From Agents to Artifacts Back and Forth: Operational and Doxastic use of Artifacts in MAS

Michele Piunti
Università degli studi di Bologna
DEIS - Bologna, Italy
Email: michele.piunti@unibo.it

Alessandro Ricci
Università degli studi di Bologna
DEIS - Bologna, Italy
Email: a.ricci@unibo.it

Abstract— **Recent approaches in Multi-Agent Systems are focusing on providing models and methodologies for the design of environments and special purpose tools supposed to ease programming in the large and scale up growing complexities. Among others, the Agents and Artifacts (A&A) approach introduced the notion af artifact as first class abstraction providing agents with external facilities, services and coordination medium explicitely conceived for promoting their activities. In this paper we analyse A&A systems by focusing on the *functional roles* played by artifacts. In particular, we here investigate the function of artifacts once they are employed in the context of societies of cognitve agents, i.e. agents capable to reason about their epistemic and motivational states. In this context, a twofold kind of interaction is envisaged. On the one side, artifact rapresentational function allows agent to improve epistemic states, i.e., by representing and sharing strategic knowledge in the overall system (*doxastic use*). On the other side, artifacts operational function allows agents to improve the repertoire of actions, i.e., by providing additional means which can be purposively triggered by agents to achieve goals (*operational use*). Some of the outcomes of this approach are discussed along with test cases showing agents engaged in goal-oriented activities relying on the transmission of relevant knowledge and the operations provided by artifacts.**

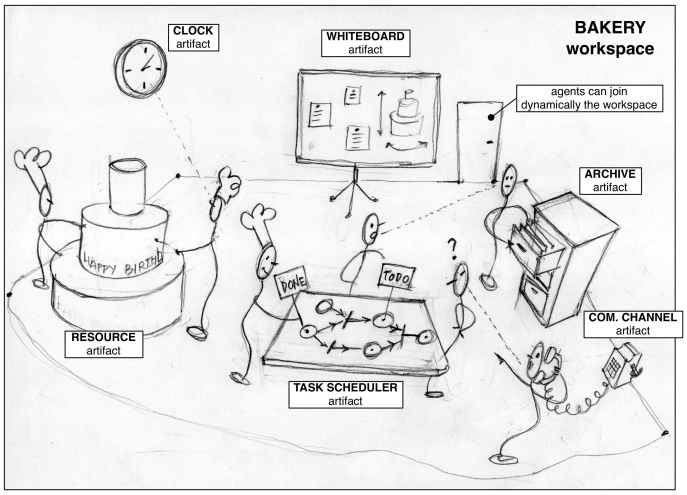


Fig. 1. A metaphorical representation of a MAS according to A&A.

I. Introduction

The *artifact* abstraction has been recently introduced in Multi-Agent System (MAS) [13] and MAS programming [20] as a basic building block to model and design agent environments and, more exactly, agent *work environments*. The notion of work environment used here refers to that part of the MAS – so developed by MAS designers and developers – which is perceived and used by agents as a first-class entity of their world, and which provides suitable functionalities and services that agents can exploit to ease their individual and social activities [18]. Artifacts – as introduced by the A&A conceptual model – can be conceived as basic module to structure such work environments, representing *non-autonomous* computational objects[1] that agents can dynamically instantiate, share and use as *resources* and *tools* to support and promote their activities. Mutuating the notion of ecosystems[2] or human societies, where individuals are supposed to behave and interact by means of shared knowledge, cultural transmission, memes [6], A&A states a clear separation of concern between the entities dwelling in a MAS: whereas agents can be considered as the proactive actors of the systems, exhibiting a purposive and autonomous behavior, artifacts are the non-autonomous entities, providing agent with facilities and special purpose tools to be exploited as external resources in order to serve a functional role [14]. According to this view, a MAS is designed and programmed in terms of an ensemble of agents that play together in a common (work) environment not only by communicating through some high-level Agent Communication Languages, but also co-constructing and co-using different kinds of artifacts, organised in *workspaces* (Figure 1 shows a metaphorical picture of a MAS in this perspective). The main source of inspiration underlying this view comes from human societies and research works in Activity Theory [12], remarking the fundamental role that play artifacts in our society in mediating and supporting human work, in particular cooperative work. Besides A&A, CARTAGO [20] has been introduced as a platform and infrastructure providing a concrete computational model to program artifacts and a distributed runtime for executing artifact-based work environments, making it possible for agents developed on different agent platforms to dynamically join and work inside such environments [18].

[1]The notion of object is used here in its general term, meaning a dynamic entity with a proper identity

[2]Introduced by Cristopherson in [5], ecosystem has been defined as "a natural unit consisting of all plants, animals and micro-organisms (biotic factors) in an area functioning together with all of the non-living physical (abiotic) factors of the environment".

