Programming Multiagent Systems with JaCaMo — Environment Part —

Olivier Boissier\textsuperscript{1}, Rafael H. Bordini\textsuperscript{2},

Jomi F. Hübner\textsuperscript{3}, Alessandro Ricci\textsuperscript{4}

\textsuperscript{1}EMSE, France
Olivier.Boissier@emse.fr

\textsuperscript{2}PUC-RS, Brazil
R.Bordini@pucrs.br

\textsuperscript{3}DAS-UFSC, Brazil
jomi@das.ufsc.br

\textsuperscript{4}University of Bologna, Italy
a.ricci@unibo.it

WESAAC 2014 — Porto Alegre
Outline of the course

- Introduction
- **Agent** Oriented Programming: *Jason* (by João)
- **Environment** Oriented Programming: CArtAgO
- **Organisation** Oriented Programming: *Moise* (by Olivier)
- Conclusions
- Practical Exercise: a hands-on lab session!
Introduction
Abstractions in Multi-Agent Systems

- **Individual Agent Level**: autonomy, situatedness
  - **Cognitive Concepts**: beliefs, desires, goals, intentions, plans
  - **Reasoning Cycle**: sense/reason/act, reactive/pro-active behaviour

- **Environment Level**: resources and services that agents can access and control; sensing and acting in an environment

- **Social and Organisation Level**: cooperation, coordination, regulation patterns
  - **Roles**: rights, responsibilities, ...
  - **Organisational Rules**: constraints on roles and their interactions, norms, deadlines, ...
  - **Organisational Structures**: topology of interaction patterns and relations over activity control
Abstractions in Multi-Agent Systems
Multi Agent Oriented Programming!

- MAS is not only agents
- MAS is not only organisation
- MAS is not only environment
- MAS is not only interaction

MAS has many dimensions all as first class entities
Environment Oriented Programming — EOP —
Outline

1. Introduction
2. Origins and Fundamentals
3. Environment Oriented Programming
4. Agent & Artifact Model
5. CArtAgO
6. Programming Artifacts
7. Programming Jason Agents & Artifacts
Notion of Environment in MAS

- The notion of environment is intrinsically related to the notion of agent and multi-agent system
  - “An agent is a computer system that is situated in some environment and that is capable of autonomous action in this environment in order to meet its design objective” [Wooldridge, 2002]
  - “An agent is anything that can be viewed as perceiving its environment through sensors and acting upon the environment through effectors.” [Russell and Norvig, 2003]
- This notion includes both physical and software environments
Classic Properties of Environment in MAS

- **Basic classification** [Russell and Norvig, 2003]
  - **Accessible** versus **inaccessible**: indicates whether the agents have access to the complete state of the environment or not
  - **Deterministic** versus **non deterministic**: indicates whether a stage change of the environment is uniquely determined by its current state and the actions selected by the agents or not
  - **Static** versus **Dynamic**: indicates whether the environment can change while an agent deliberates or not
  - **Discrete** versus **Continuous**: indicates whether the number or percepts and actions are limited or not

- **Further classification** [Ferber, 1999]
  - **Centralized** versus **Distributed**: indicates whether the environment is a single monolithic system or a set of cells or places assembled in a network
  - **Generalized** versus **Specialized**: indicates whether the environment is independent of the kind of actions that can be performed by agents or not.
Action Models

- Action defined as a transition of the environment state:
  - from an observational point of view, the result of the behavior of an agent -its action- is directly modelled by modifying the environmental state variables
  - not fully adequate for modelling Multi-Agent Systems: several agents are acting concurrently on a shared environment (concurrent actions)

- Influence & reactions [Ferber and Muller, 1996]: clear distinction between the products of the agents’ behavior and the reaction of the environment
  - influences come from inside the agents and are attempts to modify the course of events in the world
  - reactions are produced by the environment by combining influences of all agents, given the local state of the environment and the laws of the world

handling simultaneous activity in the MAS
Environment along the Agent Perspective

- Agent-Oriented Programming perspective
  - languages / platforms for programming agents and MAS
    - Agent-0, Placa, April, Concurrent Metatem, ConGolog / IndiGolog, AgentSpeak, AgentSpeak(L) / Jason, 3APL, IMPACT, Claim/Sympa, 2APL, GOAL, Dribble, etc
    - Jack, JADE, JADEX, AgentFactory, Brahms, JIAC, etc

- Environment support
  - typically minimal: most of the focus is on agent architecture & agent communication
  - in some cases: basic environment API: for “customising” the MAS with a specific kind of environment
Environment in the Jason Platform

Environment
- `globalPercepts`: List of Literals
- `agPercepts`: Map of Agent Name to List of Literals

