The Knowledge Engineering Review, Vol. 20:2, 127–141. © 2005, Cambridge University Press
doi:10.1017/S0269888905000457 Printed in the United Kingdom

Environments in multiagent systems

DANNY WEYNS[1], MICHAEL SCHUMACHER[2], ALESSANDRO RICCI[3], MIRKO VIROLI[3] and TOM HOLVOET[1]

[1]*Katholieke Universiteit Leuven, Celestijnenlaan 200A, 3001 Leuven, Belgium;*
e-mail: danny.weyns@cs.kuleuven.be, tom.holvoet@cs.kuleuven.be
[2]*École Polytechnique Fédérale de Lausanne, CH-1015, Lausanne, Switzerland;*
e-mail: michael.schumacher@epfl.ch
[3]*Università di Bologna, 47032 Cesena, Italy;*
e-mail: a.ricci@unibo.it, mirko.viroli@unibo.it

Abstract

There is a growing awareness in the multiagent systems research community that the environment plays a prominent role in multiagent systems. Originating from research on behavior-based agent systems and situated multiagent systems, the importance of the environment is now gradually being accepted in the multiagent system community in general. In this paper, we put forward the environment as a first-order abstraction in multiagent systems. This position is motivated by the fact that several aspects of multiagent systems that conceptually do not belong to agents themselves should not be assigned to, or hosted inside the agents. Examples are infrastructure for communication, the topology of a spatial domain or support for the action model. These and other aspects should be considered explicitly. The environment is the natural candidate to encapsulate these aspects. We elaborate on environment engineering, and we illustrate how the environment plays a central role in a real-world multiagent system application.

1 Introduction

Multiagent systems are an approach to building complex distributed applications. A multiagent system consists of a population of autonomous entities (agents) situated in a shared structured entity (the environment). One classic definition of an autonomous agent is 'a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future' (Franklin & Graesser, 1996). This definition stresses the importance of the environment as the medium for an agent to live, or the first entity an agent interacts with. The environment is strongly related to the notion of *embodiment*, which refers to the fact that an autonomous agent has a 'body' that delineates it from the environment in which the agent is situated. It is in this environment that an agent senses and acts.

The importance of the environment in agent systems is not new. Researchers working in the domain of behavior-based agents have been considering interaction in the environment as an essential feature for intelligent behavior for a long time, see, e.g., Brooks (1991), Maes (1994), or Arkin (1998). Originally, the main focus of this research community was on systems where agents interact in a physical environment, such as robots. Gradually, this work has influenced the software agent community. Today, researchers working in the domain of what is known as *situated multiagent systems* consider logical environments as essential parts of their multiagent systems; examples are Odell *et al*. (2003), Parunak (1997), or Mamei & Zambonelli (2004). These researchers have shown that the environment can serve as a robust, self-revising shared memory, and an excellent medium for indirect coordination of agents (Weyns *et al*., 2005b). Several practical

applications have shown how indirect interaction through the environment increases the power and expressiveness of multiagent systems, enabling solutions that would otherwise be impossible or at least impractically complex. There are examples in domains such as supply chain systems (Sauter & Parunak, 1999), network support (Bonabeau *et al.*, 1998), peer-to-peer systems (Babaoglu *et al.*, 2002), manufacturing control (Parunak, 2001), and support for automatic logistic services (Weyns *et al.*, 2005d). For an overview of industrial applications, see chapter 9 of Weiss (1998).

In spite of the promising results obtained in situated multiagent systems, most approaches in agent research still view the environment as something being modeled in the 'minds' of the agents, thus using a minimal and implicit environment that is not a first-order abstraction, but somehow the sum of all data structures within agents that represent an environment. This is a typical *subjective view* of a multiagent system inherited from Distributed Artificial Intelligence, which contrasts with an *objective view* that deals with the system from an external point of view of the agents; see, e.g., Schumacher (2001) or Omicini & Ossowski (2003). In this paper, we claim that the environment should be viewed as a *first-order abstraction* in the software engineering process of multiagent systems. This position links up with recent initiatives that consider the environment as the focus of research in its own right; see E4MAS (2005) and AgentLink-III (2005) Technical Forum.

This paper is structured as follows. In Section 2, we elaborate on the environment abstraction. Section 3 discusses engineering issues for environments. Next, in Section 4, we zoom in on a real-world application in which the environment plays a central role. Finally, we draw conclusions and look at venues for future work.