At first, for MAS designers the usefulness of the artifact abstraction concerns the availability of an explicit level of abstraction and technologies – based on A&A concepts – for modeling, design and programming work environments for different kinds of purposes. This is the main line followed by most of the existing work exploiting the notion of artifact, where work environments are mainly devoted at ruling and promoting complex social interactions. Recent examples are [9], [8], where the notion of artifacts is used respectively to conceive and design organisation infrastructures for open MAS and to support the design of reputation mechanisms.

Then, a further – more challenging – level can be devised, in which MAS designers conceive and design agents – typically *cognitive* agents – which are capable to reason about their work environment and dynamically decide how to exploit it depending on their goals and tasks. This level is fundamental when *open* MAS are of concerns, and introduces many interesting and challenging issues, both on the artifact side and the agent side.

On this line, in this paper we report on preliminary investigation concerning the *functional roles* played by artifacts in the context of cognitive agent societies, and we relate such roles to the *epistemic* and *motivational* states of agents working with artifacts. As a result, we identified two fundamental general functions (described in Section III): (i) *doxastic*, which allows agent to improve their epistemic states, by representing and sharing strategic knowledge in the overall system; (ii) *purposive*, which allows agents to improve the repertoire of actions, by providing additional means which can be purposively triggered by agents in order to achieve their goals. Besides the conceptual aspects, in Section III we show some practical outcomes of the work reporting simple examples involving agents programmed with the *Jason* agent platform [1] (based on the AgentSpeak BDI-based agent language) engaged in goal-oriented activities involving artifacts programmed with CARTAGO. In order to ground the discussion and the examples provided in Section III, Section II provides an overview about the computational model of artifacts provided in CARTAGO and the repertoire of actions provided to agents for playing within artifact-based environments. Concluding remarks on the approach are reported in Section IV.

II. The CARTAGO Platform

CARTAGO (Common ARtifact infrastructure for AGent Open environment) is an infrastructure and a platform for programming and executing artifact-based work environments for MAS [20], implementing the A&A conceptual model. In detail, the platform includes a Java-based API for programming artifacts, defining new types of artifacts following the A&A programming model, an agent API to be used in agent-oriented programming platforms to play within CARTAGO environments – including a basic set of actions for creating and interacting with artifacts, and managing and joining workspace – and a runtime and related tools, to execute and manage the artifact-based environments. CARTAGO technology is open-source[3] and implemented on top of the Java platform.

A. Environment Model

A work environment in CARTAGO is conceived as collection of *workspaces* possibly distributed on different nodes where the infrastructure has been installed (referred in the following as CARTAGO nodes). Agents (possibly in execution on multiple and heterogeneous agent platforms) can dynamically join and quit the workspaces, possibly working in multiple (and distributed) workspaces at the same time. A Role-Based Access Control (RBAC) model is adopted for specifying and managing security aspects at workspace level, ruling agent entrance and exit, and agent access and interaction with artifacts.

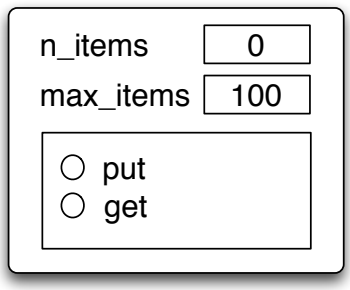
By default, each workspace contains a set of pre-defined artifacts, created at the workspace creation time, which provides some fundamental functionalities and facilities for agents working inside the workspace and for workspace(s) management. Among the others, a *factory artifact* is used to instantiate artifacts, specifying the artifact template and a name and a *registry* artifact is used to keep track of the set of artifacts actually available in the workspace. The general idea is to *reify* in terms of a suitably designed artifact every infrastructure part and functionality so as to make it observable, controllable, adaptable by agents themselves (meaning agents that have the permission to do that according to their role), besides human administrators.

B. Programming Artifacts: An Example

To give a taste of artifact programming model, here we briefly describe a simple example of artifact, a *bounded-inventory*, which contains main aspects of the artifact model, namely observable properties and a usage interface, besides basic synchronization functionalities. The bounded-inventory is a kind of coordination artifact designed to function as a shared inventory mediating the exchange of some kind of *items* between a possibly dynamic number of *producer* agents and *consumer* agents [11]. The producers-consumers problem is typical in concurrent systems, where agents are supposed to adopt effective strategies with respect of the shared resorce and taking into account further bounded resources like time and space (memory). This requires some kind of coordination strategy between agents, i.e., in order to coordinate the cyclic production of items by producer and the activities performed by consumer agents. To this end, the bounded-inventory provides a coordination mechanism to uncouple and – at the same time – synchronize the activities of producers and consumers, thus providing a locus of design (the size of the inventory) for tuning the performance of the system.

