : not friend(A) <- !say(mood afternoon(A))

+leav(0) <\_ nrint(0) - wait(1000) - leav(0)</pre>

+busy(bob) <- .drop\_intention(say(\_)).

2024-11-11



Jason

## Jason

#### AgentSpeak: The foundational language for Jason



-Jason └─AgentSpeak: The foundational language for Jason Programming happage for BD1 aprox
 Originally proposed by Rao (BRA), 1996)
 Elegran rotation, based on leajer programming
 Imprired by PRS (Georgeff & Landey), dMARS (Kinny), and BD1
 Legics (Rao & Georgeff
 Abstrace programming language aimed at theoretical results

- 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

2024-11-11

└─*Jason* └─*Jason*: A practical implementation of AgentSpeak  Jacen 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

• *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

agent dimension in JaCaMo



2024-11-11

└─*Jason* └─Main Language Constructs

Beliefic represent the information available to an agent (e.g. about the environment or other agents) Goale: represent states of affairs the agent warms to bring about Plane: are recipes for action, representing the agent's know-how

# **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



2024-11-11

Jason **Beliefs** — Representation

Symax Beliefs are represented by annotated literals of first onder logic functor (letrm<sub>1</sub>, ..., letrm<sub>n</sub>) [sonal<sub>1</sub>, ..., snoal<sub>n</sub>] Example (belief hose of spury Tom) resultman1 (located spury Tom) (friend (beb, Alico) [source (stor), ...) [friend(beb, Alico) [source (stor), ...]

#### Syntax

Beliefs are represented by annotated literals of first order logic

functor  $(term_1, \ldots, term_n)$  [annot<sub>1</sub>, ..., annot<sub>m</sub>]

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)].

annotations is a set of terms with special unification — not available in Prolog



Beliefs – Dynamics i

2024-11-11

41

-Jason

-Beliefs - Dynamics

#### 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)]

by intention the plan operators - and - can be used to add and remove beliefs annotated with source(self) (mental nones)

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

accordingly to the perception of the agent

Beliefs – Dynamics ii

2024-11-11

Jason

-Beliefs - Dynamics

#### 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)[searce(bob)] in Tom's BB

.send(tom,untell,lier(alice)); // sent by bob // removes lier(alice)[source(bob)] from Tom's DB

#### 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

• • •

© 0

.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).

2024-11-11

### *Jason* └─**Goals** — Representation

Types of goals • A holescencer goal goal to do a • Tore goal goal to know: Symm Catals have do some syntax as is blefte, host are prefined by 1 calculatered to some syntax as is blefte, host are prefined by 2 core goal Example chickle goal of goars Tonol Herst Chicks?

jason uses procedural goals (goals to do) and not declarative goals (goals to be, as in planning, 2APL, ...).

it comes from the orignal PRS inspiration where we are specifying behaviour instead of (env) states [?]

PRS also proposes maintenance goal, that is available in Jason(ER)

C 0

#### 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);

• • •



By incremine the planeyement and team be used to skill a new guitanneounal with assumediant ... // skills are skillerowast guit fortactoold (sames(skill) fortat(sees); // skills are skillerowast guit fortactoold(sames(skill) fortactoold(skill); // skills are skillerowast

Goals - Dynamics ii

#### Goals – Dynamics ii

2024-11-11

#### └─*Jason* └─Goals — Dynamics

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

send(tom,achieve,write(book)); // sent by Rob

.send(tom,umachieve,write(book)); // sent by Bob

#### 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

2024-11-11

Jason

Goals — Dynamics

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 Hob // adis new goal ?publisher(P)[source(bob)] for Tum // the response of Tum unifies with Answer

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

#### Triggering Events — Representation

2024-11-11

-*Jason* — Triggering Events — Representation  Even hopera a consequence to damps in the agent's helief or good:
 An agent means to even by executing plane
 An agent means
 An agent means to even by executing plane
 An agent means to even by executing plane

- 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)

  - +!g (achievement-goal addition)
  - -!g (achievement-goal deletion)
  - +?g (test-goal addition)
  - $-\ensuremath{\underline{?g}}$  (test-goal deletion)

