AgentSpeak(L) and Jason: Environment & Agent Interaction

Laboratory of Multiagent Systems LM
Laboratorio di Sistemi Multiagente LM

Elena Nardini
elena.nardini@unibo.it

Ingegneria Due
Alma Mater Studiorum—Università di Bologna a Cesena

Academic Year 2010/2011
1. Environment

2. Agent Interaction

3. Example: Domestic Robot

4. Exercises
   - Exercise 1
   - Exercise 2

5. Conclusion
Agents are Situated

- Autonomous agents live *situated* in an environment
- In MAS, the environment is shared by multiple agents, so an agent’s actions are likely to interfere with those of other agent

→ Having an explicit notion of environment, although not mandatory, is an important aspect in MAS developing
There are two ways to design and implement the MAS environment:

1. Defining perceptions and actions so to operate on specific environments
   - This is done defining in Java lower-level mechanisms, and by specializing the Agent Architecture and Agent classes

2. Creating a ‘simulated’ environment
   - This is done in Java by extending Jason’s Environment class and using methods such as addPercept(String Agent, Literal Percept)
Existent Agent architecture uses the `getPercepts` method to retrieve, form the simulated environment, the *percepts* to which that particular agent currently has access.
Agent-Environment Interaction

Actions

- When an intention is executed and the formula being executed is an environment action
- For each action execution request, the agent architecture invokes the `executeAction` method of the environment, and resumes the respective intention when the method returns (true or false)
In order to implement an environment, programmers need to extend the `Environment` class and likely to override the `executeAction` and `init` methods.
import jason.*;
import ...;

public class <EnvironmentName> extends Environment{
    // any class members needed...

    @Override
    public void init(String[] args) {
        // setting initial (global) percepts...
        addPercept(Literal.parseLiteral("p(a)"));
        // if this is to be perceived only by agent ag1
        addPercept("ag1", Literal.parseLiteral("p(a)"));
    }

    @Override
    public boolean stop() {
        // anything else to be done by the environment when
        // the system is stopped...
    }

    ...

Example of a User Environment

```java
...  

@Override
public boolean executeAction(String ag, Term action) {
    if (action.equals(...)) {
        addPercept("ag1", Literal.parseLiteral("p(b)");
    }
    ...
    return true;
}
```
Agents are Social

- Autonomous agents live and interact within agent societies & MAS
- Since agents are autonomous, only data (knowledge, information) crosses agent boundaries
Agent Interaction in Jason

Receiving Messages

- At the beginning of each reasoning cycle, agents check for messages they might have received from other agents.
- Any message received by the checkMail method has the structure: 
  \(<\text{sender, illoc, force, content}>\)
Agent Interaction in Jason

Sending Messages

- Messages are sent with the use (in plan bodies) of a special pre-defined internal action
- The general form of such an internal action is:
  \[ .send(\text{receiver}, \text{illocutionary force}, \text{propositional content}) \]

Performatives

- **tell** \( s \) intends \( r \) to believe \( c \) to be true
- **untell** \( s \) intends \( r \) not to believe \( c \) to be true
- **achieve** \( s \) intends \( r \) to try and achieve \( c \)
- **unachieve** \( s \) intends \( r \) to drop the goal \( c \)
- **askOne** \( s \) wants to know if \( c \) is true for \( c \)
- **askAll** \( s \) wants all of \( r \)'s answers to a question
- **tellHow** \( s \) informs \( r \) of a plan
- **untellHow** \( s \) requests that \( r \) discard a certain plan
- **askHow** \( s \) wants all of \( r \)'s plans that are relevant for the triggering event \( c \)
Domestic Robot Environment

Pattern Model-View-Control

- Its design is based on a common object-oriented design pattern: 
  *Model-View-Control* (MVC)

- The environment design is thus based on the following three components:
  - **model**: maintains the information about the environment state and the dynamics of the environment
  - **view**: renders the model into a form suitable for visualisation
  - **control**: interacts with the agents and invokes changes in the model and perhaps the view
Modelling the Environment

Percepts

- \text{at}(\text{robot}, \text{Place}). Only two places are perceived, fridge and owner. Thus, depending on its location in the house, the robot will perceive either \text{at}(\text{robot}, \text{fridge}) or \text{at}(\text{robot}, \text{owner})

