aliCE

SISMA 2008/2009 - Seminar

ENVIRONMENT PROGRAMMING IN MAS WITH CARTAGO

Alessandro Ricci

aliCE group at DEIS, Università di Bologna, Cesena

a.ricci@unibo.it

joint work with:
- Michele Piunti, PhD student at DEIS
- Mirko Viroli, Andrea Omicini

OUTLINE

- Environment Programming in (Programming) MAS
  - the road to artifacts and CARTAGO
- A&A model and CARTAGO platform
  - programming model and technology
  - integration with existing cognitive agent platforms
- Ongoing work & available theses
  - Goal-Directed use of artifacts
  - Agent-Based SOA/Web Services Applications

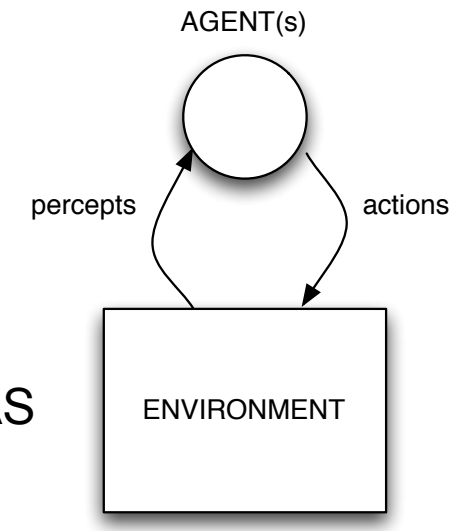
PART I

ENVIRONMENT PROGRAMMING IN (PROGRAMMING) MAS

- The ROAD to CARTAGO -

THE ROLE OF ENVIRONMENT IN MAS

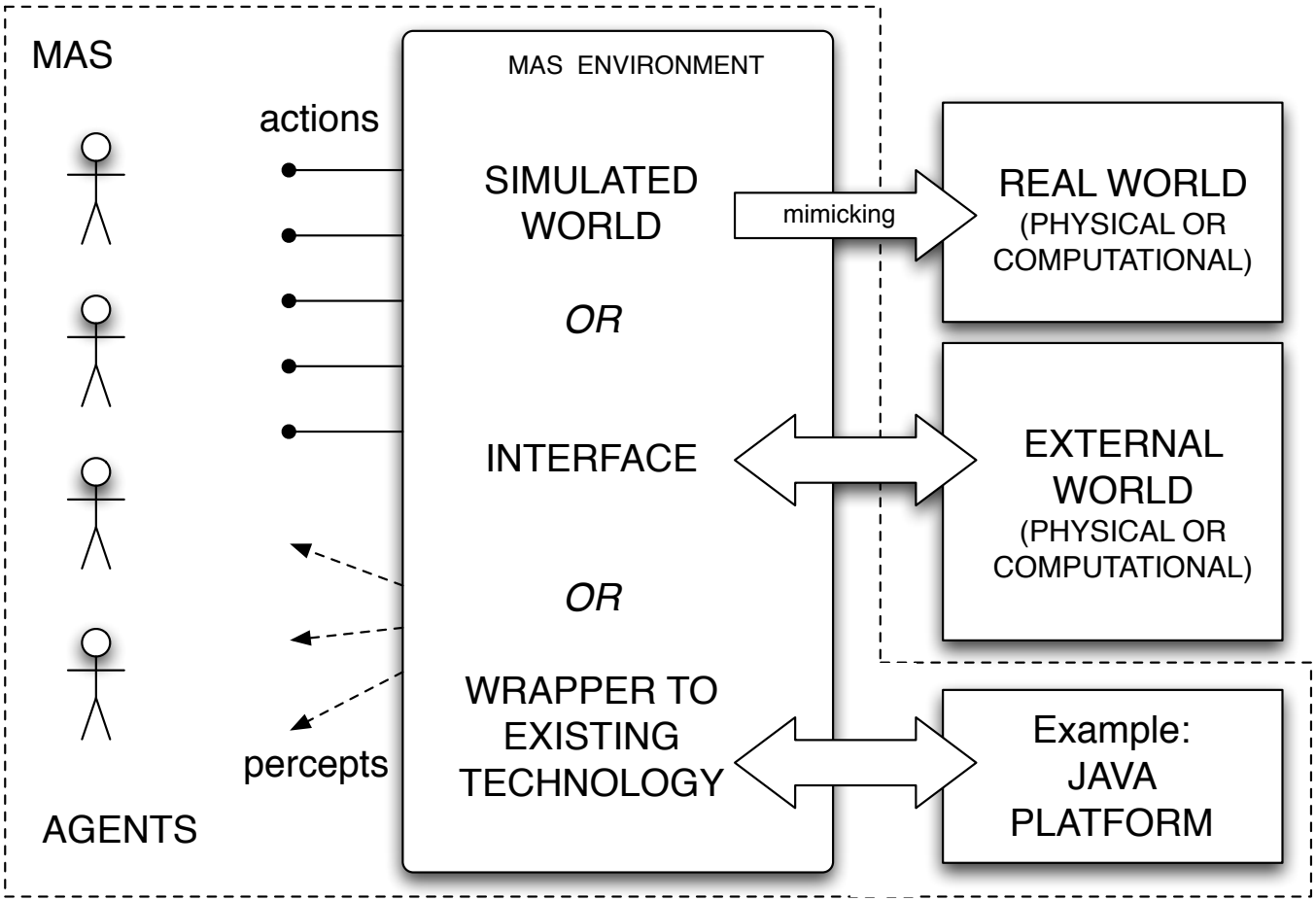
- "Traditional" (D)AI / agent / MAS view
  - the target of agent actions and source of agents perception
  - something out of MAS design / engineering
- New perspective in recent works
  - environment as first-class aspect in engineering MAS
    - mediating interaction among agents
    - encapsulating functionalities for managing such interactions
      - coordination, organisation, security,...



FROM MAS TO *MAS PROGRAMMING*

- Specific perspective on “MAS programming” adopted here
  - agents (and MAS) as a paradigm to design and program *software* systems
    - computer programming perspective
      - computational models, languages,...
    - software engineering perspective
      - architectures, methodologies, specification, verification,...
- Underlying objective in the long term
  - using agent-orientation as *general-purpose* post-OO paradigm for computer programming
    - concurrent / multi-core / distributed programming in particular

THE ROLE OF SW ENVIRONMENT IN MAS PROGRAMMING (SO FAR)



ENVIRONMENT MODEL IN MAS PROGRAMMING

- Environment as monolithic / centralised block
  - defining agent (external) actions
    - typically a static list of actions, shared by all the agents
  - generator of percepts
    - establishing which percepts for which agents
- No specific programming model for defining structure and behaviour
  - including concurrency management
  - relying on lower-level language feature
    - e.g. Java
- Typically enough for building simulated world

JASON EXAMPLE
- GOLD-MINER DEMO -

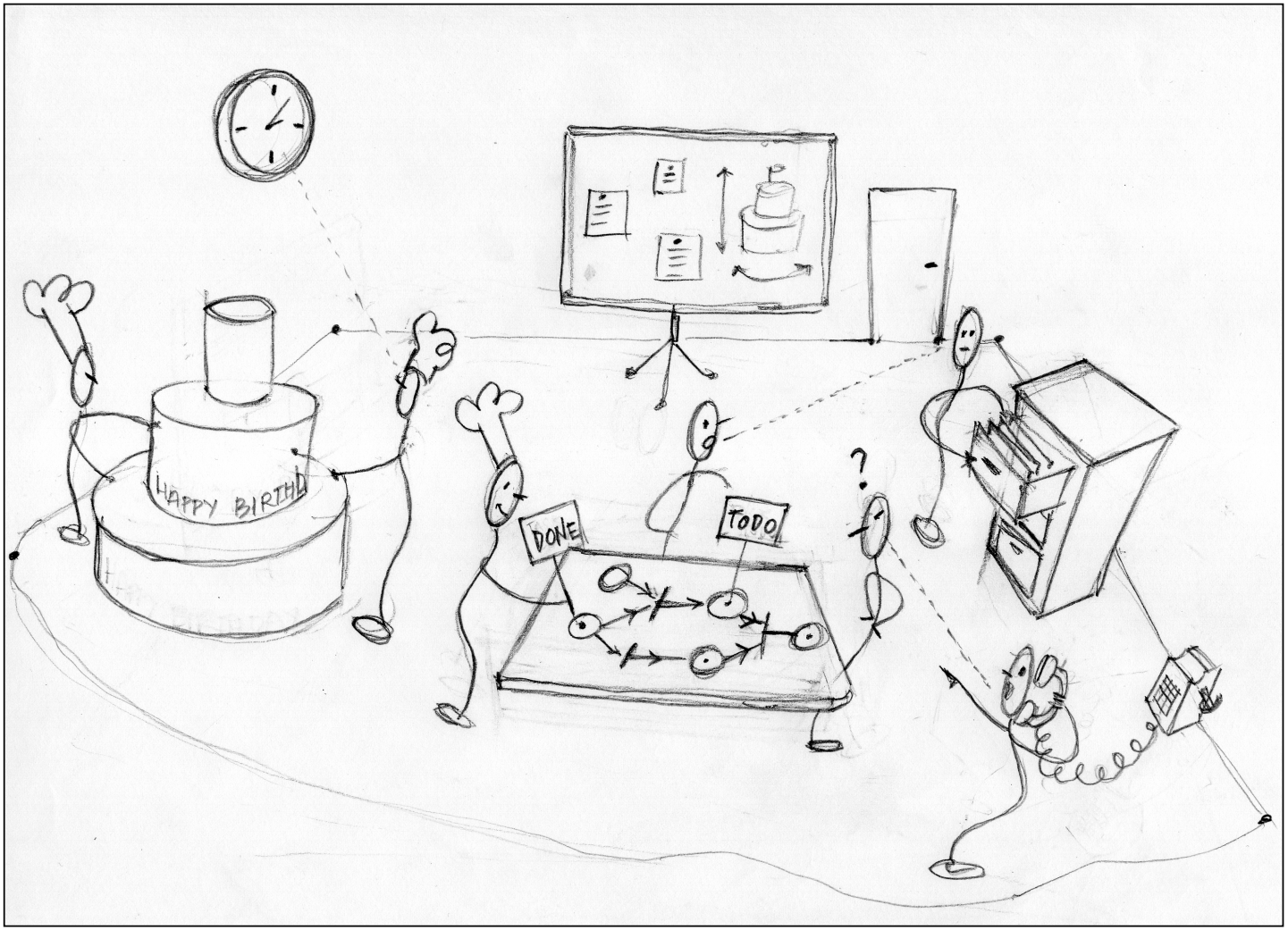
```
public class MiningPlanet extends jason.environment.Environment {
  ...
  public void init(String[] args) {...}

  public boolean executeAction(String ag, Structure action) {
      boolean result = false;
      int agId = getAgIdBasedOnName(ag);
      if (action.equals(up)) {
        result = model.move(Move.UP, agId);
      } else if (action.equals(down)) {
        result = model.move(Move.DOWN, agId);
      } else if (action.equals(right)) {
        ...
      }
      return result;
  }

  private void updateAgPercept(String agName, int ag) {clearPercepts(agName);
    // its location
    Location l = model.getAgPos(ag);
    addPercept(agName, Literal.parseLiteral("pos(" + l.x + "," + l.y + ")"));
    if (model.isCarryingGold(ag)) {
      addPercept(agName, Literal.parseLiteral("carrying_gold"));
    }
    // what's around
    updateAgPercept(agName, l.x - 1, l.y - 1);
    updateAgPercept(agName, l.x - 1, l.y);
    ...
  }
}
```

