Agent-Environment Interaction in MAS - Introduction and Survey

Joonas Kesäniemi and Vagan Terziyan
University of Jyväskylä
Finland

1. Introduction

Although agent and its environment have been inseparable concepts since the beginning of agent related research, there are quite a few opinions what the concept ”environment” actually comprises in the context of multi-agent systems. Environment is often treated either in an implicit or ad hoc way, by referring basically anything outside the boundaries of an individual agent as the environment. Only recently environment has been considered as a first class abstraction with its own clear cut responsibilities apart from agents (Weyns et al., 2005). This progress has been made as part of the agent-oriented software engineering (AOSE) field, which aims at incorporating agent-oriented concepts into the disciplined processes of software engineering.
In the layered view of MAS with environment-based supports by Viroli et al. Viroli et al. (2007) the application level environmental abstractions are supported by specialized platform level infrastructure. Using this infrastructure, the environmental abstraction can interact with each other and with the agents. According to the classic definition of agenthood by Russell and Norvig [4], agents can receive information from the environment through sensors and affect it via actuators. This agent-environment interface will be the focal point of this chapter.
The goal of this chapter is two fold. First of all, we want to give an introduction to the relationship between agent and its environment. This age-old issue is covered from the perspective of individual agent and multi-agent system. Second, we focus on the agent-environment interface and survey the different aspects of agent-environment interaction. Survey emphasizes the engineering side of the multi-agent systems, as we go through the meta-models, methodologies and infrastructures presented in the recent research literature. However, details of any application-specific environmental infrastructures or agent’s inner workings are out of the scope of this paper. We limit ourselves to the action and observation related issues that make it possible for an agent to perceive and affect its environment.
By turning the attention to the design and implementation details of agent-oriented systems, AOSE approach offers results that pave the way for turning agent systems from research prototypes into serious alternatives for mainstream software development. In order for this to happen, it is crucial to understand the different roles that agent related concepts, such as agent, environment and agent-environment interaction, play in the context of multi-agent systems.

2. Role of the environment

”Agents cannot be considered independently of the environment in which they exist and through which they interact” (Müller, 1996)

This chapter elaborates the different aspects of environment in the agent related research literature. According to the survey by Weyns et al (Weyns et al., 2005) one of the main causes of confusion what is actually included to the environment, is brought about by the fact that the environment, as an abstract, logical concept of agent related theories and models, is mixed up with the environment as infrastructure for engineering agent based systems. This of course stems partially from the inherited generality of the term ”environment” and the confusion can be alleviated by proper definitions for every given context. However, since one of the goals of this chapter is to contribute to the general discussion of the role of environment in multi-agent systems, we are going to have to mix it up and cover both theoretical and practical sides of the term. The main distinction is made between the artificial intelligence (AI) and agent-oriented software engineering (AOSE) related uses of the environment in an effort to facilitate the discussion about the relationship between agent and its environment in both MAS theory and systems engineering.

2.1 Agent and its environment

The ”classic” agent research that can be classified under AI discipline has been quite agent centric. The research has concentrated on areas such as agent architectures, agent communication languages and agent coordination models based on direct agent-to-agent communication. In the process, the role of the environment has often remained ambiguous or implicit.
One of the classic definitions of agent-hood by Russell and Norvig (Russell & Norvig, 1995) defines agent as being ”anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors.”, which in other words proposes that environment is basically anything outside the boundaries of an individual agent and that all the interaction in multi-agent system happens between agent and its environment. (Russell & Norvig, 1995) also describes some characteristics of the agent environments:

- Accessible vs. inaccessible: indicates whether agent can sense the complete state of the world or not.
- Deterministic vs. nondeterministic: indicates whether the state changes of the environment are completely determined by its current state or not.
- Episodic vs. non-episodic: indicates whether the agent’s interaction sequences with its environment (or ”episodes”) can be though of as independent or not.
- Static vs. Dynamic: indicates whether the state of the environment can change while the agent deliberates or not.
- Discrete vs. continuous: indicates whether the agent’s interactions with the environment are limited or not.

Back in the nineteen nineties, environment was generally handled in a application specific manner as demonstrated by examples of agent environment in (Russell & Norvig, 1995), which uses chessboard as an example of simple accessible, deterministic, episodic and discrete environment. Chess pawns are modeled as agents.

Another prominent definition from the mid-nineties comes from Maes, who describes agents as "computational systems that inhabit some complex dynamic environment, sense and act autonomously in this environment." (Maes, 1995). The same requirements for agent an being able to sense and act in the environment are present as in the previous definition by Russell and Norwig, but Maes emphasizes the complexity and dynamic nature of the environment.
Agent definition from Hayes-Roth et al. explicitly links the environmental state changes to the actions of the agents by stating that an intelligent agent "continuously performs three functions: perception of dynamic conditions in the environment; action to affect conditions in the environment; and reasoning to interpret perceptions, solve problems, draw inferences and determine actions"(Hayes-Roth, 1995). From the environment's perspective, this basic relationship between agent and its environment is nicely illustrated in the generic environment program (Russell & Norvig, 1995), which also takes in to consideration the dynamics of the environment (Listing 2.1).

Listing 1. Generic environment program (Russell & Norvig, 1995)

```
procedure RUN–ENVIRONMENT(state ,UPDATE–FN, agents , termination)
inputs:
  state , the initial state of the environment
  UPDATE–FN, function to modify the environment
  agents , a set of agents
  termination , a predicate to test when we are done

repeat
  for each agent in agents do
    PERCEPT[agent] <– GET–PERCEPT(agent, state)
  end
  for each agent in agents do
    ACTION[agent] <– PROGRAM[agent](PERCEPT[agent])
  end
  state <– UPDATE–FN(actions ,agents , state)
until termination(state)
```

The environment program first resolves perceptions for every agent based on the current state of the environment. The perceptions generated by agents are then turned into actions by feeding the perceptions back to the agents. Finally the state of the environment is updated using the actions of the agents and the actions resulting from the possible environmental processes running separately from agents.
The environmental processes was also covered in (Ferber & Müller, 1996), which presented a new model for agent-environment interaction in situated multi-agent systems. The details of the model are out of the scope of this chapter, but we want to highlight couple of the main concepts of the model that concern the agent-environment interface. First of all, the model distinguishes between action and the effect caused by that action. Ferber refers to the attempts of an agent to modify the state of its environment as influences. The actual state changes are referred as reactions, which are produced as the combination of influences from all the agents, the current state of the environment and the possible laws or other restrictions imposed by the environment. Another important concept presented in the article, is the decomposition of

the system into dynamics of the environment and the dynamics of the agents situated in that environment, effectively emphasizing the role of the environment as distinct and active entity in MAS.

Clear distinction between agent and environment was further promoted by Parunak, who in (Parunak, 1997) presented a set of general principles for emergent behavior in multi-agent systems inspired by natural systems such as ant colonies and flock of birds. He argues against agent-based functional decomposition of distributed MAS in favor of more multifaceted, yet still simple agents, so that the functions can emerge from the interactions between agents. He claims that functional decompositions appears most natural when the distinction between agents and their environment is overlooked. When it is recognized that environment's own processes mediate agent interactions, it is more difficult to assign agents to arbitrarily conceived functions. Although Parunak was concentrating on agents situated in physical environment, the principles presented in the paper have later been successfully applied to agents inhabiting also virtual environments (Mamei & Zambonelli, 2005).

Odell et al. (Odell, Parunak & Fleischer, 2003) defines the environment as something that "provides the conditions under which the an entity (agent of object) exits". The definition adds an important aspect to the role of the environment by including the management of other entities than just agents to the responsibilities of the environment. In an effort to present a more detailed view of environment in MAS, Odell et al. (Odell, Parunak & Fleischer, 2003) distinguish between three different environments: physical, communication and social. Physical environment provides the laws, rules, constraints, and policies that govern and support the existence of agents and objects that make up the system, just like the environment we humans live in, is constrained for example by the laws of physics. Even though the idea is modeled after natural physical environment, it does not rule out software agents and virtual environments. Communication environment provides 1) principles and processes that govern and support exchange of ideas, knowledge, information and data, and 2) the functions and structures that are used to enhance communication. Social environment is a subset of communication environment in which the agents interact in a coordinated manner. The social environment consists of 1) groups in which the agent participates, 2) roles of the agent, and 3) all the members who play roles in these social groups.

