Programming Intentional Agents: Exercises in *Jason*

Autonomous Systems / Technologies

Sistemi Autonomi / Tecnologie

Giovanni Ciatto Stefano Mariani Andrea Omicini

{giovanni.ciatto, s.mariani, andrea.omicini}@unibo.it

Dipartimento di Informatica – Scienza e Ingegneria (DISI)
Alma Mater Studiorum – Università di Bologna

Academic Year 2017/2018

1. Getting Started
2. Basic Exercises
3. AgentSpeak(L) Exercises
4. *Jason* Exercise: Domestic Robot
5. *Jason* Example: ContractNet Protocol

Next in Line. . .

1. Getting Started
2. Basic Exercises
3. AgentSpeak(L) Exercises
4. *Jason* Exercise: Domestic Robot
5. *Jason* Example: ContractNet Protocol

Get *Jason*

1. Go to *Jason* home page

   http://jason.sourceforge.net/wp/

2. On the right, click on "DOWNLOAD" button to go to *Jason* download page, on SourceForge

   http://sourceforge.net/projects/jason/files/

3. Click on the quick link next to *"Looking for the latest version?"* to obtain *Jason* latest version (currently, **2.3**)

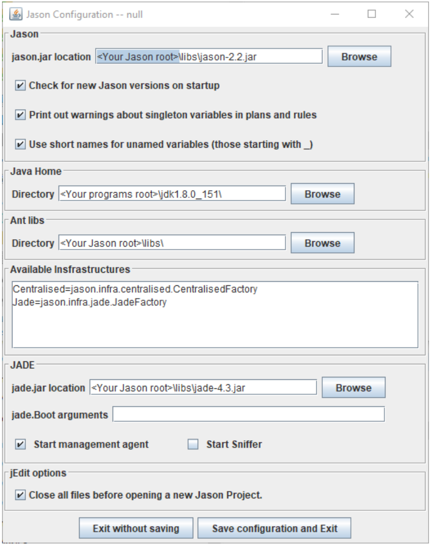
Install *Jason* I

1. *Jason* Comes with its own IDE (jEdit enhanced with *Jason* plugin), but also a Eclipse plugin exists: we will use Eclipse
2. Once downloaded *Jason* bundle, unpack it in any directory, position yourself within *Jason* directory, e.g. `jason-2.3/`
3. Run *Jason* JAR file in a command prompt, e.g.,

   ```
   java -jar libs/jason-2.3-SNAPSHOT.jar
   ```

4. A configuration window should pop-up, letting you set up the *Jason* runtime environment properties—e.g., location of *Jason* jar, available distribution infrastructures, etc.: be sure the “Java Home” field points to the JVM you want to use:

Install *Jason* II



Jason Configuration -- null

Jason

jason.jar location <Your Jason root>\libs\jason-2.2.jar

☑ Check for new Jason versions on startup

☑ Print out warnings about singleton variables in plans and rules

☑ Use short names for unamed variables (those starting with _)

Java Home

Directory <Your programs root>\jdk1.8.0_151\

Ant libs

Directory <Your Jason root>\libs\

Available Insfrastructures

```
Centralised=jason.infra.centralised.CentralisedFactory
Jade=jason.infra.jade.JadeFactory
```

JADE

jade.jar location <Your Jason root>\libs\jade-4.3.jar

jade.Boot arguments

☑ Start management agent

☐ Start Sniffer

jEdit options

☑ Close all files before opening a new Jason Project.

Install *Jason* III

5. Now open Eclipse and click "Help > Install New Software. . .' > Add..", then type in the "Location" field

   `http://jason.sourceforge.net/eclipseplugin/juno/`

   for Juno or newer

   `http://jason.sourceforge.net/eclipseplugin/`

   for Indigo

6. Click "Ok" and wait for the "jasonide" feature to appear, then tick the checkbox and step through the installation process (Eclipse restart included)

Clone/Download the exercises I

1. "File > Import > Git > Projects from Git > Clone URI " and then please paste the following URL into the "URI" field:

   `https://gitlab.com/das-lab/courses/as/jason-exercises-aa1718.git`

2. Click on "Next" until the last page is shown where a number of projects should appear
3. Select them all and then click on "Finish"
4. Ensure that *all* projects are correctly working (i.e., no errors or exclamation marks). Call the teacher otherwise

Next in Line. . .