ENRICHING THE VIEW: WORK ENVIRONMENTS

- Perspective: *designing worlds in agent worlds* for agents' *use*
  - designing good and effective place for agents to live and work in
    - environment as the context of agent activities *inside the MAS*
  - beyond simulated worlds
- "**Work environment**" notion
  - that *part of the MAS* that is *designed* and *programmed* so as to ease agent activities and work
    - first-class entity of the agent world
    - cooperation, coordination, organisation, security... functionalities
- Work environment as part of MAS design and programming
  - abstractions? computational models? languages? platforms? methodologies?

A HUMAN WORK ENVIRONMENT

(~BAKERY)

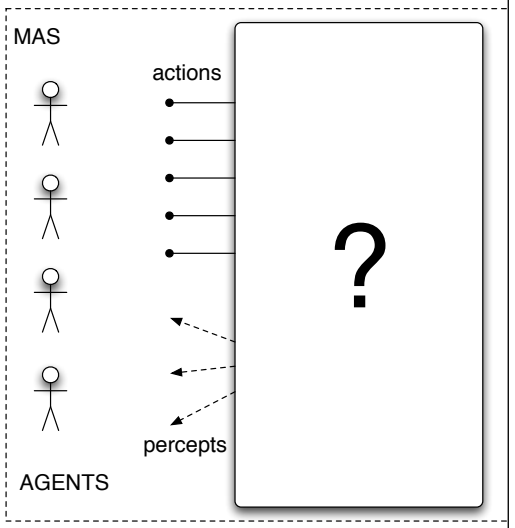


BACKGROUND LITERATURE

- In human science
  - Activity Theory, Distributed Cognition
    - importance of the environment, *mediation*, interaction for human activity development
  - Active Externalism / extended mind (Clark, Chalmer)
    - environment's obejcts role in aiding cognitive processes
- CSCW and HCI
  - importance of artifacts and tools for coordination and collaboration in human work
- Distributed Artificial Intelligence
  - Agre & Horswil work ("Lifeworld"...)
  - Kirsch ("The Intelligent Use of Space"...)
  - ...

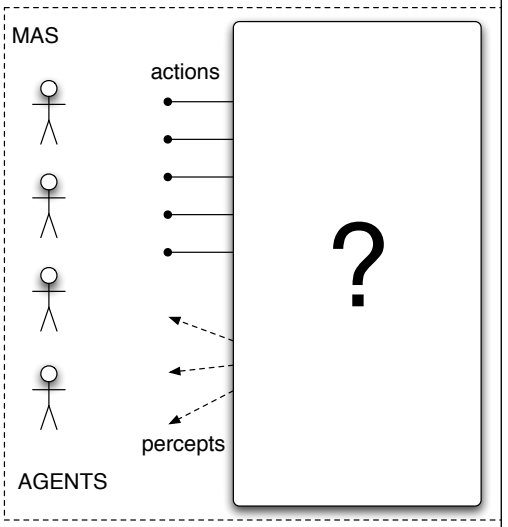
DESIDERATA FOR A WORK ENV. PROGRAMMING MODEL (1/2)

- **Abstraction**
  - keeping the agent abstraction level
    - e.g. no agents sharing and calling *OO objects*
  - effective programming models
    - for controllable and observable computational entities
- **Modularity**
  - away from the monolithic and centralised view
- **Orthogonality**
  - wrt agent models, architectures, platforms
  - support for heterogeneous systems



DESIDERATA FOR A WORK ENV. PROGRAMMING MODEL (2/2)

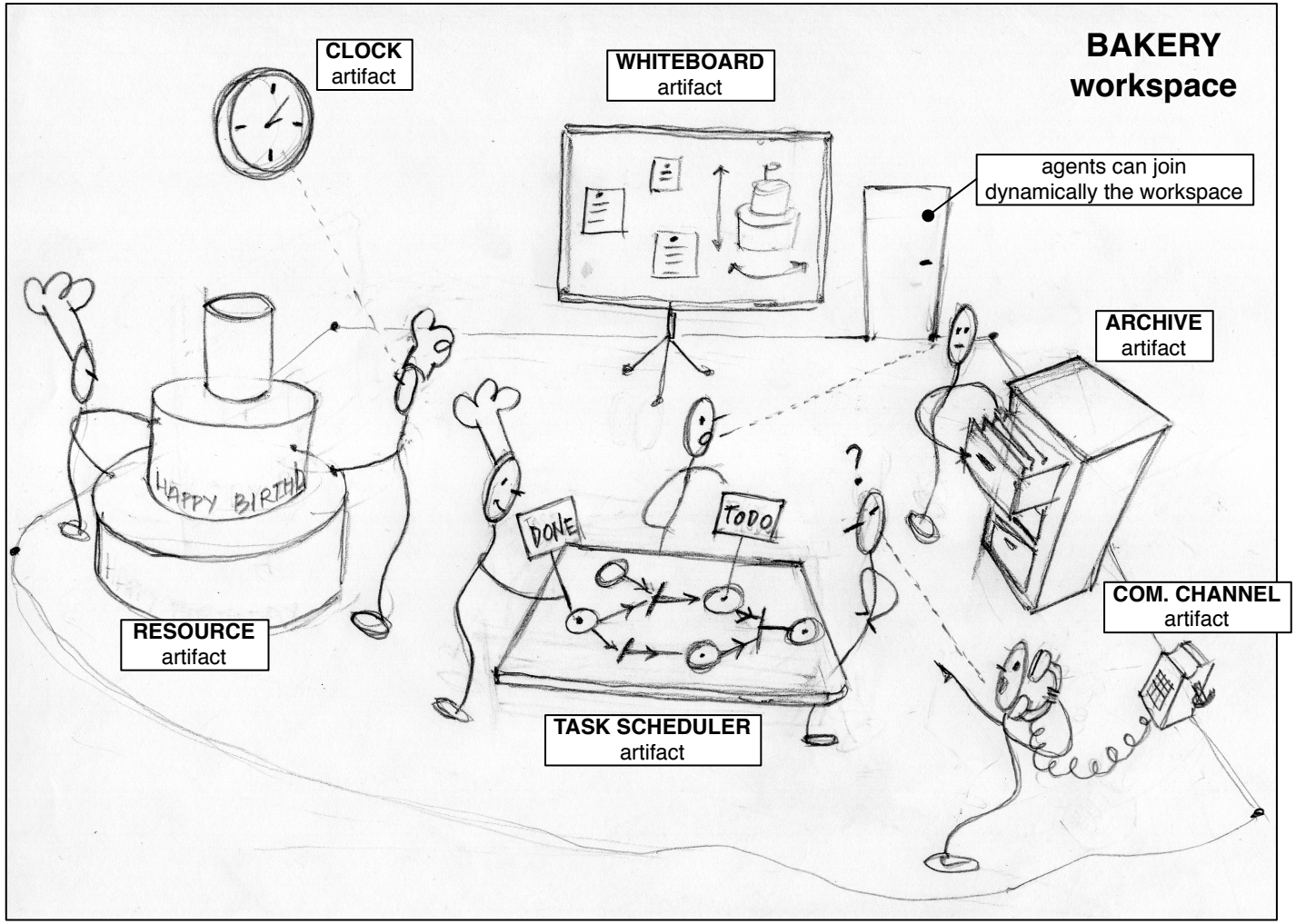
- **(Dynamic) extendibility**
  - dynamic construction, replacement, extension of environment parts
  - support for *open* systems
- **Reusability**
  - reuse of environment parts in different application contexts / domains



PART II

A&A MODEL and CARTAGO
PROGRAMMING MODEL & PLATFORM

AGENTS & ARTIFACTS (A&A) MODEL: BASIC IDEA IN A PICTURE

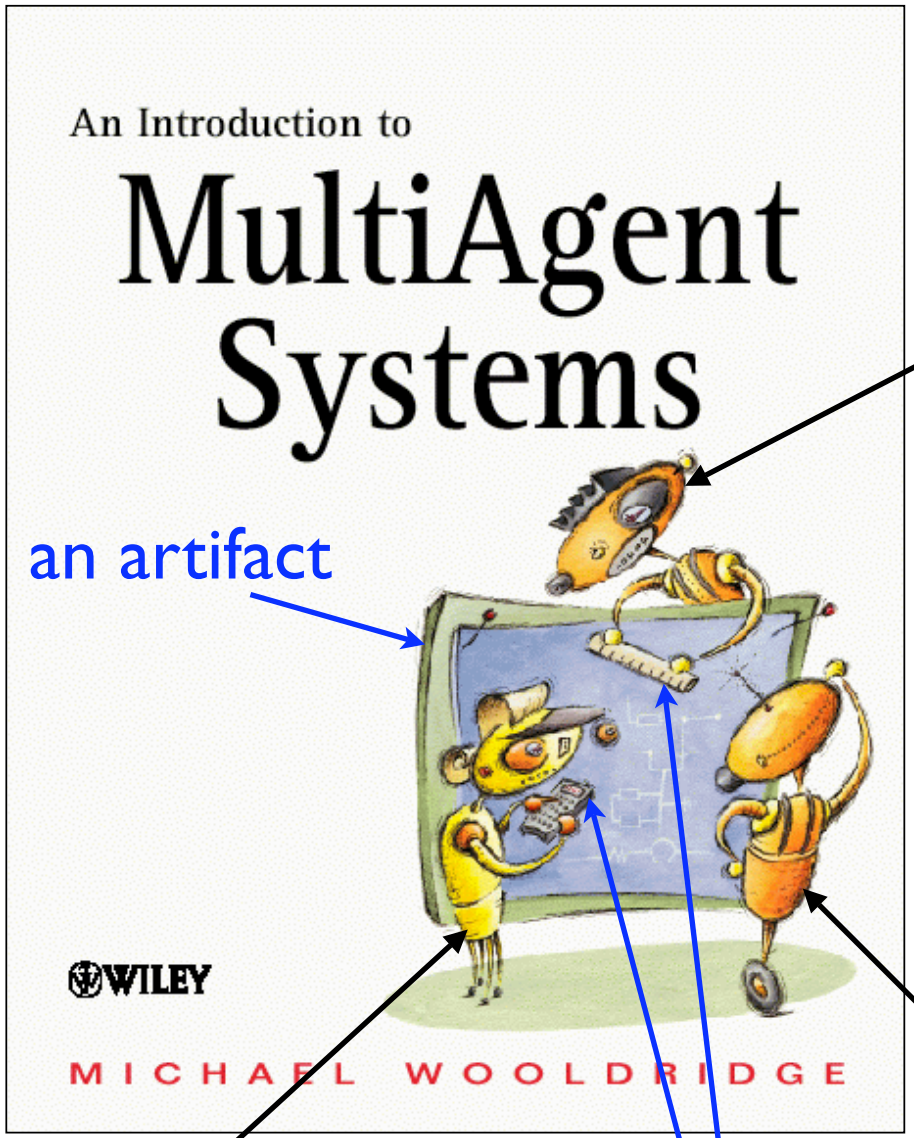


A&A BASIC CONCEPTS

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

ARTIFACTS ARE IN THE MAINSTREAM

...not really, actually...