- \text{stock}(\text{beer}, N). When the fridge is open, the robot will perceive how many beers are stored in the fridge

- \text{has}(\text{owner}, \text{beer}). It is perceived by the robot and the owner when the owner has a (non-empty) bottle of beer
Modelling the Environment

- The model of the Domestic Robot environment should maintain:
  - the number of available beers in the fridge (attribute `availableBeers`)
  - whether the owner currently has a bottle of beer (the percept `has(owner, beer)` is based on the `sipCount` value)
  - the robot’s location (the location is maintained through the use of the class `GridWorldModel` modelling an $n \times m$ grid)
Example: Domestic Robot

HouseEnv

```java
import jason.asSyntax.*;

public class HouseEnv extends Environment {

    // common literals
    public static final Literal of = Literal.parseLiteral("open(fridge)");
    public static final Literal clf = Literal.parseLiteral("close(fridge)");
    public static final Literal gb = Literal.parseLiteral("get(beer)");
    public static final Literal hb = Literal.parseLiteral("hand_in(beer)");
    public static final Literal sb = Literal.parseLiteral("sip(beer)");
    public static final Literal hob = Literal.parseLiteral("has(owner,beer)");
    public static final Literal af = Literal.parseLiteral("at(robot,fridge)");
    public static final Literal ao = Literal.parseLiteral("at(robot,owner)");

    HouseModel model; // the model of the grid

    @Override
    public void init(String[] args) {
        model = new HouseModel();

        if (args.length == 1 && args[0].equals("gui")) {
            HouseView view = new HouseView(model);
            model.setView(view);
        }

        updatePercepts();
    }

    public void step() {
        // action
        updatePercepts();
    }
}
```
/** creates the agents percepts based on the HouseModel */

```c
void updatePercepts()
{
    // clear the percepts of the agents
    clearPercepts("robot");
    clearPercepts("owner");

    // get the robot location
    Location lRobot = model.getAgPos(0);

    // add agent location to its percepts
    if (lRobot.equals(model.lFridge))
    {
        addPercept("robot", af);
    }
    if (lRobot.equals(model.lOwner))
    {
        addPercept("robot", ao);
    }

    // add beer "status" to the percepts
    if (model.fridgeOpen)
    {
        addPercept("robot", Literal.parseDouble("stock(beer," + model.availableBeers + ")");
    }
    if (model.sipCount > 0)
    {
        addPercept("robot", hob);
        addPercept("owner", hob);
    }
}
```
@Override
public boolean executeAction(String ag, Structure action)
{
    System.out.println("[" + ag + "] doing: " + action);
    boolean result = false;

    if (action.equals(of))
    {
        // of = open(fridge)
        result = model.openFridge();
    }
    else if (action.equals(clf))
    {
        // clf = close(fridge)
        result = model.closeFridge();
    }
    else if (action.getFunctor().equals("move_towards"))
    {
        String l = action.getTerm(0).toString();
        Location dest = null;
        if (l.equals("fridge"))
        {
            dest = model.lFridge;
        }
        else if (l.equals("owner"))
        {
            dest = model.lOwner;
        }
Example: Domestic Robot

HouseEnv

```java
try {
    result = model.moveTowards(dest);
} catch (Exception e) {
    e.printStackTrace();
}
else if (action.equals(gb)) {
    result = model.getBeer();
} else if (action.equals(hb)) {
    result = model.handInBeer();
} else if (action.equals(sb)) {
    result = model.sipBeer();
}
```
else if (action.getFunctor().equals("deliver"))
{
    // wait 4 seconds to finish "deliver"
    try {
        Thread.sleep(4000);
    } catch (Exception e) {
    }
    result = model.addBeer((int) ((NumberTerm) action.getTerm(1)).solve());
} else {
    System.err.println("Failed to execute action " + action);
}

if (result) {
    updatePercepts();
    try {
        Thread.sleep(100);
    } catch (Exception e) {
    }
}
return result;
import jason.environment.grid.GridWorldModel;