#### Plans — Representation

Plans - Representation

where

An AgentSpeak plan has the following general structure:

#### triggering\_event : context <- body

here: • 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 context represent the circumstances in which the plan can be used

the context is believed true at the time a plan is being chosen to handle the event

An AgentSpeak plan has the following general structure:

triggering\_event : context <- body.</pre>

#### 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





2024-11-11

#### Plans — Operators for Plan Context

#### Plans — Operators for Plan Context

2024-11-11

*└─Jason* └─Plans — Operators for Plan **Context** 

oolean oper:	itors	Arithmetic op	octators
\$	(and)		(sum)
	(ot)		(subtraction)
net	(not)		(multiply)
-	(unification)	1	(divide)
>,>=	(relational)	div	(divide - integer)
<, <=	(relational)	mod	(remainder)
	(equals)		(power)
\	(different)		

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

# 2024-11-11

#### └─*Jason* └─Plans — Operators for Plan **Body**

<- ig1;	// new sub-goal
11g2	// 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
close(door) !g3[hard_de	<pre>;// external action adline(3000)]. // goal with </pre>

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
	<pre>jia.get(X);</pre>	//	internal action
	X > 10;	//	constraint to carry on
	<pre>close(door);</pre>	//	external action
<pre>!g3[hard_deadline(3000)]. // goal with deadline</pre>			

2024-11-11

└─*Jason* └─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
- prans received from
   selltion messare
- uncellHow

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



© 0

2024-11-11

*-Jason* └─A note about "Control" Agents can control (manipulate) their own (and influence the others) • locks • grads • grads By doing so they control their behaviour The developer provide initial Values of these of energy and thus also influence the behaviour of the agent

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

Object Oriented encapsulates "beliefs" and "plans", but not "goals", no "thread of execution"



-Jason

Reasoning Cycle

# Reasoning Cycle
2024-11-11

*─Jason* └─Runtime Structures for the Reasoning Cycle Beliefic represent the information available to an agent (e.g. about the environment or other agents) Goale: represent states of affairs the agent wants to bring about Plane: 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

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 aboutPlans: 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

The former three come from the agent program (syntax), the latter two exist at runtime to support the interpretation (semantics) of Jason

© 0

### Basic Reasoning Cycle — runtime interpreter



Jason -Basic Reasoning Cycle — runtime interpreter



sic Reasoning Cycle — runtime interpreter

- 1 perceive the environment and update belief base
- 1 process new messages
- 1 select event

2024-11-11

- 2 select relevant plans
- 2 select applicable plans
- 3 create/update intention
- 4 select intention to execute
- 4 execute one step of the selected intention





Jason Reasoning Cycle



56





- machine perception
- belief revison
- knowledge representation
- communication,

argumentation

- trust
- social power

└─*Jason* └─*Jason* Reasoning Cycle

2024-11-11



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

2024-11-11

└─*Jason* └─*Jason* Reasoning Cycle





- intention reconsideration
- scheduling
- action theories

2024-11-11

└─*Jason* └─*Jason* Reasoning Cycle





-Jason

Other Features

# Other Features

### Failure Handling: Contingency Plans

# 2024-11-11

### └─*Jason* └─Failure Handling: Contingency Plans

Example (an agent blindly committed to g) +1g : g. // g is a declarative goal +1g : ... <- a1; ?g. +1g : ... <- a2; ?g.

+1g : ... <- a3; ?g. +1g <- 1g. // keep trying

Failure Handling: Contingency Plans

-1g <- 1g. // in case of some failure

```
\texttt{*g} <\_\texttt{.succeed_goal(g)}.
```

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</pre>

+g <-.succeed\_goal(g).

© 0

### Failure Handling: Contingency Plans

### Example (single minded commitment) +!g : g. // q is a declarative qoal

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

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

+g <-.succeed\_goal(g).

