

EOS Final Meeting, Torino 9-10 Gennaio 2009

From Objects based Middleware to Artifact based Work Environment in Multi-Agent Systems

Andrea Omicini ⁽¹⁾, **Michele Piunti** ^(1,2),

Alessandro Ricci ⁽¹⁾, Mirko Viroli ⁽¹⁾

⁽¹⁾ aliCE group at DEIS, Università di Bologna, Cesena

⁽²⁾ Istituto Scienze e Tecnologie della Cognizione, ISTC-CNR, Roma

michele.piunti@unibo.it

OUTLINE

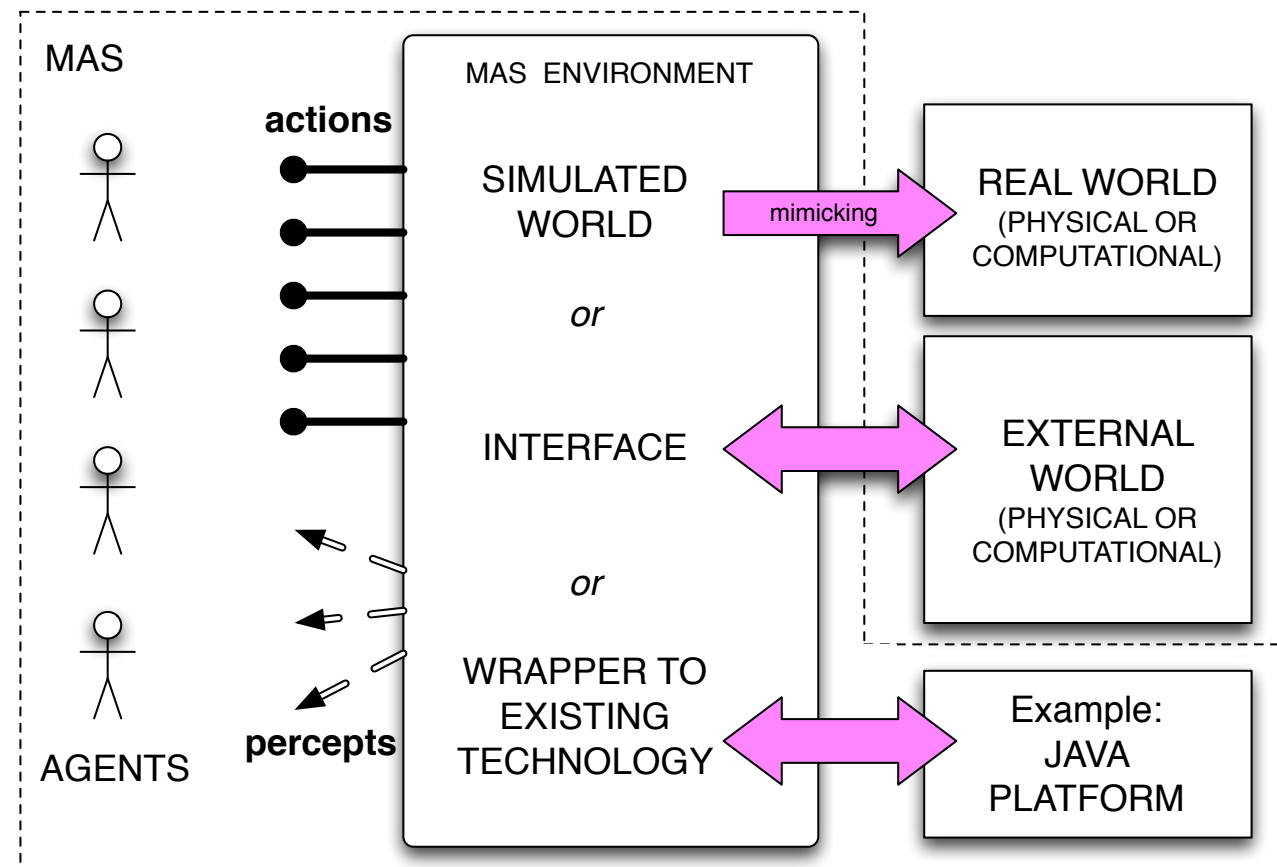
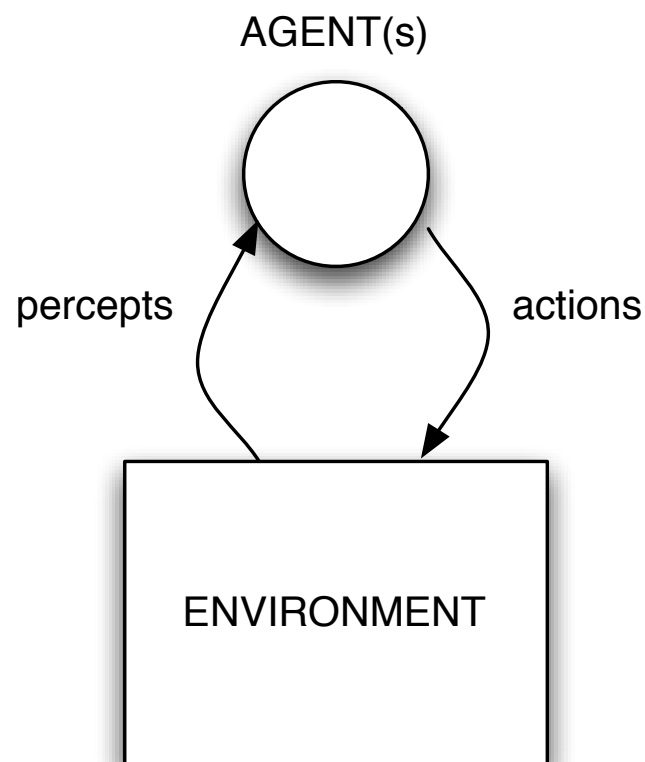
- Agents and Artifacts (A&A) approach to MAS
- Shaping Agents and Artifacts Interactions
- Examples
- Conclusions

ENVIRONMENT MODELS IN MAS

TRADITIONAL VIEW:

Infrastructure for *message exchange*

Medium for external worlds

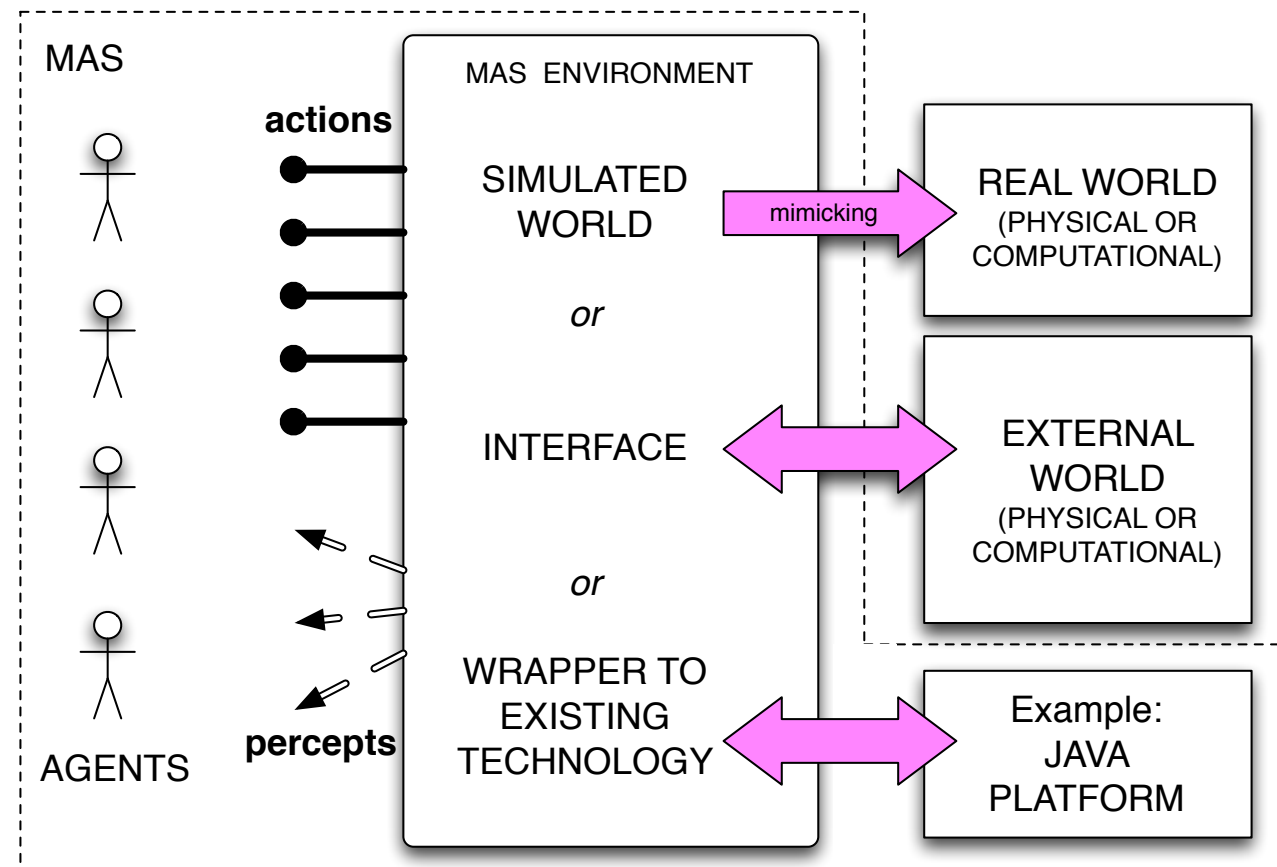
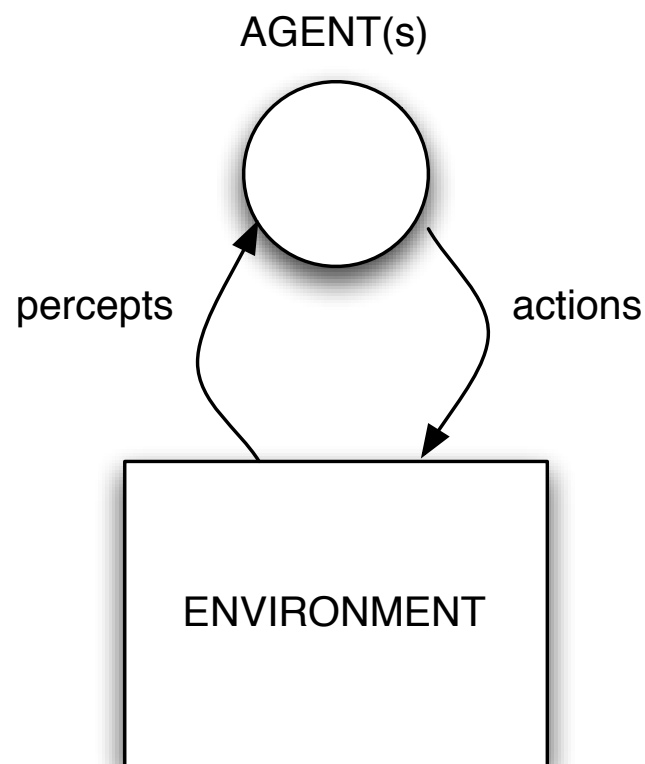


ENVIRONMENT MODELS IN MAS

TRADITIONAL VIEW:

Infrastructure for *message exchange*

Medium for external worlds



Environment as monolithic / centralised component :

- defining agent (external) actions
 - Static list of actions, shared by all the agents
- generator of (domain dependent) percepts
 - which percepts for which agents

RATIONALE

Environment as first class entity in agent dynamics
[Weyns, Omicini, Odell]

Can we leave aside Agents-Environments Interactions ?

RATIONALE

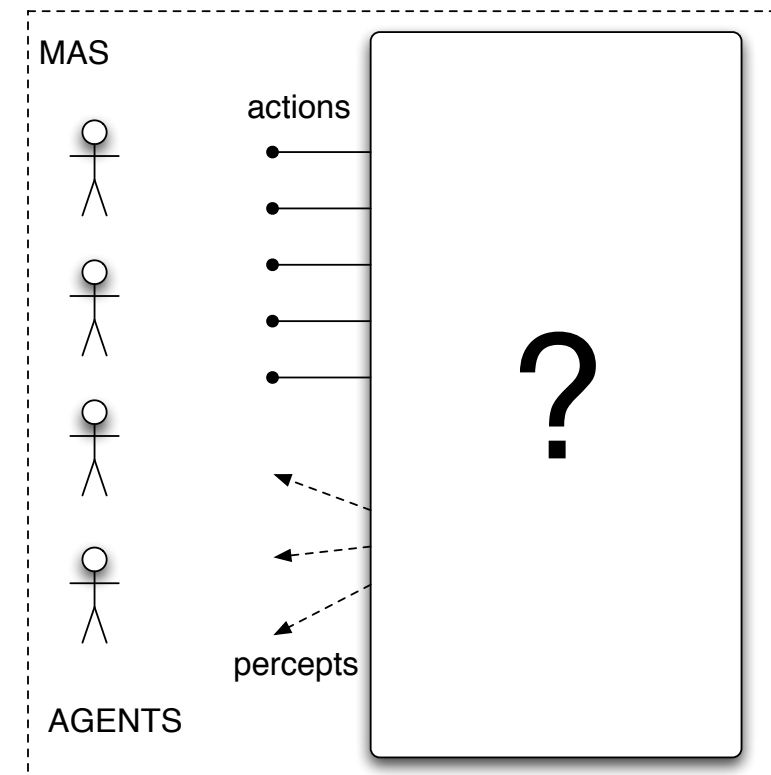
Environment as first class entity in agent dynamics
[Weyns, Omicini, Odell]

Can we leave aside Agents-Environments Interactions ?

A-E Interactions, Cognitive Agents, Opennes...