agent

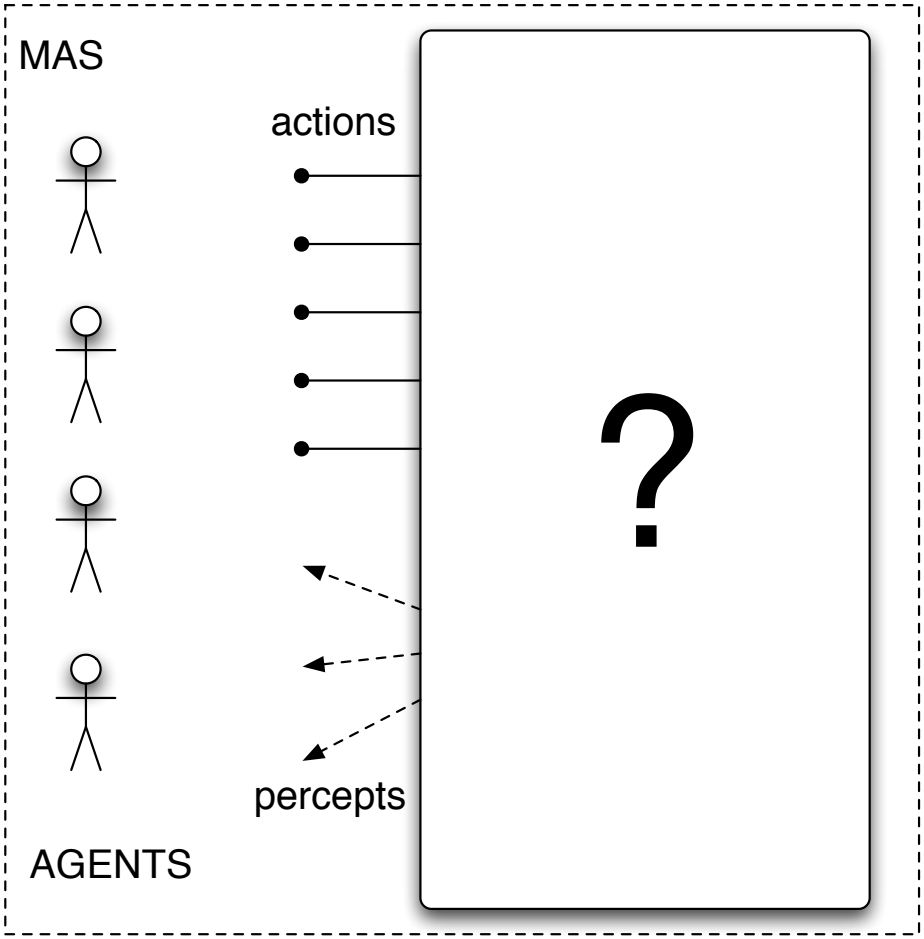
an artifact

agent

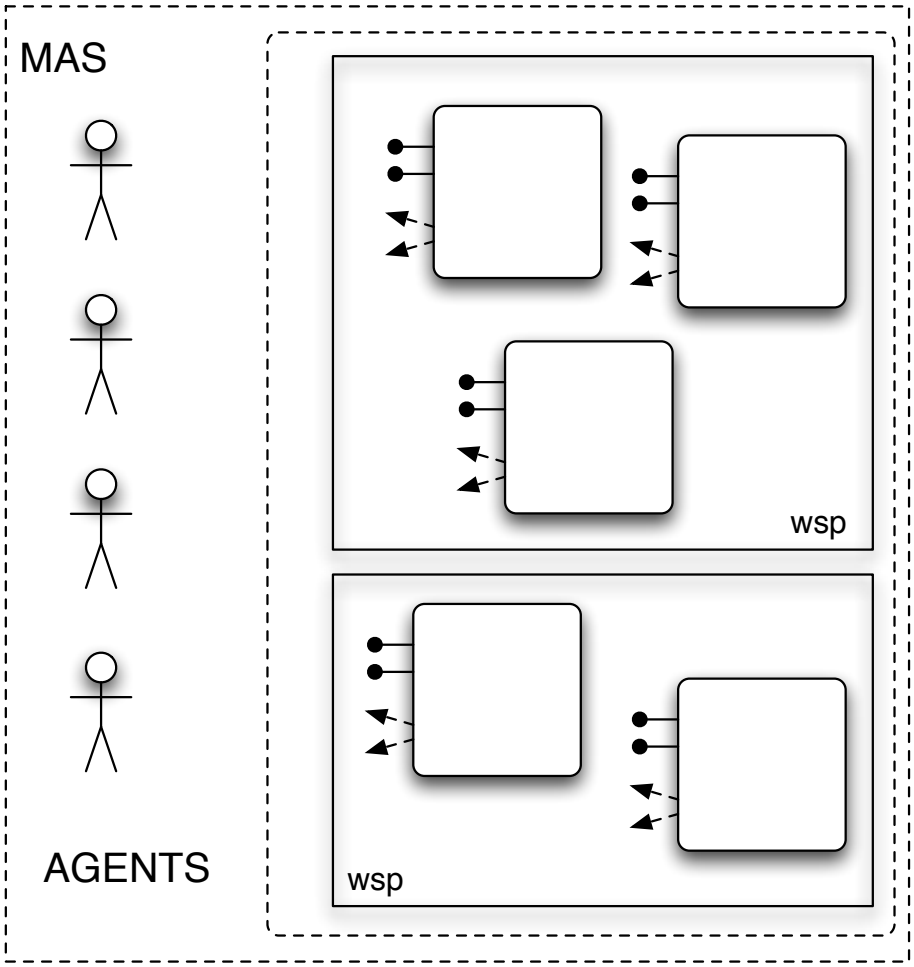
agent

artifacts

WORK ENVIRONMENT IN A&A

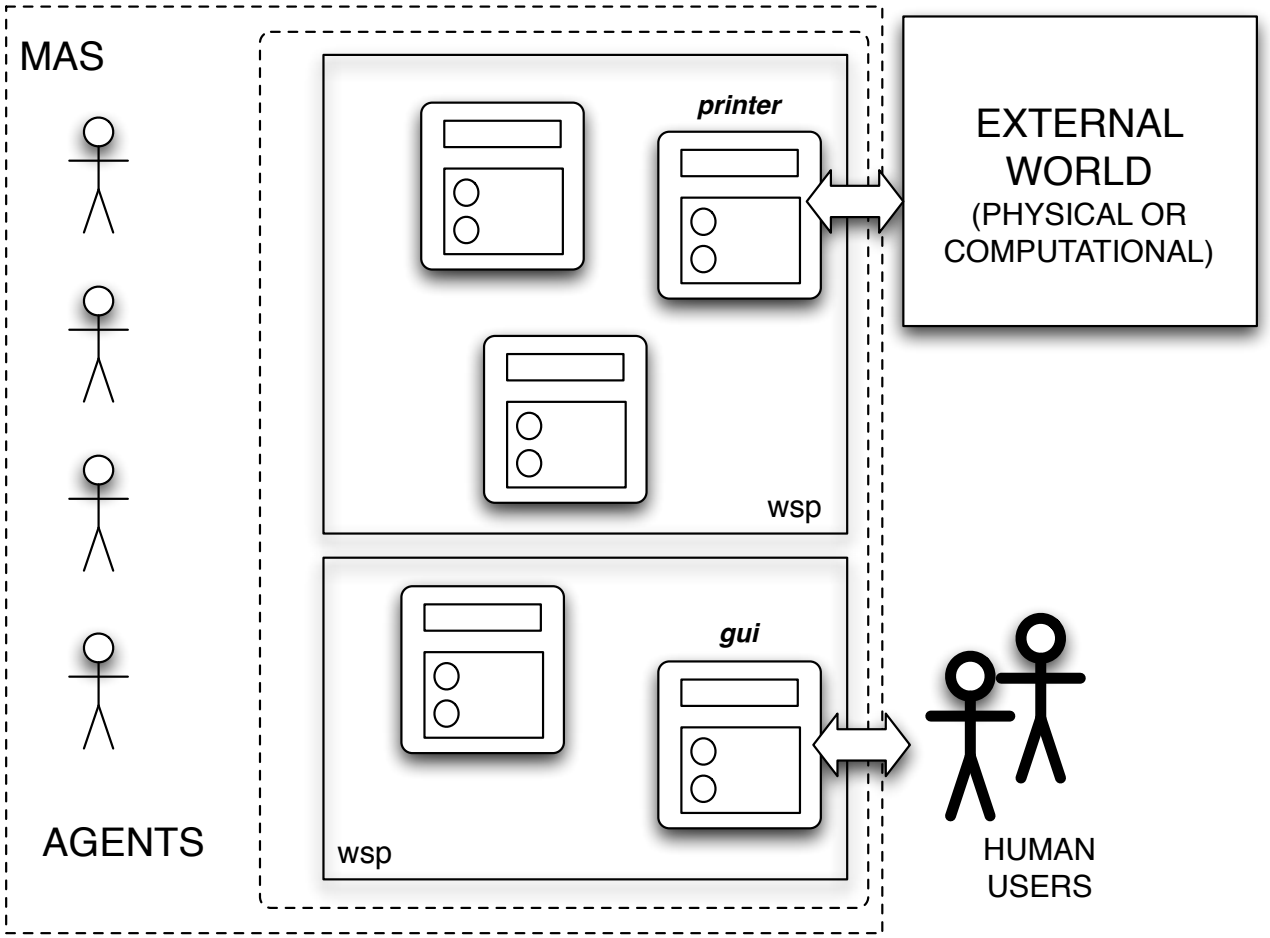


WORK ENVIRONMENT IN A&A

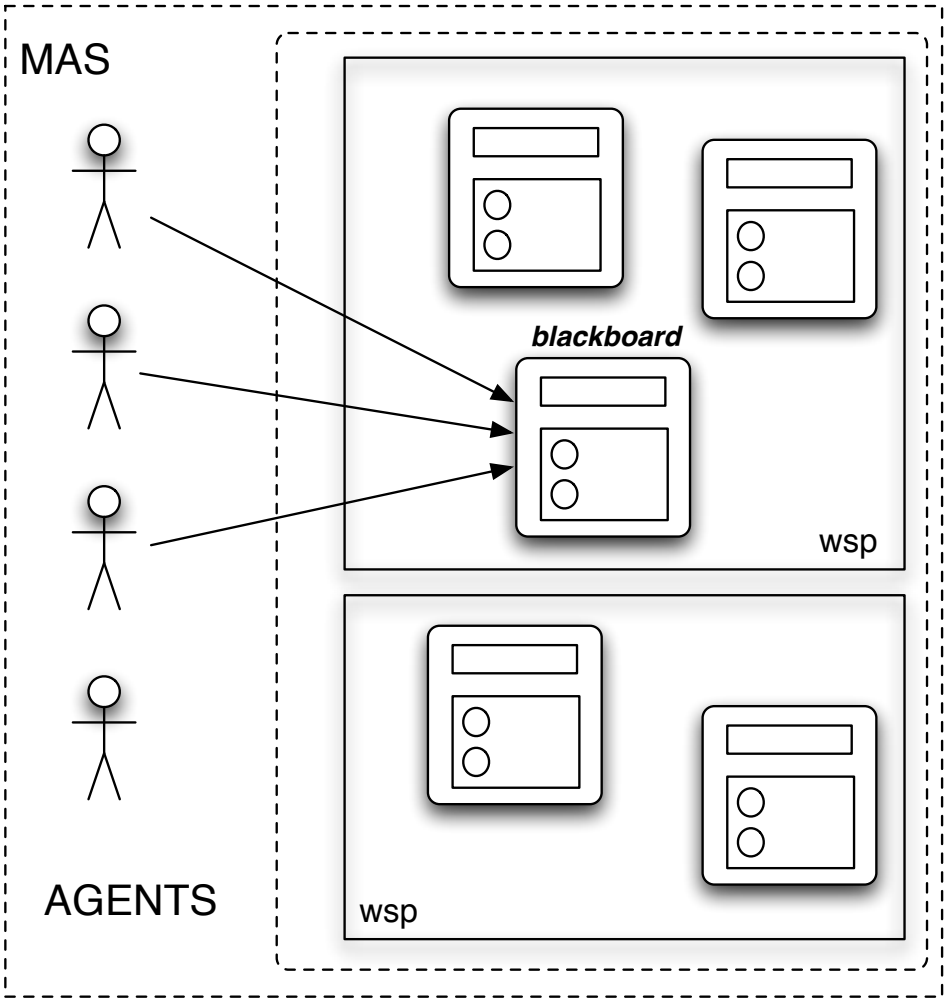


- Abstraction
  - encapsulation
  - information hiding
- Modularization
  - extendibility
  - reuse