+ `init(String[] args)`
+ `stop()`

+ `getPercepts(String agName)`: List of Literals
+ `executeAction(String agName, Structure action)`: boolean

+ `addPercept(String agName, Literal p)`
+ `removePercept(String agName, Literal p)`

UserEnvironment
+ `init(String[] args)`
+ `executeAction(String agName, Structure action)`: boolean

Agent Architecture
- `getPercepts` to User Environment
- `executeAction` to User Environment
- `change percepts` back to User Environment
SUMMARY (1)

**MAS**
- Actions
- Percepts

**SIMULATED WORLD**
- OR
- Interface
- OR

**WRAPPER TO EXISTING TECHNOLOGY**

**REAL WORLD**
- PHYSICAL OR COMPUTATIONAL

**EXTERNAL WORLD**
- PHYSICAL OR COMPUTATIONAL

**Example:** JAVA PLATFORM AGENTS

**MAS ENVIRONMENT**

**Example:**

JAVA PLATFORM AGENTS
In most cases, no direct support.

Indirectly supported by lower-level implementing technology (e.g. Java)

In some cases, first environment API

useful to create simulated environments or to interface with external resources

- simple model: a single / centralised object
- defining agent (external) actions: typically a static list of actions, shared by all the agents
- generator of percepts: establishing which percepts for which agents
1 Introduction

2 Origins and Fundamentals

3 Environment Oriented Programming

4 Agent & Artifact Model

5 CArtAgO

6 Programming Artifacts

7 Programming Jason Agents & Artifacts

8 Conclusion
Environment as a first-class abstraction in MAS

- Considering environment as an explicit part of the MAS
- Providing an exploitable design and programming abstraction to build MAS applications

Outcome
- Clear distinction between the responsibilities of the agent and those of the environment
- Separation of concerns

Improving the engineering practice with three support levels
- basic interface support
- abstraction support
- interaction-mediation support
Basic Interface Support

The environment enables agents to access the deployment context

- i.e. the hardware and software and external resources with which the MAS interacts
- e.g. sensors and actuators, a printer, a network, a database, a Web service, etc.

Figure from [Weyns et al., 2007]
Abstraction Support

Bridges the conceptual gap between the agent abstraction and low-level details of the deployment context

- Shields low-level details of the deployment context

Figure from [Weyns et al., 2007]
Interaction-Mediation Support

- Regulate the access to shared resources
- Mediate interaction between agents

Figure from [Weyns et al., 2007]
The environment is a **first-class abstraction** that provides the **surrounding conditions** for agents to exist and that **mediates** both the interaction among agents and the access to resources. [Weyns et al., 2007]
Highlights 1/2

- **First-class abstraction**
  - Environment as an independent building block in the MAS,
  - encapsulating its own clear-cut responsibilities, irrespective of the agents

- The environment provides the **surrounding conditions** for agents to exist
  - environment as an essential part of every MAS
  - the part of the world with which the agents interact, in which the effects of the agents will be observed and evaluated
The environment mediates both the interaction among agents and the access to resources.

- It provides a medium for sharing information and mediating coordination among agents.
- As a mediator, the environment not only enables interaction, it also constrains it.
- As such, the environment provides a design space that can be exploited by the designer.
Responsibilities 1/3

- **Structuring** the MAS
  - the environment is a shared “space” for the agents, resources, and services which structures the whole system
  - in terms of:
    - **physical** structure
      - refers to spatial structure, topology, and possibly distribution
    - **interaction** structure
      - refers to infrastructure for message transfer, infrastructure for stigmergy, or support for implicit communication
    - **social** structure
      - refers to the embodiment of the organizational structure within the environment
Embedding resources and services
- Resources and services can be situated either in the physical structure or in the abstraction layer introduced by the environment.
- The environment should provide support at the abstraction level shielding low-level details of resources and services to the agents.

Encapsulating a state and processes
- Besides the activity of the agents, the environment can have processes of its own, independent of agents.
  - Example: Evaporation, aggregation, and diffusion of digital pheromones.
- It may also provide support for maintaining agent-related state.
  - For example, the normative state of an electronic institution or tags for reputation mechanisms.
Ruling and governing function

- the environment can define different types of rules on all entities in the MAS.
  - constraints imposed by the domain at hand or laws imposed by the designer
  - may restrict the access of specific resources or services to particular types of agents, or determine the outcome of agent interactions
  - preserving the agent system in a consistent state according to the properties and requirements of the application domain

Examples

- coordination infrastructures
- e-Institutions
Figure from [Weyns et al., 2007]
Approaches