- No specific (programming) model for defining structure and behaviour
- Interactions based on lower-level language features and mechanisms (i.e. Java)
- No support for interoperability across different platforms

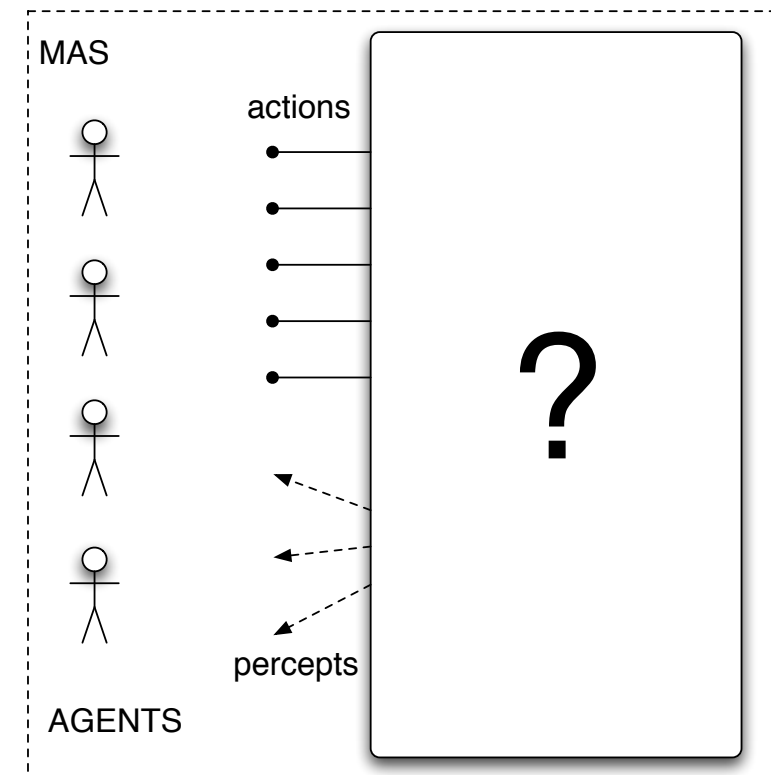
DESIDERATA FOR A WORK. ENV. PROGRAMMING MODEL (1/2)



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

- **Abstraction**

- keeping the agent abstraction level
 - concepts like Action, Perception, Observation
 - i.e., no agents sharing and calling *OO objects*
- effective programming models
 - for controllable and observable computational entities



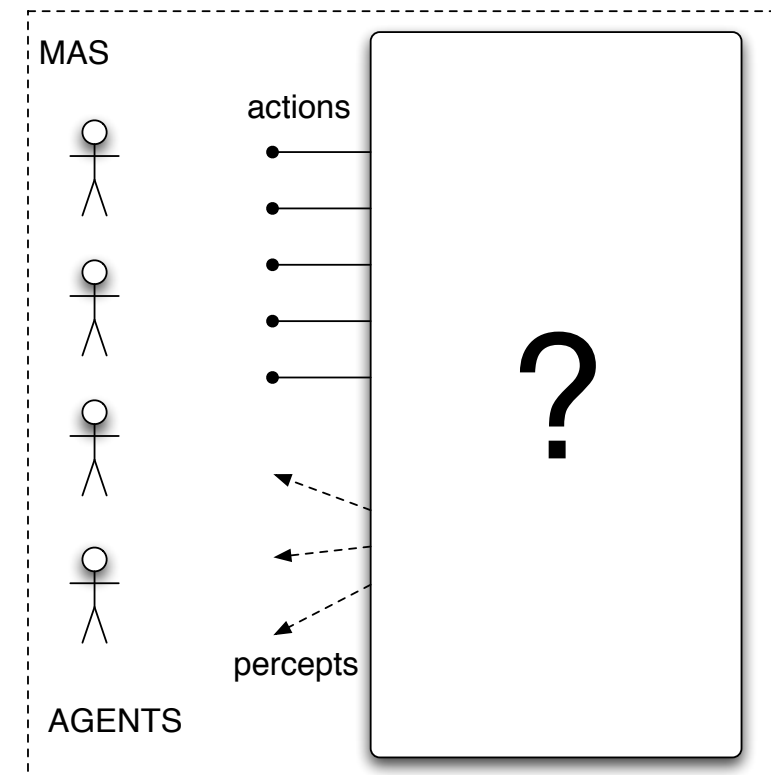
DESIDERATA FOR A WORK. ENV. PROGRAMMING MODEL (1/2)

- **Abstraction**

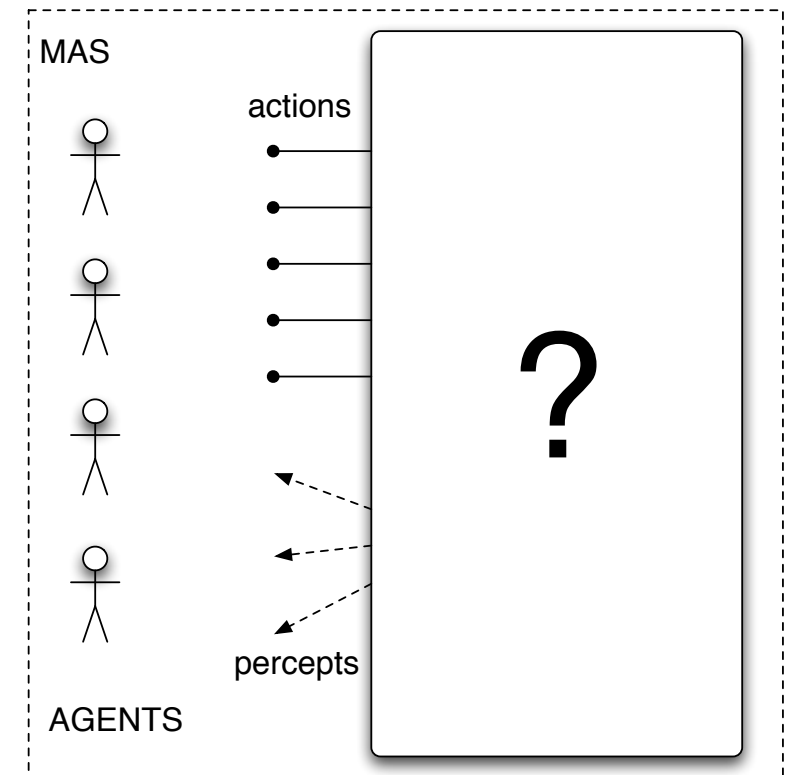
- keeping the agent abstraction level
 - concepts like Action, Perception, Observation
 - i.e., no agents sharing and calling *OO objects*
- effective programming models
 - for controllable and observable computational entities

- **Orthogonality**

- wrt agent models, architectures, platforms
- support for heterogeneous systems



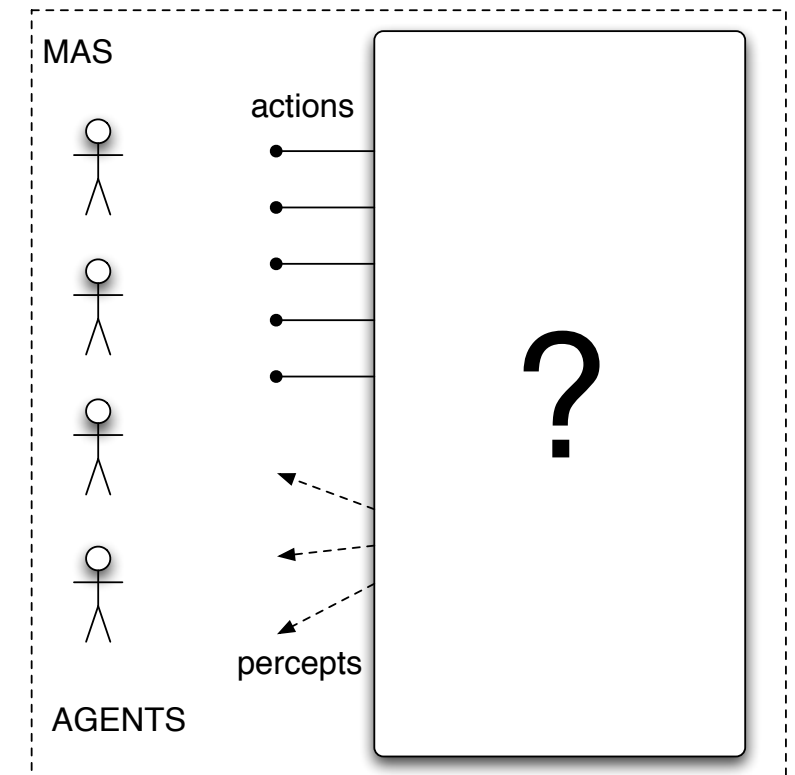
DESIDERATA FOR A WORK ENV. PROGRAMMING MODEL (2/2)



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

- **(Dynamic) extendibility**

- dynamic construction, replacement, extension of environment parts
- support for *open* systems



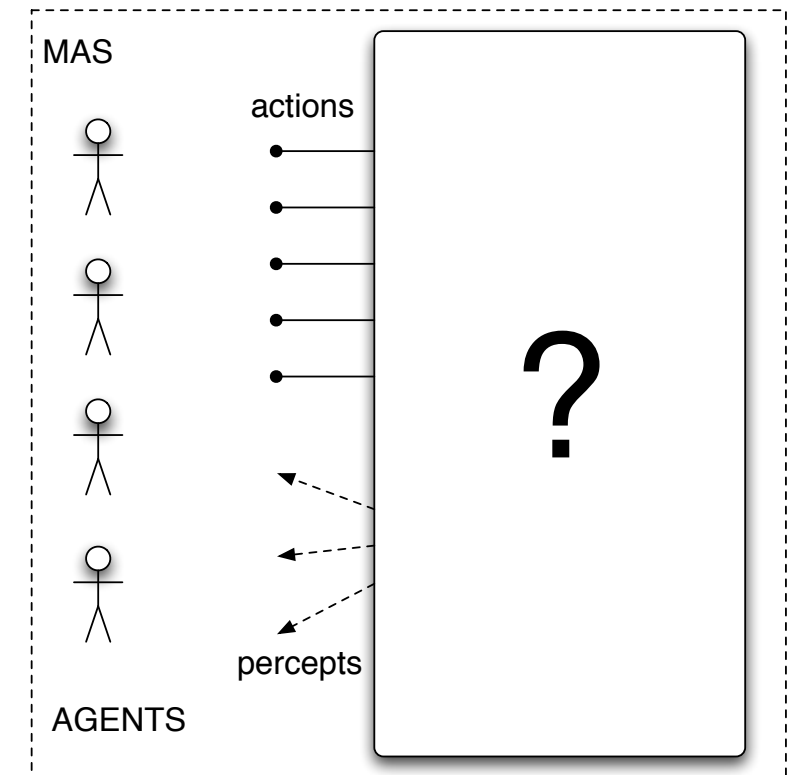
DESIDERATA FOR A WORK ENV. PROGRAMMING MODEL (2/2)

- **(Dynamic) extendibility**

- dynamic construction, replacement, extension of environment parts
- support for *open* systems

- **Modularity**

- away from the monolithic and centralised view



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

- **(Dynamic) extendibility**

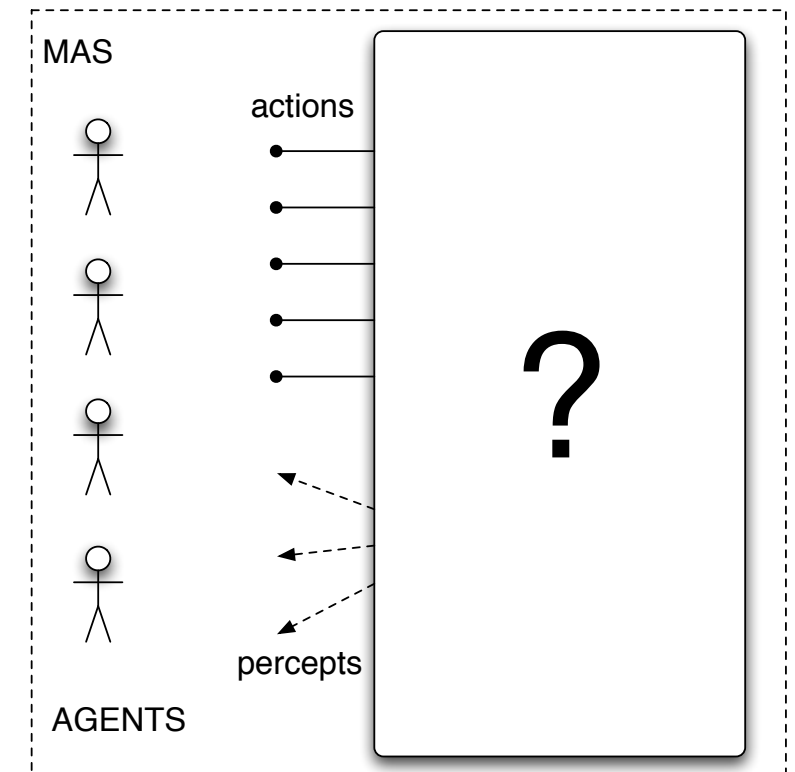
- dynamic construction, replacement, extension of environment parts
- support for *open* systems