/** class that implements the Model of Domestic Robot application */
public class HouseModel extends GridWorldModel {
    // constants for the grid objects
    public static final int FRIDGE = 16;
    public static final int OWNER = 32;
    // the grid size
    public static final int GSize = 7;

    boolean fridgeOpen = false; // whether the fridge is open
    boolean carryingBeer = false; // whether the robot is carrying beer
    int sipCount = 0; // how many sip the owner did
    int availableBeers = 2; // how many beers are available

    Location lFridge = new Location(0, 0);
    Location lOwner = new Location(GSize - 1, GSize - 1);
public HouseModel()
{
    // create a 7x7 grid with one mobile agent
    super(GSize, GSize, 1);

    // initial location of robot (column 3, line 3)
    // ag code 0 means the robot
    setAgPos(0, GSize / 2, GSize / 2);

    // initial location of fridge and owner
    add(FRIDGE, lFridge);
    add(OWNER, lOwner);
}

boolean openFridge()
{
    if (!fridgeOpen)
    {
        fridgeOpen = true;
        return true;
    }
    else
    {
        return false;
    }
}
HouseModel

```java
boolean closeFridge()
{
    if (fridgeOpen)
    {
        fridgeOpen = false;
        return true;
    }
    else
    {
        return false;
    }
}

boolean moveTowards(Location dest)
{
    Location r1 = getAgPos(0);
    if (r1.x < dest.x)
        r1.x++;
    else if (r1.x > dest.x)
        r1.x--;
    if (r1.y < dest.y)
        r1.y++;
    else if (r1.y > dest.y)
        r1.y--;
    setAgPos(0, r1); // move the robot in the grid

    // repaint the fridge and owner locations
    view.update(lFridge.x, lFridge.y);
    view.update(lOwner.x, lOwner.y);
    return true;
}
```
HouseModel

```java
boolean getBeer()
{
    if (fridgeOpen && availableBeers > 0 && !carryingBeer)
    {
        availableBeers--;  // Decrease the number of available beers
        carryingBeer = true;  // Set as carrying beer
        view.update(lFridge.x, lFridge.y);  // Update the view
        return true;
    }
    else
    {
        return false;
    }
}

boolean addBeer(int n)
{
    availableBeers += n;  // Increase the number of available beers
    view.update(lFridge.x, lFridge.y);  // Update the view
    return true;
}
```
boolean handInBeer() {
    if (carryingBeer) {
        sipCount = 10;
        carryingBeer = false;
        view.update(owner.x, owner.y);
        return true;
    } else {
        return false;
    }
}

boolean sipBeer() {
    if (sipCount > 0) {
        sipCount--;
        view.update(owner.x, owner.y);
        return true;
    } else {
        return false;
    }
}
import jason.environment.grid.*;

/** class that implements the View of Domestic Robot application */
@SuppressWarnings("serial")
public class HouseView extends GridWorldView
{
    HouseModel hmodel;

    public HouseView(HouseModel model)
    {
        super(model, "Domestic Robot", 700);
        hmodel = model;
        defaultFont = new Font("Arial", Font.BOLD, 16); // change default font
        setVisible(true);
        repaint();
    }
}
```java
/** draw application objects */
@Override
public void draw(Graphics g, int x, int y, int object) {
    Location lRobot = hmodel.getAgPos(0);
    super.drawAgent(g, x, y, Color.lightGray, -1);
    switch (object) {
        case HouseModel.FRIDGEG:
            if (lRobot.equals(hmodel.lFridge)) {
                super.drawAgent(g, x, y, Color.yellow, -1);
            }
            g.setColor(Color.black);
            drawString(g, x, y, defaultFont, "Fridge (" + hmodel.availableBeers + ")");
            break;
        case HouseModel.OWNER:
            if (lRobot.equals(hmodel.lOwner)) {
                super.drawAgent(g, x, y, Color.yellow, -1);
            }
            String o = "Owner";
            if (hmodel.sipCount > 0) {
                o += " (" + hmodel.sipCount + ")";
            }
            g.setColor(Color.black);
            drawString(g, x, y, defaultFont, o);
            break;
    }
}
```
```java
@override
public void drawAgent(Graphics g, int x, int y, Color c, int id)
{
    Location lRobot = hmodel.getAgPos(0);
    if (!lRobot.equals(hmodel.lOwner) && !lRobot.equals(hmodel.lFridge))
    {
        c = Color.yellow;
        if (hmodel.carryingBeer)
            c = Color.orange;
        super.drawAgent(g, x, y, c, -1);
        g.setColor(Color.BLACK);
        super.drawString(g, x, y, defaultFont, "Robot");
    }
}
```
MAS domestic_robot {
    environment: HouseEnv(gui)
    agents: robot;
           owner;
           supermarket agentArchClass SupermarketArch;
}
Outline