Looking at the CARTAGO implementation in Figure 2, the bounded-inventory artifact provides a usage interface with two operation controls to respectively insert (`put`) e consume (`get`) items, and two observable properties,

[3] Available at `http://cartago.sourceforge.org`.



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;
  }
}
```

Fig. 2. A simple bounded-inventory artifact, exploiting operation control guards to synchronize agent use of the inventory.

`max_nitems`, showing the maximum capacity of the inventory, and `n_items`, showing the current number of items stored in the inventory. Internally, a simple linked list is used to store items. The synchronization functionality provided by the artifact is realised here by exploiting a basic feature of the artifact programming model, which accounts for the possibility of defining *guards* that specify when an operation controls is either enabled or disabled. In the example the `put` control is allowed only when the inventory is not full (`inventoryNotFull` guard), and `get` is allowed when the inventory is not empty (`itemAvailable` guard). Hence, if an agent selects the `put` operation control and the inventory is full, the action is suspended. Analogously for the `get` control, if the inventory is empty. A most detailed description of the Java-based API and of this use case, along with other examples, can be found in [19].

C. Integration with Agent Platforms

CARTAGO is envisaged for enabling full interoperability across heterogeneous agent platforms, hence it has been designed to be orthogonal to the specific models and technologies adopted for the agents working in it. It allows integration and exploitation in principle of *any* agent programming platform, enabling agents of heterogeneous platforms (and finally models and architectures) to interact and interoperate as part of the same MAS, sharing common artifact-based environments [18].

To realise such integration, both from a conceptual and engineering point of view, the notion of *agent body* is exploited, as that part of an agent which is belonging to a CARTAGOworkspace. Whereas the *agent mind* remains in execution externally – whithin the agent platform – an agent body is physically running in a CARTAGOsystem. Hence, the agent body logically and physically *situates* an agent in a CARTAGOworkspace: in particular, it contains proper *effectors* that make it possible essentially to act upon (*use*) artifacts, and *sensors* to detect and perceive observable events generated by artifacts, possibly applying filters and specific kinds of "buffering" policies. From an architectural point of view, to connect agent mind and agent body, platform-specific *bridges* are introduced, functioning as wrappers on the agent mind side to control the body and perceive stimuli collected by body's sensors. Currently, bridges exists for *Jason* [1], an interpreter for an extended version of AgentSpeak, Jadex [17], a BDI agent platform based on Java and XML as mainstream language / technologies to develop intelligent software agent systems, and simpA [21], a Java-based agent-oriented framework based on the A&A conceptual model.

D. The Tenet of Agent-Artifact Interaction Model: Use and Observation

To enable interactions between agents and artifacts the repertoire of actions natively provided by the agent platforms

is extended with a new set of special-purpose actions envisaged for playing inside an artifact-based environment. The overall set of new actions can be grouped in four groups, as depicted in Table I: *(i)* join and leave workspaces; *(ii)* create, lookup, dispose of an artifact; *(iii)* use an artifact; *(iv)* observe an artifact without directly using it.

The core part of this set is given by actions in the last two groups, concerning artifact *use* and *observation*. The agent activities bleonging to artifact use and observation are the tenet of the agent-artifact interaction model and their understanding is the pivotal underpinning of this work.

The `use` action is provided to agents so as to act upon the artifact by selecting an operation control. To use an operation its description needs to be part of the artifact usage interface, and eventually specifying parameters required by the control (see Figure 3). If the use action succeeds, then a new instance of the operation linked to the operation control starts its execution inside the artifact. The execution of the operation eventually generates a stream of observable events that may be perceived both by the agent which is responsible of the operation execution and by all the agents that are *observing* the artifact. Some basic types of events are meant to be generated by default by artifacts, in spite of their specific type, in correspondence to particular situations (i.e., the completion or the failure of an operation, the update of an observable property, or rather the disposal of an artifact). Two aspects are important here. First, the execution of a use action upon an operation control involves a *synchronous* interaction between the agent and the artifact: action success means that the operation linked to the control has started its execution. Second, the execution of the operation is completely *asynchronous with respect to agent activity*. Hence, use does not involve any transfer of control as it happens in the case of remote procedure call or method invocation in procedure-based or object-oriented systems.

Besides use, *observation* is the second main aspect concerning agent-artifact interaction. To perceive the observable events generated by the artifact two basic modalities are possible, called here *active* and *passive*. In the active modality, the agent doing a `use` explicitly indicates a *sensor* as a parameter. The sensor is thus meant to collect all the observable events (which can be reffered to the triggered operation) as soon as they are generated; then, a further `sense` action is provided to the agent to actively fetch those events as percepts, possibly specifying filtering rules. In so doing the agent actively retrieve the percepts from the sensor on demand, *as soon as it needs it*. Sensors in this case play the role of *perceptual memory* explicitly manageable by the agent, who can use them to organise in a flexible way the processing of the events, possibly generated by multiple different artifacts that an agent can be using for different, even concurrent, activities.