1. Getting Started
2. Basic Exercises
3. AgentSpeak(L) Exercises
4. *Jason* Exercise: Domestic Robot
5. *Jason* Example: ContractNet Protocol

Exercise 0a – *Jason* projects & Hello World

1. Create a new *Jason* project by clicking on "New" icon on top left corner then selects "*Jason* Project"
   - Yes, you want to automatically switch to the *Jason* Perspective
2. Name the project `ex_0_helloWorld` and leave default options (centralised infrastructure without environment)
3. Two files are automatically created (filenames *do* matter: change them wisely)

   `.asl` is the *Jason* agent source file
   `.mas2j` is the *Jason* MAS configuration file

   Try to understand each line of the `.mas2j` file
4. Right-click on `.mas2j` file and "Run *Jason* Application": the MAS console window should pop-up, showing `sample_agent` printing "hello world."
5. Try to understand the purpose & functioning of each button

Exercise 0b – Basic computations I

Edit `sample_agent`, making it

- print “hello world.” infinitely many times

Exercise 0b – Basic computations I

Edit `sample_agent`, making it

- print “hello world.” infinitely many times

Edit `sample_agent`, making it

- continuously print “hello world *N*”, where *N* is a progressive integer starting from 0

Exercise 0b – Basic computations I

Edit `sample_agent`, making it

- print “hello world.” infinitely many times

Edit `sample_agent`, making it

- continuously print “hello world N”, where N is a progressive integer starting from 0

Edit `sample_agent`, making it

- print “hello world N” $M+1$ times, where N is a progressive integer starting from 0 and M is an arbitrary positive integer

Exercise 0b – Basic computations I

Edit `sample_agent`, making it

- print “hello world.” infinitely many times

Edit `sample_agent`, making it

- continuously print “hello world N”, where N is a progressive integer starting from 0

Edit `sample_agent`, making it

- print “hello world N” $M+1$ times, where N is a progressive integer starting from 0 and M is an arbitrary positive integer

Edit `sample_agent`, making it

- print “hello world N E” $M+1$ times, where N is a progressive integer starting from 0, M is an arbitrary positive integer, and E is “even” if N is even, “odd” otherwise

Exercise 0c – Computation I

1. Create a new agent by means of the Eclipse wizard
   - Right-click on the `src/asl` directory > New > Agent
   - Select the `ex_0_helloWorld` project
   - Name the agent "mathAgent"
   - Click on Finish
2. Look at *all* project files: how many of them changed? Why?
3. Edit the agent making it able to compute the Fibonacci sequence, according to the following constraints

$$\begin{aligned} F_0 &= 1 \\ F_1 &= 1 \\ F_n &= F_{n-1} + F_{n-2} \end{aligned}$$

Exercise 0c – Computation II

mathAgent constraints

- It initially believes that `fib(0, 1)` and `fib(1, 1)`
- Its initial achievement plan is `!compute_fibonacci_until(0, 100)`
  - Computing the Fibonacci sequence values from 0 to 100 (incl.)
  - Printing each step on a different line
  - Printing `‘‘Done!’’` when done
- It should take advantage of the `!compute_fibonacci(N)` achievement goal taking care of each step

! It may leverage on test goals

! It may leverage on rules

! It may exploit the BB in several ways

Exercise 0c – Computation III

Edit `mathAgent` as described below

- No initial achievement goal must be provided by the programmer
- The `!compute_fibonacci_until(0, Whatever)` achievement goal must be provided dynamically by another agent

! Maybe the REPL agent

! Maybe exploiting the “.send” internal action

  - `http://jason.sourceforge.net/api/jason/stdlib/send.html`

Next in Line. . .

