Jason: A Java-based interpreter for an extended version of AgentSpeak

developed by

Rafael H. Bordini and Jomi F. Hübner

http://jason.sourceforge.net/Jason.pdf

The work leading to ***Jason received many contributions, in particular from:*** *Michael Fisher*, *Joyce Martins*, *Álvaro F. Moreira*, *Renata Vieira*, *Willem Visser*, *Michael Wooldridge*, and many others

Jason

Part 1

- What is ***AgentSpeak***?
- What is ***Jason***?
- Features
- Basic Notions
- AgentSpeak(L) Syntax
- AgentSpeak(L) Informal Semantics
- Jason Reasoning Cycle
- A Simple Examples

What is ***AgentSpeak***?

- A simple but powerful programming language for building rational agents based on the ***Belief-Desire-Intention*** paradigm.

- Intellectual heritage:
  - The Procedural Reasoning Systems (PRS) developed at Stanford Research Institute in late 1980s
  - Logic Programming/Prolog

What is ***Jason***?

Jasón y Medea, de Gustave Moreau (*Museo de Orsay*).

Jason is the first fully-fledged interpreter for a much improved version of ***AgentSpeak***, including also speech-act based inter-agent communication.

Features (1)

- **Strong negation**, so both closed-world assumption and open-world are available;
  - Closed World Assumption: anything that is neither known to be true, nor derivable from the known facts or inferences, is assumed to be false.
  - “not” operator means that the negation of the formula is true if the interpreter *fails* to to derive the formula given the fatcs and rules.
  - “~” or strong negation means that the formula is false.
  - Example:
    - colour(box1,white), when the agent believes that box1 is white.
    - ~colour(box1,white), when the agent belives that box1 is not white.
    - If both (not p) and (not ~p) are true, the agent has no information about whether p is true or not

Features (2)

- **Speech-act** based inter-agent communication (and belief annotations on information sources);
- Possibility **to run** a multi-agent system **distributed** over a network (using SACI);
- A library of essential "**internal actions**" which are programmed in Java.
- Handling of **plan**,
- …

Basic Notions

- An **AgentSpeak(L)** agent is created by the specification of a set of base **beliefs** and a set of **plans**.
- A **belief** atom is simply a **first-order predicate** in the usual notation- **publisher(wiley)**
- A **triggering event** defines which events may initiate the execution of a plan.
- The **goals** are also predicates prefixed with operators: '**!**' (**achievement** goals)- **!write(book)** and '**?**' (**test** goals)- **?publisher(P)**

Basic Notions

- An **event** can be **internal**, when a subgoal needs to be **achieved**, or **external**, when generated from **belief** updates as a result of perceiving the environment.
- There are two types of triggering events: those related to the **addition** (‘**+**’) and **deletion** (‘**-**’) of mental attitudes (beliefs or goals).
- **Plans** refer to the basic actions that an agent is able to perform on its environment.

Basic Notions

- **Actions** are also defined as first-order predicates, but with special predicate symbols (called action symbols) used to distinguish them from other predicates.
- A **plan** is formed by a **triggering event** (denoting the purpose for that plan), followed by a conjunction of belief literals representing a **context** (applicable) and a sequence of **basic actions or (sub)goals:**

  triggering_event : context <- body.

+concert(A,V) : likes(A) ← !book_tickets(A,V).
+!book_tickets(A,V) : ¬busy(phone) ←
call(V); . . .; !choose_seats(A,V).

Basic Notions

- **Triggering event s:**
  - +b (belief addition)
  - -b (belief deletion)
  - +!g (achievement-goal addition)
  - -!g (achievement-goal deletion)
  - +?g (test-goal addition)
  - -?g (test-goal deletion)

AgentSpeak(L) Syntax

ag ::= bs ps

bs ::= at1. . . . atn. (n ≥ 0)

at ::= P(t1, . . . ,tn) (n ≥ 0)

ps ::= p1 . . . pn (n ≥ 1)

p ::= te : ct <- h .