In the passive modality events generated by an artifact are authomatically made observable to the agent directly as native/internal events, without the explicit mediation of sensors. In other terms, the bridge mechanism translate the events coming from the scrutinized artifact into events holding the agent architecture. Besides, the agent passively receives those events as native signals to be handled within the reasoning cycles. Those events are supposed to contain relevant information about the occurrence of the originating artifact event (i.e. , the source of the event, associated labels, contents etc.).

As an additional interaction modality, besides perceiving the events related to a previous use, a support for *pure observation* – that is, observation without use – is provided, concerning both observable properties and observable events. For *continuous* observation of properties and events a specific action called `focus` is provided (see Figure 4): by executing a focus on a specific artifact, an agent can continuously perceive the state of artifact observable properties and thus is notified of all the observable events that the artifact will generate from that moment on, even if it is not actually using it. Observable properties are directly mapped onto agent percepts and then, for cognitive agent architecture in particular, can be related to percepts or beliefs indicating the situated state of the artifact. For observable events, the two perceptive (active and passive) modalities are available also for the `focus` action, either specifying or not a sensor. The semantics is the same as in the use case: by specifying a sensor all the observable events generated by the artifact are detected by the sensor and eventually fetched by the agent through a `sense` internal action. A further action concerning observation is `observeProperty`, which makes it possible read the current value of a specific observable property, which is returned directly as feedback of the action.

It's worth noting that continuous observation of properties and perception of events have different characteristics (and then purposes, from the designer point of view). In particular, observable properties represent the *state* of the environment (structured in terms of artifacts) and, as such, it could change with a frequency that could be beyond agent perceiving (and related) capabilities. Instead, observable events represent *changes* in the world and typically are buffered and processed in some kind of order that could depend on event priorities or agent actual promptness. This is true in particular for cognitive agents which can indeed follow adaptable strategies in allocating their attentive resources.

Agents using mental states are the ideal candidate to manage complex interactions from agents to artifacts involving interleaved operation calls performed on heterogeneous artifacts distributed across nodes and workspaces. Based on the various execution models employed by the various integrated agent platforms, more complex form of loosely coupled interaction between agents and artifacts can be suitably conceived. In what follows we'll provide a systematization on the functional terms at which a complex interaction can be conceived in cognitive terms. Whereas agents perceptive capabilities allow to dynamically store and situate information which is relevant for the ongoing purposes, reasoning capabilities may promote the use of external services provided by artifacts, orchestrating a tight composition of chained sequences of operation calls.

(1) `joinWorkspace(+Workspace[,Node])`
(2) `quitWorkspace`
(3) `makeArtifact(+Artifact,+ArtifactType[,ArtifactConfig])`
(4) `lookupArtifact(+ArtifactDesc,?Artifact)`
(5) `disposeArtifact(+Artifact)`
(6) `use(+Artifact,+UIControl([Params])[,Sensor][,Timeout][,Filter])`
(7) `sense(+Sensor,?PerceivedEvent[,Filter][,Timeout])`
(8) `focus(+Artifact[,Sensor] [,Filter])`
(9) `stopFocussing(+Artifact)`
(10) `observeProperty(+Artifact,+Property,?PropertyValue)`

TABLE I

ACTIONS INTRODUCED IN AGENT'S REPERTOIRE ALLOWS THE INTERACTION IN CARTAGOWORKING ENVIRONMENT. THEY ARE FUNCTIONALLY DIVIDED IN FOUR MAIN GROUPS: FOR MANAGING WORKSPACES (1–2), FOR CREATING, DISPOSING AND LOOKING UP ARTIFACTS (3–5), FOR USING ARTIFACTS (6–7), AND FOR OBSERVING ARTIFACTS (8–10). SYNTAX IS EXPRESSED IN A LOGIC-LIKE NOTATION, WHERE ITALICISED ITEMS IN SQUARE BRACKETS ARE OPTIONAL. CONCRETE REALISATION OF THE ACTIONS DEPENDS ON THE SPECIFIC AGENT PROGRAMMING PLATFORM WHICH HAS BEEN INTEGRATED [18], [15]

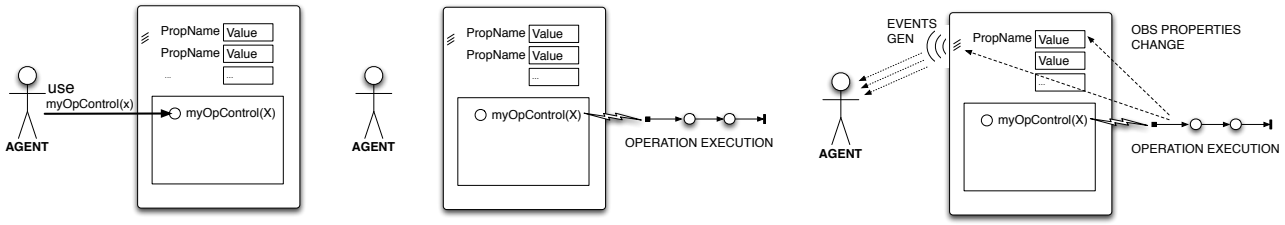


Fig. 3. Using an artifact: by selecting the `myOpControl` control belonging to the usage interface, a new operation instance starts its execution inside the artifact. The execution of the operation will eventually generate events observable to the user agents – and to all the agents observing the artifact – and possibly update artifact observable properties.

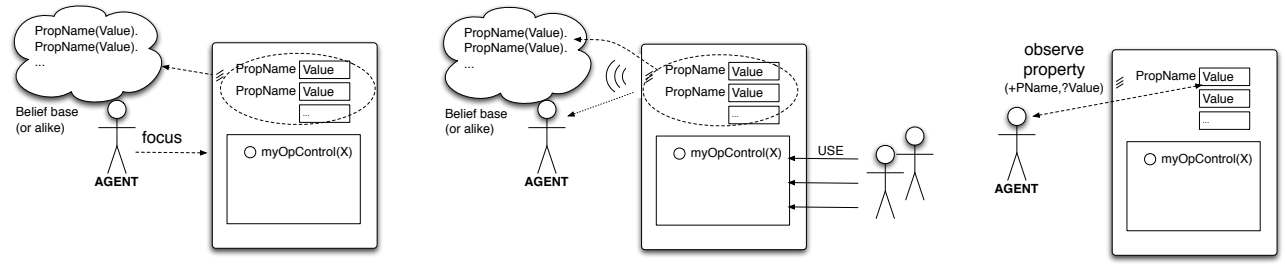


Fig. 4. Observing an artifact: by focussing an artifact, an agent is (1) continuously made aware of observable properties value as percepts typically mapped into agent belief base, and (2) receives all the observable events generated by the artifact in executing operations possibly triggered by other user agents.

III. COGNITIVE USE OF ARTIFACTS