The support required by physical environment as defined in (Odell, Associates, Arbor & Parunak, 2003) would consist of three layers of platform level environmental support: 1) Application support required by the entities running in the environment, such as directory, ontology and security related services, 2) communication and data transfer facilities required by application abstractions, and 3) physical linkage, meaning the hardware such as sensors and actuators needed to affect and sense the physical world. Of course, for purely virtual systems, there is no need for physical linkage. (Odell, Parunak & Fleischer, 2003) lists similar requirements for communication environment, which should provide support for processes such as interaction management, policy enforcement, coordination services and services related to managing group and role based social environment. The article discusses a common processing platform that would provide a foundation upon which agent societies could be built to leverage their application specific environmental requirements. Next section zooms in to this notion of environment as common processing platform for MAS.

2.2 Environment in MAS

(Huhns & Stephens, 1999) considers the role of the environment in multi-agent systems as the computational infrastructure for enabling and ruling interactions. As depicted in the previous

section, this interaction can involve agent-to-agent communication, agent-to-environment actions and interactions induced by environmental processes. From the architectural perspective, infrastructure is usually modeled as a middleware layer providing some domain specific services between the entities implementing the application specific features and the hardware infrastructure.
Designing and implementing complex MAS environments calls for detailed discussion of the concrete features that should be part of the environment for MAS. But before going into environment architectures, let us get a look into different levels of support available from the environment and basic functionalities bestowed upon them.
It was not until the latter half of the first decade of the 21st century, when researchers started to promote environment as a legitimate abstraction of its own. Weyns et al. were the first to put forward the idea of environment as first class abstraction. (Weyns et al., 2006) defines environment as "first-class abstraction that provides the surrounding conditions for agents to exist and that mediates both the interaction among agents and the access to resources". In the context of information system science, first class abstraction is defined as a piece of software that provides an abstraction or information hiding mechanism so that the actual implementation can be changed without requiring changes to the other programs depending on it. Authors argue that the reasons behind this promotion of emphasized role of the environment are related to fact that agents are used as an interface to many services that should not conceptually be assigned to them (Weyns et al., 2006), for example communication and coordination infrastructures. Another important point is that the environmental aspects should be made explicit instead of relying on implicitly stated functionalities of the presumed environment and its ad hod implementations. (Weyns et al., 2006) argues also that by treating both agents and environment as first-class abstractions will contribute the separation of concerns, which in turn can help managing the complexity of building real-world applications. In an effort to get the role of the environment up to date, (Weyns et al., 2006) presents the following list of functionalities that the MAS environment should support.

- **Provide structure for the MAS:** Environment maintains relationships between entities in the environment and enforces rules to which the relationships must comply. Different forms of structuring include physical, communication and social structures.
- **Embed resources and services:** Environment acts as the source for resources and services.
- **Maintain dynamics:** This refers to the active nature of the environment.
- **Be locally observable:** Agents should be able to observe the state of environment. The observable environment might be limited by the current spatial, social or communication context of the agent. Providing observability can be seen as the environmental counter-part to the agent's ability to perceive its environment.
- **Be locally accessible:** Naturally, the agents must be able to act in their environment, but their actions might be constraint by their current context.
- **Define and govern environmental rules:** By enforcing the rules, whether they are restrictions imposed by the application domain at hand or arbitrary laws defined by designer of the system, environment attempts to keep the system in consistent state.

The degree of functionality provided by the environment can be categorized using three level model by Weyns et al. (Weyns et al., 2006) (see figure 1).

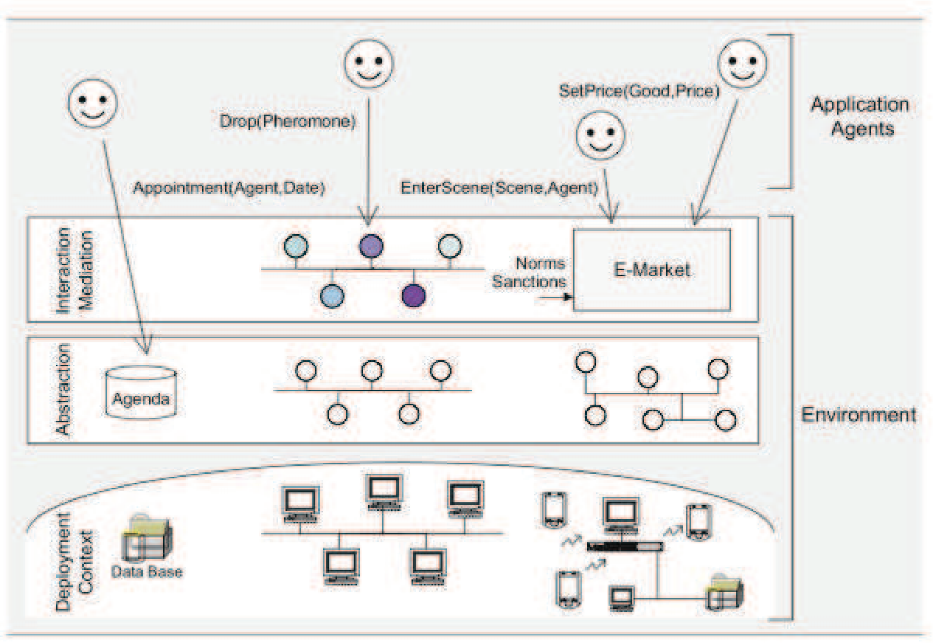

Fig. 1. Degrees of environmental support (Weyns et al., 2006)

The basic level of support is referred as the deployment context of MAS. It includes any hardware or software resources outside the boundaries of MAS. Examples of such resources are hardware sensors, databases and web services. Proving access to deployment context is one of the fundamental properties of the environment, but interacting directly with the deployment context forces agents to work with low-level details of both software and hardware environment and is not well suited for situations when the MAS is deployed in an unpredictable and highly dynamic environment. At the second level, the environment provides agents with abstractions that shield them from the low-level details of the deployment context. For example, instead of interacting with the humidity sensor directly using byte-level interface of the embedded software, agent can use a service to request a higher level representation of the state of the sensor. The abstractions are usually supported by some middleware software, which hides the possible complexity of the underlying deployment context. At the third level, the environment provides services for mediated interaction between agents. This requires environment to become an active entity that can change its state without any agent involvement. Examples of interaction-mediation level services are infrastructures for computational fields (Mamei & Zambonelli, 2005) and electronic institutions (Esteva et al., 2004).

We use FIPA (*FIPA: Foundation for Intelligent Physical Agents.*, n.d.) standards as a starting point for the more fine grained view of the responsibilities of the environmental infrastructure. The agent management reference model defines a logical MAS component called agent platform, which embeds agents as well as infrastructural components for agent communication, deployment and discovery. Figure 2 shows the reference model of the FIPA architecture.

Agent platform is responsible for running the agents. Agent management system (AMS) controls the life-cycle of agents in the system and takes care of life-cycle management services. It also maintains a directory of agent names and provides white-paper services for other agents. Message Transport System is responsible for providing ACL-based facilities for direct

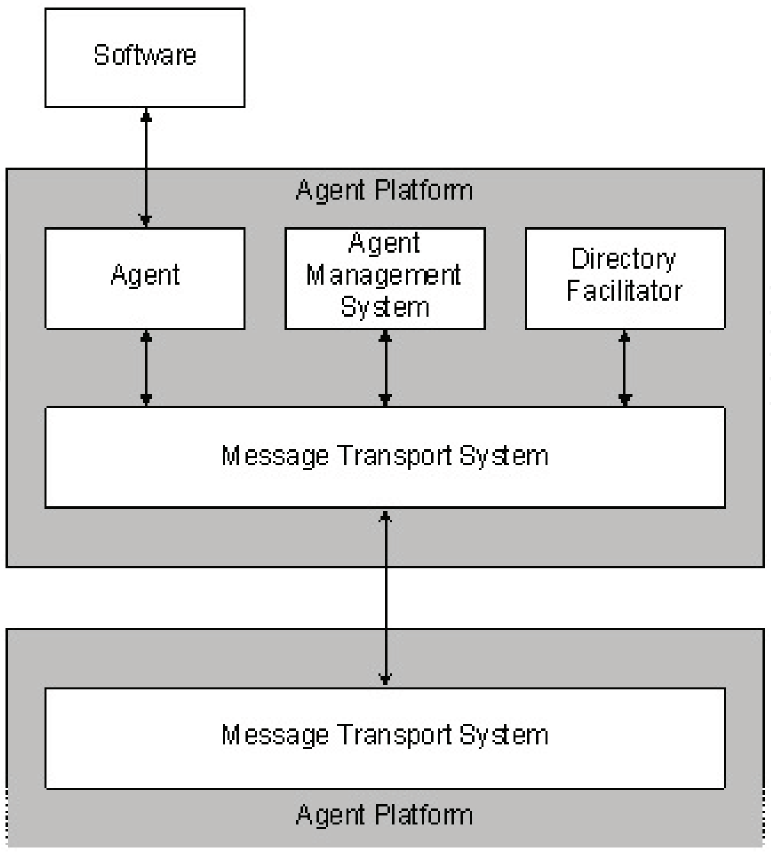

Fig. 2. FIPA abstract architecture (*FIPA: Foundation for Intelligent Physical Agents.*, n.d.)