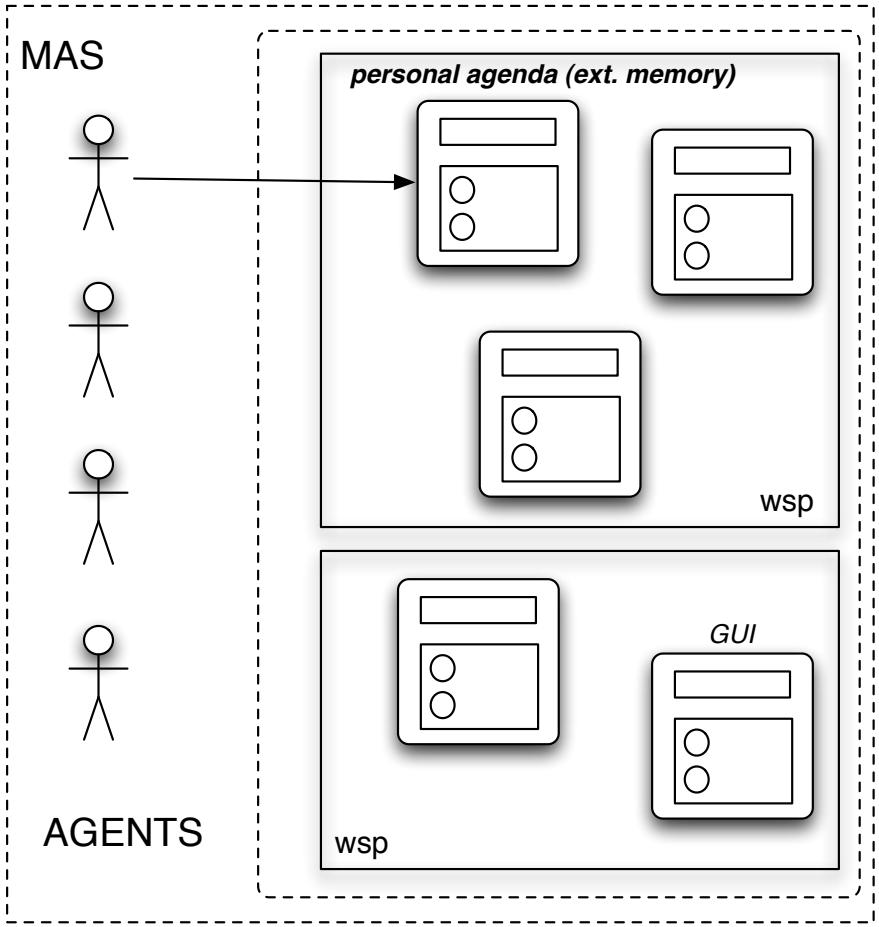
WORK ENVIRONMENT IN A&A



WORK ENVIRONMENT IN A&A

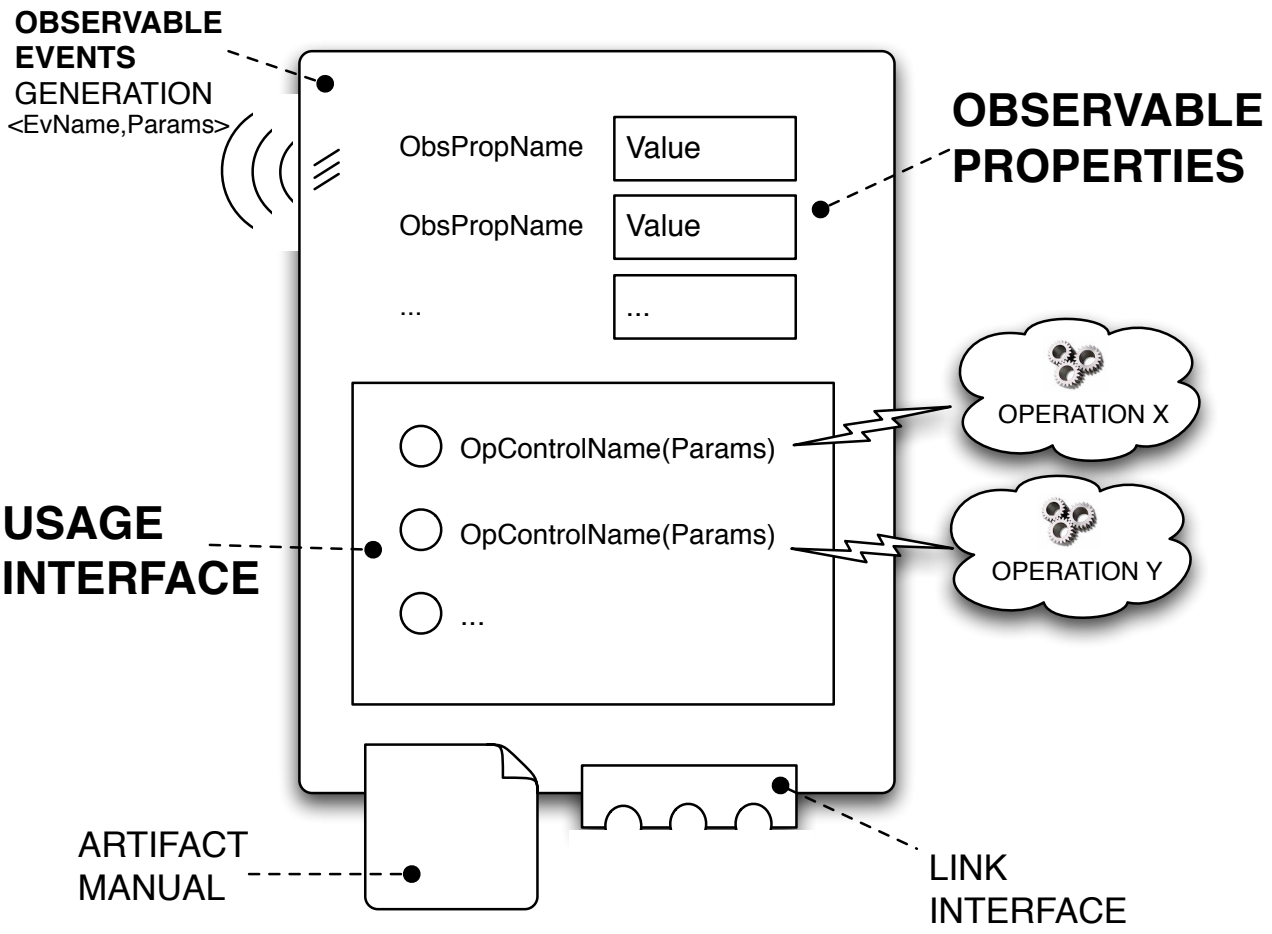


WORK ENVIRONMENT IN A&A

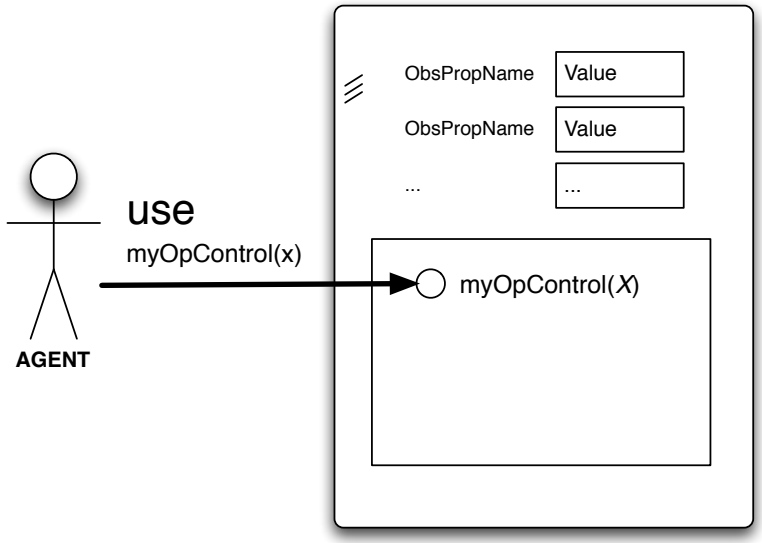


ARTIFACT COMPUTATIONAL MODEL

- “COFFEE MACHINE METAPHOR” -

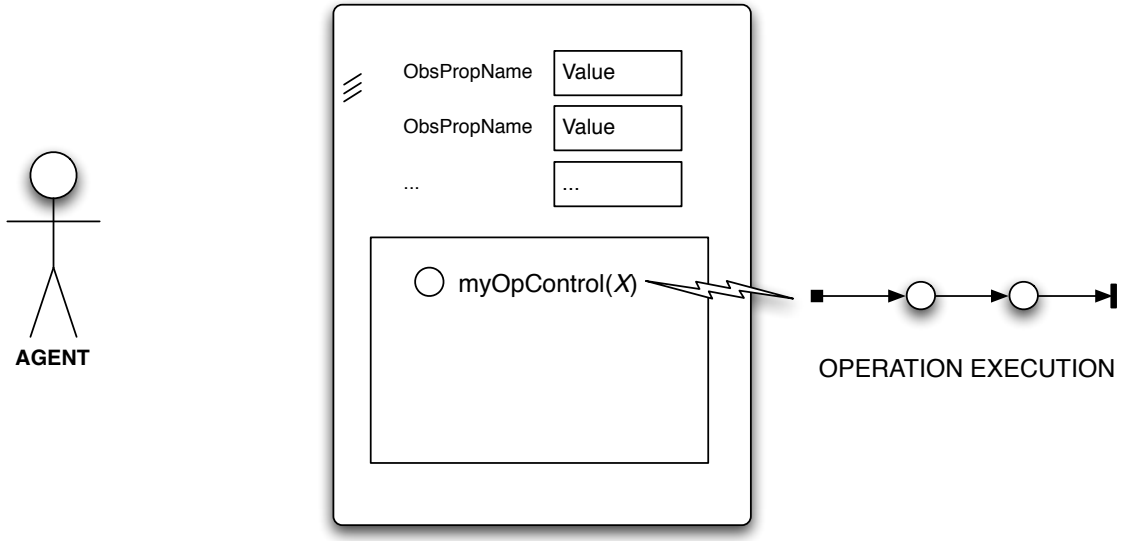


INTERACTION MODEL: **USE** & OBSERVATION



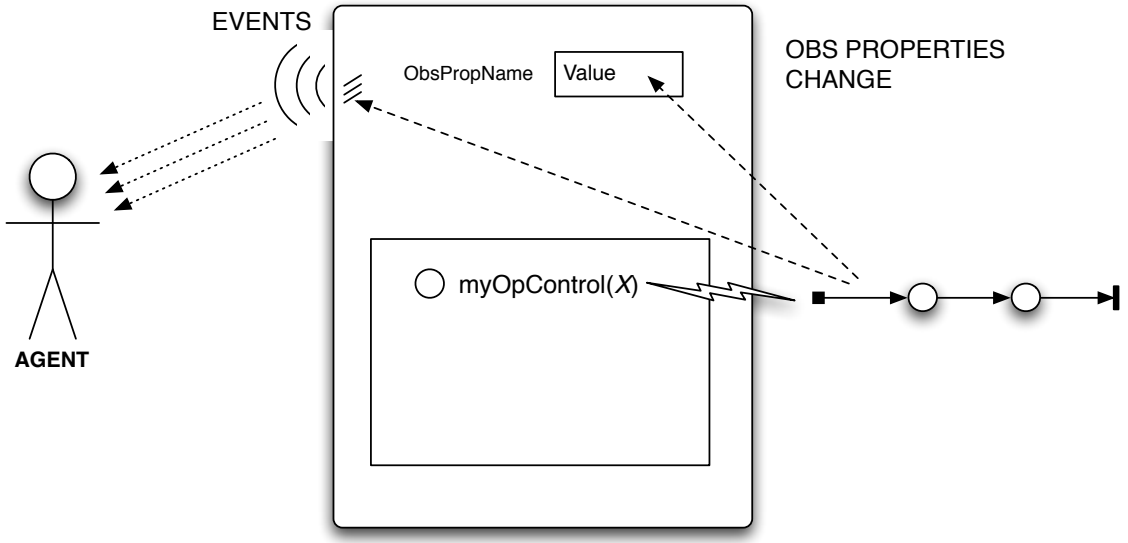
- `use` action
  - acting on op. controls to trigger op execution