- Looking for general-purpose approaches for conceiving, designing, programming, executing the environment as agents’ world
  - orthogonality
  - generality
  - expressiveness
- Uniformly integrating different MAS aspects
  - coordination, organisation, institutions, ...
- Examples of concrete models and technologies
  - AGRE/AGREEN/MASQ [Baez-Barranco et al., 2007]
  - GOLEM [Bromuri and Stathis, 2008]
  - A&A, CArtAgO [Ricci et al., 2007]
Introduction

Origins and Fundamentals

Environment Oriented Programming

Agent & Artifact Model

CArtAgO

Programming Artifacts

Programming Jason Agents & Artifacts

Conclusion
Background Human Metaphor

agents can join dynamically the workspace
### Agent & Artifacts (A&A) Basic Concepts

#### Agents
- autonomous, goal-oriented pro-active entities
- create and co-use artifacts for supporting their activities,
  - besides direct communication

#### Artifacts
- non-autonomous, function-oriented, stateful entities
  - controllable and observable
- modelling the tools and resources used by agents
  - designed by MAS programmers

#### Workspaces
- grouping agents & artifacts
- defining the topology of the computational environment
A&A Programming Model Features

**Abstraction**
- artifacts as first-class resources and tools for agents

**Modularisation**
- artifacts as modules encapsulating functionalities, organized in workspaces

**Extensibility and openness**
- artifacts can be created and destroyed at runtime by agents

**Reusability**
- artifacts (types) as reusable entities, for setting up different kinds of environments
A&A Meta-Model in more Details

- **Workspace**
- **Artifact**
- **Operation**
- **Observable Event**
- **Observable Property**
- **Manual**
- **Agent**

Relations:
- **Workspace** links to **Artifact** and **Environment**
- **Artifact** links to **Operation** and **Observable Property**
- **Operation** links to **Observable Event**
- **Manual** links to **Artifact**
- **Agent** links with **create**, **dispose**, **link**, **use**, **generate**, **update**, **perceive**, **observe**, **join**, **quit**

Actions:
- **has**
- **link**
- **consult**
A World of Artifacts

- **A counter**: count = 5
  - inc
  - reset
- **A flag**: state = true
- **A Stock Quote Web Service**
  - state = available
- **A data-base**
  - n_records = 1001
  - table_names = ...
- **A bounded buffer**
  - n_items = 0
  - max_items = 100
  - put
  - get
- **An agenda**
  - next_todo = check plant
  - last_todo = ...
  - setTodo
cancelTodo
- **An event service**
  - clearEvents
  - postEvent
  - registerForEvs
- **A tuple space**
  - out
  - in
  - rd
Simple Artifacts Taxonomy

Individual or Personal Artifacts
- designed to provide functionalities for a single agent use
- e.g. agenda for managing deadlines, a library, ...

Social Artifacts
- designed to provide functionalities for structuring and managing the interaction in a MAS
- coordination artifacts, organisation artifacts, ...
- e.g. blackboard, game-board, ...

Boundary artifacts
- to represent external resources/services (e.g. a printer, a Web Service)
- to represent devices enabling I/O with users (e.g. GUI, Console, etc)
Explicit semantics refined by the (endogenous) environment:

- success/failure semantics, execution semantics,
- actions and Percepts constitute the **Contract** (in the Software Engineering meaning) provided by the environment

### Action Repertoire (actions $\leftarrow \rightarrow$ artifacts’ operations)

- is given by the dynamic set of operations provided by the overall set of artifacts available in the workspace
- can be changed by creating/disposing artifacts.

### Percept Repertoire (percepts $\leftarrow \rightarrow$ artifacts’ obs. prop.+signals)

- is given by the dynamic set of **properties** representing the state of the environment and by the **signals** concerning events signalled by the environment
- can be changed by creating/disposing artifacts.
Performing an action corresponds to triggering the execution of an operation

\[ \rightarrow \text{acting on artifact’s usage interface} \]
Operation execution is:

- a process structured in one or multiple transactional steps
- asynchronous with respect to agent ...which can proceed possibly reacting to percepts and executing actions of other plans/activities

Operation completion causes action completion, generating events with success or failure, possibly with action feedbacks
Agents can dynamically select which artifacts to observe
- predefined focus/stopFocus actions
By focussing an artifact

- observable properties are mapped into agent dynamic knowledge about the state of the world, as percepts (e.g. belief base)
- signals are mapped into percepts related to observable events
**Artifact Linkability**

- Basic mechanism to enable inter-artifact interaction
  - linking artifacts through interfaces (link interfaces)
    - operations triggered by an artifact over an other artifact
  - Useful to design & program distributed environments
    - realised by set of artifacts linked together
    - possibly hosted in different workspaces
Artifact Manual