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

```
f is the drop condition for goal g
```



#### Failure Handling: Contingency Plans

Example (single minded commitment) +!g : g. // g is a declarative goal

+1g <- 1g. // keep trying -1g <- 1g. // in case of some failure

+g <-.succeed\_goal(g).
+f : .super\_goal(g,SG) <-.fail\_goal(SG)</pre>

f is the drop condition for goal g

### Compiler pre-processing – directives

2024-11-11

└─*Jason* └─Compiler pre-processing – directives  $\begin{array}{l} \mbox{Example (single minded commitment)} \\ \{ \mbox{ bugin } mc(g,f) \} \\ +ig: \ldots < -a1. \\ +ig: \ldots < -a2. \\ +ig: \ldots < -a3. \\ \{ \mbox{ end } \} \end{array}$ 

Example (single minded commitment)
<pre>{ begin smc(g,f) }</pre>
+!g : <- a1.
+!g : <- a2.
+!g : <- a3.
{ end }

### Meta Programming

© 0

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

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

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



Meta Programming

Example (an agent that asks for plans on densesd) -1G[error(no\_relevant)] : teacher(T) <- .send(T, ashidow, { +1G }, Plans); .add\_plan(Plans); 10

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 annorated with the error type, line, source, error(no, relevant) means no plan in the agent's plan library to achieve G
 (-1C) is the syntax to enclose triggers/plans as terms

63

2024-11-11

└─*Jason* └─Other Language Features: Strong Negation +!leave(home) : -raining

<- open(curtains); ...

+!leave(home)

not raining & not -raining
 .send(mum,askOne,raining,Answer,3000);

### +!leave(home)

- : ~raining
- <- open(curtains); ...

### +!leave(home)

- : not raining & not ~raining
- <- .send(mum,askOne,raining,Answer,3000); ...



2024-11-11

└─*Jason* └─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.

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

### **Internal Actions**

2024-11-11

Jason

-Internal Actions

Unlike actions, internal actions do not change the environmen
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(...)

- 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.



 Standaul (pse-defined) internal actions have an empty library name - print (strm<sub>1</sub>, strm<sub>2</sub>,...) - action(s(st<sub>1</sub>, lit<sub>2</sub>, lit<sub>2</sub>), - arg\_name(st<sub>2</sub>), str<sub>1</sub>(st<sub>2</sub>) - arg\_n(st<sub>2</sub>), str<sub>1</sub>(st<sub>1</sub>) - interst(litral) - drog\_name(st<sub>1</sub>))

Standard Internal Actions

Many others available for: printing, sorting, list/string operations, manipulating the beliefs/annotations/plan library, creating agents, waiting/generating events, etc.

### Namespaces & Modularity

2024-11-11







### Namespaces & Modularity

### Inspection of agent alice

- Beliefs

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

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

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

tv:: introduction(participant)<sub>[source(compar</sub> propose(11.075337225252543)<sub>[sourc</sub> propose(12.043311087442898)<sub>[sourc</sub> propose(12.81277904935436)<sub>[source</sub> winner(company\_A1)<sub>[source(self)]</sub>.

**#8priv::** state(finished)<sub>[source(self)]</sub>.

pc::

introduction(participant)<sub>[source(compar</sub> propose(11.389500048463455)<sub>[sourc</sub> propose(11.392553683771682)<sub>[sourc</sub> propose(12.348901000262853)<sub>[sourc</sub> winner(company\_A2)<sub>[source(self)]</sub>. └─*Jason* └─Namespaces & Modularity

2024-11-11

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

!pc::startCMP(fix(pc))
!tv::startCMP(fix(ty))

:::vinner(I)

Concurrent Plans

2024-11-11



+1ga <- ...; 1gb; ... +1gb <- ...; 1g1 |k| 1g2; a1; ...

+1ga <- ...; 1gb; ... +1gb <- ...; 1g1 ||| 1g2; a1; ...