  - **synchronisation point** with artifact time/state

INTERACTION MODEL: **USE** & OBSERVATION



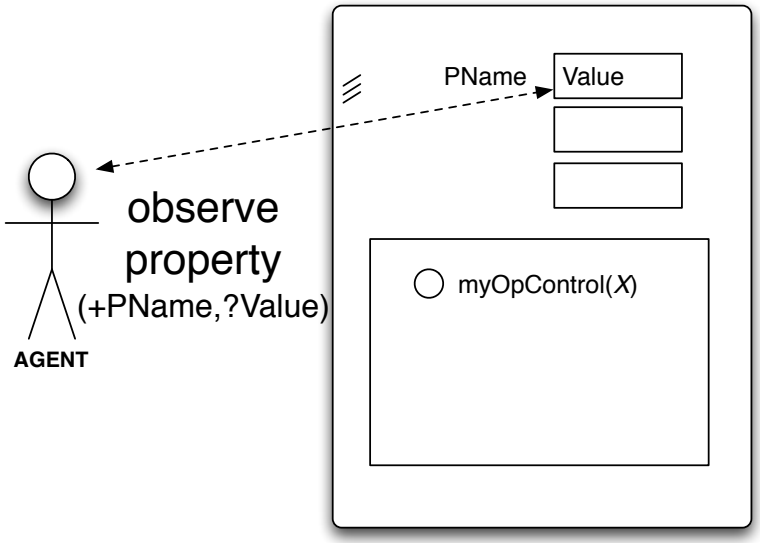
- artifact operation execution
  - asynchronous wrt agent
  - possibly a process structured in multiple atomic steps

INTERACTION MODEL: **USE** & OBSERVATION



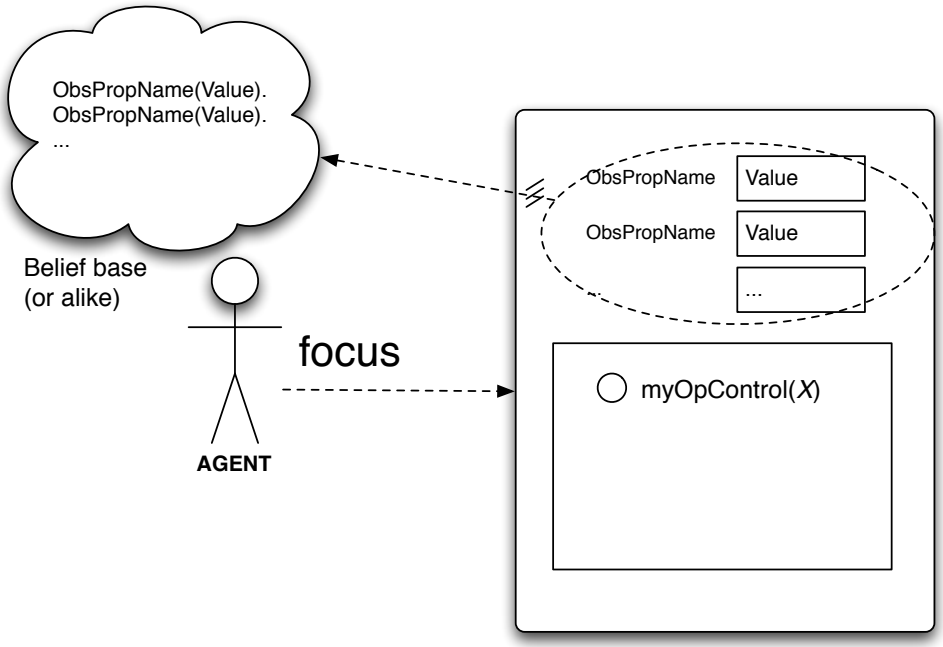
- observable effects
  - observable events & changes in obs property
  - perceived by agents either as (external) events