A main aspect of cognitive system study concerns the investigation of how information is represented and how those representations are transformed, combined and propagated so as to form a behavior [24]. In particular, we here refer to a general explanation of cognitive agents built upon the two pronged notion of epistemic and motivational states. In this view, cognitive agents can be described as *intentional systems* able to autonomously reason about their resources – mappable upon internal representations – in order to pro-actively reach a desired state of affairs. On the epistemic dimension, cognitive agents are assumed to support their reasoning processes on the basis of their internal knowledge, namely "beliefs". Beliefs can be viewed as those *doxastic representation* related on the information agents are able to find, integrate and take into account. Besides epistemic states, motivational states allow agents to "pursue" a given course of actions, i.e. by committing an intention (among the achievable ones) through the execution of some action or plan they have in repertoire. On these bases, in this section we refer to artifacts which can be cognitively used, read and created by cognitive agents.

In this view, artifact are supposed to be (not only) computational components structuring the environment (but also) resources which can be interactively and cognitively exploited by agents to attain their goals. Given the model abstractly described in Section II the following sections provide a deeper analysis on the cognitive terms underlying interactions from agents to artifacts. Before detailing the operational and dox-

astic use of artifacts, the next section describes the cognitive use of active and passive perception styles.

A. Active and Passive Interaction Styles

As seen in Subsection II-D, two basic approaches have been envisaged for agents in order to manage their perceptive activities upon scrutinized artifacts, namely *active* and one *passive* modalities. Sensors can be seen as part of an *agent body*, logically situating an agent into a workspace and containing both sensors and effectors to act upon artifacts of that workspace. Hence, in the active modality sensors play the role of *perceptual memory* or external working memory, whose functionality accounts for keeping track of stimuli arrived from the CARTAGO environment. Accordingly, sensors can be programmed by defining rules, filters and specific kinds of "buffering" policies. This allows agents to retrieve relevant events, even interleaved and generated by multiple artifacts that the agent may use for different, even concurrent, purposes. This approach provides to agent developers the possibility to customize the perceptive activities at an intentional level. Percepts generated by artifacts can be situated in the context of the adopted goals, and thus managed through internal actions to be executed within the plan workflow. In so doing, active perception makes it possible for agents developper to organize perceptive activities – at the programming level – as flexibly as they wish. For instance, in active modality, a given sensor can be devoted to filter relevant events coming from a scrutinized artifact so as to suddenly become aware on artifact changes. Accordingly, filterd events can be proactively and intentionally processed, i.e. in order to update beliefs or check goal achievement.

The passive modality allows the automatic propagation of native internal events at runtime, generated by translating on the fly the events coming from the scrutinized artifacts. This makes it possible for agents to react to observable event asynchronously, as soon as they are perceived. This functioning is supposed to ease agent's reasoning allowing pivotal processes as goal adoption and plan selection to be governed by internal events, which in turns can be targeted on the basis of the events coming from artifacts. This might be the case when agents perform activities in a reactive fashion, for instance when they have to check the execution of operations wich has been externalised in artifacts, as well as a control activity is being automated in the human case. As showed in [15], [16], using events coming from artifacts the agent can handle events so to trigger plans and thus decide the next course of actions. This has a special importance once the agent needs to manage low level and routinized interaction activities. Besides reactive behavior, events coming from artifacts can signal to the agent situation requiring particular servicing: once abnormal values are encountered or exceptional situations arrive, agents can arouse and suitably exploit such signals for reentering the deliberation process or for reconsider their intention. Besides, becoming aware about those relevant facts, agents can elicit reallocation of resources, recovery policies, exception handling etc.

B. Operational function

Artifact operations, controlled by the usage interface, encapsulate artifact's intended purposes[4]. From the agent viewpoint, operations can be suitably used to improve agent repertoire of actions, providing additional means to achieve agents' goals. They can be targeted dynamically by agents so as to *externalise* and distribute (part of) their goal-oriented activities. For doing this, operation outcomes have to be taken into account by agents in their practical reasoning. In fact, by changing the actions required for achieving a given goal, artifact operations change agent means-end reasoning[5] stages.

This aspect can be tackled at different conceptual levels. The first, most obvious, solution is to integrate artifacts functionalities in the agent's developing phases. In so doing artifacts use can be defined at the language level, by defining the operation control in an off-line fashion, at design time. Referring to the bounded-inventory example introduced in Subsection II-B, an agent having the goal to produce a new item and put it in the buffer may use the following intention (agent's specification is provided with *Jason* language):