agent-to-agent communication. Directory facilitator is an optional component that can used to discover agents based on their registered services.
FIPA reference model only covers the interaction between agents. However, reducing environment to a transfer medium of direct interaction underplays the potential of the environment as an active support for communication. Agents interact also with other entities in the environment using sensors and actuators. For example an agent can interact with a file system to sense any new files added to a certain directory, or it can call a web service to store some data to the database. Since there is no explicit abstraction for environment in FIPA reference model, agent-environment interaction is out of its scope.
Similarly as in (Weyns et al., 2006) where environment is divided into interaction-mediation and abstraction layers based on the degree of functionality provided, the survey conducted by Platon et al about the mechanisms for MAS environment (Platon et al., 2006) argues that infrastructures for environment can be classified into mechanisms that provide means for agents to interact with each other and mechanisms for agents to acquire and manage resources and contextual information. Two main classes of mechanisms are further divided in the following way: They further divide the two main classes of mechanisms in the following way:

- Interaction mediation
  - Environment-mediated interaction channels
  - Synchronization mechanisms
  - Overlay networks

- Resource and context management
  - Resources and context manager
  - Notification of contextual events
  - Overlay data structures

Different types are described using the same set of characteristics for easy comparison and evaluation. The examples given for each type range from quite low level infrastructures such as distributed hash tables (Rowstron & Druschel, 2001) and synchronization mechanisms to complete systems implementing digital pheromones, exemplifying the different layers of environmental infrastructures. For example, the pheromone infrastructure could depend on distributed hash table as its data management implementation. The higher the level of abstraction provided by the infrastructure is, the greater the potential benefits of adopting or reusing it are.

In the context of designing self-organizing emergent applications, to which many multi-agent systems can be included in, (De Wolf & Holvoet, 2006) catalogues mechanisms for de-centralized coordination using the design patterns as the basis for classification. Design pattern is a software engineering concept that captures the best practices and known solutions in a structured way. Patterns usually follow a common structure known to the mainstream software engineering, which includes sections such as context/applicability, problem/intent, solution, related patterns, examples and known uses (Meszaros & Doble, 1996). Presenting environmental mechanisms as designs patterns makes it easy to find, evaluate and compare possible solutions. It also integrates MAS infrastructures to the mainstream software engineering practices.

According to the 3-layer model for MAS by Weyns et al. (Weyns et al., 2005) both agents and environmental mechanisms are part of the MAS application layer . Other main layers include execution platform, which is composed of generic middleware infrastructure and virtual machines that run on top of operating system, and physical infrastructure that represent the hardware and network infrastructure that runs and connects the nodes hosting the MAS (see figure 3).

Application environment embeds agents that contain the application specific logic. MAS framework sub-layer includes infrastructures that offer agents high level programming abstractions for communication and agent-environment interaction. Low level infrastructures, such as distributed hash tables or synchronization mechanisms are then considered as part of the execution platform.

The 3-layer view for MAS (Weyns et al., 2005) successfully represents the role of the environment in the abstract architecture of MAS, but fails to take into account the possible middleware nature of the MAS infrastructures. The model was refined in (Viroli et al., 2007) by a) moving the environmental infrastructures down to the execution platform level, leaving only the abstractions they provide as part of the MAS application, and b) dividing the execution platform into MAS middleware and deployment context layers (see figure 4).

The logical view of the environment presented in (Viroli et al., 2007) emphasizes the role of the environment infrastructures as not necessarily application specific mechanisms built from scratch, but reusable and time-tested mechanisms selected to address certain problem in a more general domain of MAS environment. Their model also clearly separates the environmental abstractions and their interfaces from the infrastructure on top of which they are run. In other words, environmental abstractions are treated as first-class entities. The MAS middleware layer can consist of multiple agent and environment infrastructures. For example

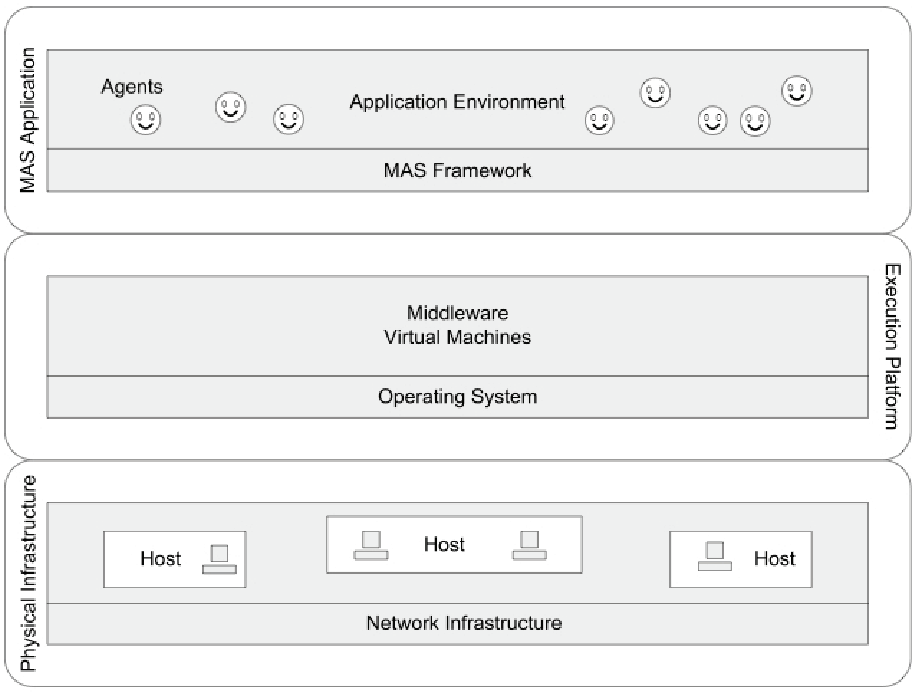

Fig. 3. 3-layer model for MAS (Weyns et al., 2005)

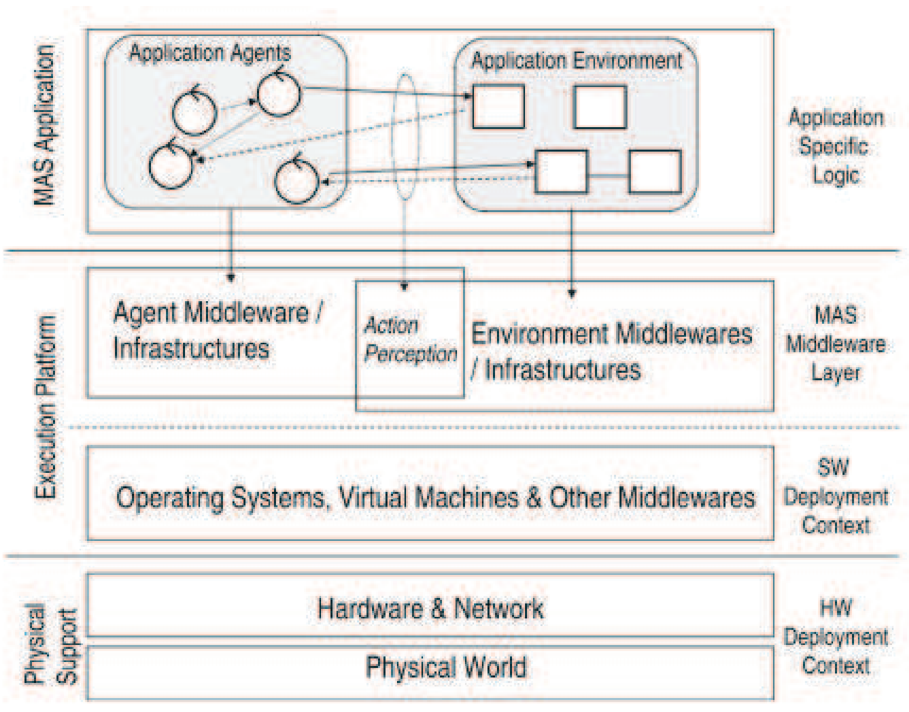

Fig. 4. MAS layers with environmental support (Viroli et al., 2007)

there could be agents running on JADE [1] and Cougaar [2] agent platforms that need to use tuple spaces, digital pheromones and electronic institutions to achieve their goals.

[1] http://jade.tilab.com/
[2] http://www.cougaar.org/

In figure 4, the arrows going from agents to environmental abstractions represent actions and the dashed lines in the opposite direction represent perceptions. We will focus on this agent-environment interface in the next section.