INTERACTION MODEL: USE & **OBSERVATION**



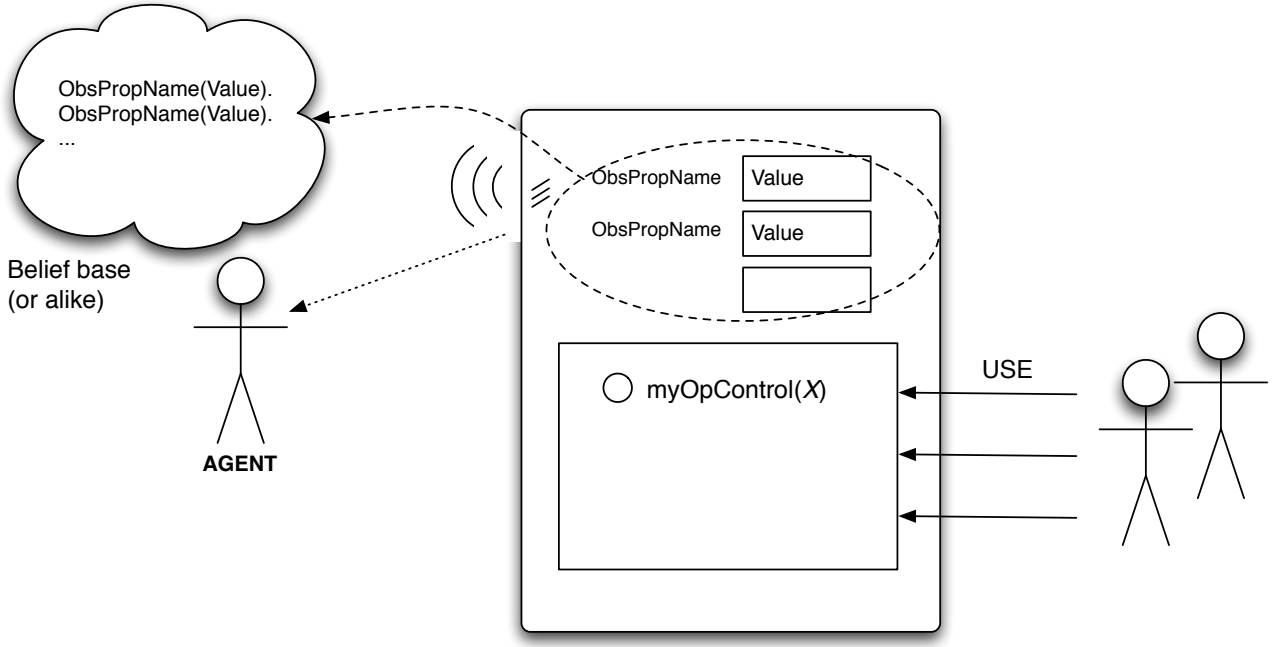
- `observeProperty` action
  - value of an obs. property as action feedback
  - *no interaction*

INTERACTION MODEL: USE & **OBSERVATION**



- `focus` / `stopFocus` action
  - start / stop a continuous observation of an artifact
    - possibly specifying filters
  - observable properties mapped into percepts

INTERACTION MODEL: USE & **OBSERVATION**



- continuous observation
  - observable events (=> agent events)
  - observable properties (=> belief base update)

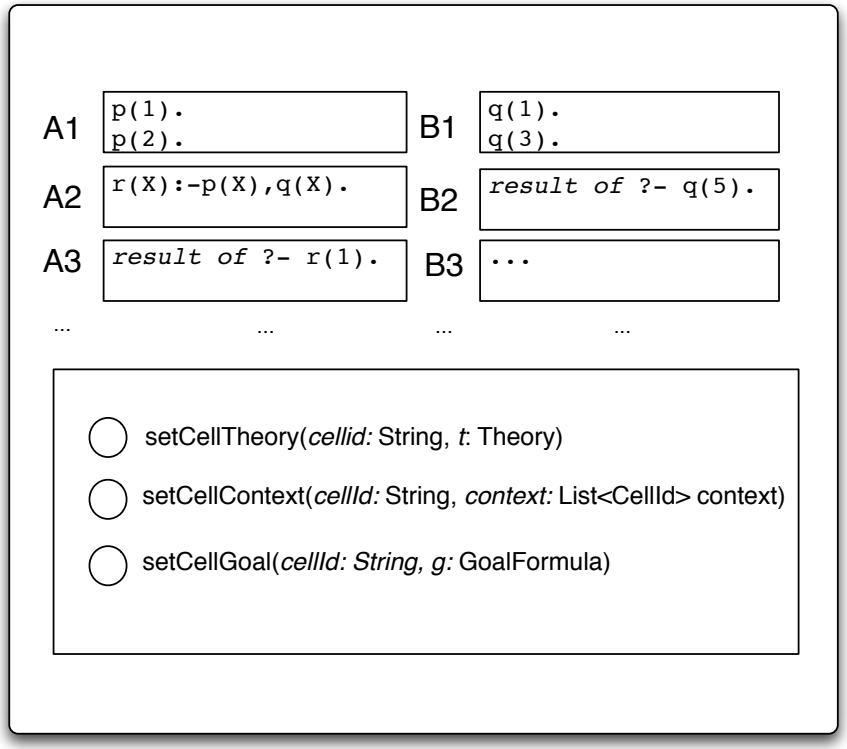
ARTIFACT COMPUTATIONAL MODEL HIGHLIGHTS

- Artifacts as **controllable** and **observable** devices
  - operation execution as a controllable process
    - possibly long-term, articulated
  - two observable levels
    - properties, events
  - transparent management of concurrency issues
    - synchronisation, mutual-exclusion, etc
- Composability through linking
  - also across workspaces
- Cognitive use of artifacts through the *manual*
  - function description, operating instructions
  - work in progress

EXAMPLES OF ARTIFACTS

- Common tools and resources in MAS
  - blackboards, tuple centres, synchronisers,...
  - maps, calendars, shared agenda,...
  - data-base, shared knowledge base,...
  - hardware res. wrappers
  - GUI artifacts
  - ...
  - principled way to design / program / use them inside MAS
- Specific & articulated purposes
  - example: *logic-based spreadsheet artifact*

LOGIC-BASED SPREADSHEET ARTIFACT SKETCH



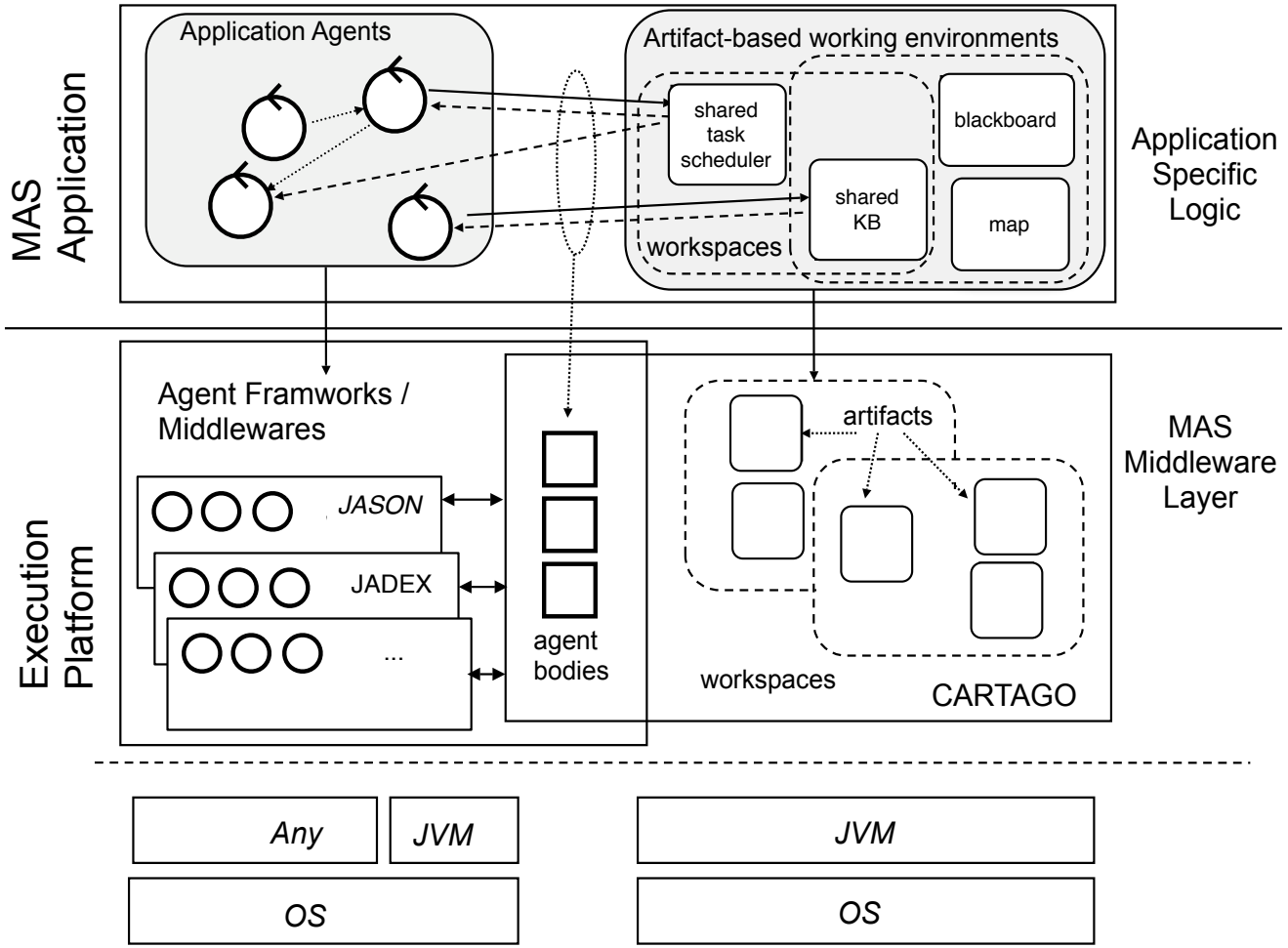
CARTAGO

- **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 plaforms to work wihin artifact-based environment
    - integration with agent programming platforms
- 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
  - available at http://cartago.sourceforge.net

...AND FRIENDS

- Integration with existing agent platforms
  - cognitive agent platforms in particular
    - ongoing cooperation with Jomi, Rafael, Alexander, Lars, Mehdi
  - available bridges: ***Jason***, **Jadex**, simpA
    - ongoing: **2APL**, Jade
  - "agent body" notion for technically realising the integration
    - effectors and sensors to act upon and sense artifacts
    - controlled by an agent mind executed on some agent platform
- Outcome
  - developing open and heterogenous MAS
  - different perspective on *interoperability*
    - sharing and working in a common work environment
    - common data-model based on Object-Oriented or XML-based data structures

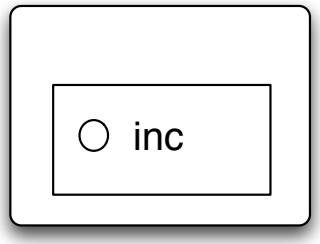
CARTAGO ARCHITECTURE



DEFINING ARTIFACTS IN CARTAGO

- Single class extending `alice.simpa.Artifact`
- Specifying the operations
  - atomic: **`@OPERATION`** methods
    - name+params -> usage interface control
    - no return value
  - structured
    - linear composition of atomic operation steps composed dynamically
  - `init` operation
    - automatically executed when the artifact is created
- Specifying artifact state
  - instance fields of the class

SIMPLE EXAMPLE #1



USAGE INTERFACE:

inc: [op_exec_completed]

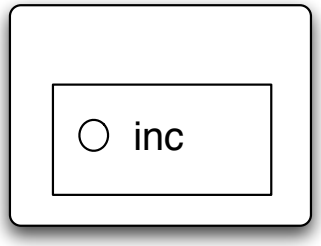
```
public class Count extends Artifact {
  int count;

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

  @OPERATION void inc(){
     count++;
  }
}
```

ARTIFACT OBSERVABLE EVENTS

- Observable events
  - generated by **signal** primitive
  - represented as labelled tuples
    - event_name(Arg0,Arg1,...)