 $+!g < x; \ (a;b) \ |k| \ (c;d) \ ||| \ (e;f); \ y.$ 

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

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

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



### Jason(ER) — motivation



```
+accept_proposal(Id) : my_offer(Task)
<- !do(Task);
    -my_offer(Task).</pre>
```

jood(EA) - maximum \*cfp(iA,tua) [server(a)] // maxer is Cull for Prepara \* prior(Take,D(fer y) is = ty\_affec(Take) < = referes(Take) \* mark(s,ull,repare(A,d(fer))). \* dry(IA\_1) [server(A)] < = refer(A\_1) [server(A)] \* except\_propulat(IA) = ty\_affec(Take) <= In (trake). \* expect\_propulat(IA) = ty\_affec(Take) < = -m\_affec(Take).</pre>

what is the goal related to the action .send(A,tell,propose(Id,Offer))
when executing actions for goal "do", if we ask "why" we can track back to accepted\_ proposal, but not to the cfp or even some implicit goal that is "participate in CNP"

some jason intentions have no explicit goal no explicit causal link among plans

### Scope & sub-plans & goal conditions

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

# 2024-11-11

### *─Jason* └─Scope & sub-plans & goal conditions

+1g(X) : c < gg c <- a1; 1g1. { +e : c1 <- a2(X). +1g1 .... }

Scope & sub-plans & goal conditions

- main objective: all behaviour is the result of an (explicit) goal (not a Jason intention, that can have no explicit goal)

- new syntax: goal condition after <: and sub-plans enclosed by { and }

relevant event are defined by the current intentions
event +e is relevant only the agent intends g
relevant plans are defined by the scope of some goal
plan for g1 is visible only in scope of g
g is dropped only when gc is true: maintenance goal
variables have a broader scope (X is visible in sub-plans

### Example of a participant in a CNP

+!do(T) <- ...

```
+!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,Off
       +accept_proposal(Id) <- !do(Task); -my_offer(Task,_)</pre>
       +reject_proposal(Id) <- -my_offer(Task,_); .done.</pre>
   +cfp(Id,_)[source(A)] <- .send(A,tell,refuse(Id)).
```

```
II-11 L_Jason

147 Lexample of a participant in a CNP
```

```
complet of purchases in a CAP
(purchases) = Calase (
    refr(ids,ima) purce(ids)) // senser (s full for Preparal
    prior(CHAses))
    faise
    competence of the sense of the sens
```

the intention for +participate\_cfp never finishes
e-plan for +cfp is triggered only if the agent has goal participate\_cnp
the progression of the intention due to +cfp is finished only by .done, since the goal condition ('false') will never hold
the e-plans enclosed by { and } are relevant only while the progression for +cfp is "running"

- consider that some progress in the intention is created from cfp(10,"banana"), only events accept\_proposal(10) and reject\_proposal(10) are relevant to trigger the subplans.

```
- enforce that every behaviour is due to a goal
```

2024-11-11

Jason Jason Customisations  Agent class castomisation: selectMessage, selectEvent, selectOption, selectIntention, buf, brf,

 Agent architecture customisation: perceive, act, sendMag, checkMail,...

Belief base customisation

 Example available with Jason previnent belief base (in usar files, in data bases, ...)

- 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, ...)

 $Jason \times Java$ 

2024-11-11

←Comparison with other paradigms └─*Jason* × Java

Consider a very simple robot with two goals:

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

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
```

© 0

ר ר **ר** 

2024-11-11

76

```
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));
```

```
└─Comparison with other paradigms
└─Java code – go to gold
```

Java code – go to gold

a = selectDirection();
doAction(mo(a));

### Java code – charge battery

© 0

ר ר **ר** 

2024-11-11

```
└─Comparison with other paradigms
└─Java code – charge battery
```

ava code - charge battery public class Robot extends Thread { boolean seeGold, lowSattery;

doAction(go(a));
if (lowBattery) charge();

```
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

© 0

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)).

2024-11-11

└─Comparison with other paradigms └─*Jason* code

direction(gold) :- see(gol direction(random) :- not see	id). (gold).
+!find(gold)	// long term goal
<- Ydirection(A); go(A);	
ffind(gold).	
<pre>*battery(low)</pre>	// reactivity
<pre>^!charge[state(executing)]      <suspend(find(gold)).< pre=""></suspend(find(gold)).<></pre>	// goal meta-events
<pre>^!charge[state(finished)]</pre>	

Issen code

2024-11-11

−Comparison with other paradigms └─*Jason* × Prolog  With the Jasen extensions, nice separation of theoretical and practical reasoning

BDI architecture allows

Issen × Prolog

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

- 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



- https://jason-lang.github.io
- R.H. Bordini, J.F. Hübner, and M. Wooldrige

Programming Multi-Agent Systems in AgentSpeak using Jason John Wiley & Sons, 2007.