- **Modularity**

- away from the monolithic and centralised view

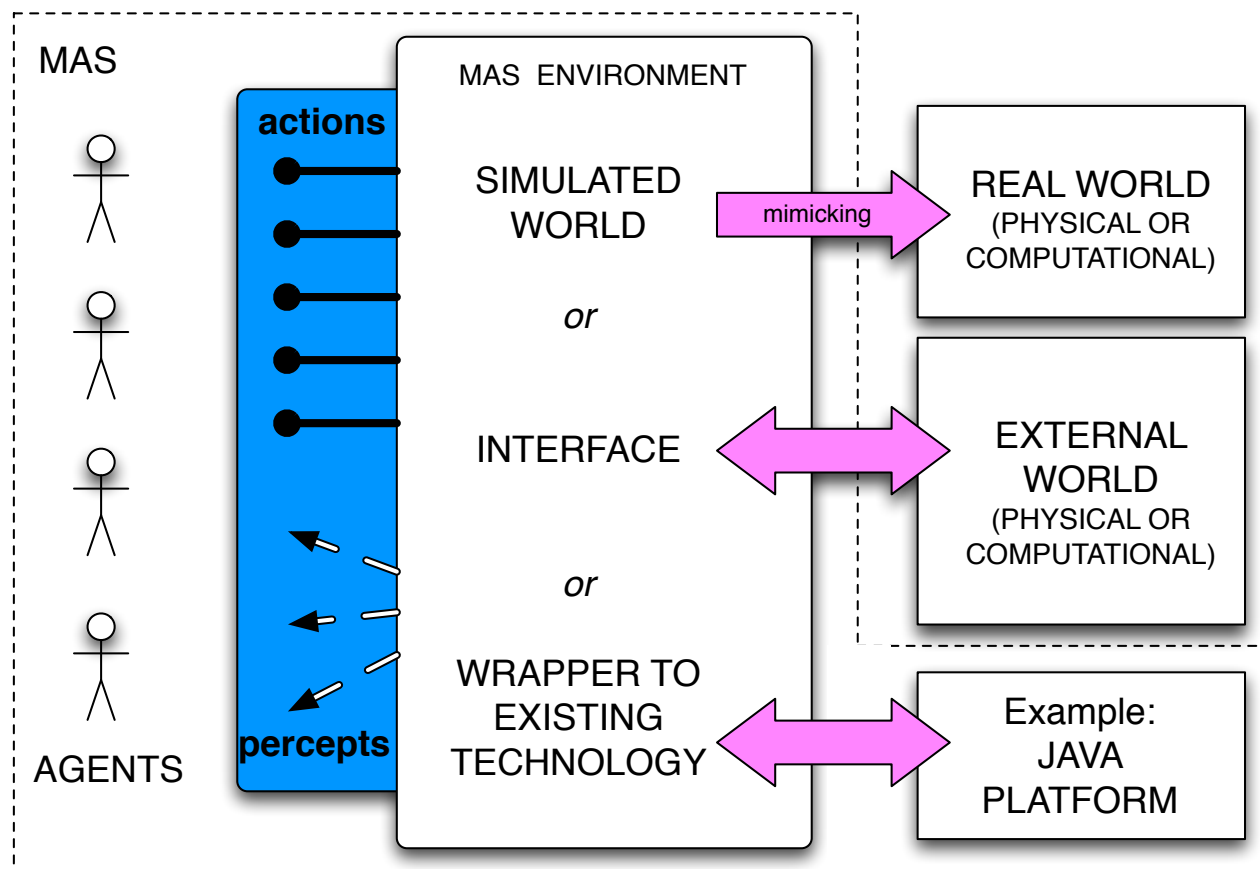
- **Reusability**

- reuse of environment parts in different application contexts / domains



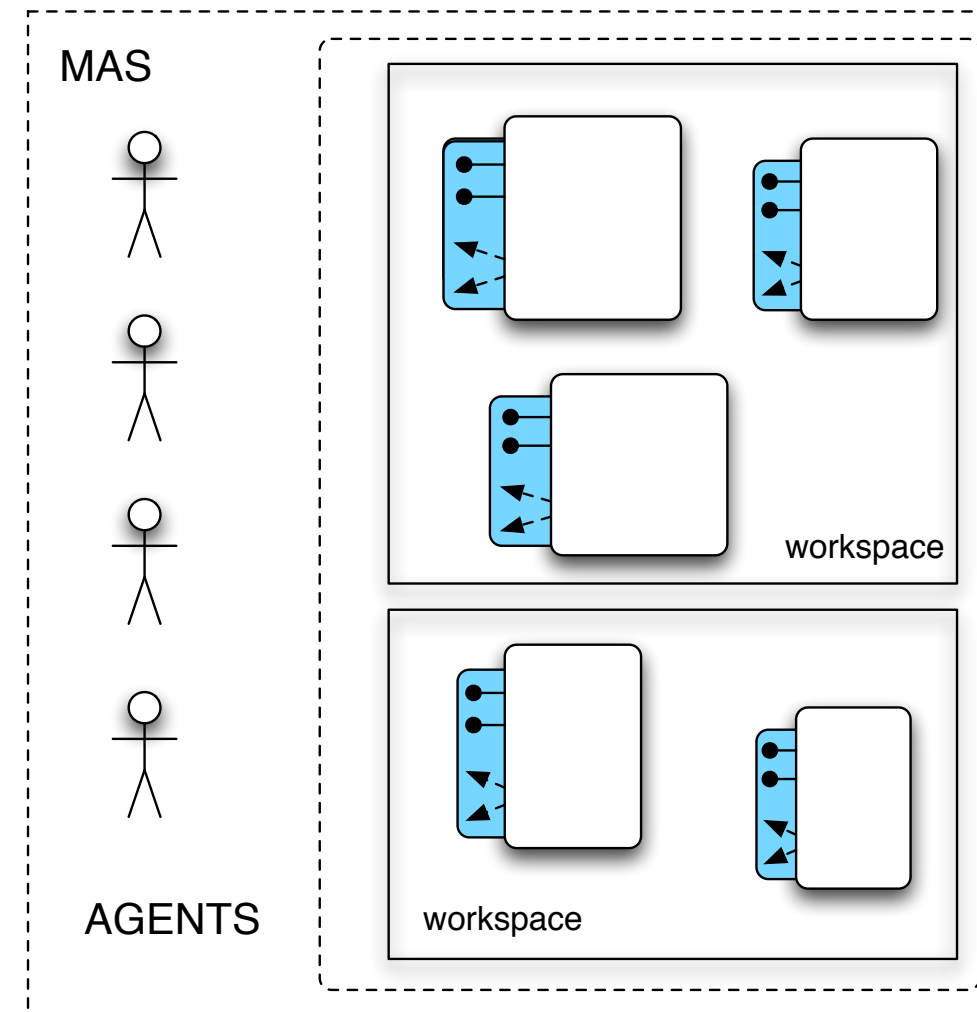
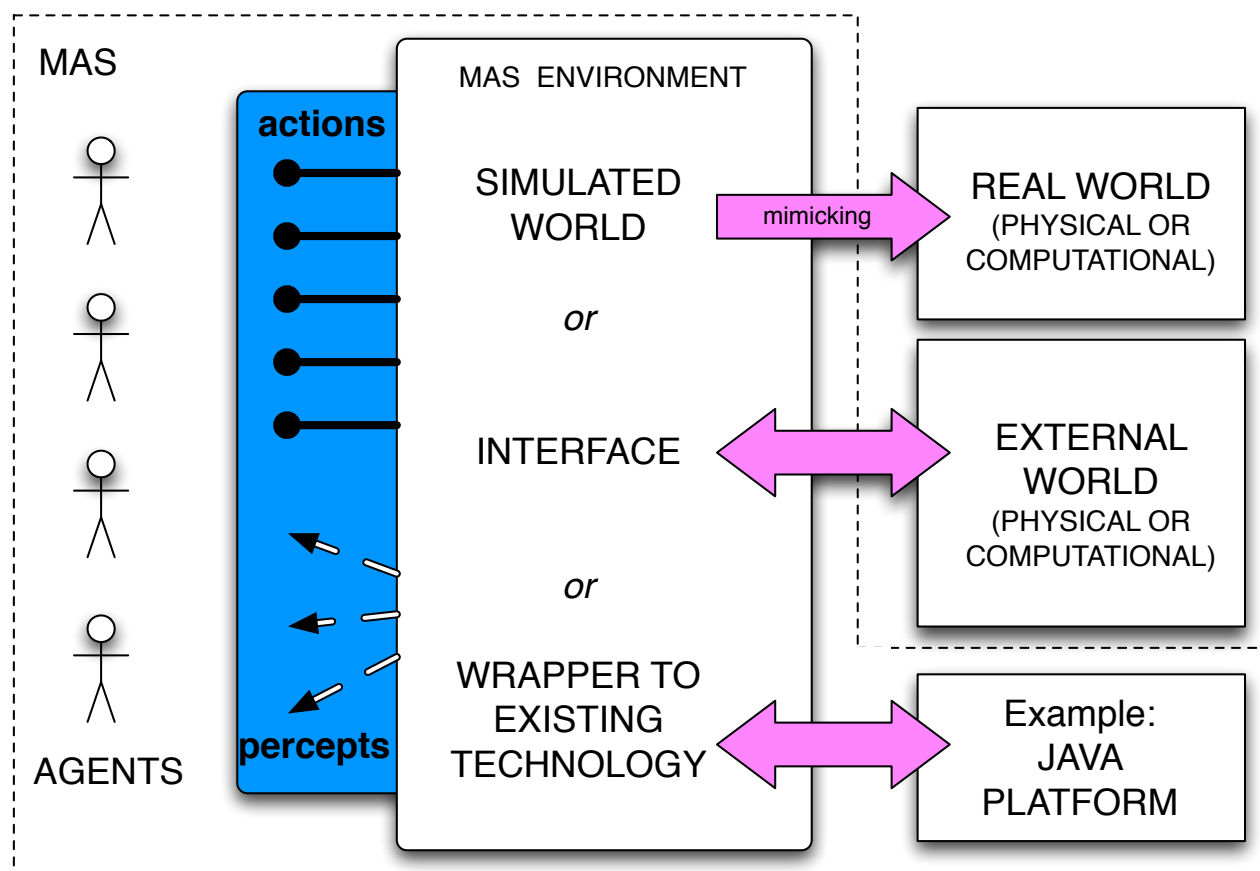
SHAPING WORKING ENVIRONMENTS

- Clear understanding of (cognitive) agent basic activities
 - Action (to change the world) and perception (to get information)
- Define the “rules of encounter”
- *Divide and Conquer*

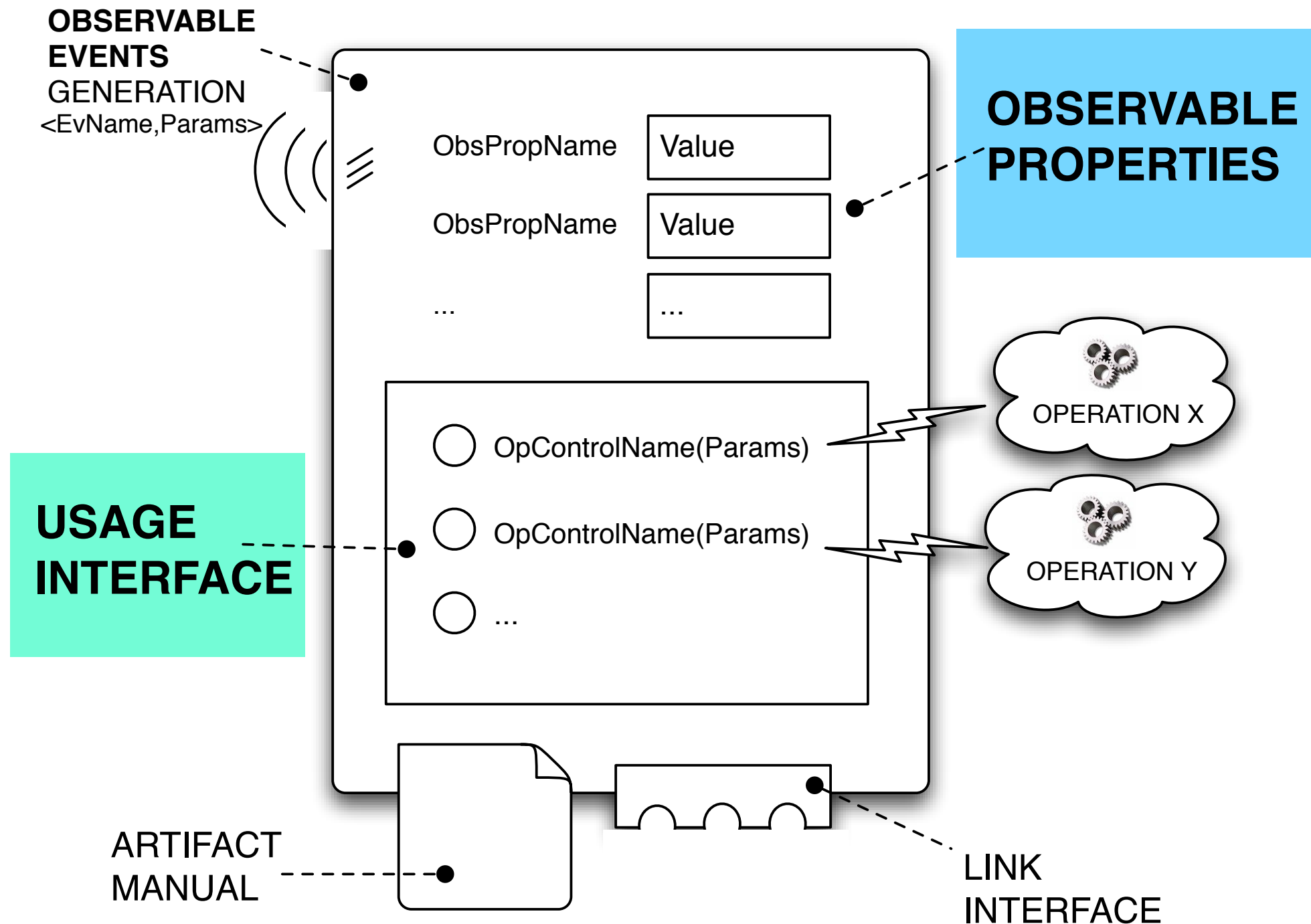


SHAPING WORKING ENVIRONMENTS

- Clear understanding of (cognitive) agent basic activities
 - Action (to change the world) and perception (to get information)
- Define the “rules of encounter”
- *Divide and Conquer*



ARTIFACT COMPUTATIONAL MODEL



ARTIFACTS ARE NOT AGENTS

- As agents, artifacts are entities existing (persisting) in a given work environment
- Differently from agents, artifacts don't encapsulate their control
 - Artifacts have *interfaces* (to be used, read)
 - Not pro-active but *passive* behavior
- Target of agent (goal oriented) activities
- Sources of domain independent percepts

ARTIFACTS ARE NOT “OO”

- Artifacts are still "objects" (in general sense), but shaped in agent oriented fashion:
 - Artifact operations are not object methods in the form of system calls
 - Designed to be **exploited by agents with mental attitudes** (so they have informational and operational functionalities to be mapped in agent mental states)
- Artifacts are located in workspaces where they exist, and persist even beyond agents
 - supported by a run time, i.e. to locate them, resolve conflicts (cuncurrency, consistency, distribution)

CARTAGO

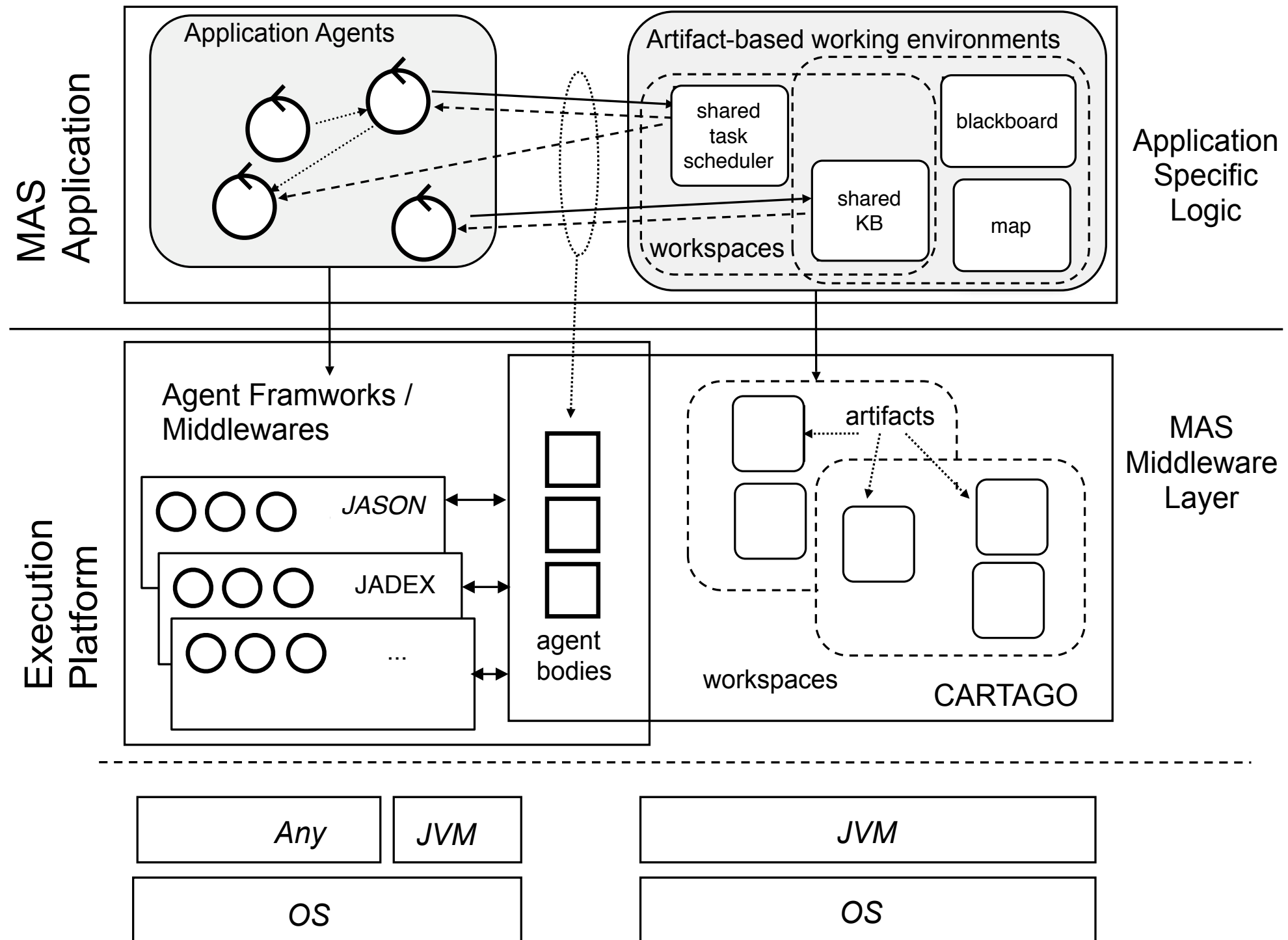
CARTAGO

- **CARTAGO** platform / infrastructure
 - runtime environment for executing (possibly distributed) artifact-based environments
 - Java-based programming model for defining artifacts
 - set of basic AGENT-API to work within artifact-based environment
 - integration with agent programming platforms: Jason, Jadex, SimpA, Jade