The resources modeled as part of the application environment are considered as passive and reactive. They don't exhibit the characteristics associated with agents, such as autonomous and pro-active behavior or social abilities (Omicini et al., 2008). If more intelligent environment is needed, nothing prevents us from creating another layer of application environment using special type of agents. These infrastructural agents can be organized as an intelligent and self-organizing environmental layer and can have for example some control over the population of the application agents. Infrastructural agents could be for example in charge of managing self-configuration of the whole MAS. The idea of agents as part of the environment is demonstrated in (Mili & Steiner, 2008), which presents a model of Agent-Environment Systems (AES) based on layered view of agents. (Mili & Steiner, 2008) defines AES as "as a system composed of a) a set of interacting social-agents, b) a distinct open environment in which these agents are situated, and c) a mechanism for social-agent/ environment interactions.". Open environment is split into cells that provide manageable division of large open MAS and social-agent/environment interaction is executed via specialized cell controller agents that form a middle layer between application environment and application agents (see figure 5).

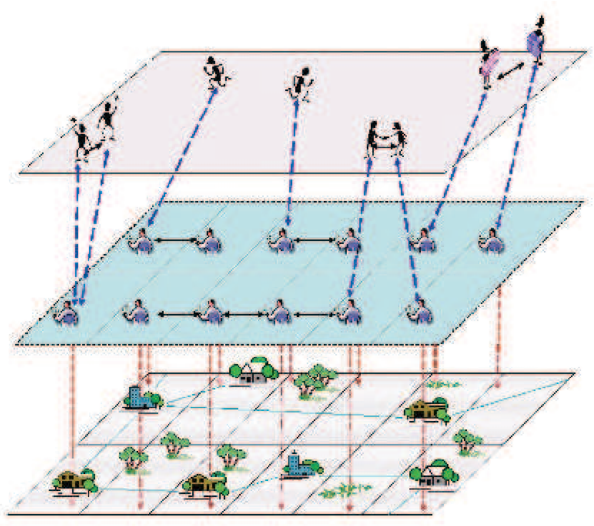

Fig. 5. AES layers (Mili & Steiner, 2008)

Social-agent can observe the state of the environment by querying the controller agents for perceptions. Controllers manage access to the environmental resources and provide synchronization services. Controller agents can also inform other controllers of any changes that may effect their cells or push the latest state of the environment to their local agents.

There is a need to model, develop and research infrastructures for both agents and environment. FIPA abstract architecture and related specifications are a popular way of modeling the agent platform, but they only cover the agent side of the execution context and miss the benefits of explicitly modeled environment. The environmental infrastructure can be seen as service to the agents that provides abstractions for engineering the environment. One of the main challenges of engineering service-oriented environments is the integration of services Weyns et al. (2007). Horizontal integration deals with problem of integrating services at the same level of abstraction by allowing flexible composition of available services based on the application specific requirements. This could mean for example allowing environmental process running on infrastructure X to observe the state of the resources supported by infrastructure Y. Vertical integration concerns both the integration of domain-specific services upwards with the agents and downwards with the more general services. The former is critical to the interoperability between agent and its environment and the latter to the efficient engineering of infrastructures for environment (Weyns et al., 2007).

3. Agent-environment interaction

Based on the research presented in the previous section, it should be clear that environment can play an important part in MAS interaction when its role is extended beyond mere transfer medium for direct agent-to-agent messaging. The models and mechanisms for environmental support for indirect interaction extend the traditional means and models of interaction in MAS and provide valuable ways of managing complexity. On the other hand, infrastructural or middleware approach to engineering agent-based systems contributes to the creation of generic and reusable components for implementing MAS environments. Infrastructures use domain specific abstractions such as digital pheromones, co-fields or organizational concepts to tackle interaction problems. Each infrastructure usually also provides their own unique interface for agents to use. Complex and open MAS environments can utilize multiple infrastructures, all coming from different sources with their own usage interfaces. This infrastructural complexity of the environment leads to more complex agents, who have to be able to adapt to different ways of sensing and acting in the environment.

We argue, that it is crucial to keep the interface between agent and its environment as simple as possible, without sacrificing the benefits of the environment-mediated interaction. In an ideal situation, agent could sense and act in the environment through a single interface, which would provide access to the whole environment and the major integration task would be moved from the agent side to the side of the environment. This is a reasonable assumption, because although it is possible to think of environment as an unbounded, in practice it is implemented as a set of situated interaction spaces, confined by real-world or artificial restrictions. In other words, the agents explicitly enter and leave the environment. For example, for agents inhabiting a physical object, such as a mobile phone, entering an environment might mean that the object is carried inside a certain smart room. Virtual agent on the other hand might be required to present valid credentials before gaining access to a shared virtual environment. So, even though the environment could be made up of components from different sources it is usually maintained by a single stakeholder.

In contrast to the models and infrastructures presented in the earlier sections, this chapter raises the level of abstraction to generic perception and actions. So, instead of discussing about depositing and observing digital pheromones, we look into the problem of how agents can observe the environment in a generic way and discover, choose and perform any generic action that will help them to reach their goals. This must also include the modeling of the environment upon which the action are executed. The goal of this section is to provide a survey of the current research literature concerning the theory, design and implementation aspects of generic interface for agent-environment interaction.

The first sub-section covers artifacts as the unifying concepts of agent-environment interaction. Second subsection takes an AOSE approach and discusses the impact of explicit environment to the agent-based engineering methodologies. In the third subsection we discuss the context of agent actions. Fourth subsection present two infrastructures available that support the high level modeling concepts for resources and observation. Finally the last subsection discusses the semantic description of the environment.

3.1 Artifacts as unifying abstraction for engineering environment-mediated interaction

One of the most general approaches to modeling environment-mediated interaction comes from Ricci at al. (Ricci et al., 2006)in the form of artifacts as the functionality-oriented building block of the MAS environment. Artifact is used as a first-class abstraction to represent everything in MAS context that is not suitably modeled as an agent i.e. everything that is not characterized as goal- or task-oriented. The idea is that the same concept of artifact is used through out the whole process of developing MAS from modeling to implementation and all the way to deployment, where the artifacts are supported by suitable infrastructures. (Ricci et al., 2006) defines artifact as "a computational device populating agents' environment, designed to provide some kind of function or service, to be used by agents - either individually or collectively - to achieve their goals and to support their tasks". Multi-agent system is then defined as a computational system that consists of agents and artifacts (Omicini et al., 2008). Agents represent the pro-active and autonomous components of the system whereas artifacts are passive and reactive.
Figure 6 shows the abstract representation of an artifact. Artifact is described by its function, usage interface and operating instructions. The definition of artifact says nothing about its internals. Function describes the intended use of the artifact provided by its designer, but it doesn't necessarily determine the actual usage of the artifact. Usage interface is defined as the set of operations, observable states and events provided by the artifact.

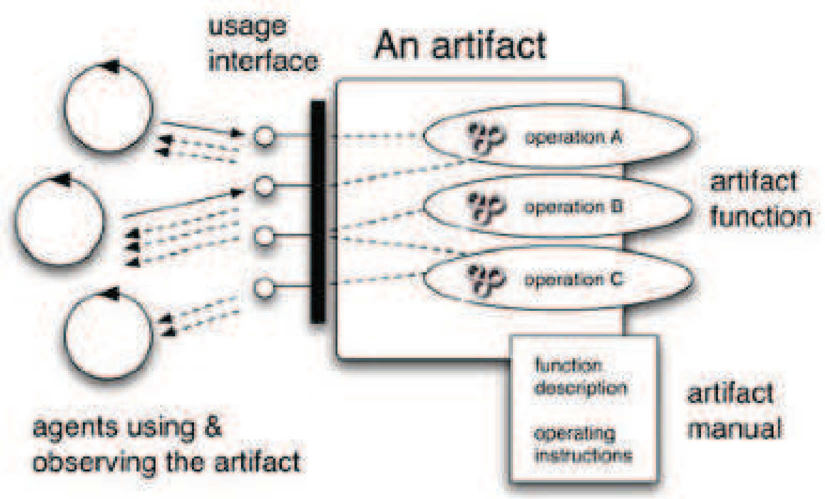

Fig. 6. Abstract representation of the artifact. (Ricci et al., 2007b)

Observable state consists of dynamic and persistent attributes that belong to the artifact and can be observed without interacting with the actual artifact. Observable events on the other hand are non-persistent information that carry information about the evolution of the artifact's observable state (Ricci et al., 2008).
(Ricci et al., 2006) presents some of the basic artifact types common in the context of MAS engineering: coordination artifacts, boundary artifacts and resource artifacts. Another

categorization is given in (Ricci et al., 2007b), which categorizes artifacts into resources and tools. According to (Ricci et al., 2007b) resources are the primary source and target of the agent activities, whereas the tools are used as means to achieve some goal. In other words, agent can interact directly with the resource or use an appropriate tool. (Viroli et al., 2005) divides artifacts into individual, social and resource artifacts as depicted in figure 7.