- Agent-readable description of artifact’s...
  - functionality
    - what functions/services artifacts of that type provide
  - operating instructions
    - how to use artifacts of that type
- Towards advanced use of artifacts by intelligent agents
  - dynamically choosing which artifacts to use to accomplish their tasks and how to use them
  - strong link with Semantic Web research issues
- Work in progress
  - defining ontologies and languages for describing the manuals
CArtAgO

- CArtAgO framework / infrastructure
  1. environment for programming and executing artifact based environments
  2. Java-based programming model for defining artifacts
  3. set of basic API for agent platforms to work within artifact-based environment

- Integration with agent programming platforms: available bridges for Jason, Jadex, AgentFactory, simpA, ongoing for 2APL and Jade

- Distributed and open MAS: workspaces distributed on Internet nodes

- Agents can join and work in multiple workspace at a time (Role-Based Access Control (RBAC) security model)

- Open-source technology
CArtAgO Architecture

**MAS Application**
- Application Agents
- Artifact-based working environments
  - shared task scheduler
  - shared KB
  - blackboard
  - map

**Execution Platform**
- Agent Frameworks / Middlewares
  - JASON
  - JADEX
  - ...
- agent bodies

**CARTAGO**
- workspaces
- artifacts

**Any**
- JVM

**OS**
- JVM

**OS**
Pre-defined Artifacts

- Each workspace contains by default a predefined set of artifacts
  - providing core and auxiliary functionalities
  - i.e. a pre-defined repertoire of actions available to agents...
- Among the others
  - `workspace`, type: `cartago.WorkspaceArtifact`
    - functionalities to manage the workspace, including security
    - operations: `makeArtifact`, `lookupArtifact`, `focus`, ...
  - `node`, type: `cartago.NodeArtifact`
    - core functionalities related to a node
    - operations: `createWorkspace`, `joinWorkspace`, ...
  - `console`, type `cartago.tools.Console`
    - operations: `println`, ...
  - `blackboard`, type `cartago.tools.TupleSpace`
    - operations: `out`, `in`, `rd`, ...
  - ....
1 Introduction

2 Origins and Fundamentals

3 Environment Oriented Programming

4 Agent & Artifact Model

5 CArtAgO

6 Programming Artifacts
   - Observable Property
   - Operations
   - Links between Artifacts

7 Programming Jason Agents & Artifacts

8 Conclusion
Defining an Artifact

- An artifact type extends the `cartago.Artifact` class
- An artifact is composed of:
  - **state variables**: class instance fields
  - **observable properties** with a set of primitives to define/update/.. them
  - **signal** primitive to generate signals
  - **operation controls**: methods annotated with `@OPERATION`
    - The operation `init` is the operation which is automatically executed when the artifact is created (analogous to constructor in objects).
  - **internal operations**: operations triggered by other operations, methods annotated with `@INTERNAL_OPERATION`
  - **await** primitive to define the operation steps
  - **guards** - both for operation controls and operation steps -: methods annotated with `@GUARD`
Observable property

- Observable property is defined by a name and a value.
- The value can change dynamically according to artifact behaviour.
- The change is made automatically observable to all the agents focussing the artifact.
- Defined by using `defineObsProperty`, specifying
  - the name of the property
  - the initial value (that can be of any type, including objects)
- Accessed by
  - `getObsProperty`
  - `updateObsProperty`
Change of the value of a property using primitive

\[ \text{getObsProperty(String name).updateValue(Object value)} \]

or \[ \text{updateObsProperty(String name, Object value)} \]

- the specified value must be compatible with the type of the corresponding field
- the value of the property is updated with the new value
- an event is generated (content is the value of the property) \[ \text{property\_updated(PropertyName,NewValue,OldValue)} \]
- the event is made observable to all the agents focussing the artifact
**Example**

```java
public class Counter extends Artifact {
    void init() {
        defineObsProperty("count", 0);
    }

    @OPERATION void inc() {
        ObsProperty prop = getObsProperty("count");
        prop.updateValues(prop.intValue() + 1);
    }
}
```
Operations

- Operation $\text{op}(\text{param1},\text{param2},\ldots)$ is defined as:
  - a method $\text{op}$, in the artifact class returning $\text{void}$
  - annotated with $\text{@OPERATION}$

- Parameters can be input/output operation parameters
  - Output operation parameters ($\text{OpFeedbackParam}<T>$) can be used to specify the operation results and related action feedback

- Operation can be composed of zero, one or multiple atomic computational steps

- $\text{init}$ method (defined or not as an operation) is called at the initialisation of the artefact.