```
+!produceItems : nextItemToProduce(Item)
  <- cartago.lookupArtifact("my-inventory", InvID)
     cartago.use(InvID,put(Item),5000).

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

The agent here selects the intention to store an item on the inventory once an `Item` has been prepared and is available in the belief base. Then the adopted intention first lookups the `my-inventory` artifact to retrieve its system identifier `InvID` and then stores the item by selecting the `put` control provided by the artifact usage interface. Notice here the presence of a 5000 milliseconds timeout, after which the action (and the plan) is considered failed and a message is printed on the console by the goal deletion plan `-!produceItems`.

Besides, a consumer agent can cyclically adopt the following plan to attain an item on the inventory:

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

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

Notice that a sensor identified by `S` is explicitly used by the consumer to detect and manage the final event of type `new_item(Item)` generated by the artifact at the end of the `get` operation. This event represents for the agent the signal indicating a goal achievement.

[4]Notice that before beeing in the intention of an agent who wants to use the artifact, the intended purpose is in the mind of artifact designer, who conceive it in order to serve an operation or a function.

[5]We here refer to the notion of cognitive agents able to find a successful sequence of actions, between the ones he has in repertoire, in order to attain an adopted goal. Several agent architectures founded on this reasoning principle have been presented in the last decades, many of which can be related to the conceptual model provided by [2]

C. Doxastic function

A secondary function, dual to operational one, is about informational, observable and retrievable knowledge provided by artifacts and represented by their observable properties. In this case, from an agent point of view, artifacts are informational units functioning to maintain, make it observable and pre-process information which is exploitable in a situated way. In other terms, by embedding machine-readable representations, an artifact can be a target for agents epistemic actions [10]. This entails for agents the opportunity to use, read and observe artifacts to attain new information and possibly update beliefs, solely with the aim to improve the knowledge base with information which is strategic for their tasks. In this view, artifacts are supposed to provide observable cues in order to highlight relevant information (thus improving agent's situated cognition). This turns to be important for shaping *goal-supporting beliefs*, i.e. those beliefs required to agents for ruling over deliberation and practical reasoning [4]. Accordingly, information available with artifacts can ease agent reasoning, for instance simplifying and improving agent's decision making and remarkably easing belief update processes.

