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" [Wooldrige and Jennings, 1995]
- "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.

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

Example of “Agents in Environment” Approach

Example of “Environment in Agents” Approach

Example of MANTA Programming [Drogoul, 2003]
Environment in the Jason Platform

Summary (1)

MAS

SIMULATED WORLD

OR

REAL WORLD

(_PHYSICAL OR COMPUTATIONAL)

INTERFACE

OR

EXTERNAL WORLD

(_PHYSICAL OR COMPUTATIONAL)

WRAPPER TO EXISTING TECHNOLOGY

Example: JAVA PLATFORM

Summary (2)

- 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
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.

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

Interaction-Mediation Support

- Regulate the access to shared resources
- Mediate interaction between agents
Environment Definition Revised

Environment Definition [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.

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

Highlights 2/2

- **Environment as a glue**
  - on their own, agents are just individual loci of control.
  - to build a useful system out of individual agents, agents must be able to interact
  - the environment provides the glue that connects agents into a working system
  - 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.

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 Stathi, 2007].
  - A&A, CArAgO [Ricci et al., 2007].
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

Background Human Metaphor

![Diagram of a workspace with various artifacts and agents, including Clock, Archive, Task Scheduler, and BAKERY workspace.](image)
A&A Meta-Model in more Details

Artifact Abstract Representation

A World of Artifacts

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 \(< \rightarrow \) 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 \(< \rightarrow \) 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.

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

Performing an action corresponds to triggering the execution of an operation

... acting on artifact’s usage interface

Agents can dynamically select which artifacts to observe

... predefined focus/stopFocus actions
Interaction Model: Observation (2)

By focusing 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 another artifact
  - Useful to design and program distributed environments
  - Realized by sets of artifacts linked together
  - Possibly hosted in different workspaces

Artifact Manual

- Agent-readable description of artifacts...
  - 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 framework / infrastructure
- environment for programming and executing artifact based environments
- Java-based programming model for defining artifacts
- 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

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,...
  - ....
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/...
  - *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 property

Change of the value of a property using primitive

```java
getObsProperty(String name).updateValue(Object value)
```

or

```java
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)
- the event is made observable to all the agents focussing the artifact

Example

```java
void init() {  
    defineObsProperty("count",0);  
}
@OPERATION void inc() {  
    int count = getObsProperty("count").intValue();  
    updateObsProperty("count",count+1);  
}
```

Example

```java
public class Counter extends Artifact {  
    void init() {  
        defineObsProperty("count",0);  
    }
    @OPERATION void inc() {  
        int count = getObsProperty("count").intValue();  
        updateObsProperty("count",count+1);  
    }
```
Example (revisited)

Observable Properties:
- **count**: int

Usage Interface:
- **inc**: (op_exec_started, inc, count[X], op_exec_completed)

Example

```java
public class Counter extends Artifact {
    void init() {
        defineObsProperty("count", 0);
    }
    @OPERATION void inc() {
        ObsProperty prop = getObsProperty("count");
        prop.updateValues(prop.intValue()+1);
    }
}
```

Observables 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

```java
public class Count extends Artifact {
    int count; // state variable
    void init() { count = 0; }
    @OPERATION void inc(OpFeedbackParam<Int> res) {
        res.set(++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

```java
public class BoundedCounter extends Artifact {
    private int max;
    private int this.max = 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);
        }
    }
}
```

Example: Bounded Buffer with Output Parameters

```java
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

```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 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

Example of Multi-step Operation

```java
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

```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 sync
        nReady++;
        await("allReady");
        getObsProperty("all_ready").updateValue(true);
    }

