Programming Multi-Agent Systems in AgentSpeak Using Jason

Published by John Wiley & Sons Ltd., October 2007.

http://jason.sf.net/jBook/
AgentSpeak

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

- The main language constructs of AgentSpeak are:
  - Beliefs
  - Goals
  - Plans

- The architecture of an AgentSpeak agent has four main components:
  - Belief Base
  - Plan Library
  - Set of Events
  - Set of Intentions
Syntax of AgentSpeak (Beliefs and Goals)

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

  `publisher(wiley)`

• **Goals** represent states of affairs the agent wants to bring about (come to believe, when goals are used declaratively)

  • Achievement goals:

    `!write(book)`

  Or attempts to retrieve information from the belief base

  • Test goals:

    `?publisher(P)`
Syntax of AgentSpeak (Events and Plans)

• An agent reacts to **events** by executing plans

• Events happen as a consequence to changes in the agent’s beliefs or goals

• **Plans** are recipes for action, representing the agent’s know-how

• An AgentSpeak plan has the following general structure:

  \[
  \text{triggering\_event} : \text{context} \leftarrow \text{body}.
  \]

• 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.
Syntax of AgentSpeak (Plans Cont.)

- AgentSpeak triggering events:
  - $+b$ (belief addition)
  - $-b$ (belief deletion)
  - $+!g$ (achievement-goal addition)
  - $-!g$ (achievement-goal deletion)
  - $+?g$ (test-goal addition)
  - $-?g$ (test-goal deletion)

- The context is logical expression, typically a conjunction of literals to be checked whether they follow from the current state of the belief base.

- The body is a sequence of actions and (sub) goals to achieve.

- NB: This is the original AgentSpeak syntax; *Jason* allows other things in the context and body of plans.
Jason

- *Jason* implements the operational semantics of a variant of AgentSpeak
- Various extensions aimed at a more practical programming language
- Platform for developing multi-agent systems
- Developed by Jomi F. Hübner and Rafael H. Bordini
- We’ll look at the *Jason* additions to AgentSpeak and its features
Reasoning Cycle (Steps)

1. Perceiving the Environment
2. Updating the Belief Base
3. Receiving Communication from Other Agents
4. Selecting ‘Socially Acceptable’ Messages
5. Selecting an Event
Reasoning Cycle (Steps)

6. Retrieving all Relevant Plans

7. Determining the Applicable Plans

8. Selecting one Applicable Plan

9. Selecting an Intention for Further Execution

10. Executing one step of an Intention
10. Intention Execution

a. Environment actions

b. Achievement goals

c. Test goals

d. Mental notes

e. Internal actions

f. Expressions
Example of Annotations

- An agent’s belief base with a user-defined `doc` annotation (degree of certainty)

  blue(box1)[source(ag1)].
  red(box1)[source(percept)].
  colourblind(ag1)[source(self),doc(0.7)].
  liar(ag1)[source(self),doc(0.2)].
Belief-Base Rules

• Prolog-like rules in the belief base

\[
\text{likely\_color(Obj,C)} \\
\quad :\quad \text{colour(Obj,C)[degOfCert(D1)]} \\
\quad \quad \quad \& \quad \text{not (} \\
\quad \quad \quad \quad \text{colour(Obj,\_)[degOfCert(D2)]} \\
\quad \quad \quad \quad \& \quad D2 > D1 \quad \text{)} \\
\quad \quad \& \quad \text{not \sim colour(C,B).}
\]
Handling Plan Failure

- Goal-deletion events were syntactically defined, but no semantics

- We use them for a plan failure handling mechanism (probably not what they were meant for)

- Handling plan failures is very important as agents are situated in dynamic environments

- A form of “contingency plan”, possibly to “clean up” before attempting another plan
To create an agent that is blindly committed to goal $g$:

```plaintext
+!g : g <- true.
+!g : ... <- ... ?g.
...
-!g : true <- !g.
```
Internal Actions

- Unlike actions, internal actions do not change the environment.

- Code to be executed as part of the agent reasoning cycle.

- AgentSpeak is meant as a high-level language for the agent’s practical reasoning.

- Internal actions can be used for invoking legacy code elegantly.
AgentSpeak X 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

- Direct integration with Java
Overall Agent Architecture

- Users customise the AgentArch class to change the way the agent interacts with the infrastructure: perception, action, and communication.

- Helps switching between simulation for testing and real deployment.
  
  - perceive()
  
  - act()
  
  - sendMsg()
  
  - broadcast()
  
  - checkMail()
Belief Base Customisation

- Logical belief base might not be appropriate for large applications

- Jason has an alternative belief base combined with a database

- Users can create other customisations
  - add()
  - remove()
  - contains()
  - getRelevant()
Customised MAS

MAS Custom {
agents:

a1 agentClass MyAg

agentArchClass MyAgArch

beliefBaseClass Jason.bb.JDBCPersistentBB("org.hsqldb.jdbcDriver",
"jdbc:hsqldb:bookstore",
...
"[count_exec(1,tablece)]");
}
Environments

• In actual deployment, there will normally be an environment where the agents are situated.

• As discussed earlier, the AgentArchitecture needs to be customised to get perceptions and act on such environment.

• We often want a simulated environment (e.g., to test a MAS application).

• This is done in Java by extending Jason’s Environment class and using methods such as addPercept(String Agent, Literal Percept).
Jason for jEdit

/* Initial beliefs and rules */

available(beer, fridge). // initially, I believe that there are some beer in the fridge.
limit(beer, 10). // my owner should not consume more than 10 beers a day.

too_much(B) :-
    .date(YY, MM, DD) & .count(consumed(YY, MM, DD, ..., B), QtDB) &
    limit(B, Limit) & QtDB > Limit.

/* Plans */

+!has(owner, beer) : available(beer, fridge) & not too_much(beer)
    <= !lat(robot, fridge);
        open(fridge);
        get(beer);
        close(fridge);
        !lat(robot, owner);
        hand_in(beer);
        ?has(owner, beer);
Jason’s Mind Inspector

**Inspection of agent r1 (cycle #12)**

- **Beliefs**
  - pos(back,3,0) [source (self)]
  - pos(r1,3,0) [source (percept)]
  - pos(r2,3,3) [source (percept)]
  - garbage(r1) [source (percept)]

- **Events**
  - Sel Trigger Intention
    - X +lensure_pick(garb) 4

- **Options**

- **Intentions**
  - Sel Id Pen Intended Means Stack (show details)
    - X 4 +lensure_pick(S) { S = garb }
    - +ltake(S,L) { S = garb, L = r2 }
    - +lcarry_to(R) { R = r2, Y = 0, X = 3 }
    - +garbage(r1) [source (percept)]

- **Actions**
  - Pend Feed Sel Term Result Intention
    - X X pick (garb) false 4

**Agent History**

Cycle 0 10 20 Cycle 22

Run 5 cycle(s) for all agents view as: html
Jason is available Open Source under GNU LGPL at:

http://jason.sf.net

(kindly hosted by SourceForge)
programming multi-agent systems in AgentSpeak using Jason

Rafael H. Bordini
Jomi Fred Hübner
Michael Wooldridge