te ::= +at | -at | +g | -g

ct ::= true | l1 & . . . & ln (n ≥ 1)

h ::= true | f1 ; . . . ; fn (n ≥ 1)

l ::= at | not at

f ::= A(t1, . . . ,tn) | g | u (n ≥ 0)

g ::= !at | ?at

u ::= +at | -at

ag- agent ::= bs- belives ps- plans

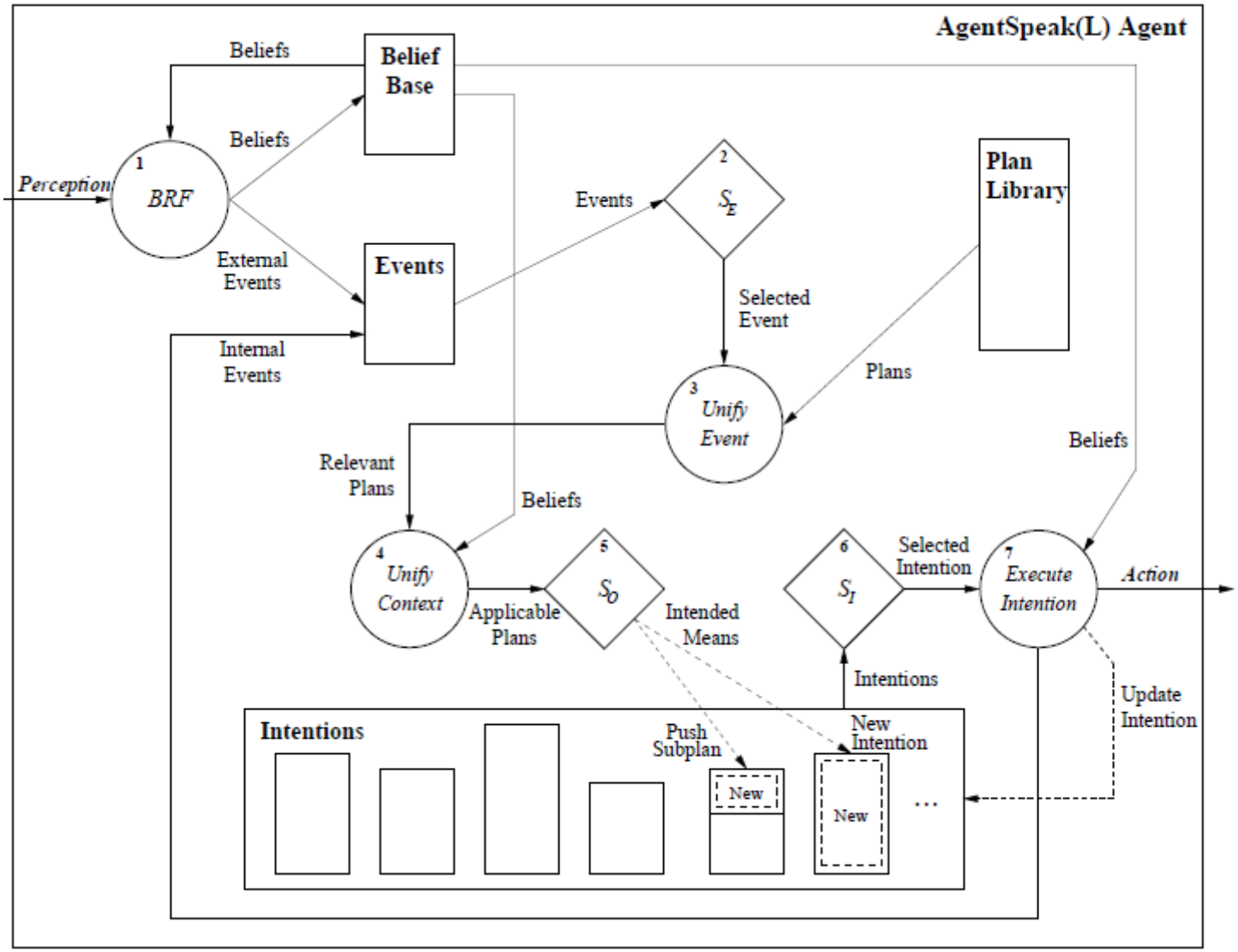
p ::= te-trigering event : ct-context <- h- sequence of actions, goals, or belief updates.

at - addition or the deletion of a belief

g- addition or the deletion of a goal

A(t1, . . . ,tn) - accion

Jason Reasoning Cycle



A Simple Example

Collecting Garbage

The scenario used here involves two robots that are collecting garbage on **planet Mars(1)**. **Robot r1** searches for pieces of garbage and when one is found, the robot picks it up, take it to the location of r2, drops the garbage there, and return to the location where it found the garbage and continues its search from that position. **Robot r2** is situated at an incinerator; whenever garbage is taken to its location by r1, r2 just puts it in the incinerator.