2 The environment abstraction

In this paper, we put forward the environment as a *first-order abstraction* in the software engineering process of multiagent systems. Considering the environment as a first-order abstraction allows us to clearly define the environment responsibilities that differ from the agent responsibilities. A first-class module can be defined as 'a program building block, an independent piece of software which [. . .] provides an abstraction or information hiding mechanism so that a module's implementation can be changed without requiring any change to other modules'.[1] The environment should therefore be an independent building block that encapsulates its own clear-cut responsibilities in a multiagent system. Motivations to put forward the environment as first-order abstraction are the following.

1. Several aspects of multiagent systems that conceptually do not belong to agents themselves should not be assigned to, or hosted inside, agents. Examples are infrastructure for communication, the topology of a spatial domain, or support for the action model.
2. The above and other aspects should be explicitly considered. The environment is the natural candidate to encapsulate these aspects.
3. The environment can be a creative part of a designed solution of a multiagent system, helping to manage the huge complexity of engineering complex real-world applications.

In the following two sections, we further elaborate on the position and the role of the environment as a first-order abstraction in multiagent systems.

2.1 A three-layer model of multiagent systems

One problem with the specification of the environment is the confusion between the logical entity of an environment in the application and the underlying infrastructure of the multiagent system. To unravel this confusion, we discuss a deployment model for multiagent system applications that describes the position of agents and the environment at three levels (see Figure 1):

[1] The Free Online Dictionary of Computing, `http://foldoc.doc.ic.ac.uk/foldoc/`, 8/2005.

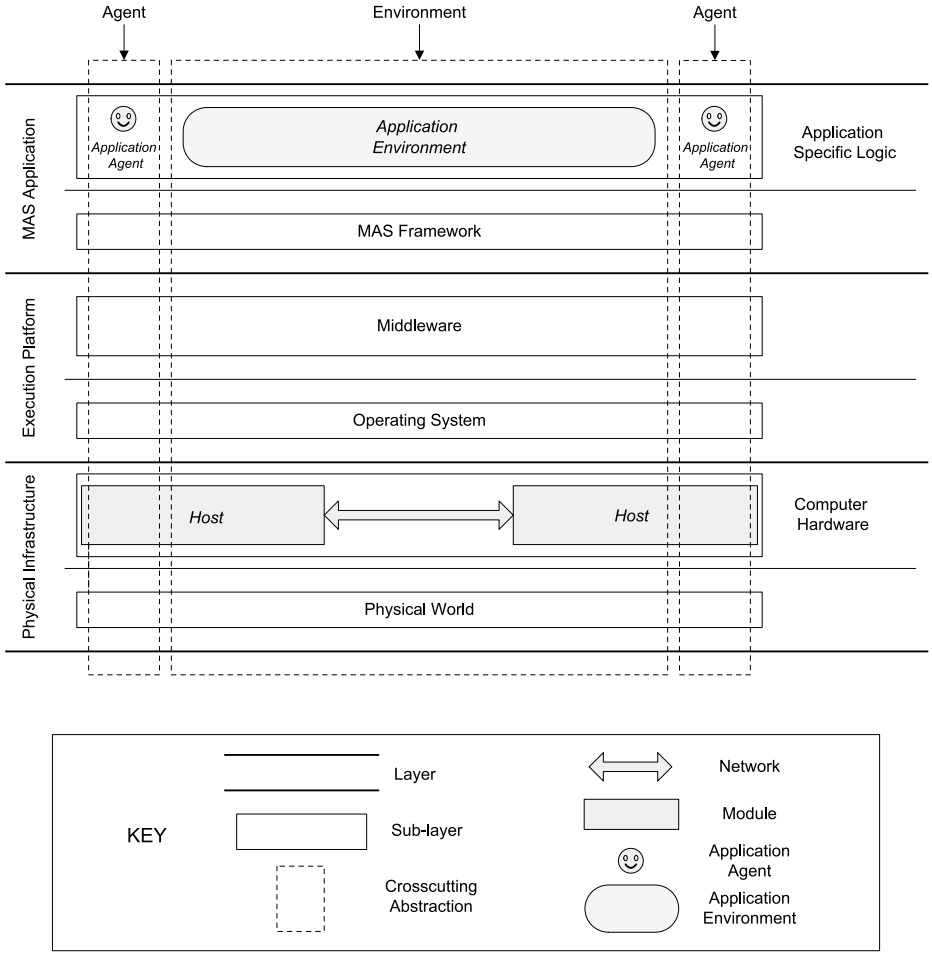

Figure 1 Three-layer model for multiagent systems

- the *multiagent application* layer at the top (i.e. the application logic);
- the *execution platform* (i.e. middleware infrastructure and the operating system); and
- the *physical infrastructure* at the bottom (i.e. processors, network, etc.).