CARTAGO

- **CARTAGO** platform / infrastructure
 - runtime environment for executing (possibly distributed) artifact-based environments
 - Java-based programming model for defining artifacts
 - set of basic AGENT-API to work within artifact-based environment
 - integration with agent programming platforms: Jason, Jadex, SimpA, 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

CARTAGO ARCHITECTURE



SHAPING AGENTS AND ARTIFACTS INTERACTIONS

A&A INTERACTIONS

A&A INTERACTIONS

Ease agents (and developers) to deal with mechanisms and low level interaction constructs

To tackle problems at the right DOMAIN LEVEL...
...and maintain the same level of ABSTRACTION in conceiving interaction from A to A

A&A INTERACTIONS

Ease agents (and developers) to deal with mechanisms and low level interaction constructs

To tackle problems at the right DOMAIN LEVEL...
...and maintain the same level of ABSTRACTION in conceiving interaction from A to A

- No more Agents operating upon “component based” middleware, nor Java Objects....

A&A INTERACTIONS

Ease agents (and developers) to deal with mechanisms and low level interaction constructs

To tackle problems at the right DOMAIN LEVEL...
...and maintain the same level of ABSTRACTION in conceiving interaction from A to A

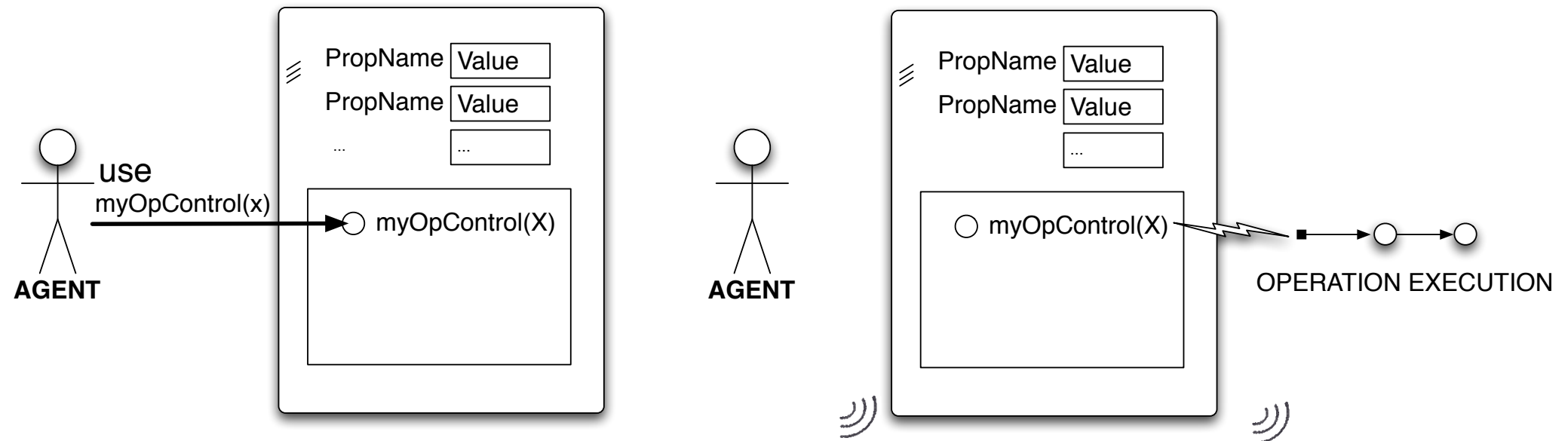
- No more Agents operating upon “component based” middleware, nor Java Objects....
- Shaping interaction based on agents basic activities:
Action and Perception

INTEGRATING HETEROGENEOUS AGENTS PLATFORMS

Extending agents repertoire with a basic set of actions enables playing in artifact-based work environment

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

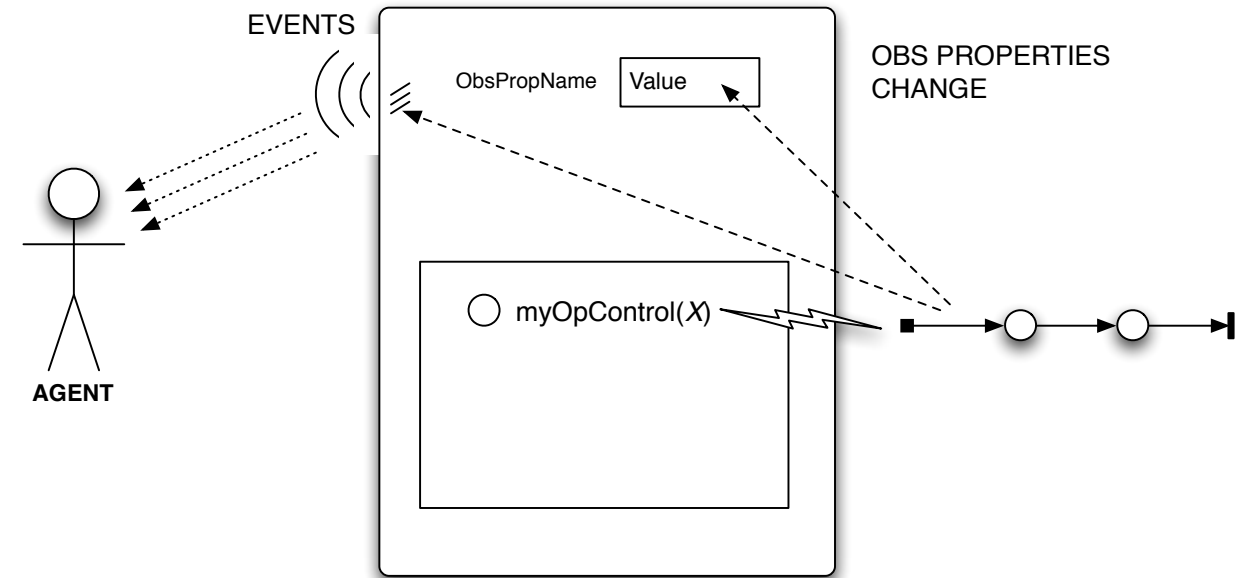
USE (OPERATIONS)



- Use Action acts upon an operation control of the usage interface to trigger the execution of an operation
 - `use(Aid, OpCntrName(Params), +Sensor, +Timeout, +Filter)`
- Operations are executed as atomic steps
 - Cuncurrency and Consistency

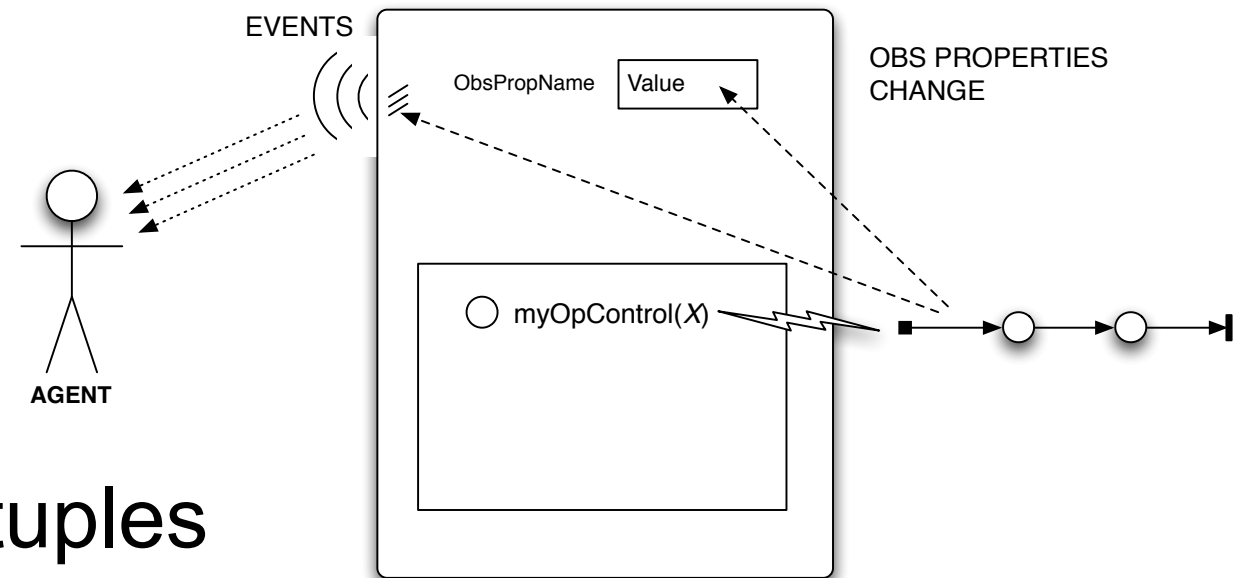
OBSERVABLE EVENTS

Generated, within artifact operations, by the **signal** primitive



OBSERVABLE EVENTS

Generated, within artifact operations, by the **signal** primitive

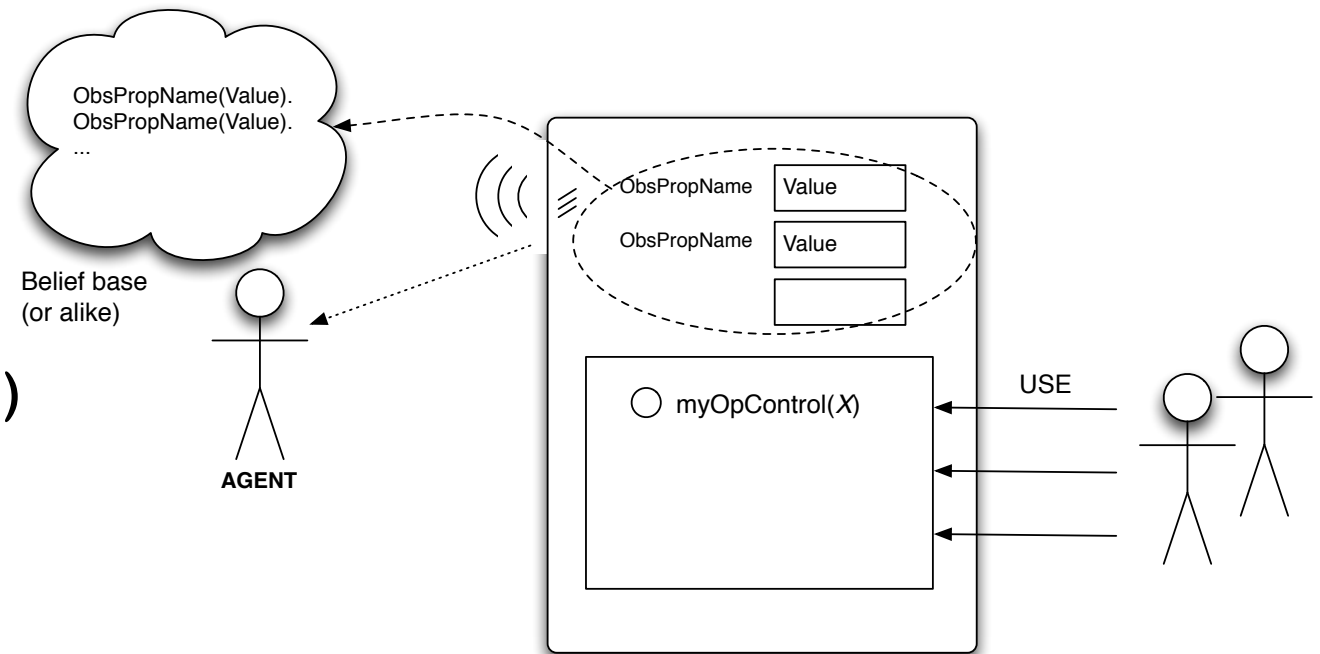


- Represented as labelled tuples
 - *event_name(Arg0, Arg1, ...)*
- Propagate *non persistent* signals which are automatically observable to:
 - the agent who executed the operation sense
 - **sense**(Sensor, ?Perception, +Filter, +Timeout)
 - all the agents focusing the artifact
 - **focus**(Aid, +Sensor, +Filter)

OBSERVABLE PROPERTIES

Declared by :

defineObsProperty(name, value)