1. Getting Started
2. Basic Exercises
3. AgentSpeak(L) Exercises
4. *Jason* Exercise: Domestic Robot
5. *Jason* Example: ContractNet Protocol

Exercise 1 – AgentSpeak(L) vs *Jason* I

- Consider now project `ex_1_agentSpeakL_basics`
- It is a very basic example of AgentSpeak(L) program featuring
  - achievement-goal addition events
  - belief addition events
  - plan contexts
  - test-goals
- Can you spot what should *not* be in an AgentSpeak(L) program?
  - ! Notice that this is a valid *Jason* program. . .
- Listen to the teacher explaining the source code

Exercise 1 – AgentSpeak(L) vs *Jason* II

Consider the guard of the plan for `+!book_tickets(A,D,V)`

- What is its meaning?
- Change it in such a way the plan can be selected if the phone is *certainly* not busy

Exercise 1 – AgentSpeak(L) vs *Jason* III

Edit the MAS as described below

- The phone is shared by `fanboy` and its `mum`: they may compete for using the phone
- `mum` simply randomly tries to use the phone to call her friends
  - When she gets the phone she keeps talking for a while
  - `http://jason.sourceforge.net/api/jason/stdlib/random.html`
- Every agent getting the phone must first of all inform the other one that the phone is busy
  - `http://jason.sourceforge.net/api/jason/stdlib/send.html`
- Every agents must inform the other one that the phone is not busy anymore when it ends the call
- Every agent must wait for the phone to be free in case it is needed
- The phone is initially free

Next in Line. . .

1. Getting Started
2. Basic Exercises
3. AgentSpeak(L) Exercises
4. *Jason* Exercise: Domestic Robot
5. *Jason* Example: ContractNet Protocol

Exercise 2 – Domestic Robot

1. Consider now project `ex_2_domesticRobot`
   - a domestic robot has the goal of serving beer to its owner
   - thus, it receives beer requests from the owner, goes to the fridge, takes out a beer, brings it back to the owner
   - the robot eventually orders beer using a nearby supermarket's home delivery service
   - also, the robot obeys hard-wired rules from the Department of Health (e.g. "do not serve more than 10 beers a day")
2. Listen to the teacher explaining the source code

Highlights

- The robot should remember if beer is available irrespective of its location in the environment, because the perception about beer stock is available *only when the robot is in front of the open fridge*—as soon as it closes the fridge, the perception is gone
- How to ensure that the robot will respond *only* to its owner's requests?
- What happens if we change the order of `+!at(robot, P)` plans? And if we drop the *plan context* from either one? Again, what happens if we change plans order in this case?
- How to ensure each external action is available only to the right agent?

Next in Line. . .

1. Getting Started
2. Basic Exercises
3. AgentSpeak(L) Exercises
4. *Jason* Exercise: Domestic Robot
5. *Jason* Example: ContractNet Protocol

Exercise 3 – ContractNet protocol

1. Consider now project `ex_3_contractNetProtocol`
    - one agent, called “initiator”, wishes to have some tasks performed, thus asks other agents, called “participants”, to bid to perform that task
    - this asking message is called “call for proposals” (cfp, for short), and participants may reply by either sending their proposals or by refusing the call
    - it can happen that participants do not even reply, so when a deadline chosen by the initiator expires, it evaluates the received proposals and selects one agent to perform the task
2. Listen to the teacher explaining the source code

Highlights

- Note that plan `@contracting` is `atomic`: when it starts executing, *no other intention is selected for execution* before it finishes:
    - the first action of the plan is to change the protocol state, which is also used in the context of the plan
    - this ensures that the intention for the goal `!contract` is never performed twice
- Suppose the initiator wants to cancel the CFP
    - add a plan in the initiator program for events such as `+!abort(CNPId)`
    - which kind of *illocutionary force* (or performative) may this plan exploit to inform participants accordingly?

hint basically, we want to remove an agent's belief from another agent...

1. Getting Started
2. Basic Exercises
3. AgentSpeak(L) Exercises
4. *Jason* Exercise: Domestic Robot
5. *Jason* Example: ContractNet Protocol

Programming Intentional Agents: Exercises in *Jason*

Autonomous Systems / Technologies

Sistemi Autonomi / Tecnologie

Giovanni Ciatto Stefano Mariani Andrea Omicini

{giovanni.ciatto, s.mariani, andrea.omicini}@unibo.it

Dipartimento di Informatica – Scienza e Ingegneria (DISI)
Alma Mater Studiorum – Università di Bologna

Academic Year 2017/2018