As a simple example of doxastic use, we consider here an extension of the producer-consumer scenario in which two bounded inventories are used instead of one (to avoid centralisations, for instance). By continuously observing the number of items of both the inventories, consumer agents must dynamically decide which artifact to use to consume a new item, choosing the one with more items so as to minimise the probability to get stuck because of the inventory is empty. To this end the continuous observation of artifact observable properties is exploited:

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

+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: ", Item));
     !consumeAction.
```

The agent here uses the goal-supporting belief `selectedInv(InventoryID,NItems)` to store the identifier of the inventory with the greatest number of items, among the observed inventories. Such a belief is initially set for `my-inventory-1` artifact in the `consumeActivity` plan, then it is updated by the second plan of the agent each time a new percept about the actual value of the observable property `n_items` of any observed inventory is perceived. In the plan, the annotations `[source(percept), artifact(InventoryID)]` make it possible to retrieve the identifier of the artifact source of the percept.

So, in this case agents are aware of the current state of the artifacts and can rule their means-end reasoning based on goal-supporting beliefs which are *read* on the artifacts. A similar strategy can be implemented for the producer agents (the code is here omitted for brevity) that can use a twofold strategy for choosing the inventory where to put a new item.

D. Discussion

Some final remarks are worth to be taken into account on these cognitive aspects. A first pivotal aspect in threating operational functionalities of artifacts relates on the motivational attitudes of agents. Actually abilities to handle goals are variously characterized by mainstream agent platforms [22]. The procedural goal approaches can be related to agents functioning according to transitions within the action selection policy, where the goal is not explicit in agents specification and where a behavioral policy is rather constructed by the programmer through procedures taking into account the intended goal states. We refer, in the case of agents adopting procedural goals, to a *goal-oriented* use of artifacts. On the contrary, declarative goal approaches refer to agents able to process goals explicitly represented as internal states, where declarativeness stands for explicit representation of goals described either in terms of end-states either in terms of execution states within the reasoning process. As discussed in [15] describing integration between the Jadex agent platform and CARTAGO, we refer, in the case of agents dealing declarative goals, to a stronger notion of usabilty, namely *goal-directed* use of artifacts. A more advanced approach in exploiting operational function envisages an on-line integration, by which agents are enabled to dynamically discover and afford artifact which are not known at design time. This approach requires the additional capability for agents to afford artifacts and map operations, learned form artifact's machine readable descriptions, in their planning and means-end processes. As in the human case, once once an artifact has been acknowledged in terms of its descriptions (i.e. through *manuals*), agents can learn to use operations. In addition, by introducing planning capabilities, an agent can switch actions of his repertoire with operations provided by artifacts to achieve goals.

As for the doxastic function, the contribute of artifacts in easing epistemic activities is remarkable also in the context of Multi Agent scenario. Here the pivotal aspect is the distribution of information in the overall society of agents. In particular, information can be spread over several orthogonal dimensions: *(i)* across agents: by organising and making available relevant information as permanent side-effect of artifact use (modification of artifact state); *(ii)* across platforms: once interactions between agents are mediated by artifacts, heterogeneous platforms can be integrated at the same domain level. Moreover agents acquire an additional option to communicate, being artifacts a suitable alternative to protocols based on message exchange; *(iii)* across time: artifacts are designed to hold strategic information whic can persist also over interleaved presence of individual agents; *(iv)* across space: the topological

notion of work environments makes it possible for agents to distribute their activities between many nodes and workspaces. This entails no need for agents mutual presence within a given location/place.

IV. Conclusion and Related Works

In this work we provided a common grounding for theories and programming approaches based on A&A interaction. In particular, we investigated cognitive aspects of interactions between agents and artifacts, describing the terms of the interaction since the definition of the perceptive activities needed for agents to cognitively operate with artifacts. By adopting a functional approach, we described the twofold role played by artifact once they are used by a cognitive agent. On the one side artifacts are supposed to provide operations, wich agents can exploit to perform activities and attain goals (operational function). On the other side artifacts embeds information which is readable by agents to improve their epistemic states and can be considered as repositories of relevant information in working environments (doxastic function).