**Example**

```java
@OPERATION void inc(OpFeedbackParam<Int> res) {
    res.set(++count);
}
```
Observable Events

Observable events are generated by default:

- `op_execution_completed`, `op_execution_failed`, `op_execution_aborted` ...

Observable event can be generated explicitly, within an operation by the method

```java
signal(String evType, Object variable params)
```

- Generated event is a tuple, with `evType` label, composed of the sequence of passed parameters
- Generated event can be observed by
  - the agent responsible of the execution of the operation
  - all the agents observing the artifact

```java
signal(AgentId id, String evType, Object variable params)
```

- Generated event is perceivable only by the specified agent that must be observing the artifact, anyway.
Example of Observable Events

Example

```java
public class Count extends Artifact {
    int count;
    void init() {
        count = 0;
    }
    @OPERATION void inc() {
        count++;
        signal("new_value", count);
    }
}
```
Failed primitive

- `failed(String failureMsg)`
- `failed(String failureMsg, String descr, Object... args)`

An action feedback is generated, reporting a failure msg and optionally also a tuple descr(Object...) describing the failure.
Example of Observable Events

Example

```java
public class BoundedCounter extends Artifact {
    private int max;
    void init(int max) {
        defineObsProperty("count", 0);
        this.max = max;
    }

    @OPERATION void inc() {
        ObsProperty prop = getObsProperty("count");
        if (prop.intValue() < max) {
            prop.updateValue(prop.intValue()+1);
            signal("tick");
        } else {
            failed("inc failed","inc_failed","max_value_reached",max);
        }
    }
}
```
public class BBuffer extends Artifact {
    private LinkedList<Item> items;
    private int nmax;
    void init(int nmax) {
        items = new LinkedList<Item>();
        this.nmax = nmax;
        defineObsProperty("n_items",0);
    }
    @OPERATION(guard="bufferNotFull") void put(Item obj) {
        items.add(obj);
        getObsProperty("n_items").updateValue(items.size());
    }
    @OPERATION void get(OpFeedbackParam<Item> res) {
        await("itemAvailable");
        Item item = items.removeFirst();
        res.set(item);
        getObsProperty("n_items").updateValue(items.size());
    }
    @GUARD boolean bufferNotFull(Item obj) { return items.size() < nmax; }
    @GUARD boolean itemAvailable() { return items.size() > 0; }
}
Operation Guards

Guard on an operation is specified as:

- a `boolean` method annotated with `@GUARD`, having the same number and type of parameters of the guarded operation
- Its name is included as the attribute `guard` of the `@OPERATION` annotation
- or used as parameter of the method `await` in the body of the operation
- The operation will be enabled only if (when) the guard is satisfied

**Example**

```java
public class MyArtifact extends Artifact {
    int m;
    void init() { m=0; }
    @OPERATION(guard="canExecOp1") void op1() { ... }
    @OPERATION void op2() { m++; }
    @GUARD boolean canExecOp1() { return m == 5; }
}
```
Example: Bounded Buffer with Guarded Operations

<table>
<thead>
<tr>
<th>n_items</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>max_items</td>
<td>100</td>
</tr>
<tr>
<td>put</td>
<td></td>
</tr>
<tr>
<td>get</td>
<td></td>
</tr>
</tbody>
</table>

**OBSERVABLE PROPERTIES:**

- **n_items**: int+
- **max_items**: int

**USAGE INTERFACE:**

- **put(item:Item)** / (n_items < max_items): {...}

- **get** / (n_items >= 0):
  
  { new_item(item:Item),...}

```java
public class BBuffer extends Artifact {
    private LinkedList<Item> items;
    private int nmax;
    void init(int nmax) {
        items = new LinkedList<Item>();
        defineObsProperty("max_items", nmax);
        defineObsProperty("n_items", 0);
    }

    @OPERATION(guard="bufferNotFull") void put(Object obj) {
        items.add(obj);
        getObsProperty("n_items").updateValue(items.size());
    }

    @GUARD boolean bufferNotFull(Item obj) {
        int maxItems = getObsProperty("max_items").intValue();
        return items.size() < maxItems;
    }

    @OPERATION(guard="itemAvailable") void get() {
        Object item = items.removeFirst();
        getObsProperty("n_items").updateValue(items.size());
        signal("new_item", item);
    }

    @GUARD boolean itemAvailable() { return items.size() > 0; }
}
```
Multi-step Operation

