From Agents to Artifacts Back and Forth
Purposive and Doxastic Use of Artifacts in MAS

Michele Piunti (1,2), Alessandro Ricci (1)

(1) aliCE group at DEIS, Università di Bologna, Cesena
(2) Istituto Scienze e Tecnologie della Cognizione, ISTC-CNR, Roma

[michele.piunti, alessandro.ricci]@unibo.it
OUTLINE

• Agents and Artifacts (A&A) approach to MAS

• Agents and Artifacts in the Loop of Cognitive Interactions

• Example: Producers-Consumers

• Conclusions
ENVIRONMENT MODELS IN MAS

- Infrastructure for message exchange
- Medium for external worlds

Example: JAVA PLATFORM

REAL WORLD (PHYSICAL OR COMPUTATIONAL)

EXTERNAL WORLD (PHYSICAL OR COMPUTATIONAL)

MAS ENVIRONMENT

SIMULATED WORLD

or

INTERFACE

or

WRAPPER TO EXISTING TECHNOLOGY

mimicking

REAL WORLD (PHYSICAL OR COMPUTATIONAL)

EXTERNAL WORLD (PHYSICAL OR COMPUTATIONAL)

Example: JAVA PLATFORM

MAS

agents

actions

percepts

AGENTS

EUMAS 2008, Bath UK
ENVIRONMENT MODELS IN MAS

- **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
PROBLEMS

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

Can we leave aside Agents-Environments Interactions?
PROBLEMS

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
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*
AGENTS AND ARTIFACTS (A&A)

- Artifact as First Class Abstraction in design of MAS
  - Dual notion wrt Agent
  - Self-contained computational units defined based on their (special purpose) functionalities

- Workspaces
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
  - Action Perception
  - 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, distribution)
CARTAGO ARCHITECTURE

MAS Application

Artifact-based working environments
- shared task scheduler
- shared KB
- workspaces
- blackboard
- map

Application Specific Logic

Agent Framworks / Middlewares
- JASON
- JADEX

Execution Platform

Any JVM
OS

MAS Middleware Layer

JVM
OS

From Agents to Artifacts Back and Forth
### INTEGRATING HETEROGENEOUS AGENTS PLATFORMS

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

| workspace management          | $\text{joinWsp}(\text{Name}, \text {?WspId}, \text{+Node}, \text{+Role}, \text{+Cred})$  
|                              | $\text{quitWsp}(\text{Wid})$ |
| artifact use                  | $\text{use}(\text{Aid}, \text{OpCntrName}(\text{Params}), \text{+Sensor}, \text{+Timeout}, \text{+Filter})$  
|                              | $\text{sense}(\text{Sensor}, \text {?Perception}, \text{+Filter}, \text{+Timeout})$  
|                              | $\text{grab}([\text{Aid}])$  
|                              | $\text{release}([\text{Aid}])$ |
| artifact observation          | $\text{observeProperty}(\text{Aid}, \text{PName}, \text {?PValue})$  
|                              | $\text{focus}(\text{Aid}, \text{+Sensor}, \text{+Filter})$  
|                              | $\text{stopFocus}(\text{Aid})$ |
| artifact instantiation, discovery, management | $\text{makeArtifact}(\text{Name}, \text{Template}, \text{+ArtifactConfig}, \text{?Aid})$  
|                              | $\text{lookupArtifact}(\text{Name}, \text{?Aid})$  
|                              | $\text{disposeArtifact}(\text{Aid})$ |
AGENTS AND ARTIFACTS IN THE LOOP OF COGNITIVE INTERACTIONS
Artifacts provide serviceable operations, pre-processed information and coordination facilities conceived by the MAS designer for easing agents work.
Artifacts provide serviceable operations, pre-processed information and coordination facilities conceived by the MAS designer for easing agents work

- **PROBLEM:**
  What’s for agent architectures and systems...
  - ...When Agents can reason? /w mental states (Beliefs, Goals, Intentions)?
  - Why agents should allocate resources to use artifacts ?
  - What is going to change in reasoning model to bring about artifacts?
CONCEIVING A&A INTERACTIONS

**Agents using Artifacts:**

[Dependance Theory in Social Systems: Castelfranchi, 92]

An agent $AG$ depends on an Artifact $AR$ for a given goal $G_i$, according to a set of plans $P_{ji}$ if:

1. It has $G_i$ in its set of goals
2. It is not autonomous for $G_i$
   - Lacks at least one of the resources (actions/information) necessary to achieve $G_i$
3. There is a plan $p_{ji} \in P_{ji}$ that achieves $G_i$ where at least one resource (action/information) used in this plan is in $AR$’s set of available operations or $AR$’s observable state
   - Artifact Control Interface or Observable Properties

**Interaction : Functional Approach**
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]
Operational Function

- Operations are Artifacts *intended* purposes
  - “Intended” is in the mind of Artifact Designers before being in agents intentions

- Exploitable as self contained operations:
  - Improving repertoire of actions, as additional means to achieve Goals
  - Externalise and distribute part of agent activities

EUMAS 2008, Bath UK
Operational Function

- Goals are achieved by the mean of operations which have been defined -by the artifact designer- within artifacts control interface.