    @GUARD boolean allReady() {
        return nReady >= nParticipants;
    }
}
```
Example: Bounded Buffer with Guarded Steps

public class Buffer extends Artifact {
    private List<Item> items;
    private int max_items;
    @OPERATION void init(int max_items) {
        items = new LinkedList<Item>();
        defineObsProperty("max_items", max_items);
        defineObsProperty("n_items", 0);
    }
    @OPERATION void put(Object obj) {
        await("bufferNotFull", 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;
    }
}

Example of Temporally Guarded Operation

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);
        }
    }
}

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

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

public 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.
The Simplest Artifact

public class Counter extends Artifact {
    void init() {
        defineObsProperty("count", 0);
    }
    @OPERATION void inc() {
        int count = getObsProperty("count").intValue();
        getObsProperty("count").updateValue(count + 1);
    }
}

Observable Properties:
- count

Usage Interface:
- inc: (op_exec_started(inc), count(X), op_exec_completed(inc))

Jason Agents using the Simplest Artifact (1)

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

Producer Jason Agent

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

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

+!produceItems : true
<- getNextItemToProduce(Item);
put(Item);
!produceItems.

+getNextItemToProduce(Item) : true <- -item_to_produce(Item);
+item_to_produce(Item+1).

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

+lookupArtifact("myBuffer",Buffer).
```

Consumer Jason Agent

```java
!consume.

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

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

+!consumeItem(Item) : true <- ...

+bufferToUse(BufferId) : true
<- lookupArtifact("myBuffer",BufferId).
-!bufferToUse(BufferId) : true
<- .wait(50); ?bufferToUse(BufferId).
```

Producer-Consumer Artifact

bounded-buffer artifact for producers-consumers scenarios

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

  @OPERATION(guard="bufferNotFull") void put(Object obj) {
    items.add(obj);
    getObsProperty("n_items").updateValue(items.size()+1);
  }

  @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()-1);
    signal("new_item",item);
  }

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

Producer Jason Agent

```java
!observe.

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

++!observe : true

++!observe : true
<- count(V)[artifact_name(Id,"ourCount") ] : V >= 10
<- println("stop observing.");
  stopFocus(Id).

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

?-?myTool(CounterId) : true <- .wait(10); ?myTool(CounterId).
```
Synchronisation Artifact

**Example**

```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");
    }

    @OPERATION void setAllReady() {
        getObsProperty("all_ready").updateValue(true);
    }

    @GUARD boolean allReady() {
        return nReady >= nParticipants;
    }
}
```

**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) {...}
```

**Remarks**

- Process-based action execution semantics
  - action/operation execution can be long-term
  - action/operation execution can overlap
- Key feature for implementing coordination functionalities

- Multi-step operations
  - operations composed by multiple transactional steps, possibly with guards
  - `await` primitive to define the steps
**Example 4: 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);
        }
    }
}
```

**Example 5: GUI Artifacts**

```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("" + value);
        updateObsProperty("value", value);
    }

    @INTERNAL_OPERATION void updateText(ActionEvent ev) {
        updateObsProperty("value", getValue());
    }

    private int getValue() {
        return Integer.parseInt(frame.getText());
    }
}
```

**Internal operations**
- execution of operations triggered by other operations
- implementing controllable processes
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 6: Action Execution & Blocking Behaviour

```java
// agent code
@processing_stream
+!processing_stream : true
<- makeArtifact("myStream","Stream",Id);
focus(Id);
+sum(0);
generate(1000);
?sum(S);
println(S).
@update [atomic]
+new_number : sum(S)
<- -+sum(S+V).
```

```java
// artifact code
class Stream extends Artifact {
...
@OPERATION void generate(int n){
  for (int i = 0; i < n; i++)
    signal("new_number",i);
}
}
```

- The agent perceives and processes new_number percepts as soon as they are generated 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
    - JaCa-Web
      - implementing Web 2.0 applications using JaCa
    - JaCa-Android
      - implementing mobile computing applications on top of the Android platform using JaCa

Bibliography I

  A unified model for physical and social environments.

  Situating Cognitive Agents in GOLEM.
  In Engineering Environment-Mediated Multiagent Systems (EEMMAS’07).

  De la simulation multi-agent à la résolution collective de problèmes. Une étude de l’émergence de structures d’organisation dans les systèmes multi-agents.

  Multi-Agent Systems, An Introduction to Distributed Artificial Intelligence.
  Addison-Wesley.

Bibliography II

  Influences and reaction: A model of situated multiagent systems.

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


  Environment as a First-class Abstraction in MAS.

  Intelligent agents: Theory and practice.
  The Knowledge Engineering Review, 10(2):115–152.