1. Environment
2. Agent Interaction
3. Example: Domestic Robot
4. Exercises
   - Exercise 1
   - Exercise 2
5. Conclusion
Thermostat Agent with the Environment

Requirements

- Check the environment temperature $T$.
- Until $T$ is not: $> 18$ and $< 22$:
  - Decrease $T$ of one unit if the temperature is 22
  - Increase $T$ of one unit if the temperature is 18

Constraint

- Only one agent: `thermostat.asl`
- Environment modelled with the MVC pattern: `RoomModel`, `RoomView` and `RoomEnv` classes
- `RoomModel` has to contain the class `TempMaker` extending the class `Thread` that changes the environment temperature with a random value
Thermostat Agent with the Environment

- Thermostat Agent
- MAS Console - thermostatAgent
- [thermostat] temperatura troppo alta 25
- [thermostat] doing: set_temp(24)
- [thermostat] temperatura troppo alta 24
- [thermostat] doing: set_temp(23)
- [thermostat] temperatura troppo alta 23
- [thermostat] doing: set_temp(22)
- [thermostat] temperatura troppo alta 22
- [thermostat] doing: set_temp(21)
- [thermostat] temperatura perfetta 21
- [Temperature Maker] doing: set temperature to 26
- [thermostat] temperatura troppo alta 26
- [thermostat] doing: set_temp(25)
- [thermostat] temperatura troppo alta 25
- [thermostat] doing: set_temp(24)
- [thermostat] temperatura troppo alta 24
- [thermostat] doing: set_temp(23)
- [thermostat] temperatura troppo alta 23
- [thermostat] doing: set_temp(22)
- [thermostat] temperatura troppo alta 22
Outline

1. Environment
2. Agent Interaction
3. Example: Domestic Robot
4. Exercises
   - Exercise 1
   - Exercise 2
5. Conclusion
Thermostat Agent with Agent Interaction

A New Constraint

- Two interacting agents: `thermostat.asl` and `manager.asl`
  - `thermostat` senses the temperature and sends the temperature to `manager` and sets the new temperature when it is received from `manager`
  - `manager` checks the temperature and sends the new temperature to set to `thermostat`
Thermostat Agent with Agent Interaction

[thermostat] invio la temperatura al manager 25
[manager] temperatura troppo alta 25
[thermostat] cambio la temperatura in 24
[thermostat] doing: set_temp(24)
[thermostat] invio la temperatura al manager 24
[manager] temperatura troppo alta 24
[thermostat] cambio la temperatura in 23
[thermostat] doing: set_temp(23)
[thermostat] invio la temperatura al manager 23
[manager] temperatura troppo alta 23
[thermostat] cambio la temperatura in 22
[thermostat] doing: set_temp(22)
[thermostat] invio la temperatura al manager 22
[manager] temperatura troppo alta 22
[thermostat] cambio la temperatura in 21
[thermostat] doing: set_temp(21)
[thermostat] invio la temperatura al manager 21
[manager] temperatura perfetta 21
[Temperature Maker] doing: set temperature to 19
[thermostat] invio la temperatura al manager 19
[manager] temperatura perfetta 19
Conclusion

Questions

- Centralised or distributed Agents?
- Non-simulated Environment?
AgentSpeak(L) and Jason: Environment & Agent Interaction
Laboratory of Multiagent Systems LM
Laboratorio di Sistemi Multiagente LM

Elena Nardini
elena.nardini@unibo.it

Ingegneria Due
ALMA MATER STUDIORUM—Università di Bologna a Cesena

Academic Year 2010/2011