- Automatically made observable to...
  - the agent who executed the operation
  - all the agents observing the artifact

SIMPLE EXAMPLE #2



USAGE INTERFACE:

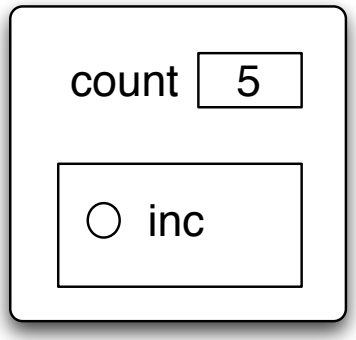
inc: [new_count_value,
op_exec_completed]

```
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 **defineObsProperty** primitive
    - characterized by a property name and a property value
  - internal primitives to read / update property value
    - **updateObsProperty**
    - **getObsProperty**
- Automatically made observable to all the agents observing the artifact

SIMPLE EXAMPLE #3



OBSERVABLE PROPERTIES:

count: int

USAGE INTERFACE:

inc: [op_exec_completed]

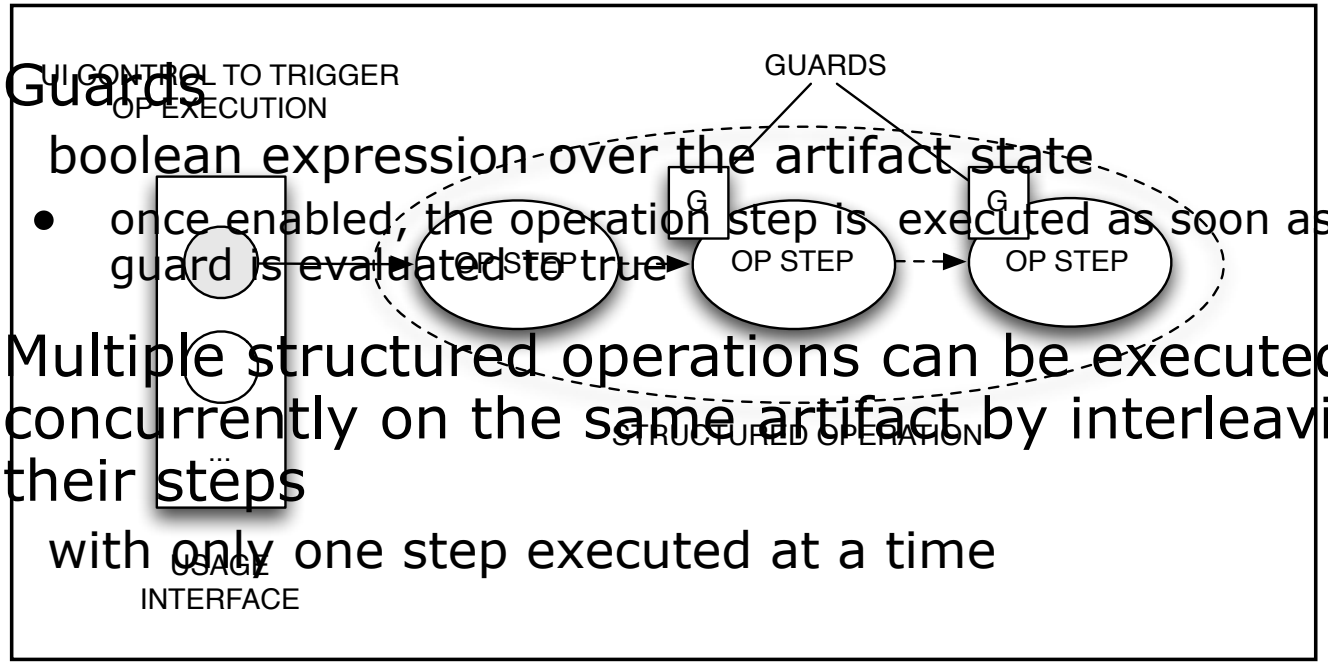
```
public class Count extends Artifact {

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

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

MORE ON ARTIFACTS

- Structured operations
  - specifying operations composed by chains of atomic operation steps
  - to support the concurrent execution of multiple operations on the same artifact
    - by interleaving steps
- Linkability
  - dynamically composing / linking multiple artifacts together
- Artifact *manual*
  - machine-readable description of artifact functionality and operating instructions

STRUCTURED OPERATIONS

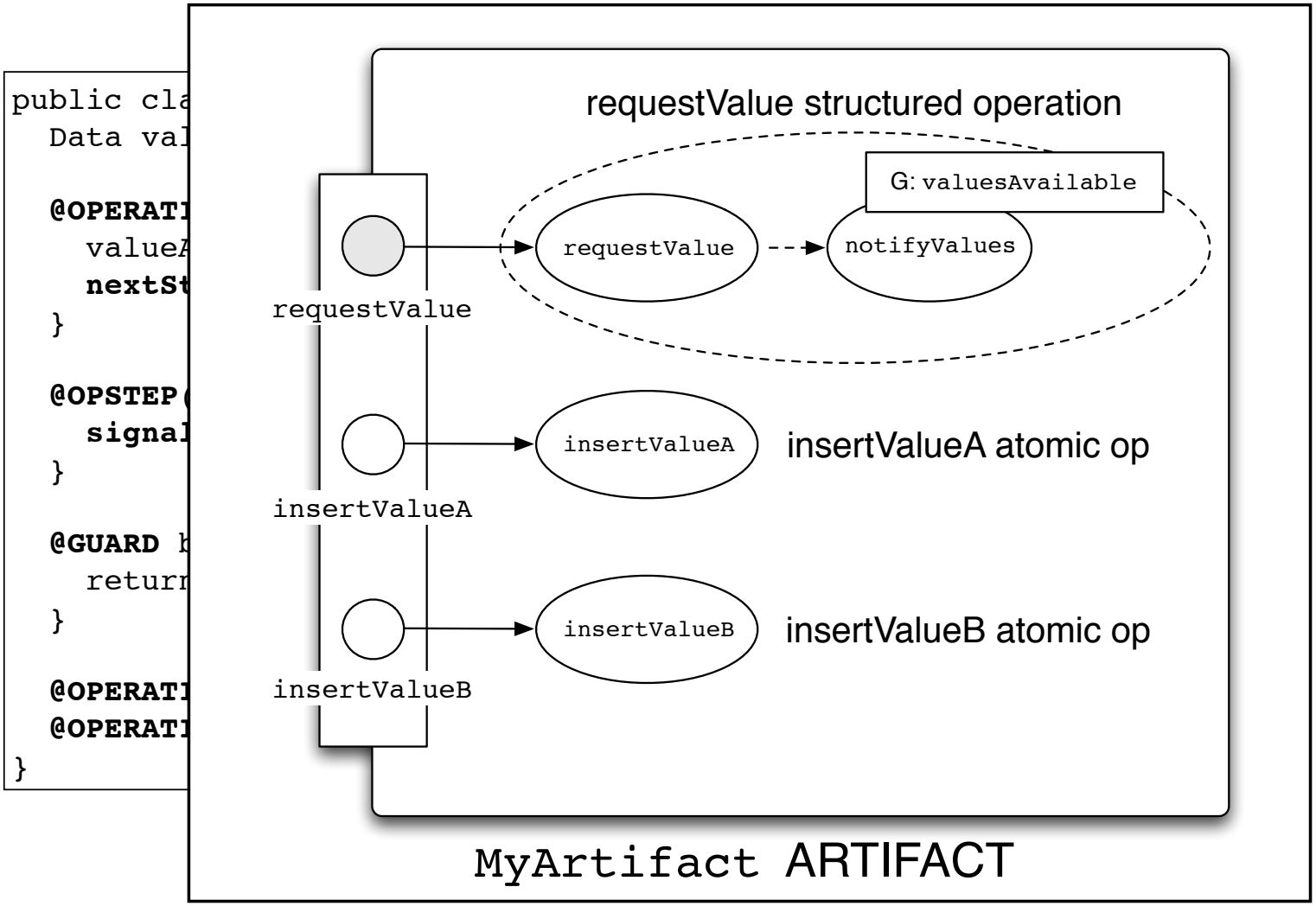
- Complex operations as chains of guarded atomic operation step execution
  - `@OPSTEP` methods



- Guards
  - boolean expression over the artifact state
    - once enabled, the operation step is executed as soon as the guard is evaluated to true

> Multiple structured operations can be executed concurrently on the same artifact by interleaving their steps
  - with only one step executed at a time

EXAMPLE


public cla
Data val
@OPERATI
valueA
nextSt
}
@OPSTEP(
signal
}
@GUARD k
return
}
@OPERATI
@OPERATI
}
requestValue structured operation
G: valuesAvailable
requestValue
notifyValues
requestValue
insertValueA
insertValueA atomic op
insertValueA
insertValueB
insertValueB atomic op
insertValueB
MyArtifact ARTIFACT

AGENT ABSTRACT API

- Extending agent actions with a basic set to work within artifact-based environments

workspace management	**joinWsp**(Name,?WspId,+Node,+Role,+Cred) **quitWsp**(Wid)
artifact use	**use**(Aid,OpCntrName(Params),+Sensor,+Timeout,+Filter) **sense**(Sensor,?Perception,+Filter,+Timeout) **grab**([Aid]) **release**([Aid])
artifact observation	**observeProperty**(Aid,PName,?PValue) **focus**(Aid,+Sensor,+Filter) **stopFocus**(Aid)
artifact instantiation, discovery, management	**makeArtifact**(Name,Template,+ArtifactConfig,?Aid) **lookupArtifact**(Name,?Aid) **disposeArtifact**(Aid)

RAW AGENT API

```
joinWsp

use
sense
focus
stopFocus

grab
release
```

+

basic set of artifacts available in each workspace

- `factory`
- `registry`
- `security-registry`
- `console`

implementing non primitive actions:

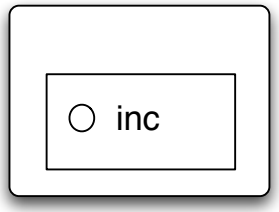
`makeArtifact` => use `factory`

`lookupArtifact` => use `registry`

CARTAGO API TASTE

EX1: SIMPLE USE 1a

- A shared counter



USAGE INTERFACE:

inc: [new_count_value,
op_exec_completed]

```
MAS mas0a {

  environment:
    alice.c4jason.CEnvStandalone

  agents:
    user0a agentArchClass alice.c4jason.CAgentArch;
}
```

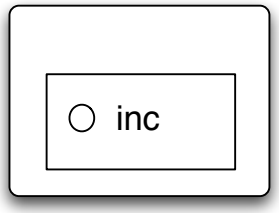
```
package test;
import alice.cartago.*;

public class Counter0 extends Artifact {

  int count;

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

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

```
// user0a

!create_and_use.