(1) Rafael H. Bordini, Michael Fisher, Willem Visser, and Michael Wooldridge. Verifiable multi-agent programs. M. Dastani, J. Dix, A. El Fallah-Seghrouchni (Eds.) In Proceedings of the First International Workshop on Programming Multiagent Systems: languages, frameworks, techniques and tools (ProMAS-03). LNAI 3067, pp. 72-89 Springer-Verlag Berling Heidelberg 2004.

A Simple Examples: Collecting Garbage

Agent r1

Beliefs

pos(r2, 2, 2).

checking(slots).

Plans

+pos(r1, X, Y) : checking(slots) & not garbage(r1) (p1)

<- **next(slot)**. (**simple action**)

+garbage(r1) : checking(slots) (p2)

<- !stop(check); !take(garb,r2); !continue(check).

+!stop(check) : true (p3)

<- ?pos(r1, X, Y); +pos(back, X, Y); -checking(slots).

+!take(S, L) : true (p4)

<- !ensure_pick(S); !go(L); **drop(S)**.

A Simple Examples: Collecting Garbage

Agent r1

Plans

```
+!ensure_pick(S) : garbage(r1)                                   (p5)
        <- pick(garb); !ensure_pick(S).
+!ensure_pick(S) : true <- true.                                 (p6)
+!continue(check) : true                                         (p7)
        <- !go(back); -pos(back, X, Y); +checking(slots); next(slot).
+!go(L) : pos(L, X, Y) & pos(r1, X, Y)                           (p8)
        <- true.
+!go(L) : true                                                   (p9)
        <- ?pos(L, X, Y); moveTowards(X, Y); !go(L).
```

A Simple Examples: **Collecting Garbage**

Agent r2

+garbage(r2) : true

<- **burn(garb)**.

Jason

Part 2

- Others Jason Notions
- Semantics to communication
- Complete Jason Reasoning Cycle
- Other Simple Examples
- MAS Configuration File
- Conclusion

Others Jason Notions

- Belief annotation:
  - **blue(box1)[source(ag1)].**
  - **red(box1)[source(percept)].**
  - **colourblind(ag1)[source(self), doc(0.7)].**
  - **lier(ag1)[source(self), doc(0.2)].**
- The operator '~' is used for strong negation:

  +!leave(home) : not raining & not ~raining

  <- open(curtains); …

Others Jason Notions

- Internal action:

  .desire(literal)

  .intend(literal)

  .drop_desires(literal)

  .drop_intentions(literal)

- Internal action for communication:

  .send(receiver, ilf, predicate-content) where

  ilf ∈ {tell, untell, achieve, unachieve,

  askOne, askAll, askHow, tellHow, untellHow, ...}

Semantics to communication

- **The Knowledge Query and Manipulation Language** (KQML) is a language that adds intentional context:
  - **tell** : S informs R that the sentence in the message content is true of S;
  - **untell**: the message content is not in the knowledge base of S;
  - **achieve**: S requests that R try to achieve a state of the world where the message content is true;
  - **unachieve**: S wants to revert the effect of an achieve previously sent.
  - http://jmvidal.cse.sc.edu/talks/agentcommunication/kqmlperformatives.html

MAS Configuration File

```
MAS my_system {
   infrastructure: Jade
   environment: MyEnv
   ExecuctionControl: …
   agents: ag1; ag2; ag3;
}
```

- Multiple instances of an agent

```
   agents: ag1 #10;
```

Conclusion

- Jason was implemented in Java by Rafael H. Bordini and Jomi F. Hübner, with contributions from various colleagues.
- Research in the area of agent-oriented programming languages is still incipient, so we expect much progress from research in the area.
- Jason is distributed completely on an “as is” basis.
- It has been developed during our spare time, we cannot guarantee much support.
- If you have questions or bug reports, you are welcome to use the mailing lists at SourceForge:
  - jason-announcement@lists.sourceforge.net (where we announce new releases and important news about Jason)
  - jason-users@lists.sourceforge.net (for questions about using Jason)
  - jason-bugs@lists.sourceforge.net (to report bugs)