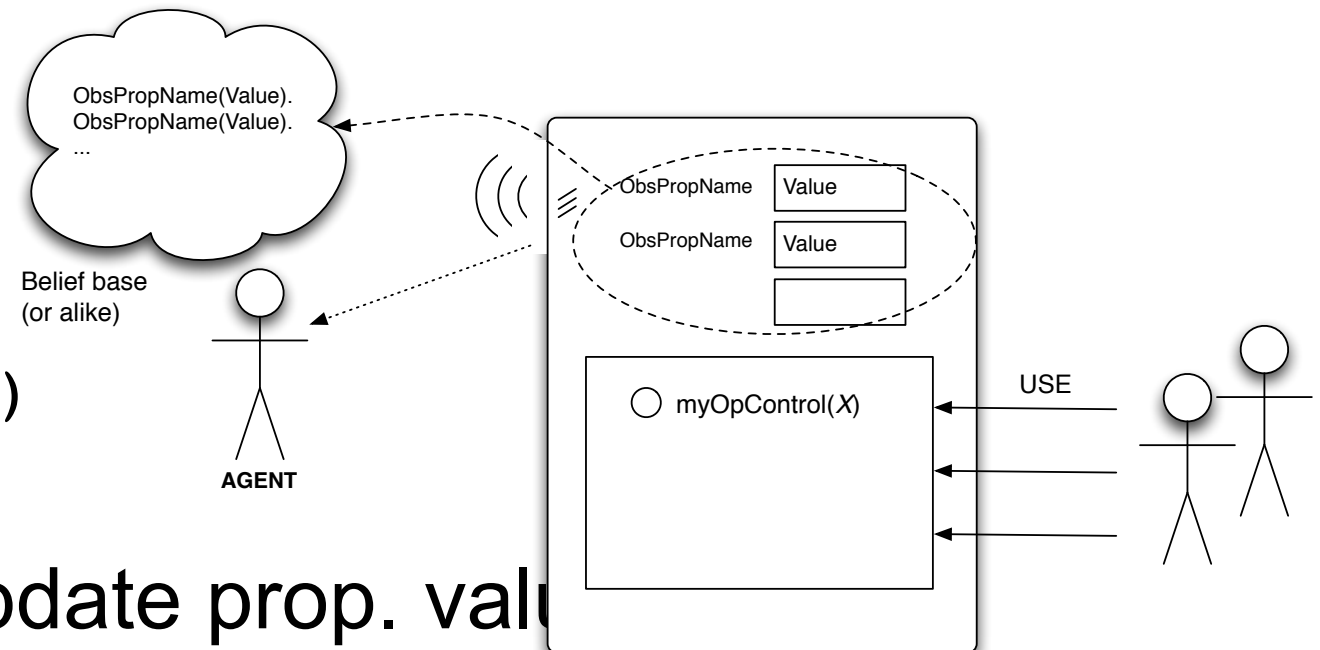


OBSERVABLE PROPERTIES

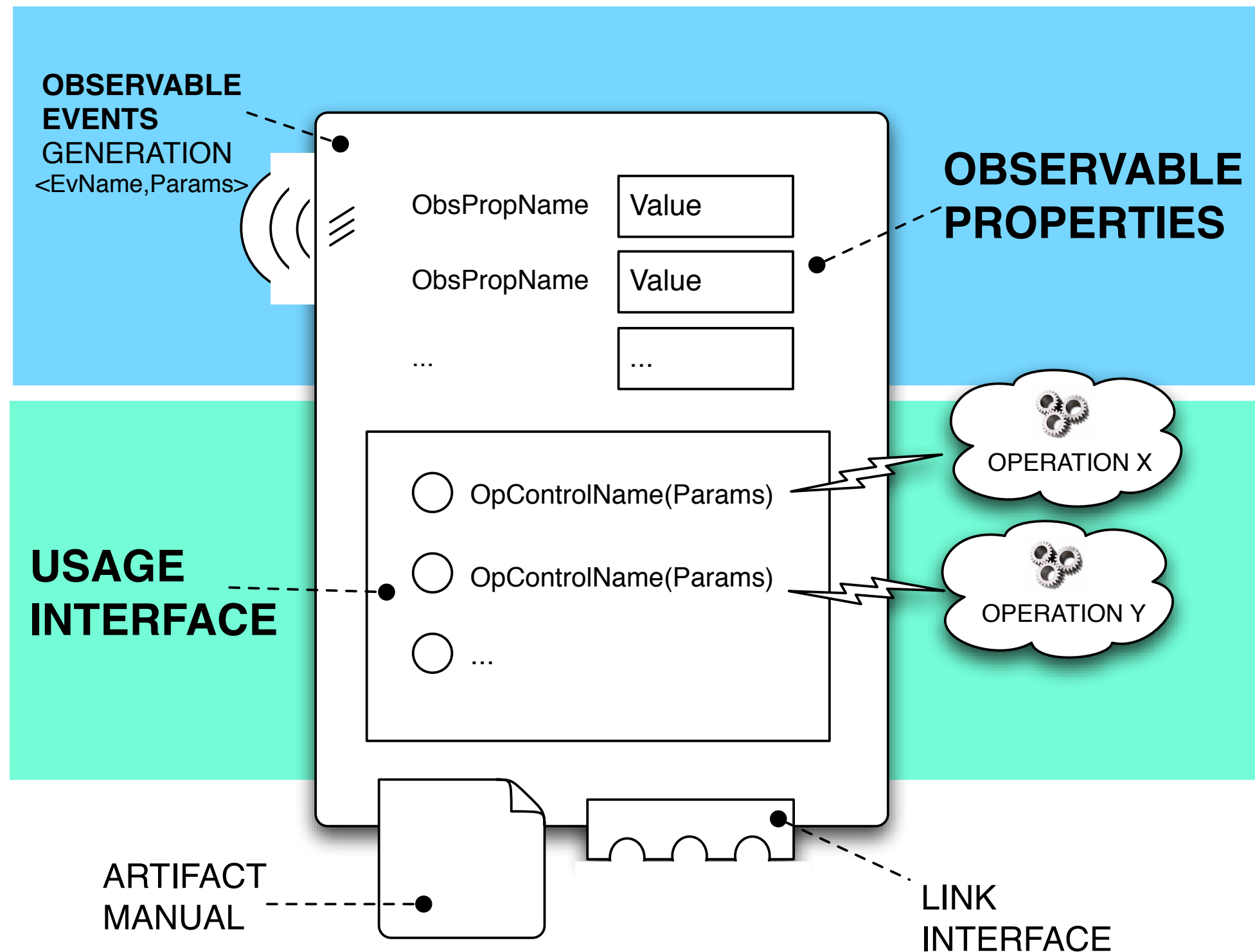
Declared by :

defineObsProperty(name, value)

- Artifact primitives to read/update prop. value
 - **updateObsProperty**
 - **getObsProperty**
- *Persistent observable state*
- Automatically observable by all the agents focusing (or observing) the artifact
 - **observeProperty**(Aid, PName, ?PValue)
 - **focus**(Aid, +Sensor, +Filter)



ARTIFACT COMPUTATIONAL MODEL



FUNCTIONAL APPROACH

From an agent view-point, artifacts embeds:

1. Operational (purposive) function

- Goals can be achieved by the mean of operations which have been defined within control interface.

2. Doxastic (epistemic) function

- Artifact representational contents can be used as an external informational structure [Normann, Kirsh]

FUNCTIONAL APPROACH

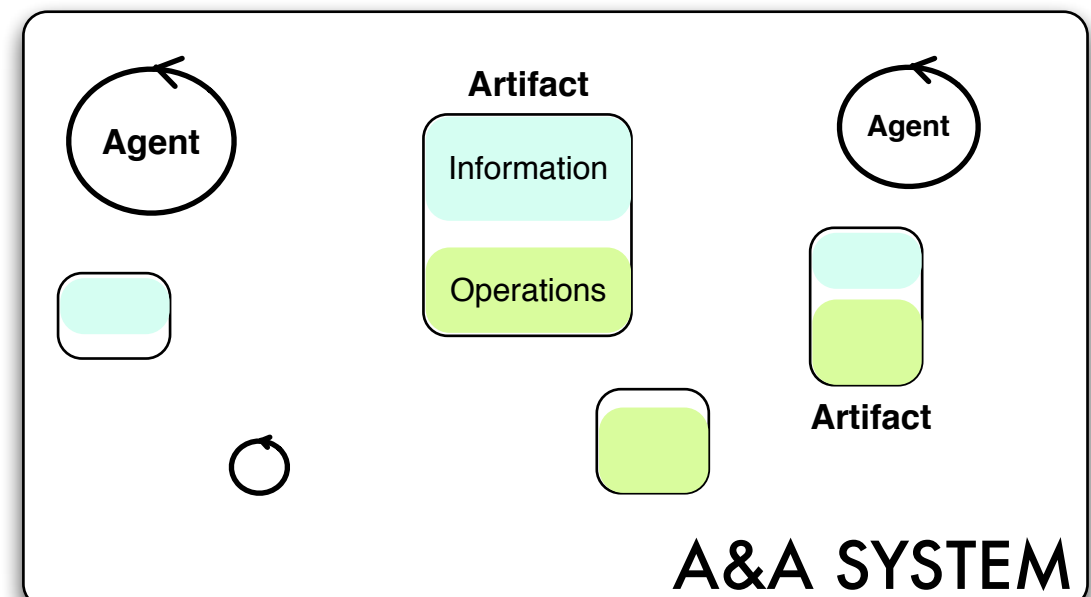
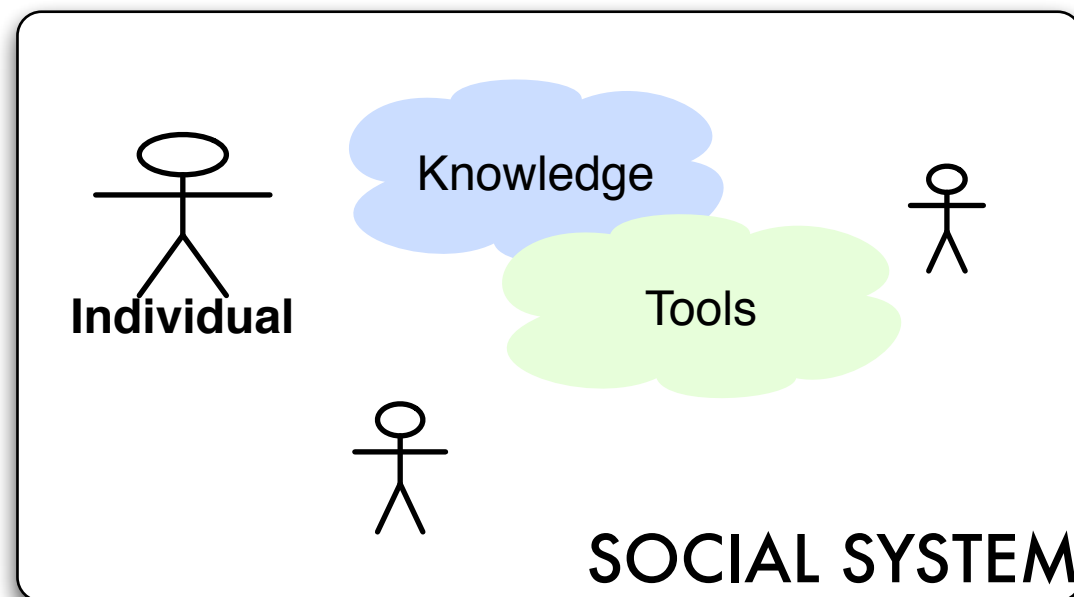
From an agent view-point, artifacts embeds:

1. Operational (purposive) function

- Goals can be achieved by the mean of operations which have been defined within control interface.

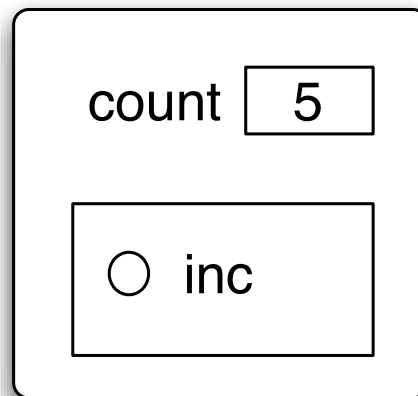
2. Doxastic (epistemic) function

- Artifact representational contents can be used as an external informational structure [Normann, Kirsh]



EXAMPLES

COUNTER



OBSERVABLE PROPERTIES:
count: int

USAGE INTERFACE:
inc: [obs_prop_updated,
op_exec_completed]

```
import alice.cartago.*;

public class Counter extends Artifact {

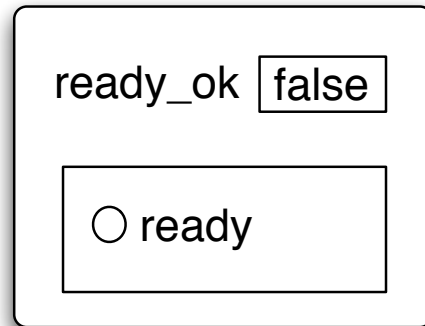
    @OPERATION void init(){
        defineObsProperty("count",0);
    }

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

```
!create_and_use.
+!create_and_use : true
<-
cartago.makeArtifact("mycount","Counter",C);
cartago.use(C,inc);
cartago.use(C, inc, s0);
cartago.sense(s0, op_exec_completed(_), 1000);
!perceive.

+!perceive : true
<-
cartago.observeProperty(C,count(V));
cartago.use(console, println("Actual Count value:",V)).
```

SYNCHRONISER



OBSERVABLE PROPERTIES:
ready_ok: {true,false}

USAGE INTERFACE:
ready: [op_exec_completed,
 prop_updated]
reset: [op_exec_completed,
 prop_updated]

```
import alice.cartago.*;

public class Synchroniser extends Artifact {
    int nReady, nParticipants;

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

    @OPERATION void ready(){
        nReady++;
        nextStep("setReadyOK");
    }

    @OPSTEP(guard="allReady") void setReadyOK(Item obj){
        updateObsProperty("ready_ok",true);
    }

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

    @OPERATION void reset(){
        nReady = 0;
        updateObsProperty("ready_ok",false);
    }
}
```

```
!work.

+!work : true
  <-
  //synchronize:
  cartago.focus(mySynch);
  cartago.use(mySynch,ready).

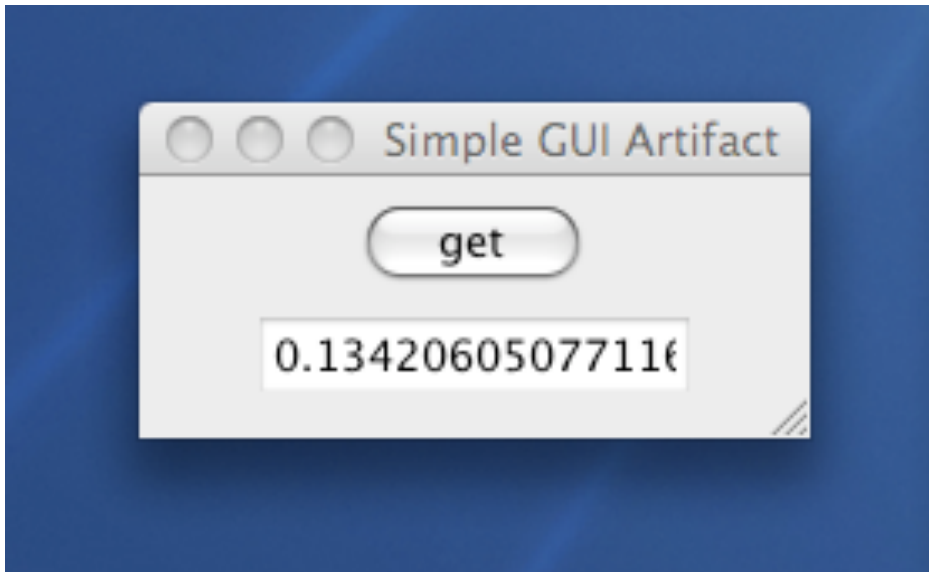
+ready_ok(true) : true
  <-
  //synchronized:go on working
```

```
!work.