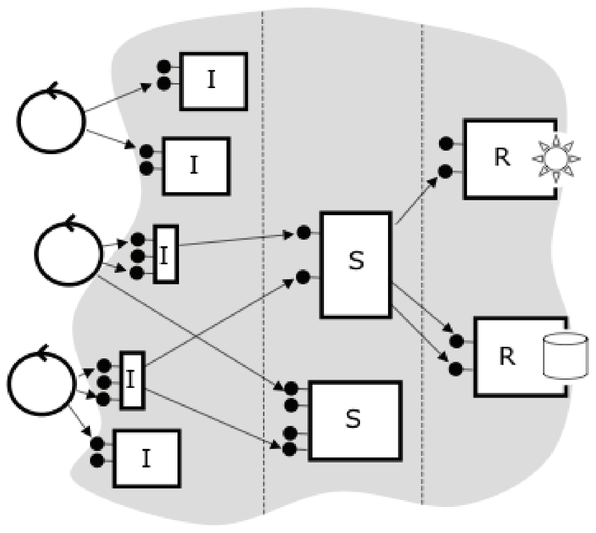

Fig. 7. Artifacts and the layered view of application environment. (Viroli et al., 2005)

Individual artifact are used by only one agent. They can be seen as the personal artifacts of the agent and can implement for example a private extension of agent's memory. As the name implies, social artifacts are used by two or more agents. They mediate interaction between agents and their functionality is usually geared towards the environmental goals instead of the goals of an individual agent. Examples of social artifacts include coordination services and shared repositories. Finally, resource artifacts are artifacts that wrap external resources such as legacy systems and physical sensors (Viroli et al., 2005). As part of the effort to further generalize the artifact based approach, Agents & Artifacts meta-model (Omicini et al., 2008) makes a distinction between artifacts that are used by cognitive and non-cognitive agents. Basically, a cognitive artifact must include the function, usage and operation instructions described above, whereas a basic artifact can be deployed without them.
Ricci et al. identify three main aspects of agent-artifacts relationship: 1) selection, 2) use and, 3) construction and manipulation (Ricci et al., 2006). The selection process relies on the observability of artifacts. Agent must be able to inspect the state and behavior of an artifact in order to be able to select the appropriate artifact for the task at hand. Agent uses artifacts to support their activities. Artifacts are situated in the environment, meaning that they are both the source and target of change in the environment. Being situated, the function of an artifact or what the artifact does when used by an agent is defined as the change it produces to the environment. Agent uses artifact through an interface, which is defined as a collection of operations. Operation changes the state of the artifact, activates it and produces the defined effects to the environment. Finally, construction and manipulation refers to linkability and malleability of the artifacts respectively. Linkability allows agents to combine them at runtime to create new and more complicated artifacts. Malleability refers to the characteristics of

artifacts that makes it possible for agent to change the behavior of an artifact at runtime in order to adapt it to the dynamic requirements of the open environment.
According to the Agents & Artifacts meta-model (Omicini et al., 2008), the agent actions in MAS can be classified in three categories: 1) Internal actions, which are actions that are part of the inner workings of an agent, 2) communicative actions, which are related to sending and receiving of ACL messages from other agents and 3) pragmatic actions that involve interaction with the environment via artifacts. Pragmatic actions cover the observation, usage, construction and modification of artifacts. A&A meta-model does not however provide any means for explicit modeling of the action itself, which it is left to the concrete models.
Artifacts represent a very general model for engineering MAS. The models presented in the following sections share many of the characteristics of artifacts and provide extra abstractions for more concrete modeling tasks.

3.2 Agent-environment interaction in AOSE methodologies

Although some of the modern AOSE methodologies emphasize the role of the environment, they rarely provide an explicit meta-model or concepts for modeling the agent-environment interaction for that matter. Even if the methodology would include tools for environmental modeling, the gap between design and implementation remains significant if the target deployment infrastructures are not taken into account.

3.2.1 GAIA

(Zambonelli et al., 2003) presents an updated version of the GAIA methodology with environment model, where the environment is modeled as set of abstract computational resources made available to agents for sensing, effecting and consuming. A simple model for agent-environment interaction is given as a list of resources each associated with the type of actions (read, change, and remove) agents can perform on it and possible informal textual descriptions of the actions. GAIA was one of the first methodology to include environmental modeling, but the proposed approach is too limited for any detailed discussion of the environmental responsibilities.

3.2.2 Agent Environment Interaction model

Agent Environment Interaction (AEI) (DeLoach & Valenzuela, 2007) is an extension to the O-MaSE methodology (Deloach, 2006) and it concentrates on the agent-environment interaction. The proposed model uses concepts from O-MaSE. AEI has three main components: Capability model, Environment model and Interactions between capabilities and the environment. Figure 8 depicts the main concepts of the model and their relationships.
Environment model is designed to support the modeling of active environment through Process entities. The environment itself is composed of objects, which can be connected with relations. Agents are considered as a special type of objects that can posses capabilities. Capability is defined as an entity that can perform one or more actions. Capabilities can be defined at different levels of abstraction based on the requirements of the application. AEI model also supports the creation of new capabilities as a composition of existing capabilities. This basically means that the new capability has access to all the actions defined in the capabilities it is composed of. Action is used to represent the actual sensor of actuator. It is defined via single operation that is used to invoke the action. The operation can be equipped with pre-conditions that must be met in order to execute it. There can also be post-conditions

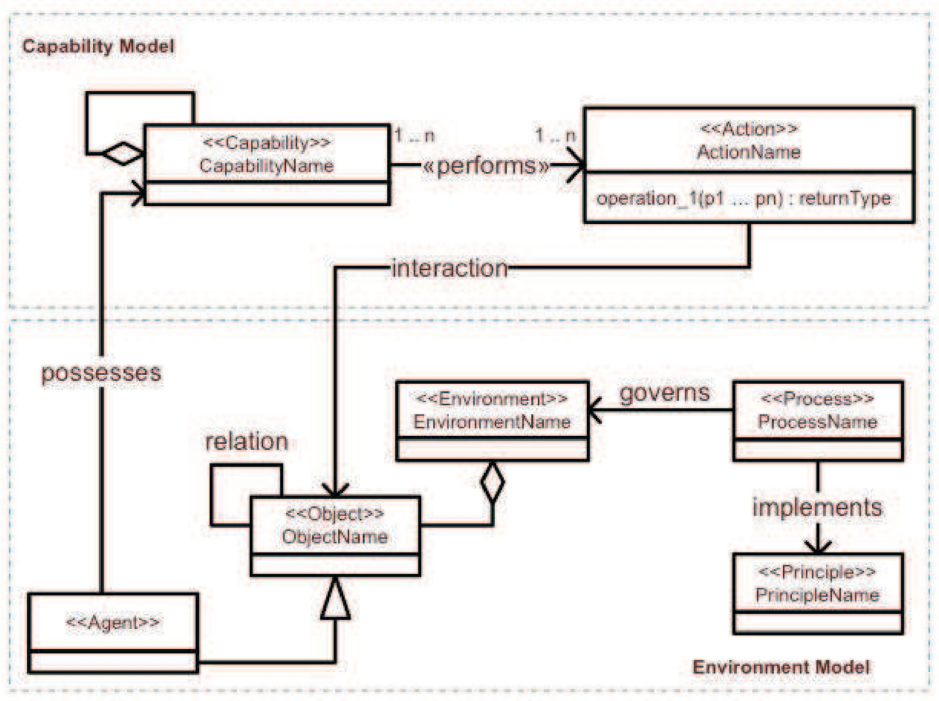

Fig. 8. Agent-Environment Interaction model. (DeLoach & Valenzuela, 2007)

that describe the desired state of the environment after operation execution. The dynamics of the environment can affect the state of the environment so that the post-conditions might not hold.

The viewpoint of the capability model is one of agent behavior as opposed to the artifact behavior emphasized by A&A meta-model. Instead of using environmental objects directly, agent is said to perform actions/operations on the environment as a whole, which may impact the state of one or more objects in the environment. There is no mention of general observability of the environment. Whether or not to allow agents to observe certain state of the environment is modeled as part of the actions. Of course, one could create a capability called Observe through which the agents could perform generic observation related actions with operations such as *getObservableObjectsByType(objectType)* and *getCurrentObservableState(objectID)*. Lack of explicitly defined observability also affects the definition of operations. In AEI every operation has a return value in contrast to the operation in artifact, which makes its new state available through asynchronously manifested observable events.

