Jason Agents in CArtAgO Working Environments
(The slides are partially taken from slides created by Prof. Alessandro Ricci)

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 Modelling an Environment in Jason
   - An Environment Model
   - Environment Model Implementation
   - Environment Model in Jason
     - Using CArtAgO Artifacts from Jason Agents

2 Exercises
   - Exercise 1
   - Exercise 2

3 Conclusion
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 specialising 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)`

Today we follow the option one. Thus, we need:

- An environment model: A&A model
- An implementation of such a model: CArtAgO
- An integration with Jason: CArtAgO for Jason
Outline

1 Modelling an Environment in Jason
   - An Environment Model
   - Environment Model Implementation
   - Environment Model in Jason
     - Using CArtAgO Artifacts from Jason Agents

2 Exercises
   - Exercise 1
   - Exercise 2

3 Conclusion
Agent & Artifact Model

Basic Concepts

1. **Agents**
   - Autonomous, goal-oriented and pro-active entities
   - Create and co-use artifacts for supporting their activities, besides direct communication

2. **Artifacts**
   - Non-autonomous, function-oriented entities; controllable and observable from agents
   - Modelling the tools and resources used by agents, designed by MAS programmers

3. **Workspaces**
   - Grouping agents & artifacts
   - Defining the topology of the computational environment
Environment in A&A

- Is called **Work Environment**
- Is composed by
  - Artifacts
  - Workspaces
Modelling an Environment in Jason

An Environment Model

Artifact Computational Model

[Diagram showing an Artifact Computational Model with observable events, properties, usage interface, operation interfaces, and a link interface.]
**Interaction Model: Use**

- **use action**: acting on operation controls to trigger operation execution

![Interaction Model Diagram]

- OBSERVABLE EVENTS
- GENERATION: <EvName, Params>
- OBSERVABLE PROPERTIES
- Link Interface
- Operation X
- Operation Y
- Usage Interface
- ObsPropName Value
- ObsPropName Value
- OpControlName(Params)
- OpControlName(Params)
- ...
Interaction Model: Use

- **Operation execution** makes observable effects:
  - Observable events & changes in observable properties
  - Perceived by agents either as external events
Interaction Model: Observation

- `observeProperty` action: value of an observable property as action feedback
Interaction Model: Observation

- **focus / stopFocus** action
  - start / stop a continuous observation of an artifact (possibly specifying filters)
  - observable properties and events are mapped into percepts
Outline

1. Modelling an Environment in Jason
   - An Environment Model
   - Environment Model Implementation
   - Environment Model in Jason
     - Using CArtAgO Artifacts from Jason Agents

2. Exercises
   - Exercise 1
   - Exercise 2

3. Conclusion
CArtAgO

CArtAgO Platform / Infrastructure

- Runtime environment for executing (possibly distributed) artifact-based environments
- Java-based programming model for defining artifacts
- Set of basic API for agent platforms to work within artifact-based environment

Open-source technology

- Available in http://cartago.sourceforge.net/
  - It is possible to download the last version cartago-2.0.1.zip
  - A Getting Started is available for the deployment
  - A CArtAgO by Examples is available to learn CArtAgO
  - Additional documentation...
Outline

1 Modelling an Environment in Jason
   - An Environment Model
   - Environment Model Implementation
   - Environment Model in Jason
     - Using CArtAgO Artifacts from Jason Agents

2 Exercises
   - Exercise 1
   - Exercise 2

3 Conclusion
Defining CArtAgO Artifacts

- Single class extending Artifact
- Specifying the operations
  1. atomic @OPERATION methods
     - name + params → usage interface control
     - no return value
  2. init operation
     - automatically executed when the artifact is created
Example 1

public class Count extends Artifact {
    int count;

    @OPERATION void init()
    {
        count = 0;
    }

    @OPERATION void inc()
    {
        count++;
    }
    ...
}
Artifact Observable Events

- Generated by the primitive `signal`
- Represented as labelled tuples
- Automatically made observable to the
  - agent who executed the operation
  - all the agents observing the artifact
public class Count extends Artifact {
    int count;

    @OPERATION void init()
    {
        count = 0;
    }

    @OPERATION void inc()
    {
        count++;
        signal("new_count_value", count);
    }

    ...
}
Artifact Observable Properties

Observable Properties

- Declared by the primitive `defineObsProperty`
- Internal primitives to read / update property value
  - `updateObsProperty`
  - `getObsProperty`
- Automatically made observable to all the agents observing the artifact
public class Count extends Artifact {