+!work : true
  <-
  //synchronize:
  cartago.use(mySynch,ready,s1);
  cartago.sense(s1,
               op_exec_completed(ready));
  //synchronized:go on working
```

GUI ARTIFACT

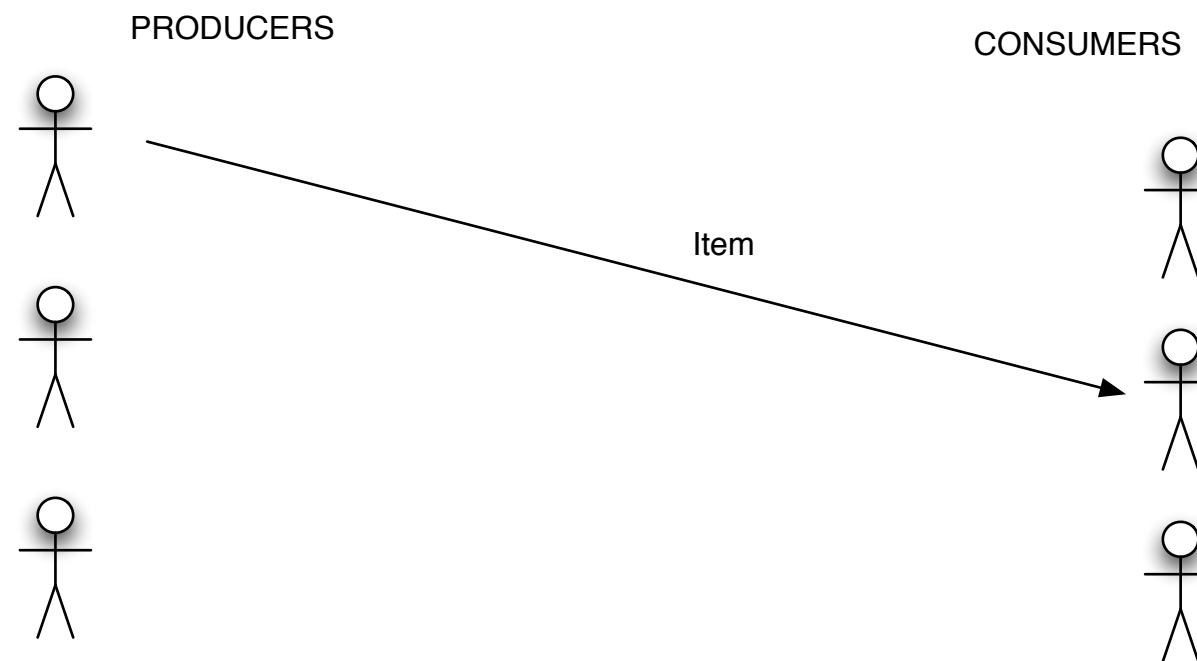
```
!manage_gui.  
  
+!manage_gui : true <-  
  cartago.makeArtifact(myGUI,  
                        "MyGUIArtifact",GUI);  
  cartago.focus(GUI).  
  
+get_selected(Ev)[source(myGUI)]: true <-  
  .random(R);  
  cartago.use(myGUI,setValue(R)).  
  
+closed(Ev)[source(myGUI)] : true <-  
  .my_name(Me);  
  .kill_agent(Me).
```



```
import javax.swing.*;  
import alice.cartago.*;  
  
public class MyGUIArtifact extends GUIArtifact {  
  private MyFrame frame;  
  
  @OPERATION void init() throws CartagoException {  
    frame = new MyFrame();  
    frame.setVisible(true);  
    registerSource(frame.getButton,"actionPerformed","get_selected");  
    registerSource(frame, "windowClosing", "closed");  
  }  
  
  @OPERATION void setValue(double value){  
    frame.text.setText(""+value);  
  }  
  
  class MyFrame extends JFrame {  
    public JTextField text;  
  