Nevertheless the role of the environment as first-class abstraction in designing complex MAS has been largely acknowledged in literature (see [25] for a survey), few works consider the issue of cognitive agents interacting in properly designed environments. Among others, Brahms [23] is a programming language and platform to develop and simulate multi-agent models of human and machine behavior, based on a theory of work practice and situated cognition. Another approach has been developed by Holvoet and Valckenaers [7], who introduce Delegate MAS as a mean to design environment in BDI-based agent architectures. A further work is GOLEM [3], that introduces a platform for modeling situated cognitive agents in distributed environments by declaratively describing the representation of the environment in a logic-based form.

References

[1] Rafael Bordini, Jomi Hübner, and Mike Wooldridge. *Programming Multi-Agent Systems in AgentSpeak Using Jason*. John Wiley & Sons, Ltd, 2007.

[2] M. Bratman. *Intention, Plans, and Practical Reason*. Harvard University Press, 1987.

[3] Stefano Bromuri and Kostas Stathis. Situating Cognitive Agents in GOLEM. In *Engineering Environment-Mediated Multiagent Systems (EEMMAS'07)*, 2007.

[4] C. Castelfranchi and F. Paglieri. The role of beliefs in goal dynamics: Prolegomena to a constructive theory of intentions. *Synthese*, 155:237–263, 2007.

[5] R.W. Christopherson. *Geosystems: An Introduction to Physical Geography*. 1996.

[6] Richard Dawkins. *The Selfish Gene*. Oxford University Press, 1976.

[7] Tom Holvoet and Paul Valckenaers. Beliefs, desires and intentions through the environment. In *AAMAS'06, Proceedings*, pages 1052–1054, New York, NY, USA, 2006. ACM.

[8] Jomi F. Hübner, Olivier Boissier, and Laurent Vercouter. Instrumenting multi-agent organisations with reputation artifacts. In Virginia Dignum and Eric Matson, editors, *Coordination, Organizations, Institutions and Norms (COIN@AAAI), held with AAAI 2008*, 2008.

[9] Rosine Kitio, Olivier Boissier, Jomi Fred Hübner, and Alessandro Ricci. Organisational artifacts and agents for open multi-agent organisations: "giving the power back to the agents". In J. Sichman, P. Noriega, J. Padget, and S. Ossowski, editors, *Coordination, Organizations, Institutions, and Norms in Agent Systems III*, LNCS. Springer, 2008.

[10] Paul P. Maglio and David Kirsh. Epistemic action increases with skill. In *18th Annual Conference of the Cognitive Science Society*, pages 391–396. Erlbaum, 1996.

[11] Thomas Malone and Kevin Crowston. The interdisciplinary study of coordination. *ACM Computing Surveys*, 26(1):87–119, 1994.

[12] B. A. Nardi. *Context and Consciousness: Activity Theory and Human-Computer Interaction*. MIT Press, 1996.

[13] Andrea Omicini, Alessandro Ricci, and Mirko Viroli. Artifacts in the A&A meta-model for multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 17 (3), December 2008.

[14] Andrea Omicini, Alessandro Ricci, Mirko Viroli, Cristiano Castelfranchi, and Luca Tummolini. Coordination Artifacts: Environment-based Coordination for Intelligent Agents. In *Proceedings of AAMAS'04*, volume 1, pages 286–293, New York, USA, 2004.

[15] M. Piunti, A. Ricci, L. Braubach, and A. Pokahr. Goal-Directed Interactions in Artifact-Based MAS: Jadex Agents playing in CARTAGO Environments. In *2008 IEEE/WIC/ACM Conferences on Web Intelligence and Intelligent Agent Technology (IAT-2008)*. IEEE, 2008.

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

[17] A. Pokahr, L. Braubach, and W. Lamersdorf. *Jadex: A BDI Reasoning Engine*, chapter Chapter of Multi-Agent Programming. Kluwer Book, 2005.

[18] A. Ricci, M. Piunti, L. D. Acay, R. Bordini, J. Hubner, and M. Dastani. Integrating Artifact-Based Environments with Heterogeneous Agent-Programming Platforms. In *AAMAS'08, Proceedings*, 2008.

[19] Alessandro Ricci, Michele Piunti, Mirko Viroli, and Andrea Omicini. Environment programming in CARTAGO. In R. H. Bordini, M. Dastani, J. Dix, and A. El Fallah Seghrouchni, editors, *Programming Multi-Agent Systems*. To appear. The draft of the chapter is available at: http://137.204.107.188/ aricci/Drafts/chapter-mas-programming.pdf.

[20] Alessandro Ricci, Mirko Viroli, and Andrea Omicini. The A&A programming model & technology for developing agent environments in MAS. In *ProMAS'07,Post-proceedings*, volume 4908 of *LNAI*, pages 91–109. Springer, 2007.

[21] Alessandro Ricci, Mirko Virolil, and Giulio Piancastelli. simpA: A simple agent-oriented Java extension for developing concurrent applications. In Mehdi Dastani, Amal El Fallah Seghrouchni, Joao Leite, and Paolo Torroni, editors, *Languages, Methodologies and Development Tools for Multi-Agent Systems (LADS 2007*, volume 5118 of *LNAI*, pages 176–191. Springer-Verlag, Durham, UK, 2007.

[22] M. Birna Van Riemsdijk, Mehdi Dastani, and Michael Winikoff. Goals in agent systems: A unifying framework. In *Intern. Conf. on Autononous agents and Multi-Agent Systems (AAMAS08)*, 2008.

[23] Marteen Sierhuis and William J. Clancey. Modeling and simulating work practice: A human-centered method for work systems design. *IEEE Intelligent Systems*, 17(5), 2002.

[24] Herbert Alexander Simon. *The Sciences of the Artificial*. MIT Press, Cambridge, Mass., 1981.

[25] Danny Weyns and H. Van Dyke Parunak. Special issue on environments for multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 14(1):1–116, February 2007.