### └─Comparison with other paradigms └─Further Resources



Further Resources

Besides the JaCaMo book (which has chapters dedicated to the agent dimension, the Jason book is all focused on this dimension

2024-11-11

### Bibliography i

2024-11-11

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.

Comparison with other paradigms Bibliography Bibliography i

 Rodlard, R. I., Bankoh, L., Dirash, M., Edhé-Sephowskin, A. G., Gioro-Sand, J., Leino, J., Olino, G. M., Rohan, A., and Kari, A. (2008).
 Anarcy of programming languages and platform for multi-gene system. *Approximat Obstantia*, 20033–44.
 Rodlard, R. H., Danzak, M.D. Ba, J. and Falls-Seglenscheit, A. E., alaben CHON. Moli-Segn Programming Language, *Teleform and Applications*, editors, editors, Moli-Segn Programming Language, *Teleform and Application*, *Moli-Segn Programming Languages*, *Teleform and Application*, editors, e

### Bibliography ii

© 0

2024-11-11

Comparison with other paradigms Bibliography Bibliography ii

- Bordini, R. H., Dastani, M., Dix, J., and Fallah-Seghrouchni, A. E editors (2009). Multi-Agent Programming: Languager, Tools and Applications.
- Springer.
  Bondini, R. H., Hübner, J. F., and Wooldridge, M. (2007).
- Programming Multi-Agent Systems in AgentSpeak Uning Jasan. Wiley Series in Agent Technology. John Wiley & Sons.

Beatman, M. E. (1987). Intention, Plans, and Practical Reson. Harvard University Press, Cambridge

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.
- Bratman, M. E. (1987). Intention, Plans, and Practical Reason.

Harvard University Press, Cambridge.

### Bibliography iii

# 2024-11-11

# 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.

### Dastani, M. (2008).

### 2apl: a practical agent programming language.

Autonomous Agents and Multi-Agent Systems, 16(3):214–248.

### └─Comparison with other paradigms └─Bibliography

#### Bibliography iii

Bearman, 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 Intellipages 410–415. Moregan Kaufmann.

#### Dastani, M. (2008).

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

### Bibliography iv

# 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.

2024-11-11

### └─Comparison with other paradigms └─Bibliography

### Bibliography iv

Fisher, M. (2005). Metatem: The story so far.

In PROMAS, pages 3-22.

Fisher, M., Bordini, R. H., Hirsch, B., and Torrori, P. (2007). Computational logics and agents: A road map of current technologies and future reads. *Compositional 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.

trrif. Intell., 121(1-2):109-16

### Bibliography v

### 📔 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.

2024-11-11

└─Comparison with other paradigms └─Bibliography

#### Bibliography v

Hindriks, K. V (2009).
 Programming rational agents in GOAL.
 In [Bendine rat., 2009], pages 119–137.
 Hindriks, K. V., de Beer, F. S., van der Hock, 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 [Bendini et al., 2005], pages 149–174.

### Bibliography vi

### 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.

Shoham, Y. (1993).

## Agent-oriented programming.

Artif. Intell., 60(1):51–92.

2024-11-11

### └─Comparison with other paradigms └─Bibliography

#### Bibliography vi

### Rao, A. S. (1996).

Agentspeak(I): Bdi agents speak out in a logical computable language.

in an vease, w. v. and retram, 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 arents: from theory to practice.

BLM agents: from theory to practice. In Lesser, V., editor, Proceedings of the First International Conference on MultiAgent Systems (ICMAS95), pages 312–319. AAAI Pess.

Shoham, Y. (1993). Agent-oriented programming Artif. Intell., 60(1):51–97

Bibliography vii

2024-11-11

−Comparison with other paradigms └─Bibliography Winikoff, M. (2005). Jack intelligent agents: An industrial strength platform in [Bendini et al., 2005], pages 175–193.

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

### 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.