Structured (non-atomic) operations are implemented with

- one `@OPERATION` representing the entry point
- one or multiple transactional steps, possibly with guards
- `await` primitive to define the steps
public class MyArtifact extends Artifact {
    int internalCount;
    @OPERATION void opWithResults(double x, double y, OpFeedbackParam<Double> sum, OpFeedbackParam<Double> sub) {
        sum.set(x+y);
        sub.set(x-y);
    }
    @OPERATION void structureOp(int ntimes) {
        internalCount=0;
        signal("step1_completed");
        await( "canExecStep2", ntimes);
        signal("step2_completed", internalCount);
    }
    @OPERATION void update(int delta) {
        internalCount += delta;
    }
    @GUARD boolean canExecStep2(int ntimes) {
        return internalCount >= ntimes;
    }
}
Example: Simple synchronisation artifact

public class SimpleSynchronizer extends Artifact {
    int nReady, nParticipants;
    void init(int nParticipants) {
        defineObsProperty("all_ready",false);
        nReady = 0;
        this.nParticipants = nParticipants;
    }
    @OPERATION void ready() { // to synch
        nReady++;
        await("allReady");
        getObsProperty("all_ready").updateValue(true);
    }
    @GUARD boolean allReady() {
        return nReady >= nParticipants;
    }
}
Example: Bounded Buffer with Guarded Steps

OBSERVABLE PROPERTIES:

<table>
<thead>
<tr>
<th>n_items</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>max_items</td>
<td>100</td>
</tr>
</tbody>
</table>

 USAGE INTERFACE:

put(item:Item) / (n_items < max_items): {...}

get / (n_items >= 0):
   { new_item(item:Item),...}
Temporal Guards on Operation Steps

- Specified with `await_time` primitive
- Parameter indicates the number of milliseconds that must elapse before the step could be executed, after having being triggered
- Its value is a long value greater than 0
**Example of Temporally Guarded Operation**

```java
public class Clock extends Artifact {
    boolean working;
    final static long TICK_TIME = 100;
    void init() {
        working = false;
    }
    @OPERATION void start() {
        if (!working) {
            working = true; execInternalOp("work");
        } else {
            failed("already_working");
        }
    }
    @OPERATION void stop() {
        working = false;
    }
    @INTERNAL_OPERATION void work() {
        while (working) {
            signal("tick");
            await_time(TICK_TIME);
        }
    }
}
```
**Link Interface**

- Set of operations that can be triggered by an artifact on another artifact
- Operations are annotated with `@LINK` (can be composed by multiple steps, can generate events, etc.)

**Example**

```java
class LinkableArtifact extends Artifact {
    int count;
    void init() { count= 0; }
    @LINK void inc() {
        log("inc invoked."); count++;
        signal("new_count_value",count);
    }
}
```

- Call of the operation from the linking Artifact is done using the `execLinkedOp` primitive.
**Linking Artifacts**

- Executing `execLinkedOp` triggers the operation
- Once triggered, linked operation execution is the same as normal operations
- The only difference is:
  - the events that are generated by a linked operations, are made observable to the agent using or observing the artifact that triggered the execution of the link operation
  - In the case of a chain, with an agent X executing an operation on an artifact, which links the operation of an artifact B, which links an operation of an artifact C, all the observable events generated by B and C linked operations are made observable to X
public class Counter extends Artifact {
    void init() {
        defineObsProperty("count", 0);
    }

    @OPERATION void inc() {
        int count = getObsProperty("count").intValue();
        getObsProperty("count").updateValue(count + 1);
    }
}
Jason Agents using the Simplest Artifact (1)

!create_and_use.
+!create_and_use

<- !setupTool(Id);
    // first use
    inc;
    // second use specifying the id
    inc [artifact_id(Id)].
+!setupTool(C)
<- makeArtifact("ourCount", "Counter", [], C).
Jason Agents observing the Simplest Artifact (2)

!observe.

+!observe
  <- ?myTool(C);                // query goal
      focus(C).

+count(V) : V < 10  <- println("count percept: ",V).

+count(V)[artifact.name(Id,"ourCount")]: V >= 10
  <- println("stop observing.");
      stopFocus(Id).

+?myTool(CounterId)
  <- lookupArtifact("ourCount",CounterId).

-?myTool(CounterId)  <- .wait(10); ?myTool(CounterId).
Producer-Consumer Artifact

- bounded-buffer artifact for producers-consumers scenarios

**OBSERVABLE PROPERTIES:**

- **n_items**: int+
- **max_items**: int

**USAGE INTERFACE:**

- **put**(item:Item) / (n_items < max_items): {...}
- **get** / (n_items >= 0):
  { new_item(item:Item),...}
Producer Jason Agent

```prolog
item_to_produce(0).
!produce.

+!produce
<- !setupTools(Buffer); !produceItems.

+!produceItems
<- ?nextItemToProduce(Item);
   put(Item);
   !!produceItems.

+?nextItemToProduce(Item)
<- -item_to_produce(Item);
   +item_to_produce(Item+1).

+!setupTools(Buffer)
<- makeArtifact("myBuffer", "BoundedBuffer", [10], Buffer).

-!setupTools(Buffer)
<- lookupArtifact("myBuffer", Buffer).
```
Consumer Jason Agent

!consume.