    @OPERATION void init()
    {
        defineObsProperty("count", 0);
    }

    @OPERATION void inc()
    {
        int count = getObsProperty("count");
        updateObsProperty("count", count + 1);
    }

    ...
}
CArtAgO Artifact: Clock

```java
package c4jexamples;

import cartago. *;

public class Clock extends Artifact {

    boolean counting;
    final static long TICK_TIME = 100;

    void init(){
        counting = false;
    }

    @OPERATION void start(){
        if (!counting){
            counting = true;
            execInternalOp("count");
        } else {
            failed("already_counting");
        }
    }

    @OPERATION void stop(){
        counting = false;
    }

    @INTERNAL_OPERATION void count(){
        while (counting){
            signal("tick");
            await_time(TICK_TIME);
        }
    }
}
```
example-clock.mas2j

```plaintext
MAS example_clock {
    environment: c4jason.CartagoEnvironment
    agents:
        clock_user agentArchClass c4jason.CAgentArch;
    classpath: "/lib/cartago.jar";"/lib/c4jason.jar";
}
```
Modelling an Environment in Jason

Environment Model in Jason

Elena Nardini (Università di Bologna)

Jason & CArtAgO
A.Y. 2010/2011

clock_user.asl

!test_clock.

+!test_clock
  <- makeArtifact("myClock","c4jexamples.Clock",[],Id);
  focus(Id);
  +n_ticks(0);
  start;
  println("clock started.").

@plan1
+tick: n_ticks(10)
  <- stop;
  println("clock stopped.").

@plan2 [atomic]
+tick: n_ticks(N)
  <- +n_ticks(N+1);
  println("tick perceived!").
Result

```
[ clock_user ] clock started.
[ clock_user ] tick perceived!
[ clock_user ] tick perceived!
[ clock_user ] tick perceived!
[ clock_user ] tick perceived!
[ clock_user ] tick perceived!
[ clock_user ] tick perceived!
[ clock_user ] tick perceived!
[ clock_user ] tick perceived!
[ clock_user ] tick perceived!
[ clock_user ] tick perceived!
[ clock_user ] tick perceived!
[ clock_user ] tick perceived!
[ clock_user ] tick perceived!
[ clock_user ] tick perceived!
[ clock_user ] tick perceived!
[ clock_user ] tick perceived!
[ clock_user ] tick perceived!
[ clock_user ] tick perceived!
[ clock_user ] tick perceived!
[ clock_user ] tick perceived!
[ clock_user ] tick perceived!
[ clock_user ] tick perceived!
[ clock_user ] tick perceived!
[ clock_user ] clock stopped.
```
Outline

1 Modelling an Environment in Jason
   - An Environment Model
   - Environment Model Implementation
   - Environment Model in Jason
     - Using CArtAgO Artifacts from Jason Agents

2 Exercises
   - Exercise 1
   - Exercise 2

3 Conclusion
Thermostat Agent with CArtAgO4Jason

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
Thermostat Agent with CArtAgO4Jason

**Constraint**

- **ThermostatGUI.java** represents the Artifact Thermostat
  - Use the example 07a in the CArtAgO distribution to create a GUI Artifact
  - Use the primitive `await_time()` in order to periodically change the environment temperature (example 06)

- There are two agents:
  - `thermostat_maker.asl` creates the artifact `thermostat_gui`
  - `thermostat_agent.asl` interact with `thermostat_gui` to sense and change the temperature
    - it can obtain `thermostat_gui` through the external action `lookupArtifact` (example 01)
Outline

1 Modelling an Environment in Jason
   • An Environment Model
   • Environment Model Implementation
   • Environment Model in Jason
     • Using CArtAgO Artifacts from Jason Agents

2 Exercises
   • Exercise 1
   • Exercise 2

3 Conclusion
New Constraints

- There are three agents:
  - `thermostat_maker.asl` creates the artifact `thermostat_gui`
  - `thermostat_agent.asl` interact with `thermostat_gui` to sense and change the temperature
  - `manager_agent.asl` interact `thermostat_agent.asl` to change the temperature if it is needed

- `thermostat_agent.asl` and `manager_agent.asl` interact with the artifact `TupleSpace`, provided by CArtAgO (example 05a)
Conclusion

Questions

- Centralised or distributed Agents?
- Direct or mediated interactions?
Conclusion

Jason Agents in CArtAgO Working Environments
(The slides are partially taken from slides created by Prof. Alessandro Ricci)

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