3.2.3 SODA

SODA (Societies in Open and Distributed Agent spaces) is an agent-oriented methodology that focuses on the inter-agent issues related to analysis and design of open agent-based systems (Omicini, 2001). SODA is one the few agent-based methodologies that deals with MAS environment explicitly. In the analysis phase resource model is created by expressing the environment as a set of services, each associated with a certain abstract resource. Resource model also includes abstract definitions for permissions associated with each role or group. The information flow between a service and its user is captured as part of the interaction model in the form of interaction protocols. Later, in the design phase, resources are mapped to infrastructural classes or components in order create concrete environment model. Each

component is associated with an interface, which is defined by its associated interaction protocol.
Original SODA methodology has been later extended with the notion of artifacts (see previous section) (Molesini et al., 2005). Artifacts in SODA are mainly considered at the design phase, although the authors do acknowledge their role in the analysis phase and especially to the interaction model. In the context of resource/environmental models the introduction of artifacts has two major consequences (Molesini et al., 2005). Replacing concrete resources with resource artifacts, which embody external resources, allows designers of MAS environment to raise the description of those resources up to the agent cognitive level. Another observation derived from the artifacts is the need for topological model. The authors argue that in order for the artifact to be applicable abstraction through out the whole engineering process from design to deployment, it is important to give the engineers of MAS environment the ability to model the topology of the system already at the design phase (Molesini et al., 2005). The interaction model is modified to coincide with the interaction categories of A&A meta-model (Omicini et al., 2008) by dividing interaction protocols into role interaction protocols that deal with the agent-to-agent communication and resource interaction protocols that are related to artifact usage.
MeNSA project [3] is an interesting initiative that has taken synthesizing approach at creating a new methodology by integrating methodologies under a new shared meta-model in order to produce as versatile meta-model as possible. (Dalpiaz et al., 2008) present the first result towards that goal by introducing the first version of MeNSA meta-model that combines strongest aspects from GAIA (Zambonelli et al., 2003), Tropos (Bresciani et al., 2004), SODA (Omicini, 2001) and PASSI (Cossentino, 2005) methodologies. Not surprisingly, MeNSA has adopted the environment model from SODA.

3.3 Environment as a context for agent actions

Agent actions are a fundamental part of the MAS and thus usually covered by AOSE methodologies either implicitly or explicitly. Action is always performed in some context and there should be a flexible way for modeling preconditions for the action in that context. In the current approaches, the context model for action usually does not take into consideration the state of the non-social part of the environment. EAI model (DeLoach & Valenzuela, 2007) has something in that direction, but based on the article, one can't conclude anything definitive about the scope of possible context for the preconditions of operations. In SODA (Omicini, 2001), the permissions regarding actions on resources are based on the role or group that the agent belongs to in the organization. This is enough for environments that don't contain many interconnected resources that might affect each others availability. For complex environments there might be requirement to include for example information about the topology of the environmental resources to context against which the preconditions of the action are evaluated.
The Environment as Active Support if Interaction (EASI) model by Saunier et al (Saunier et al., 2007) defines the context of MAS as the observable properties of all the entities in MAS. Agent's actions are therefore guarded by contextual conditions that take in to account the collective status of the whole MAS. Of course, this approach gets complicated when the environment is distributed, but then the contextual information can be restricted to the local state of the MAS. Similarly to EASI, Agent Observable Environment (Kesäniemi et al., 2009)

[3] http://www.mensa-project.org

infrastructure presented in the next section is designed to use the so called observable state of the local environment as the context for observation rules.

3.4 Infrastructural support for agent-environment interface

In the previous sections we have gone through methodologies and meta-models proposed for engineering and modeling environment and agent-environment interaction. In order to move the system from design to deployment, the design time models have to be transformed into an implementation. It is always at least theoretically possible to build the whole implementation from scratch but in practice that is rarely an acceptable route to take for obvious resource related reasons. That is why everything is usually built on top existing software, frameworks, libraries and infrastructures. MAS infrastructures have traditionally focused on supporting abstractions related to building different type of agents or facilitating agent-to-agent communication, while the environment part of the MAS, if included in the designs at all, has been left for ad hoc implementations or the infrastructural support has been for relatively low level functionality.
The recent promotion of environment as a first-class entity (Weyns et al., 2006) has fueled the development of more generic, adaptable and reusable infrastructures for environments. But the use of high level environmental infrastructures doesn't remove the fact that there is usually a conceptual gap between the abstract concepts used in the meta-model of the methodology and concepts provided by the infrastructure. The cause for this gap is argued to be in the different perspectives taken by the creators of methodologies and infrastructures (Dalpiaz et al., 2008). AOSE methodologies usually follow the top-down approach, moving from real life problems to architectures, whereas the infrastructure developers start from existing programming paradigms and work their way up to higher level programming constructs. In the end, there is usually mismatch between the approaches and some transformations or ”glue” concepts must be developed to bridge the gap. Decreasing the gap would alleviate the challenges related to the vertical integration of domain specific infrastructures upwards to agents. In this section we focus on the particular type of environmental infrastructure that tries to minimize that gap in the context of agent-environment interaction.

3.4.1 CArtAgO

CArtAgO or Common "Artifacts for Agents" Open framework, is a Java based open source software [4] created in the university of Bologna. It provides development and run-time support for artifact-based (see section 3.1) computational environments (Ricci et al., 2007a). The three main components of the framework are 1) API for designers and implementors of the environment for defining artifact types, 2) API for agents for interacting with the artifacts, and 3) infrastructure for run-time support for artifact based working environments. CArtAgO is an environmental infrastructure and it is designed to be integrated with existing agent frameworks.
Figure 9 shows the abstract architecture of CArtAgO. In addition to artifacts and workspaces, two concepts coming from A&A meta-model, CArtAgO introduces a new concept called agent body, which works as the interface between agent and its environment. It is very similar to the softbody introduced by Platon et al. in (Platon et al., 2005). Agent body is created for every agent that inhabits CArtAgO environment and it contains the effectors and sensors of the agent. Agent controls its own body and can dynamically link and unlink different kinds

[4] http://sourceforge.net/projects/cartago/

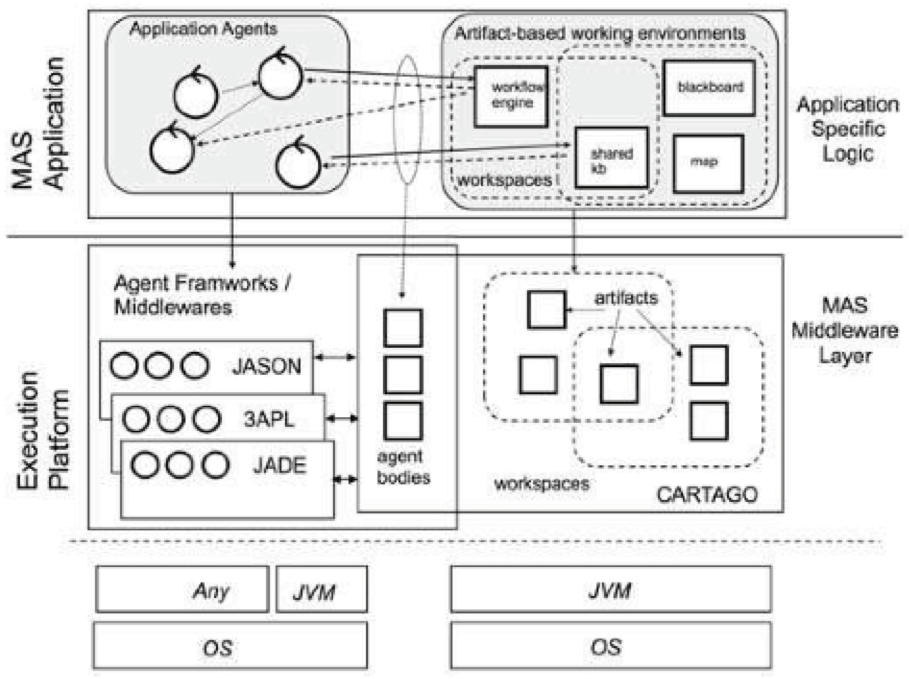

Fig. 9. Abstract representation of a MAS application using CArtAgO. (Ricci et al., 2007b)

of sensors to it. Agent-environment interaction happens through the construction, selection and use of artifacts and perception of observable event as described in section 3.1.
The agent side of the API, i.e. the way agent can interact with its environment, deals with the construction, disposal, selection, use and inspection of artifacts as well as sensor and workspace related management tasks. For example, agent can execute specific operation on a given artifact by using the action *execOP(ArtifactID, OperationName, Args, SensorID)*. The last two parameters are optional. Args refers to the parameters of the operation and SensorID to the sensor that is going to be used to collect the possible observable event generated by the action. On the artifact side the available actions deal with the generation of observable events and management of operations and observable state of the artifact. Detailed description of the API is available in (Ricci et al., 2007a).