    public MyFrame(){  
      setTitle("Simple GUI Artifact");  
      setSize(200,100);  
      JPanel panel = new JPanel();  
      setContentPane(panel);  
      JButton button = new JButton("get");  
      button.setSize(80,50);  
      panel.add(button);  
      text = new JTextField(10);  
      text.setEditable(false);  
      panel.add(text);  
    }  
  }  
}
```

PRODUCERS-CONSUMERS

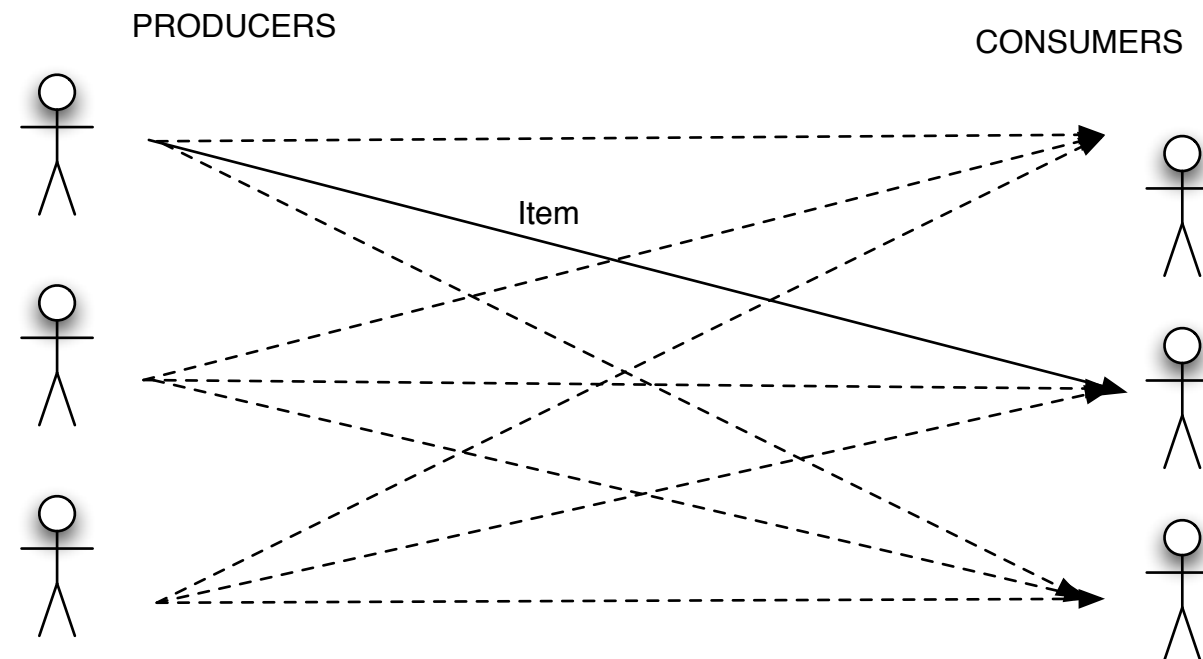
- Example: a variable number of Producer and Consumer Agents want to exchange generic Informational Items



- Bounded resources: Time, Space, Memory..
 - Need for a strategy to synchronise/coordinate exchanges
 - Well known problem in concurrent systems

PRODUCERS CONSUMERS IN MAS

- Direct Interaction: overload of messages and broadcasts (who may I ask for a relevant Item ?)

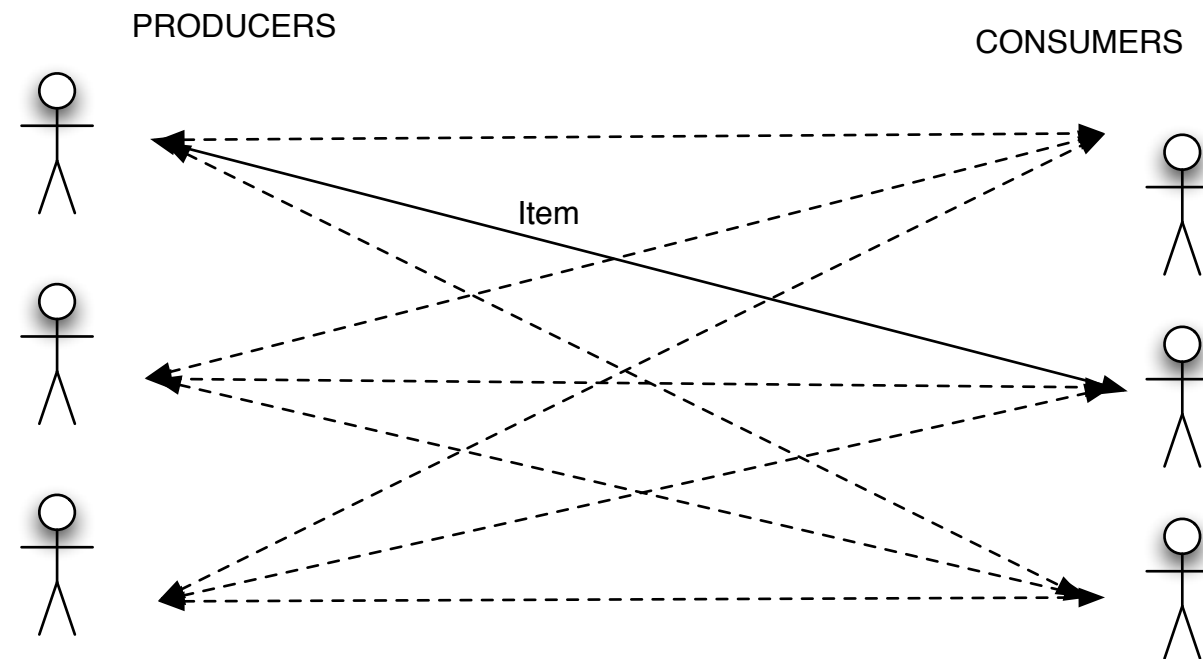


PRODUCERS CONSUMERS IN MAS

- Direct Interaction: overload of messages and broadcasts (who may I ask for a relevant Item ?)

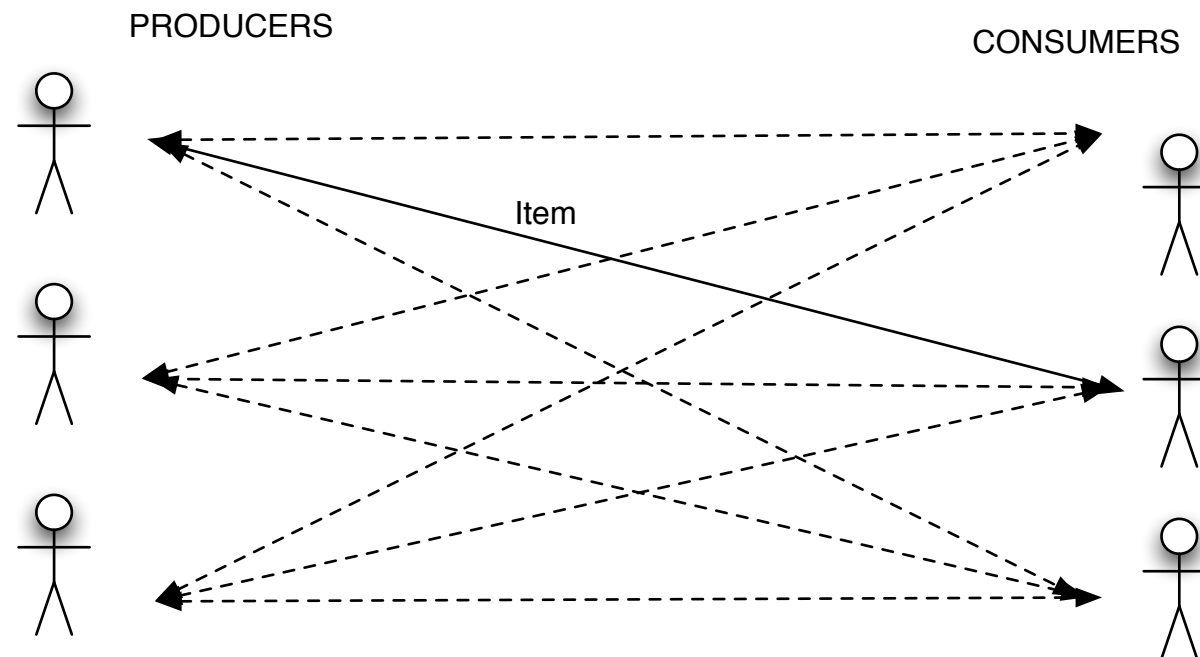
PRODUCERS CONSUMERS IN MAS

- Direct Interaction: overload of messages and broadcasts (who may I ask for a relevant Item ?)



PRODUCERS CONSUMERS IN MAS

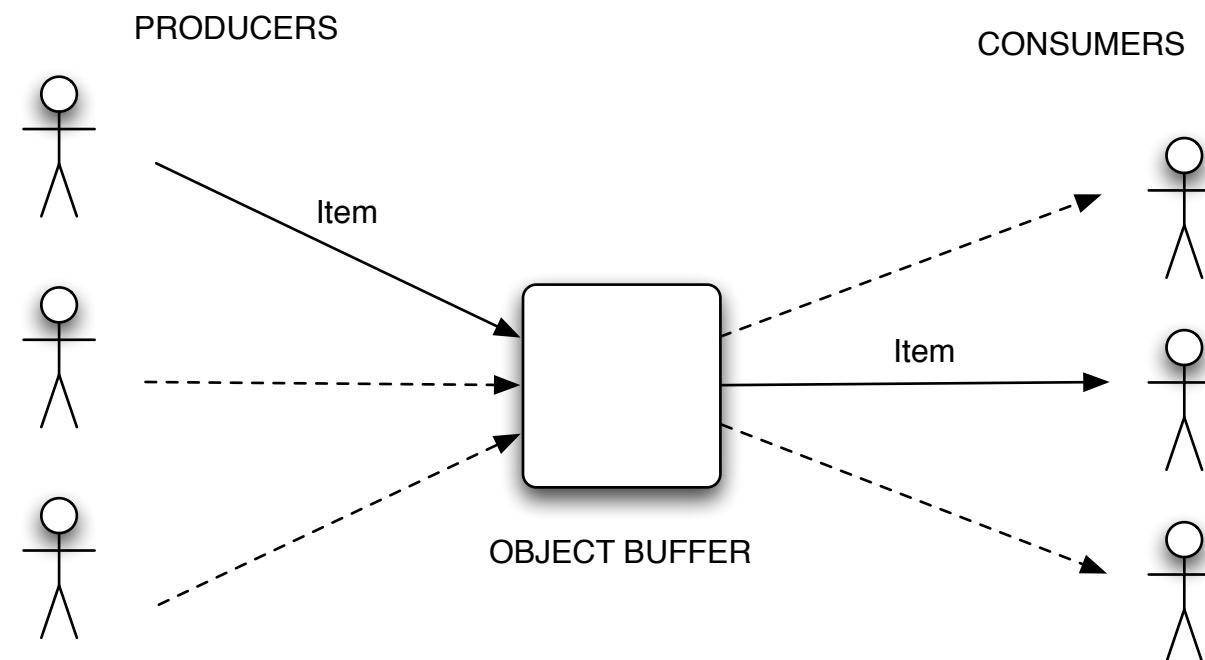
- Direct Interaction: overload of messages and broadcasts (who may I ask for a relevant Item ?)



- Which need to be typed (protocols)
- Bidirectional (Request-Response, Solicit Response...)

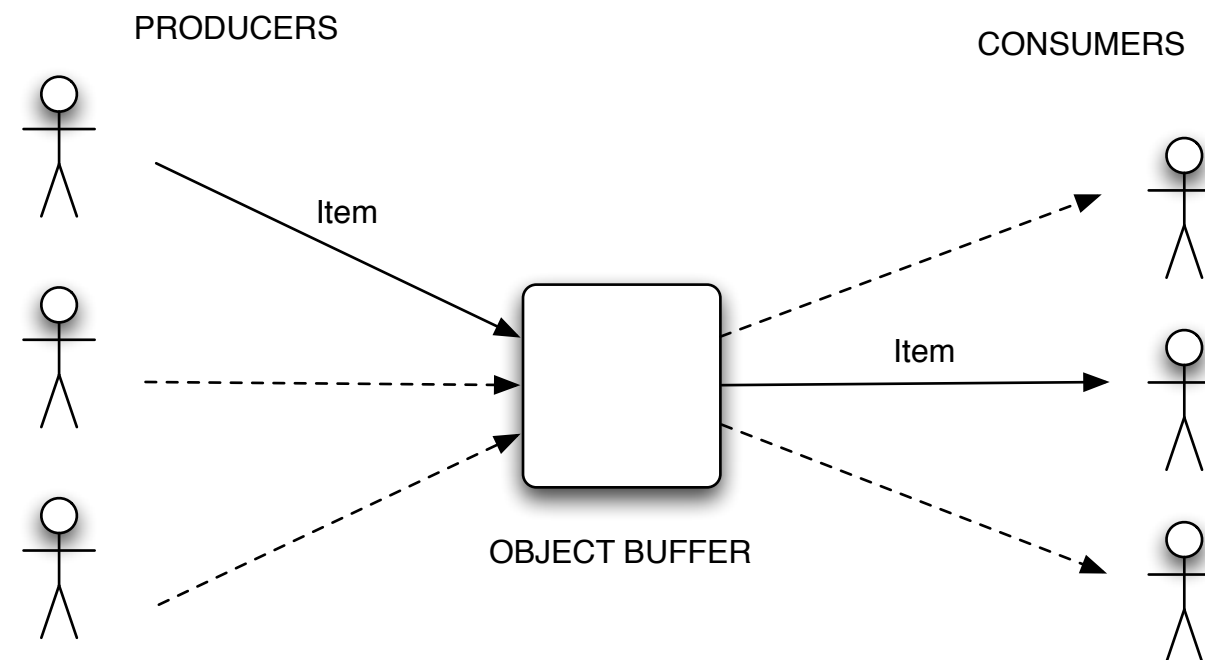
PRODUCERS CONSUMERS IN MAS

- Using an Object as a shared buffer reduces the amount of interactions



PRODUCERS CONSUMERS IN MAS

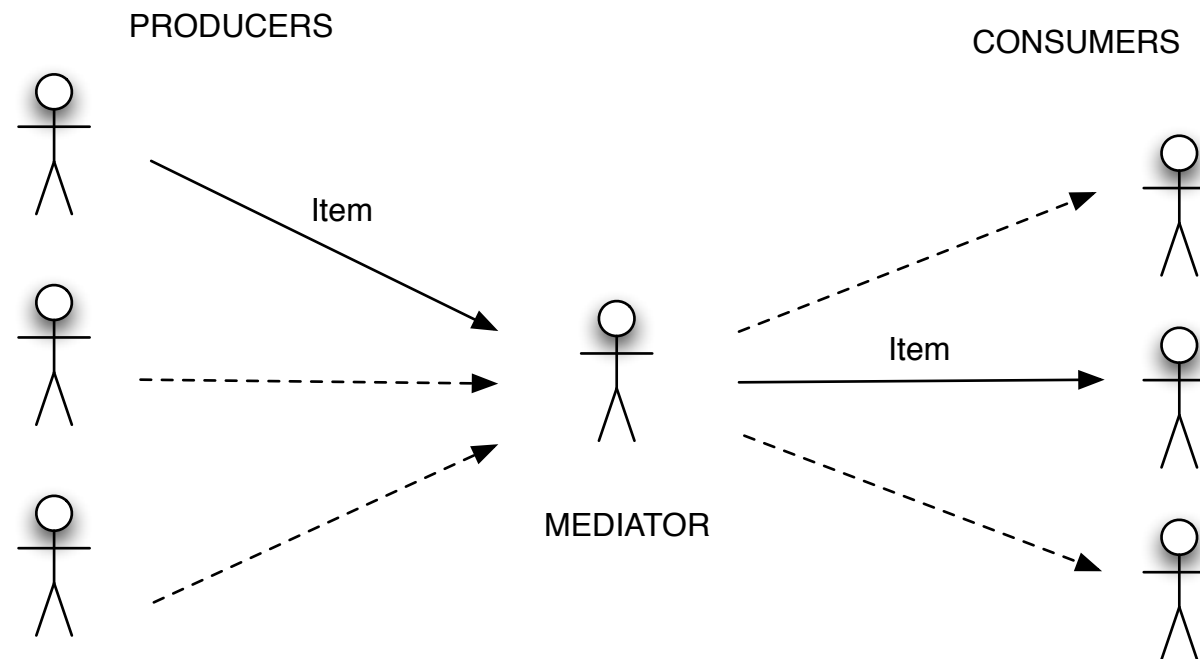
- Using an Object as a shared buffer reduces the amount of interactions



- Which kind of interaction? From Agents to (Java) Objects? RPC?
- Not the same level of abstraction for domain entities of the system

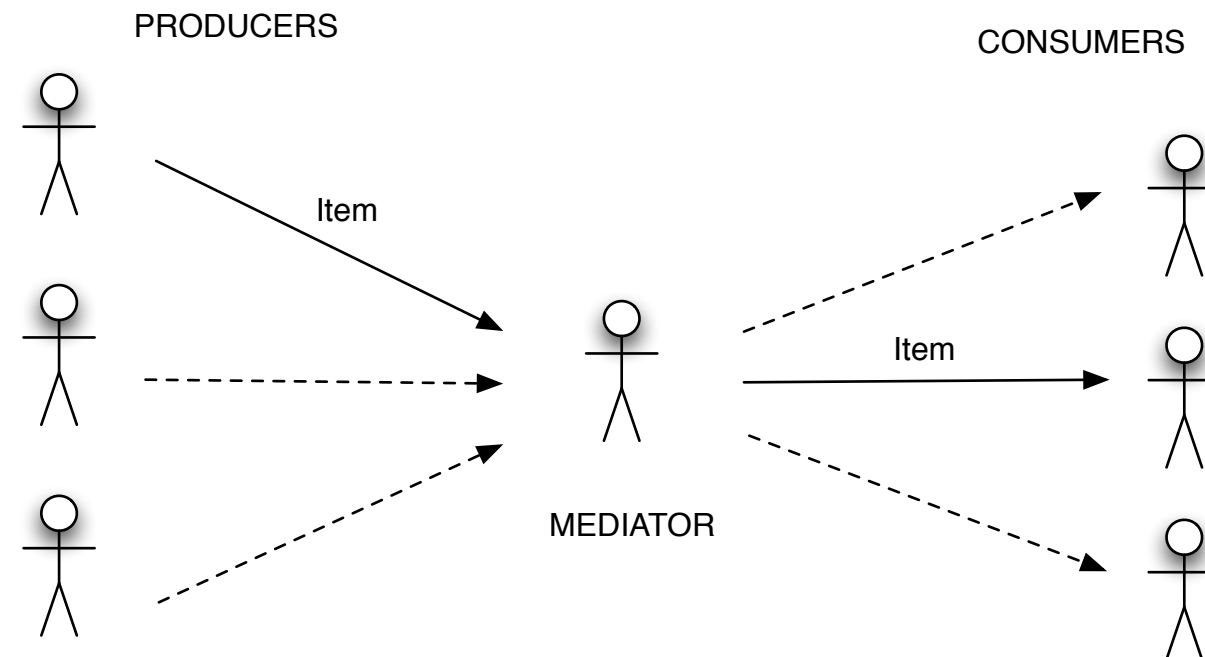
PRODUCERS CONSUMERS IN MAS

- Use a Mediator Agent maintains the same level of abstraction between entities



PRODUCERS CONSUMERS IN MAS

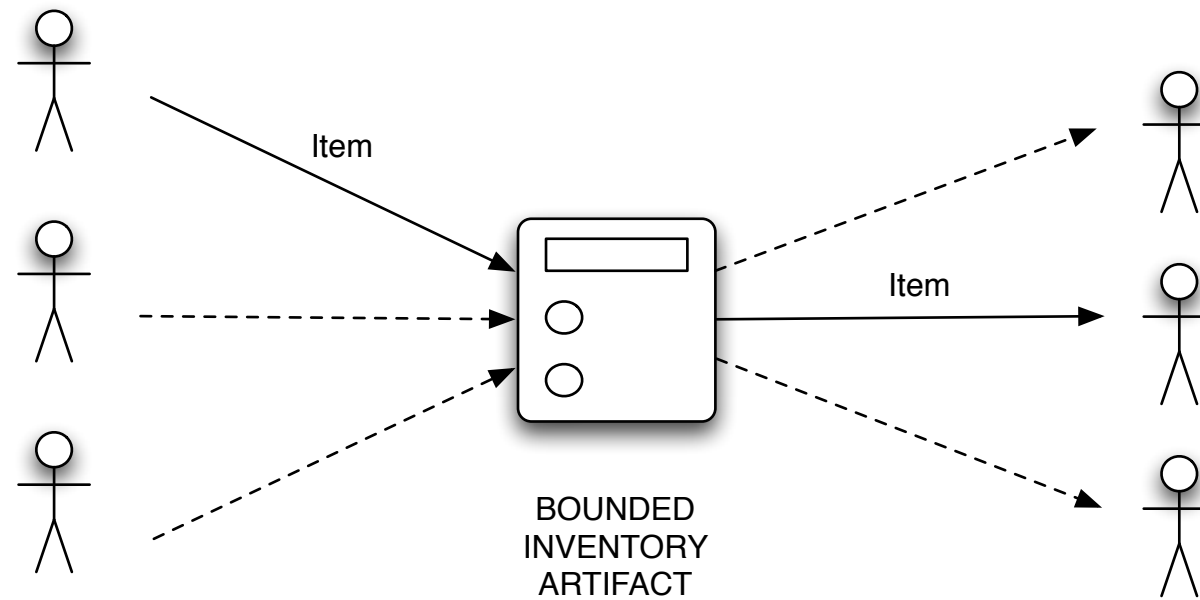
- Use a Mediator Agent maintains the same level of abstraction between entities



- But which kind of Agents? Reactive? Autonomous?
 - *(can say no!!?)*
- Still need protocols and messages to exchange:
 - *opennes?*

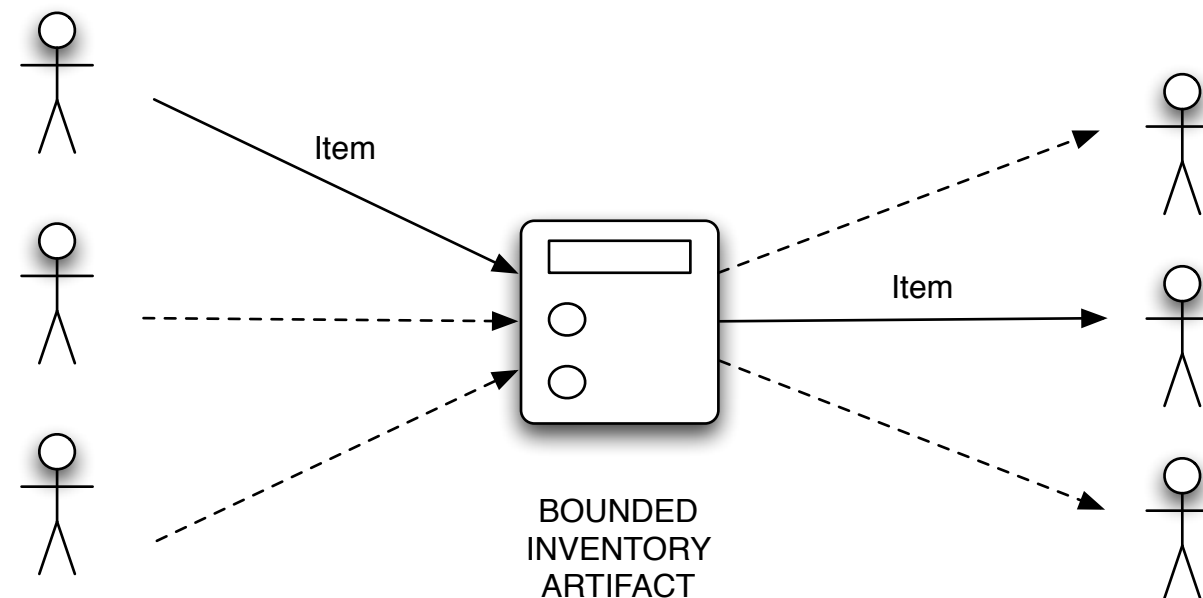
PRODUCERS CONSUMERS IN MAS

Use an Artifact !



PRODUCERS CONSUMERS IN MAS

Use an Artifact !



- A Bounded Inventory Artifact is designed to function as a shared Inventory mediating the activities in an automatic fashion
 - *Ruling and Enabling* the exchange
- Inventory can be modified to change performance of the system (i.e. capacity, size, data preprocessing)

BOUNDED INVENTORY

n_items	0
max_items	100
☐ put ☐ get	

OBSERVABLE PROPERTIES:

n_items: int+

max_items: int

Invariants:

$n_items \leq max_items$

USAGE INTERFACE:

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

get / ($n_items \geq 0$) :
[obs_prop_updated, *new_item*(item:Item),
op_exec_completed]

```
import alice.cartago.*;
import java.util.*;

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

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

    @OPERATION(guard="inventoryNotFull") void put(Item obj){
        items.add(obj);
        updateObsProperty("n_items",items.size()+1);
    }

    @OPERATION(guard="itemAvailable") void get(){
        Item item = items.removeFirst();
        updateObsProperty("n_items",items.size()-1);
        signal("new_item",item);
    }

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

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

Operational Function

```
+!produceItems : nextItemToProduce(Item)
  <- cartago.lookupArtifact("my-inv", InvID);
  cartago.use(InvID, put(Item), 500).

/* Fail Event */
-!produceItems: true
  <- cartago.use(console,
    println("Insertion failed due to timeout.")).
```

```
+!consume
  : myInventory(InvID) & mySensor(S)
  <- cartago.use(InvID, get, S, 1000);
  cartago.sense(S, new_item(Item));
  !consumeItem(Item);
  !consume.

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

Operational Function

```
+!produceItems : nextItemToProduce(Item)
  <- cartago.lookupArtifact("my-inv", InvID);
  cartago.use(InvID, put(Item), 500).

