From Objects to Agents: The Java Agent Middleware (JAM)

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. Defining Agents

2. Objects vs. Agents [Odell, 2002]

3. An Agent Middleware

4. Exercise

5. Bibliography
We will use *autonomy* as the only fundamental and definitely feature of agents.

### Computational Autonomy

- Agents are autonomous as they *encapsulate* (the thread of) *control*.
- Control does not pass through agent boundaries.
  - Only data (knowledge, information) crosses agent boundaries.
- Agents have no interface, cannot be controlled, nor they can be invoked.
- Looking at agents, MAS can be conceived as an aggregation of multi distinct *loci* of control interacting with each other by exchanging information.
“Weak” Notion of Agent

Four key qualities [Wooldridge and Jennings, 1995]

- Autonomy
- Pro-activity
- Reactivity (to change)
- Sociality
Object-Oriented Programming

- The basic unit of software are objects & classes
- Structured units of code could actually be reused under a variety of situations
- Objects have local control over variables manipulated by their own methods
  - variable state is persistent through subsequent invocations
  - object’s state is encapsulated
- Objects are passive—methods are invoked by external entities
  - modularity does not apply to unit invocation
  - object’s control is not encapsulated
Agent-Oriented Programming

- The basic unit of software are agents
  - encapsulating everything, in principle
    - by simply following the pattern of the evolution
  - whatever an agent is
    - we do not need to define them now, just to understand their desired features
- Agents could in principle be reused under a variety of situations
- Agents have control over their own state
- Agents are active
  - they cannot be invoked
  - agent's control is encapsulated
Toward an Agent Middleware

Middleware for the paradigm shift

- How are we going to *implement the paradigm shift*, under the heavy weight of legacy?
- A middleware allows agents to be used in conjunction with sub-systems adopting different component models
JAM: Java Agent Middleware

Developed by Prof. Matteo Baldoni for the course of “Programmazione in Rete” at the Politecnico di Torino

JAM Architecture

- Supports the agent creation and interaction by exploiting the Java technology that is based on the object paradigm
- Is based on Java RMI and the Agent Directory Service Layer (ADSL)
On the one hand, an agent is an object, instance of the `JAMAgent` class, describing their internal state and providing the communicative functionality.

On the other hand, unlike objects, agents are autonomous.

→ The **object** abstraction needs to be coupled with the **process** abstraction in order to model the **agent** abstraction.
JAM Agents are Autonomous

Computational autonomy

- The behaviour of an agent is defined by a set of objects of kind `JAMBehaviour`
- Each behaviour is executed by a single Java thread
JAM Agents are Pro-active

Pro-activity

- JAMBehaviour implements the method `action` accessing the agent state and the communicative functionality, and representing the actions defining the agent behaviour.

- The method action is executed once if the class implementing the method is `JAMSimpleBehaviour` (→ task-oriented).

- The method action is executed in a cyclic way until the value of the boolean variable `done` is not equals to `true`, if the class implementing the method is `JAMWhileBehaviour` (→ goal-oriented, where the goal is: `done = true`).

<table>
<thead>
<tr>
<th>JAMBehaviour</th>
</tr>
</thead>
<tbody>
<tr>
<td>+done()</td>
</tr>
<tr>
<td>+isDone()</td>
</tr>
<tr>
<td>+sleep()</td>
</tr>
<tr>
<td>+action()</td>
</tr>
</tbody>
</table>
Situated Agents and Reactive

Situatedness

- The agent environment is represented by the JVM and the *Runtime Agent Middleware* (RAM)
- RAM is built upon the Java RMI infrastructure and ADSL and it allows to host, find and execute agents, and exchange messages among agents
  -> JAM does not provide any first class abstraction to model the agent environment
  -> The agent environment is not explicitly modelled

Reactivity (to change)

- Agents can wait for something to happen
  - A message received from another agent
  - An event from the JVM
  - Example: exceptions
JAM Agents are Social

Sociality
- As MAS subsystems, agents can interact with one another
- JAM supports direct agent interaction through the runtime environment RAM

Agent interaction through RAM
- Agents can be distributed over the network
- Agents can interact by writing messages in message boxes by exploiting Java RMI
- Each agent has a message box associated
- A message and a message box are respectively represented by the objects Message and MessageBox
Messages and Message Boxes in JAM

**Message**
- `performative` : Performative
- `content` : String
- `extraArgument` : Object
- `sender` : AgentID
- `receiver` : AgentID

```
+Message()
+getSender()
+setSender()
+getReceiver()
+setReceiver()
+getPerformative()
+setPerformative()
+getContent()
+setContent()
+getExtraArgument()
+setExtraArgument()
+toString()
```

**MessageBox**

```
+MessageBox()
+readMessage()
+isThereMessage()
+writeMessage()
```
An Agent Middleware

Agent Directory Service Layer I

**Discovery agents to interact with**

- RAM provides ADSL that is exploited to collect information about the agents of the MAS in order to interact with one another
- ADSL is like *white pages* and it is remotely accessible

```java
java.rmi.server.UnicastRemoteObject
  \[\text{ADSL}\]
  + getRemoteMessagingBoxes()
  + insertRemoteMessagingBox()
  + removeRemoteMessagingBox()

ADSLImpl
- remoteMessagingBoxes : List<RemoteMessagingBox>
  + getRemoteMessagingBoxes()
  + insertRemoteMessagingBox()
  + removeRemoteMessagingBox()
```
Exploiting the ADSL service

- ADSL lookup by each agent
- Each agent has to register the remote object `MessageBox` in the ADSL
- In order to send a message
  - first, an agent search the remote message box of the agent which is the intended receiver of the message
  - then, the sender agent sends the message writing it in the remote message box
Exercise “Che ora è?”

- Download the library JAM.jar
- Download the JAM documentation JAMLibrary.pdf
- Implement the exercise “Che ora è?” provided in section 2.3
Open Points

Questions

- Which differences with an implementation exploiting the object paradigm?
- Environment not explicitly modelled from the middleware: which are the drawbacks?
- Direct communication among agents: which are the drawbacks?
Bibliography


From Objects to Agents: The Java Agent Middleware (JAM)

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