3.4.2 Agent observable environment

Agent Observable Environment (AOE) is an infrastructure designed to support the observation process of an agent through observable softbodies (Kesäniemi et al., 2009). It separates the common Observable Environment (OE) from the domain-specific infrastructures. The idea is that all the MAS infrastructures use the same OE making it the principal source and target of observation for the whole MAS. In other words, AOE works as a integration middleware between various agent and environment infrastructures. It was designed with the notion of open MAS in mind, where both the agent population and services and resources of the environment can be very dynamic. In a dynamic environment, AOE would work as the lowest common denominator between the heterogeneous entities and facilitate the indirect interactions between them. It should be noted that AOE only accounts for observation side of the perception/action interface between agent and its environment.
Figure 10 shows the logical architecture of AOE using the layered presentation of MAS adapted from (Viroli et al., 2007). The observable part of the environment is modeled as a

separate entity called Observable Environment, which contains soft-bodies and observation rules. Every soft-body is connected to either an agent or some environmental resource.

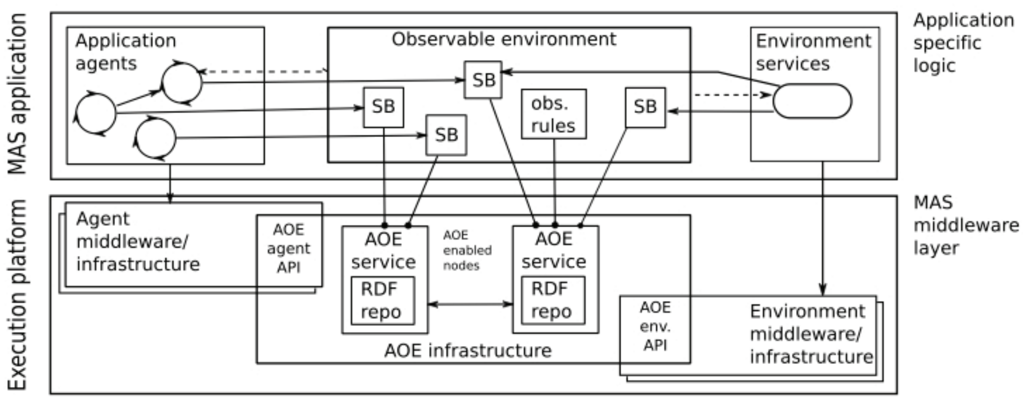

Fig. 10. Logical architecture of the AOE. (Kesäniemi et al., 2009)

Soft-body is further divided into owner and environment managed parts. The division means that the entity to which the soft-body belongs to has only partial control over its observable state. This is different from the agent body in CArtAgO, which is completely under agent's control. AOE supports both static and dynamic observation (Viroli et al., 2001). The result of a static observation is a snapshot of the observable state of the environment presented as observable items, whereas dynamic observation allows agent to observe the evolution of the state through observable events. In addition to agent initiated static or dynamic observation, AOE is able ”push” observable items or events to the agents without explicit request from the agent. This behavior takes the environment controlled awareness provided by the EASI (Saunier & Balbo, 2007) even further and is considered as a potential source for emergent behavior in MAS. Observability of the soft-body can be restricted using observation rules. There are two kind rules: ones originating from the environments side and ones set up by the agents. Environmental rules, or ”laws of nature”, take precedence over rules set by the agent.

3.5 Semantic description of environment

All the approaches covered in the paper so far, have at some level conveyed the need to provide formal, machine processable descriptions of the environment to the software agents. Actual models for such descriptions have been out of their scope and the task has been delegated as part of the development process of a system. This easily leads to ad hoc and application specific descriptions, in the same way as the early MAS environments were implemented on a application to application basis. One way of fostering interoperability between diverse sources of environmental data is to use upper level core ontology (Doerr et al., 2003) from which the infrastructure specific concepts can be extended or to which they can be mapped or transformed. The use of ontologies also allows agents to reason about the state and behavior of their observable environment. This is especially important in open and dynamic environments where the agent might not have any prior knowledge about the environment before entering it.
Descriptions of environmental resources and services are an integral part of the A&A meta-model with the so called cognitive artifacts (Omicini et al., 2008). Cognitive artifact must include function, usage and operation instructions in some machine processable form. Agent Observable Environment can also be greatly enhanced with the use of ontologies. Since

AOE uses RDF [5] as the representation of observable environment it is easy to extends it with ontologies, which can be used to enable semantic observation capabilities (Kesäniemi et al., 2009). Acay et al. coined the term extrospection in (Acay et al., 2007), which refers to a sort of cognitive situatedness in agent's ability to discover, select, use and reason about environmental resources based on their formal semantic descriptions. They argue that the combination of artifact and its usage manual, or cognitive artifact, would benefit the MAS development by making it possible for agents to complete their design at run-time through the process of discovery, use and reuse of components and allowing domain independent meta-level reasoning to be built into agents (Acay et al., 2009). Next section describes their ontology based approach to environmental modeling.

3.5.1 OWL-T

OWL-T (OWL Tool) is an ontology for describing infrastructural components (Acay et al., 2007). The approach taken in the information modeling is pragmatic as opposed to epistemic, meaning that instead of trying to classify and label thing and relations between them, the goal of OWL-T has been define the functional nature of an object independent of its physical properties. For example both paddle and a sail affords to moving a boat on the water although they are not that similar in their physical properties. As the name implies, OWL-T uses Web Ontology Language (OWL) [6] or more precisely the OWL-DL variant of OWL, as its ontology language.

OWL-T can be divided into five mutually exclusive sub-models: ActionModel, ObjectModel, OrganizationModel, AgentModel, and AbstractConcept. We will shortly describe the ones that are most relevant to the topic at hand, namely the ActivityModel, ObjectModel, and AbstractConcept models. The AgentModel is out of the scope of this chapter because it deals with the inner workings of an agent.

Objects in OWL-T are divided into *Artifacts* and *Tools*. Tool is defined by its *PhysicalProperty* and *IdealProperty*. Former supports the spatial and shape characteristics of any object and the latter refers to the functionality of the *Tool* or in other words in what kind of activity it is useful. The usage interface is formalized as an *Interface*, which represent the actions supported for a given *Role*. OWL-T defines three types of agent behavior: *Process*, *Action* and *Activity*. *Process* in an atomic, reactive behavior whereas *Action* represent goal-oriented and non-atomic agent behavior. Completion of an action might require sequence of *Processes* to be executed before completion. The order of *Processes* required by the an *Action* is defined using *ProcessModel*. *Action* takes an *Artifact* as an input parameter and transforms it into output. *Process* can then be seen as working at the operation level of an *Artifact*. Finally, an *Activity* is used to encapsulate actions of multiple agents into to a group activity with a common, system level goal where every agent plays a certain role. Full description of the ontology and related formal semantic and pragmatic for it can be found in (Acay et al., 2007).

4. Conclusions

In this chapter, we provided an introduction to the different roles of environment in agent-based systems. We also presented a survey of different aspects of agent-environment interaction from the perspective of agent-oriented software engineering. The promotion

[5] http://www.w3.org/RDF/
[6] http://www.w3.org/TR/owlref/.

of environment as a first-class abstraction has led to research results in various fields of agent-oriented software engineering ranging from new or extended methodologies that include explicit environmental models to general infrastructures that can support application-independent concepts for building and integrating environments for MAS. There has also been efforts towards semantic description of infrastructural components.

At a very abstract level, agent-environment interaction can be reduced to agent actions performed via sensors and actuators and it basically has three ways of using them. First, it can use the sensors/actuators that are part of its own "body". The features of the agent's body can be static and decided by the designer of the agent, or they can be dynamically modifiable at run-time by the agent itself. Second, if the agent does not possess some required capability, it can try to ask some other agent to provide it. Third option is to use the resources available in the environment. The discovery and usage of environmental resources are the cornerstones or agent-environment interaction and are therefore covered at some level by all the surveyed models and infrastructures. One interesting extension that is not taken into account, is the possibility for agents to deposit their own actuators and sensors to the environment for others to use so that they would be available even if the agent that created them is no longer available. For example, a mobile agent could place a specialized messaging resource to the environment that would allow other agents and resources to interact with some previously unknown external system. In all of the three cases, sensors and actuators can be seen as a sort of a service, either provided by the agent's body itself, other agents or the environment. This kind of sensor/actuator as a service view of the agent-environment interaction could be used to create unified layer for discovery and usage, both direct of indirect, of sensors and actuators regardless of their source.

5. References

Acay, L. D., Pasquier, P. & Sonenberg, L. (2007). Extrospection: Agents reasoning about the environment, *3rd IET International Conference on Intelligent Environments (IE 07)*, pp. 220–227.