/* Fail Event */
-!produceItems: true
  <- cartago.use(console,
    println("Insertion failed due to timeout.")).
```

```
+!consume
  : myInventory(InvID) & mySensor(S)
  <- cartago.use(InvID, get, S, 1000);
  cartago.sense(S, new_item(Item));
  !consumeItem(Item);
  !consume.

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

- Both Producer and Consumer Goals are in Artifact outcomes

Operational Function

```
+!produceItems : nextItemToProduce(Item)
  <- cartago.lookupArtifact("my-inv", InvID);
  cartago.use(InvID, put(Item), 500).

/* Fail Event */
-!produceItems: true
  <- cartago.use(console,
    println("Insertion failed due to timeout.")).
```

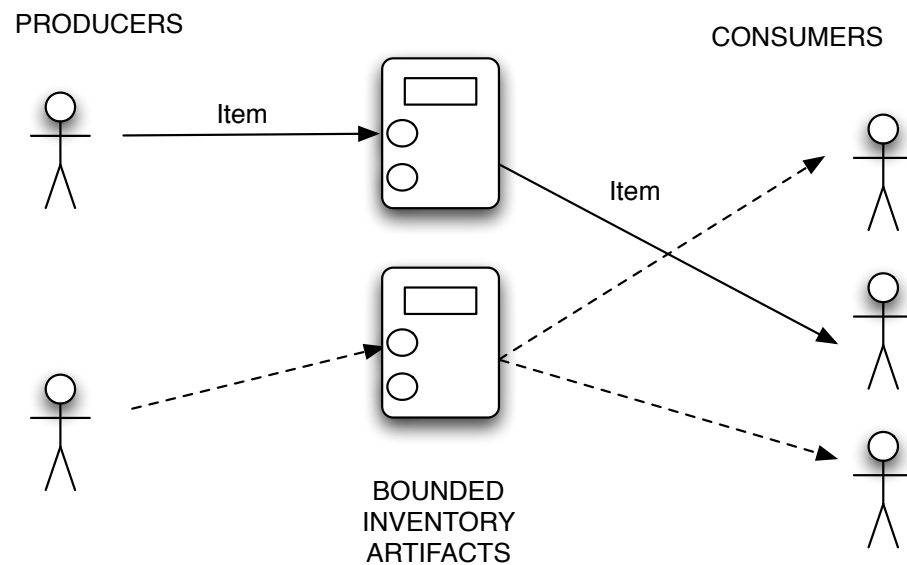
```
+!consume
  : myInventory(InvID) & mySensor(S)
  <- cartago.use(InvID, get, S, 1000);
  cartago.sense(S, new_item(Item));
  !consumeItem(Item);
  !consume.

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

- Both Producer and Consumer Goals are in Artifact outcomes
- All the computation required to synchronize interactions is externalised in the Bounded Inventory
 - It can possibly pre-process the items, e.g. translate them to different ontologies

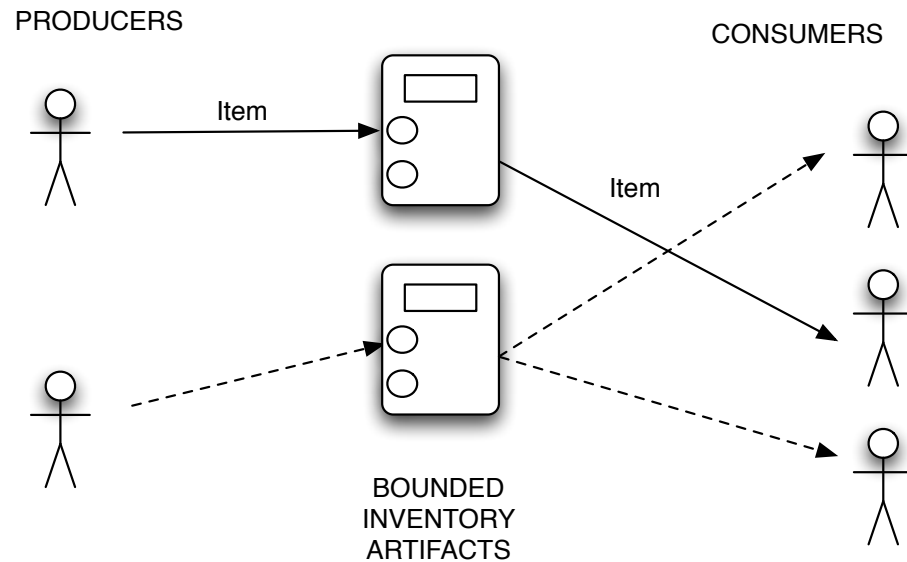
Doxastic Function

- 2 bounded buffers are in place, each with a different item 'load'



Doxastic Function

- 2 bounded buffers are in place, each with a different item 'load'



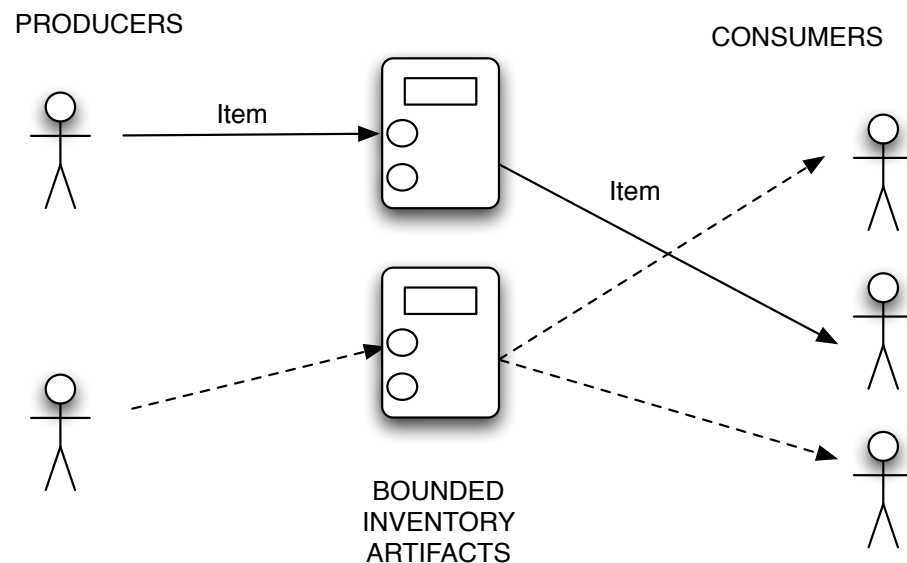
```
+!consumeActivity : true
  <- +min_items(-1);
  cartago.lookupArtifact("inventory-1", InvID1);
  cartago.focus(InvID1);
  cartago.lookupArtifact("inventory-2", InvID2);
  cartago.focus(InvID2);
  +selectedInv(InvID1,0);
  !consumeAction.

/* Updates Goal Supporting Beliefs */
+n_items(N) [source(percept), artifact(InventoryID)]
: selectedInv(_,N1) & N > N1
  <- -+selectedInv(InventoryID,N).

+!consumeAction : selectedInv(InvID,_)
  <- cartago.use(InvID, get, mySensor);
  cartago.sense(mySensor, new_item(Item));
  cartago.use(console,println("Consumed:",Item));
  !consumeAction.
```

Doxastic Function

- 2 bounded buffers are in place, each with a different item 'load'



```
+!consumeActivity : true
  <- +min_items(-1);
  cartago.lookupArtifact("inventory-1", InvID1);
  cartago.focus(InvID1);
  cartago.lookupArtifact("inventory-2", InvID2);
  cartago.focus(InvID2);
  +selectedInv(InvID1,0);
  !consumeAction.

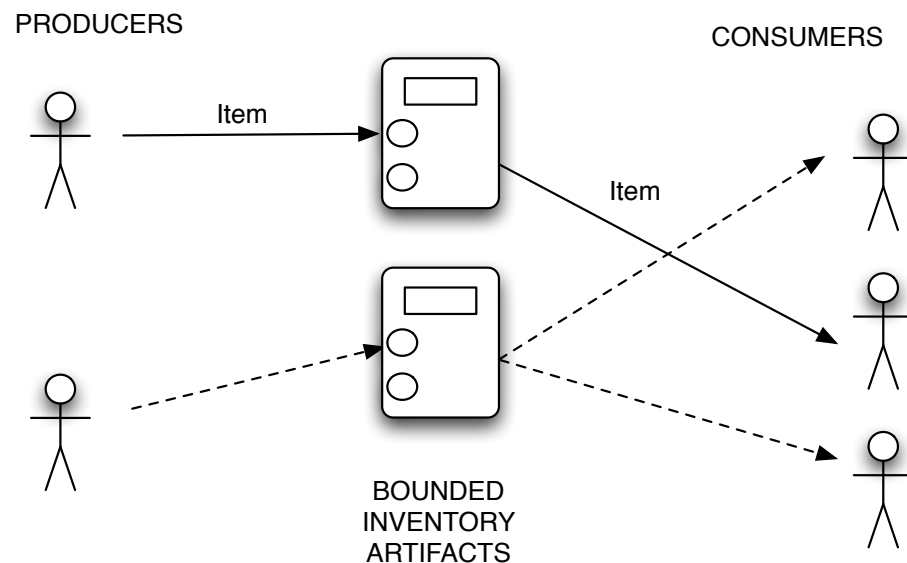
/* Updates Goal Supporting Beliefs */
+n_items(N) [source(percept), artifact(InventoryID)]
: selectedInv(_,N1) & N > N1
  <- -+selectedInv(InventoryID,N).

+!consumeAction : selectedInv(InvID,_)
  <- cartago.use(InvID, get, mySensor);
  cartago.sense(mySensor, new_item(Item));
  cartago.use(console,println("Consumed:",Item));
  !consumeAction.
```

- Consumer can decide which is the most reliable one to use (Goal Supporting Beliefs)

Doxastic Function

- 2 bounded buffers are in place, each with a different item 'load'



```
+!consumeActivity : true
  <- +min_items(-1);
  cartago.lookupArtifact("inventory-1", InvID1);
  cartago.focus(InvID1);
  cartago.lookupArtifact("inventory-2", InvID2);
  cartago.focus(InvID2);
  +selectedInv(InvID1,0);
  !consumeAction.

/* Updates Goal Supporting Beliefs */
+n_items(N) [source(percept), artifact(InventoryID)]
: selectedInv(_,N1) & N > N1
  <- -+selectedInv(InventoryID,N).

+!consumeAction : selectedInv(InvID,_)
  <- cartago.use(InvID, get, mySensor);
  cartago.sense(mySensor, new_item(Item));
  cartago.use(console,println("Consumed:",Item));
  !consumeAction.
```

- Consumer can decide which is the most reliable one to use (Goal Supporting Beliefs)
- By focusing art. properties, the update of Goal Supporting Beliefs is done automatically
 - in reaction to a “property change” event

CONCLUSIONS

ARTIFACT BASED WORK ENVIROMENT

Artifacts provide serviceable operations, pre-processed information and coordination facilities in WORK ENVIRONMENT conceived -by the MAS designer- *for easing agents work*

ARTIFACT BASED WORK ENVIROMENT

Artifacts provide serviceable operations, pre-processed information and coordination facilities in WORK ENVIRONMENT conceived -by the MAS designer- *for easing agents work*

- To distribute Information and Services in **Multi-Agent**
 1. Across *Agents*: organise and make available relevant information as permanent side-effect of artifact use (modification of artifact state)
 2. Across *Platforms*: mediated interactions
 3. Across *Time/Space*: hold strategic information persistent over agent presence and no need for agents mutual presence within a location or heavy message exchange protocols

ONGOING EVALUATION (APPLICATIONS)

- ORA4MAS (w/ Hubner, Boissier, Kitio, Ricci)
 - exploiting artifacts to build organisational infrastructures
- CARTAGO-WS
 - artifacts for SOA/WS applications
 - interacting with web services
 - implementing web services
- FORMAL MODEL
 - Interactions
- ARTIFACT LIBRARIES
 - setting up a set of reusable artifacts in MAS applications

REFERENCES

- CARTAGO is an open-source technology
 - Project HomePage: <http://cartago.sourceforge.net>
- ...and Friends:
 - <http://alice.unibo.it/xwiki/bin/view/CARTAGO/Integrations>
- Agents & Artifacts Theory, Design and Practice
 - Google Group:
 - <http://groups.google.com.au/group/agents-and-artifacts>

BIBLIOGRAPHY

Agens Faber: Toward A Theory of Artefacts for MAS *Electronic Notes in Theoretical Computer Sciences*, Elsevier Science B.V., **2006**, 150, 21-36

'Give Agents their Artifacts': The A&A Approach for Engineering Working Environments. *In 6th international Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2007). Honolulu, Hawai'i, USA, 2007*

Ricci, A., Viroli, M. , Omicini, A. The A&A Programming Model & Technology for Developing Agent Environments in MAS. *ProMAS'07, Post-proceedings, Springer, 2007*, 4908, 91-109

Piunti, M., Ricci, A. Cognitive Artifacts for Intelligent Agents in MAS: Exploiting Relevant Information residing in Environments. *Workshop on Knowledge Representation for Agents and Multi-Agent Systems (KRAMAS 2008), Sydney, 2008*

Ricci, A., Piunti, M., Acay, L. D., Bordini, R.; Hubner, J. , Dastani, M. Integrating Artifact-Based Environments with Heterogeneous Agent-Programming Platforms. *In 6th international Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'08). Estoril, 2008*

Artifacts in the A&A meta-model for multi-agent systems. *Journal of Autonomous Agents and Multi-Agent Systems*, **2008**, 17 (3)

Piunti, M.; Ricci, A.; Braubach, L. & Pokahr, A. Goal-Directed Interactions in Artifact-Based MAS: Jadex Agents playing in CARTAGO Environments *In IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology. Sydney. 2008*

Piunti, M. , Ricci, A. From Agents to Artifacts Back and Forth: Operational and Doxastic Use of Artifacts in MAS. *In Proceedings of Sixth European Workshop on Multi-Agent Systems, to appear. 2008*

DAAD A MANIFESTO

first class abstraction, dual to ones used by agents

Self-contained computational units defined based on their functionalities

Persistent entities, conforming to the interaction dynamic (action/perception laws) defined at the application level (not domain level)

Artifacts are not Agents:

As agents, artifacts are entities existing (persisting) in the same work environment

Differently from agents, artifacts don't encapsulate their control, have interfaces (to be used),

- Artifact are not objects:
 - operations are not methods in the form of system calls
 - "object" (in general sense) shaped in agentive canon. In the particular case of our work they are designed to be exploited by agent with mental attitudes (so they have informational and operational functionalities to be mapped in agent mental states).
 - Artifacts are located in workspaces where they exist and are supported by a run time environment that resolve conflicts (cuncurrency, distribution)