+!create_and_use : true
    <-  cartago.makeArtifact("mycount","test.Counter0",C);
        // first use
        cartago.use(C,inc);
        // second use
        cartago.use(C,inc,s0);
        cartago.sense(s0,new_count_value(V));
        // log the value
        cartago.use(console,println("[USER] value ",V)).
```

CARTAGO API TASTE

EX2: SIMPLE USE 1b

- A shared counter



USAGE INTERFACE:

inc: [new_count_value,
op_exec_completed]

```
MAS mas0b {

  environment:
    alice.c4jason.CEnvStandalone

  agents:
    user0b agentArchClass alice.c4jason.CAgentArch;
}
```

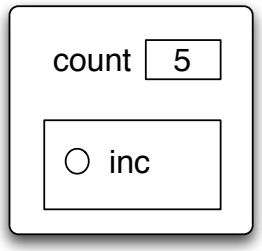
```
package test;
import alice.cartago.*;

public class Counter0 extends Artifact {

  int count;

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

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

```
// user0b

!create_and_use.

+!create_and_use : true
    <-  cartago.makeArtifact("mycount","test.Counter0",C);
        // first use
        cartago.use(C,inc);
        // second use
        cartago.use(C,inc).

+artifact_event(C,new_count_value(V)) : true
    <-  cartago.use(console,println("[USER] value ",V)).
```

CARTAGO API TASTE

EX3: USE & OBSERVATION

- Counter with obs prop



OBSERVABLE PROPERTIES:

count: int

USAGE INTERFACE:

inc: [op_exec_completed]

```
MAS mas1 {

  environment:
    alice.c4jason.CEnvStandalone

  agents:
    observer agentArchClass alice.c4jason.CAgentArch;
    user agentArchClass alice.c4jason.CAgentArch #2;
}
```

```
package test;

public class Counter1 extends Artifact {
  @OPERATION void init(){
    defineObsProperty("count",0);
  }

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

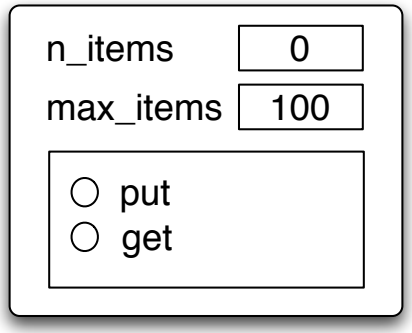
```
// observer
!observe.

+!observe : true
    <-  cartago.makeArtifact("my_counter","test.Counter1", Count);
        cartago.focus(Count).

+count(V) : true
    <-  cartago.use(console,println("current count observed: ",V)).
```

```
// user
!use_count.

+!use_count : true
  <- ?counter_to_use(Counter) ;
     +cycle(0) ;
     !use_count(Counter).

+?counter_to_use(Counter) : true
  <- cartago.lookupArtifact("my_counter",Counter).

-?counter_to_use(Counter) : true
  <- .wait(100);
     ?counter_to_use(Counter).

+!use_count(C) : cycle(N) & N < 10
  <- -cycle(N);
     cartago.use(C,inc,mySensor0);
     cartago.sense(mySensor0,"operation_completed");
     !have_a_rest ;
     +cycle(N+1) ;
     !use_count(C).

+!use_count(C) : cycle(10).

+!have_a_rest : true
  <- .wait(10).
```

CARTAGO API TASTE

EX4: USING OP CONTROLS WITH GUARDS

- *bounded-buffer* artifact for open producers-consumers scenarios



OBSERVABLE PROPERTIES:

n_items: int+
max_items: int

Invariants:
n_items <= max_items

USAGE INTERFACE:

put(*item*:Item) / (n_items < max_items):
[op_exec_completed]

get / (n_items >= 0) :
[*new_item*(item:Item), op_exec_completed]

```
public class BBuffer extends Artifact {
  private LinkedList<Item> items;

  @OPERATION void init(int nmax){
    items = new LinkedList<Item>();
    defineObsProperty("maxNItems",nmax);
    defineObsProperty("nItems",0);
  }

  @OPERATION(guard="bufferNotFull") void put(Item obj){
    items.add(obj);
    updateObsProperty("nItems",items.size()+1);
  }
  @GUARD boolean bufferNotFull(Item obj){
    int maxItems = getObsProperty("maxNItems").intValue();
    return items.size() < maxItems;
  }

  @OPERATION(guard="itemAvailable") void get(){
    Item item = items.removeFirst();
    updateObsProperty("nItems",items.size()-1);
    signal("new_item",item);
  }
  @GUARD boolean itemAvailable(){ return items.size() > 0; }
}
```

CARTAGO API TASTE

PRODUCERS & CONSUMERS IN JASON

PRODUCERS

```
!produce.

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

+!produceItems : true <-
  ?nextItemToProduce(Item);
  cartago.use(myBuffer,put(Item),5000);
  !produceItems.

+?nextItemToProduce(Item) : true <- ...

+!setupTools(Buffer) : true <-
  cartago.makeArtifact("myBuffer",
               "test.BBuffer",[10],Buffer).
-!setupTools(Buffer) : true <-
  cartago.lookupArtifact("myBuffer",Buffer).
```

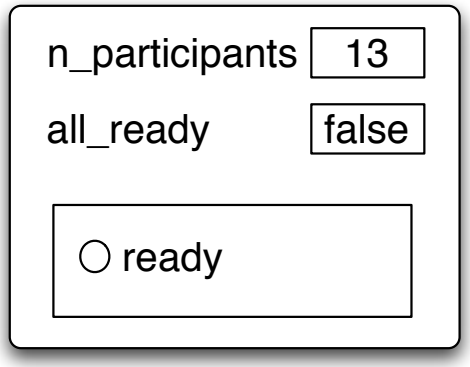
CONSUMERS

```
!consume.

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

+!consumeItems : true <-
  cartago.use(myBuffer,get,s0,5000);
  cartago.sense(s0,new_item(Item),5000);
  !consumeItem(Item);
  !consumeItems.

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

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

CARTAGO API TASTE

EX5: ARTIFACT WITH STRUCTURED OPERATIONS

- simple synchronization artifact (~barrier)



OBSERVABLE PROPERTIES:

n_participants: N > 0
all_ready: {true,false}

USAGE INTERFACE:

ready / true :
[accepted(N), op_exec_completed]

```
class SimpleSynchronizer extends Artifact {
  int nReady;

  @OPERATION void init(int nParticipants){
    defineObsProperty("all_ready",false);
    defineObsProperty("n_participants",0);
    nReady = 0;
    this.nParticipants = nParticipants;
  }

  @OPERATION void ready(){
    nReady++;
    signal("accepted",nReady);
    nextStep("setAllReady");
  }

  @OPSTEP(guard="allReady") void setAllReady(){
    updateObsProperty("all_ready",true);
  }

  @GUARD boolean allReady(){
    return nReady ==
          getObsProperty("n_participants").intValue();
   }
}
```

CARTAGO API TASTE
SYNCH USER (ACTIVE/REACTIVE)

SYNCH USER - WITH SENSOR

```
!work.

+!work: true <-
  ...
  // locate the synch tool
  cartago.lookupArtifact("mySynch",Synch);
  // ready for synch
  cartago.use(Synch,ready,sid);
  // waiting all synchs
  cartago.sense(sid,op_exec_completed(ready);
  // all ready, go on.
  ...
```

SYNCH USER - REACTIVE

```
!work.

+!work: true <-
  ...
  // locate the synch tool
  cartago.lookupArtifact("mySynch",Synch);
  // ready for synch
  cartago.use(Synch,ready).
  // observe it.
  cartago.focus(Synch).

// react to all_ready(true) percept
+artifact_event(mySynch,all_ready) : true
  <-
  // all ready, go on.
  ...
```

OPEN WORKSPACES & DISTRIBUTION

- Agents can dynamically join and quit workspaces
  - heterogeneous & “remote” agents
    - *Jason*, JADEX, simpA, etc.
  - in Jason MAS
    - `alice.c4jason.CEnv` environment class
- RBAC model for ruling agent access & use of artifacts
  - `security-registry` artifact to keep track of roles and role policies
    - making roles & policies observable and modifiable by agents themselves
- Distribution
  - agents can join and work concurrently in multiple workspaces at a time
  - workspaces can belong to different CARTAGO nodes

PART III

ADVANCED (SELECTED) ISSUES & ONGOING WORK

GOAL-DIRECTED USE OF ARTIFACTS

- Objective
  - enabling intelligent agents to dynamically discover and use (and possibly construct) artifacts according to their individual / social objectives
  - *open* systems
    - systems with different kinds of aspects not defined a priory by MAS designers
- Toward fully autono(mic/mous) systems
  - exploring self-organizing systems based on intelligent agents
    - self-CHOP+CA
      - configuring, healing, optimizing, protecting + constructing, adapting

GOAL-DIRECTED USE: SOME CORE ASPECTS

- Defining an “agent-understandable” model & semantics for artifact manual
  - how to specify artifact functionalities
  - how to specify artifact operating instructions
- How to extend agent basic reasoning cycle including reasoning about artifacts
  - relating agent goals and artifact functions
  - relating agent plans and artifact operating instructions and function description
- Reference literature
  - Artificial Intelligent and Distributed AI
  - Semantic Web / Ontologies

ONGOING EVALUATION (APPLICATIONS)

- ORA4MAS
  - exploiting artifacts to build an organisational infrastructure
- CARTAGO-WS
  - basic set of artifacts for building SOA/WS applications
    - interacting with web services
    - implementing web services
- ARTIFACT LIBRARIES
  - setting up a set of reusable artifacts in MAS applications

AVAILABLE PROJECTS & THESES

- CARTAGO extensions
  - integrations with other agent platforms
    - 2APL, JADE
- Goal-directed use of artifacts
  - models & languages for manual
  - artifacts in the loop of reasoning
- Applying intelligent agents+CARTAGO
  - SOA/WS/Web based MAS
    - engineering SOA/WS and Web systems using MAS
    - CARTAGO-WS, CARTAGO-Web
  - MAS-based Autonomic Systems / Computing & Virtualization
    - MAS for automated management of virtual machines & virtual resources