Acay, L. D., Sonenberg, L., Ricci, A. & Pasquier, P. (2009). How situated is your agent? a cognitive perspective, *Programming Multi-Agent Systems: 6th International Workshop, ProMAS 2008, Estoril, Portugal, May 13, 2008. Revised Invited and Selected Papers*, Springer-Verlag, Berlin, Heidelberg, pp. 136–151.

Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F. & Mylopoulos, J. (2004). Tropos: An Agent-Oriented Software Development Methodology, *Autonomous Agents and Multi-Agent Systems* 8(3): 203–236.

Cossentino, M. (2005). From Requirements to Code with the PASSI Methodology, *Agent-Oriented Methodologies*, Idea Group Publishing, pp. 79–106.

Dalpiaz, F., Molesini, A., Puviani, M. & Seidita, V. (2008). Towards filling the gap between AOSE methodologies and infrastructures: Requirements and meta-model, *in* M. Baldoni, M. Cossentino, F. De Paoli & V. Seidita (eds), *9th Workshop "From Objects to Agents" (WOA 2008) – Evolution of Agent Development: Methodologies, Tools, Platforms and Languages*, Seneca Edizioni, Palermo, Italy, pp. 115–121.

De Wolf, T. & Holvoet, T. (2006). A catalogue of decentralised coordination mechanisms for designing self-organising emergent applications, *Technical Report CW458*, Katholieke Universiteit Leuven, Department of Computer Science.

Deloach, S. A. (2006). Engineering organization-based multiagent systems, *LNCS*, Springer, pp. 109–125.

DeLoach, S. & Valenzuela, J. (2007). An agent-environment interaction model, *in* L. Padgham & F. Zambonelli (eds), *Agent-Oriented Software Engineering VII*, Vol. 4405 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 1–18.

Doerr, M., Hunter, J. & Lagoze, C. (2003). Towards a core ontology for information integration, *Journal of Digital Information* 4(1).

Esteva, M., Rosell, B., Rodriguez-Aguilar, J. A. & Arcos, J. L. (2004). Ameli: An agent-based middleware for electronic institutions, *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, IEEE Computer Society, Washington, DC, USA, pp. 236–243.

Ferber, J. & Müller, J.-p. (1996). *Influences and Reaction : a Model of Situated Multiagent Systems*.

FIPA: Foundation for Intelligent Physical Agents. (n.d.).
URL: *http://www.fipa.org*

Hayes-Roth, B. (1995). An architecture for adaptive intelligent systems, *Artif. Intell.* 72(1-2): 329–365.

Huhns, M. N. & Stephens, L. M. (1999). Multiagent systems and societies of agents, *in* G. Weiss (ed.), *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, The MIT Press, Cambridge, MA, USA, pp. 79–120.

Kesäniemi, J., Katasonov, A. & Terziyan, V. (2009). An Observation Framework for Multi-agent Systems, *2009 Fifth International Conference on Autonomic and Autonomous Systems* pp. 336–341.

Maes, P. (1995). Artificial life meets entertainment: lifelike autonomous agents, *Communications of the ACM* 38(11): 108–114.

Mamei, M. & Zambonelli, F. (2005). Programming stigmergic coordination with the tota middleware, *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, ACM, New York, NY, USA, pp. 415–422.

Meszaros, G. & Doble, J. (1996). MetaPatterns: A Pattern Language for Pattern Writing, *In 3rd Pattern Languages of Programming conference*, pp. 4—-6.

Mili, R. Z. & Steiner, R. (2008). Modeling agent-environment interactions in adaptive mas, pp. 135–147.

Molesini, A., Omicini, A., Denti, E. & Ricci, A. (2005). Soda: A roadmap to artefacts, *ESAW*, pp. 49–62.

Müller, J. P. (1996). *The Design of Intelligent Agents*, Springer Berlin / Heidelberg.

Odell, J., Associates, J. O., Arbor, A. & Parunak, H. V. D. (2003). Modeling agents and their environment: the physical environment, *Journal of Object Technology* 2: 43–51.

Odell, J., Parunak, H. V. D. & Fleischer, M. (2003). Modeling agents and their environment: The communication environment, *Journal of Object Technology* 2(1): 39–52.

Omicini, A. (2001). Soda: Societies and infrastructures in the analysis and design of agent-based systems, *in* P. Ciancarini & M. Wooldridge (eds), *Agent-Oriented Software Engineering*, Vol. 1957 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 311–326.

Omicini, A., Ricci, A. & Viroli, M. (2008). Artifacts in the A&A meta-model for multi-agent systems, *Vital And Health Statistics. Series 20 Data From The National Vitalstatistics System Vital Health Stat 20 Data Natl Vital Sta* (May): 432–456.

Parunak, H. V. D. (1997). " Go to the Ant ": Engineering Principles from Natural Multi-Agent Systems, *Artificial Intelligence* 4049: 1–27.

Platon, E., Honiden, S. & Sabouret, N. (2005). Oversensing with a softbody in the environment: Another dimension of observation, *Proceedings of Modeling Others from Observation at International Joint Conference on Artificial Intelligence*.

Platon, E., Mamei, M., Sabouret, N., Honiden, S. & Parunak, H. V. D. (2006). Mechanisms for environments in multi-agent systems: Survey and opportunities, *Autonomous Agents and Multi-Agent Systems* 14(1): 31–47.

Ricci, A., Piunti, M., Acay, L. D., Bordini, R. H., Hübner, J. F. & Dastani, M. (2008). Integrating heterogeneous agent programming platforms within artifact-based environments, *AAMAS '08: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems*, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, pp. 225–232.

Ricci, A., Viroli, M. & Omicini, A. (2006). Programming mas with artifacts, *in* R. Bordini, M. Dastani, J. Dix & A. Seghrouchni (eds), *Programming Multi-Agent Systems*, Vol. 3862 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 206–221.

Ricci, A., Viroli, M. & Omicini, A. (2007a). Cartago: a framework for prototyping artifact-based environments in mas, *E4MAS'06: Proceedings of the 3rd international conference on Environments for multi-agent systems III*, Springer-Verlag, Berlin, Heidelberg, pp. 67–86.

Ricci, A., Viroli, M. & Omicini, A. (2007b). Give agents their artifacts: the a&#38;a approach for engineering working environments in mas, *AAMAS '07: Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, ACM, New York, NY, USA, pp. 1–3.

Rowstron, A. & Druschel, P. (2001). Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems, *Middleware '01: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, Springer-Verlag, pp. 329–350.

Russell, S. J. & Norvig, P. (1995). *Artificial Intelligence: A Modern Approach*, Prentice-Hall International.

Saunier, J. & Balbo, F. (2007). An Environment to Support Multi-Party Communications in Multi-Agent Systems, *Environment* pp. 52–61.

Saunier, J., Balbo, F. & Badeig, F. (2007). Environment as active support of interaction, *E4MAS'06: Proceedings of the 3rd international conference on Environments for multi-agent systems III*, Springer-Verlag, Berlin, Heidelberg, pp. 87–105.

Viroli, M., Holvoet, T., Ricci, A., Schelfthout, K. & Zambonelli, F. (2007). Infrastructures for the environment of multiagent systems, *Autonomous Agents and Multi-Agent Systems* 14(1): 49–60.

Viroli, M., Moro, G. & Omicini, A. (2001). On observation as a coordination paradigm: an ontology and a formal framework, *SAC '01: Proceedings of the 2001 ACM symposium on Applied computing*, ACM, New York, NY, USA, pp. 166–175.

Viroli, M., Omicini, A. & Ricci, A. (2005). Engineering MAS environment with artifacts, *in* D. Weyns, H. V. D. Parunak & F. Michel (eds), *2nd International Workshop "Environments for Multi-Agent Systems" (E4MAS 2005)*, AAMAS 2005, Utrecht, The Netherlands, pp. 62–77.

Weyns, D., Helleboogh, A., Holvoet, T. & Schumacher, M. (2007). The Agent Environment in Multi-Agent Systems : A Middleware Perspective, *Sciences-New York* (3): 1–20.

Weyns, D., Omicini, A. & Odell, J. (2006). Environment as a first class abstraction in multiagent systems, *Autonomous Agents and Multi-Agent Systems* 14(1): 5–30.

Weyns, D., Parunak, H. V. D., Michel, F., Holvoet, T. & Ferber, J. (2005). Environments for multiagent systems, state-of-the-art and research challenges, *Lecture Notes in Computer Science Series* 3374: 2–52.

Zambonelli, F., Jennings, N. R. & Wooldridge, M. (2003). Developing multiagent systems: The gaia methodology, *ACM Trans. Softw. Eng. Methodol.* 12(3): 317–370.