+!consume
<- ?bufferToUse(Buffer);
   .print("Going to use ",Buffer);
   !consumeItems.

+!consumeItems
<- get(Item); !consumeItem(Item); !consumeItems.

+!consumeItem(Item) <- ...

+?bufferToUse(BufferId)
<- lookupArtifact("myBuffer",BufferId).

-?bufferToUse(BufferId)
<- .wait(50); ?bufferToUse(BufferId).
Synchronisation Artifact

- **OBSERVABLE PROPERTIES:**
  - all_ready: {true, false}

- **USAGE INTERFACE:**
  - ready / true: { op_exec_completed }

```java
public class SimpleSynchronizer extends Artifact {
    int nReady, nParticipants;
    void init(int nParticipants) {
        defineObsProperty("all_ready",false);
        nReady = 0;
        this.nParticipants = nParticipants;
    }
    @OPERATION void ready() { // to synch
        nReady++;
        nextStep("setAllReady");
    }
    @OPSTEP(guard="allReady") void setAllReady() {
        getObsProperty("all_ready").updateValue(true);
    }
    @GUARD boolean allReady() {
        return nReady >= nParticipants;
    }
}
```
Example

!work.
+!work <- ...

// locate the synch tool
lookupArtifact(mySynch,Synch);

// observe it.
focus(Synch);

// ready for synch
ready.

// react to all_ready(true) percept
+all_ready(true)[artifact_id(mySynch)]
<- // all ready, go on.
...

// synch tool
// all ready, go on.
Example: A Tuple-Space Artifact

```java
public class SimpleTupleSpace extends Artifact {
    TupleSet tset;

    void init()
        tset = new TupleSet();
    }

    @OPERATION void out(String name, Object... args){
        tset.add(new Tuple(name, args));
    }

    @OPERATION void in(String name, Object... params){
        TupleTemplate tt = new TupleTemplate(name, params);
        await("foundMatch", tt);
        Tuple t = tset.removeMatching(tt);
        bind(tt, t);
    }

    @OPERATION void rd(String name, Object... params){
        TupleTemplate tt = new TupleTemplate(name, params);
        await("foundMatch", tt);
        Tuple t = tset.readMatching(tt);
        bind(tt, t);
    }

    @GUARD boolean foundMatch(TupleTemplate tt){
        return tset.hasTupleMatching(tt);
    }

    private void bind(TupleTemplate tt, Tuple t){...}
}
```

- Multi-step operations
  - operations composed by multiple transactional steps, possibly with guards
  - await primitive to define the steps
Remarks

- Process-based action execution semantics
  - action/operation execution can be long-term
  - action/operation execution can overlap
- Key feature for implementing coordination functionalities
Example: Dining Philosopher Agents

**WAITER**

philo(0,"philo1",0,1).
philo(1,"philo2",1,2).
philo(2,"philo3",2,3).
philo(3,"philo4",3,4).
philo(4,"philo5",4,0).

!prepare_table.

+!prepare_table
  <- for ( .range(I,0,4) ) {
    out("fork",I);
    ?philo(I,Name,Left,Right);
    out("philo_init",Name,Left,Right);
  }
  for ( .range(I,1,4) ) {
    out("ticket");
  }
  println("done.").

**PHILOSOPHER AGENT**

!boot.

+!boot
  <- .my_name(Me);
  in("philo_init",Me,Left,Right);
  +my_left_fork(Left); +my_right_fork(Right);
  println(Me," ready.");
  !!enjoy_life.

+!enjoy_life
  <- !thinking; !eating; !!enjoy_life.

+!eating
  <- !acquireRes; !eat; !releaseRes.

+!acquireRes : my_left_fork(F1) & my_right_fork(F2)
  <- in("ticket"); in("fork",F1); in("fork",F2).

+!releaseRes: my_left_fork(F1) & my_right_fork(F2)
  <- out("fork",F1); out("fork",F2); out("ticket").