- Agent’s (sub)goals ‘are’ in Operation Outcomes:

```prolog
%!doAction
    : not done(myGoal) & mySensor(S) & artifactID(Aid)
    <- cartago.use(Aid, doAction(X, Y, ...), S);
    cartago.sense(S, done(Res) );
    +done(Res). /* Res is myGoal */

%!doAction : true
    <- ?done(myGoal).
```
Doxastic Function

AGENT

2 property updated

obs_prop

value

op_1
op_2
op_3
op_4

1 focus("artifact")

an ARTIFACT
Doxastic Function

- Artifacts include *machine-readable* Representations
  - Can maintain, make it observable, pre-process information

AGENT

AGENT

1 focus("artifact")

2 property updated

obs_prop

value

○ op_1
○ op_2
○ op_3
○ op_4

an ARTIFACT
Doxastic Function

• **Artifacts include *machine-readable* Representations**
  
  - Can maintain, make it observable, pre-process information

![Diagram of agent focusing on an artifact](image)

• **Artifacts are *informational* units, exploitable in a *situated* way:**
  
  - As external repositories, collecting and providing strategic knowledge
  - Additional memory, even shared between agent groups
  - Observable cues, to highlight relevant information (*situated cognition*)
Doxastic Function

- Artifact Observable properties can be exploited by agents to retrieve strategic information.
- The mechanism of observation is conceived as a weak interaction.
- Strategic (Goal Supporting) knowledge ‘is’ in artifacts readable properties:
Doxastic Function

• Artifact Observable properties can be exploited by agents to retrieve strategic information.

• The mechanism of observation is conceived as a weak interaction.

• **Strategic (Goal Supporting) knowledge ‘is’ in artifacts readable properties:**

```prolog
+!achieveGoal : artifactID(Aid)
   <- cartago.observeProperty(Aid, propertyName(X) );
   !doAction(X).

+!doAction(X)  /* X is a Goal Supporting Belief */
   : X > 40
   <- +done(myGoal).

+!doAction(X)
   : X > 20
   <- cartago.use(Aid, doAction(X, Y), S);
   cartago.sense(S, done(Res) );
   +done(Res).  /* Res is myGoal */
```
AN EXAMPLE:

PRODUCERS

CONSUMERS
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 synchronize/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?)

![Diagram of producers and consumers in MAS](image)
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...)
Using an Object as a shared buffer reduces the amount of interactions
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.
• Use a Mediator Agent maintains the same level of abstraction between entities
• 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!

Item

Item

BOUNDDED INVENTORY ARTIFACT

PRODUCERS CONSUMERS IN MAS

Use an Artifact!

Item

Item

BOUNDDED INVENTORY ARTIFACT

EUMAS 2008, Bath UK

From Agents to Artifacts Back and Forth
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)
BOUNDDED INVENTORY

| n_items | 0 |
| max_items | 100 |
| put | get |

OBSERVABLE PROPERTIES:

n_items: int+
max_items: int

Invariants:
n_items <= max_items

USAGE INTERFACE:

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

get / (n_items >= 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

<table>
<thead>
<tr>
<th>+!produceItems : nextItemToProduce(Item)</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;- cartago.lookupArtifact(&quot;my-INV&quot;, InvID);</td>
</tr>
<tr>
<td>cartago.use(InvID, put(Item), 500).</td>
</tr>
<tr>
<td>/* Fail Event */</td>
</tr>
<tr>
<td>-!produceItems: true</td>
</tr>
<tr>
<td>&lt;- cartago.use(console,</td>
</tr>
<tr>
<td>println(&quot;Insertion failed due to timeout.&quot;).)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>+!consume</th>
</tr>
</thead>
<tbody>
<tr>
<td>: myInventory(InvID) &amp; mySensor(S)</td>
</tr>
<tr>
<td>&lt;- cartago.use(InvID, get, S, 1000);</td>
</tr>
<tr>
<td>cartago.sense(S, new_item(Item));</td>
</tr>
<tr>
<td>!consumeItem(Item);</td>
</tr>
<tr>
<td>!consume.</td>
</tr>
<tr>
<td>+!consumeItem(Item) : true</td>
</tr>
<tr>
<td>&lt;- ...</td>
</tr>
</tbody>
</table>
Operational Function

- Both Producer and Consumer Goals are in Artifact outcomes

```plaintext
+!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

- 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 preprocess 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’

```plaintext
+!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’

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

```prolog
%!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’

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

- By focusing artifact properties, the update of Goal Supporting Beliefs is done automatically
  - in reaction to a “property change” event

```plaintext
/* 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.use(mySensor, new_item(Item));
  cartago.use(console, println("Consumed: ", Item));
  !consumeAction.
```

EUMAS 2008, Bath UK
CONCLUSIONS
Cognitive Artifacts

The notion is related to agents able to bring about artifact informational functions [Normann]
Cognitive Artifacts

The notion is related to agents able to bring about artifact informational functions [Normann]

- To provide Relevant, Strategic Information to Single-Agent
  - Easy reasoning about goals: goal supporting beliefs
  - Simplifying agent choices and easing deliberation processes
Cognitive Artifacts

The notion is related to agents able to bring about artifact informational functions [Normann]

- **To provide Relevant, Strategic Information to Single-Agent**
  - Easy reasoning about goals: *goal supporting beliefs*
  - Simplifying agent choices and easing deliberation processes

- **To distribute Information 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
Two more Steps towards an Extended Mind [Clarks]

Artifacts can be *cognitively* used once their representational and operational contents are mapped into reasoning processes

- **Artifact discovery as a process of *Epistemic Reasoning***
  - Use of MANUALS to allow agent to *Read, learn, and match* Artifact Representational contents *w* mental attituses

- **Operational contents has to be included in *Practical Reasoning* (to achieve goals)**
  - By changing the actions required for achieving a goal, artifact operations change means-end reasoning
  - Adding runtime new means: on-line planning

EUMAS 2008, Bath UK
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:
BIBLIOGRAPHY


DA A AD 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 runtime environment that resolve conflicts (concurrency, distribution)