+!thinking <- .my_name(Me); println(Me," thinking").
+!eat <- .my_name(Me); println(Me," eating").
Example: A Clock

```java
public class Clock extends Artifact {
    boolean working;
    final static long TICK_TIME = 100;

    void init(){ working = false; }

    @OPERATION void start()
    {
        if (!working){
            working = true;
            execInternalOp("work");
        } else {
            failed("already_working");
        }
    }

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

    @INTERNAL_OPERATION void work()
    {
        while (working)
        {
            signal("tick");
            await_time(TICK_TIME);
        }
    }
}
```

CLOCK USER AGENT

```
!test_clock.
+!test_clock
  <- makeArtifact("myClock","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!");
```

- Internal operations
  - execution of operations triggered by other operations
  - implementing controllable processes
Example: GUI Artifacts

- Exploiting artifacts to enable interaction between human users and agents
Example: Agent and User Interaction

**GUI ARTIFACT**

```java
public class MySimpleGUI extends GUIArtifact {
  private MyFrame frame;

  public void setup() {
    frame = new MyFrame();
    linkActionEventToOp(frame.okButton, "ok");
    linkKeyStrokeToOp(frame.text, "ENTER", "updateText");
    linkWindowClosingEventToOp(frame, "closed");
    defineObsProperty("value", getValue());
    frame.setVisible(true);
  }

  @INTERNAL_OPERATION void ok(ActionEvent ev) {
    signal("ok");
  }

  @OPERATION void setValue(double value) {
    frame.setText("+
```
Remark: Action Execution & Blocking Behaviour

- Given the action/operation map, by executing an action the intention/activity is suspended until the corresponding operation has completed or failed
  - action completion events generated by the environment and automatically processed by the agent/environment platform bridge
  - no need of explicit observation and reasoning by agents to know if an action succeeded
- However the agent execution cycle is not blocked!
  - the agent can continue to process percepts and possibly execute actions of other intentions
Example: Action Execution & Blocking Behaviour

The agent perceives and processes `new_number` percepts as soon as they are generate by the `Stream` even if the `processing_stream` plan execution is suspended, waiting for `generate` action completion.

The test goal `?sum(S)` is executed after `generate` action completion so we are sure that all numbers have been generated and processed.
Other Features

- Other CArtAgO features not discussed in this lecture
  - linkability
    - executing chains of operations across multiple artifacts
  - multiple workspaces
    - agents can join and work in multiple workspaces, concurrently
    - including remote workspaces
  - RBAC security model
    - workspace artifact provides operations to set/change the access control policies of the workspace, depending on the agent role
    - ruling agents’ access and use of artifacts of the workspace
  - ...

- See CArtAgO papers and manuals for more information
A&A and CArtAgO: Some Research Explorations

- Designing and implementing artifact-based organisation Infrastructures
  - ORA4MAS infrastructure
- Cognitive stigmergy based on artifact environments
  - Cognitive artifacts for knowledge representation and coordination
- Artifact-based environments for argumentation
- Including A&A in AOSE methodology
- ...
Applying CArtAgO and JaCa

- Using CArtAgO/JaCa for building real-world applications and infrastructures
- Some examples
  - JaCa-WS / CArtAgO-WS
    - building SOA/Web Services applications using JaCa
    - [http://cartagows.sourceforge.net](http://cartagows.sourceforge.net)
  - JaCa-Web
    - implementing Web 2.0 applications using JaCa
  - JaCa-Android
    - implementing mobile computing applications on top of the Android platform using JaCa
Conclusions
Putting the Pieces Together

BELIEFS
GOALS
PLANS
PERCEPTIONS
AGENTS
MISSIONS
ROLES
DEONTIC RELATIONS
GROUPS
ROLES
SANCTIONS
MISSIONS
REWARDS
DEONTIC RELATIONS
NORMS
ORGANISATIONS
RESOURCES
LEGACY
SERVICES
OBJECTS
ENvironments
JASON
Agent Proc.
Language
MOISE
Framework
JADE Platform
CarTaGO Platform
SPEECH ACTS
COMMUNICATION LANGUAGES
INTERACTATION PROTOCOLS
INTERACTIONS
Agent meta-model

- External Action
- Internal Action
- Action
- Agent
- Plan
- Trigger event
- Belief
- Goal

Cardinalities are not represented

- composition
- association
- dependency
- agent's actions
- concept mapping
- dimension border

Dimension border
Environment meta-model

- Manual
- Workspace
- Work Environment
- Artifact
- Operation
- Observable Property
- Observable Event

- has
- update
- generate
- composition
- association
- dependency
- cardinalities not represented
A & E Interaction meta-model
Research on Multi-Agent Systems...

Whatever you do in MAS, make it available in a programming language/platform for MAS!!!


In 6th international Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2007). Honolulu, Hawai’i, USA.

Artificial Intelligence, A Modern Approach (2nd ed.).
Prentice Hall.

Environment as a First-class Abstraction in MAS.

An Introduction to Multi-Agent Systems.
John Wiley & Sons, Ltd.