We now elaborate on each layer and illustrate how the abstraction of the environment as well as the agent span the three layers in the model. For a detailed study of the three-layer model, we refer to Weyns *et al.* (2005f).

2.1.1 Multiagent system application layer

The multiagent system application layer contains the *application specific logic*, i.e. application agents and the application environment of the multiagent system. The application agents are the autonomous entities in the multiagent system. The application environment offers a domain-specific abstraction to application agents, hiding the complexity of recourse access, interaction handling and consistency management. The application environment imposes the rules that regulate domain dynamics, e.g. application agents mobility, access to resources, etc. Section 2.2 elaborates on responsibilities of the application environment.

Consider the example of an electronic shopping mall, where consumers can buy various goods in different shops. Such a system may be constructed with virtual shops distributed on the Internet and mobile agents roaming from one place to the other. It may also be matched to a real supermarket, where personal agents on Personal Digital Assistants (PDAs) contact those local shops their users are interested in. The agents are straightforward: buyers, sellers and intermediaries

for financial transactions, logistics, etc. The environment is made up of one logical place for each shop, plus a common place sharing some characteristics for the whole supermarket. Each of these places will have specificities such as product descriptions, payment facilities, etc. The environment also ensures exchange of data, for instance between buyers and sellers.

The application logic is typically deployed on top of a *multiagent system framework*. The multiagent system framework supports predefined multiagent systems abstractions, such as loci for places, different types of agents, support for indirect coordination such as a infrastructure for digital pheromones, etc. The multiagent system framework provides reuse across different applications.

2.1.2 Execution platform

The execution platform is composed of a *middleware* on top of an *operating system*. Middleware offers transparent access to lower level resources and serves as the glue between (distributed) components, it hides hardware and platform details, and offers powerful capabilities such as remote method invocation, threading, transaction, persistence or load balancing. Middleware offers programming abstractions, which hide system hardware, network and distribution, and implements at the same time a software platform on which (distributed) applications can be executed. For instance, the electronic shopping mall application would certainly need a middleware with capabilities for database access, load balancing of threads, monitoring, security, etc.

The operating system enables the execution of the application on the physical hardware; it offers basic functionality to applications, hiding low-level details of the underlying physical platform. The operating system manages memory usage and offers transparent access to lower level resources such as files, it provides network facilities, it handles the intervention of the users, it provides basic support for timing, etc.

2.1.3 Physical infrastructure

The execution platform runs on top of the *physical infrastructure*, which is composed of the *computer hardware* with hosts and a network, and the *physical world*, if present in the application. The shopping mall application would be run on a computer farm at different locations. If the supermarket is only virtual, no physical world comes into play. However, if it runs in a real shopping mall with an *ad hoc* intelligent agent based peer-to-peer network, as proposed in Helin *et al.* (2005), the physical world would be composed of hosts and users' PDAs.

2.2 The role of environments in multiagent systems

Having clarified how the notion of environment is being used, we now elaborate on the logical functionalities of the environment.

Until now, the majority of researchers of the multiagent systems research community have mainly considered the environment as an implicit means for agent communication and as an agent container. Typically, environment issues are treated implicitly. Recent research efforts propose ideas that go beyond this view; see, for instance, Odell *et al.* (2003) or Weyns *et al.* (2005b). Following this line, we want to make the environment responsibilities explicit, i.e. as concerns of environments as a first-order abstraction. In this section, we elaborate on the role an environment plays in a multiagent system, linking up with the discussion started in Weyns *et al.* (2005b).

2.2.1 Responsibilities of the environment

Intuitively, the environment can be seen as a place where agents interact with domain objects and resources, and with other agents. Important responsibilities of the environment are as follows.

- *Structuring*. The environment is first of all a shared common 'space' for the agents, which structures the whole system. Such a structuring can be spatial (Bandini *et al.*, 2005), or

organizational (Ferber *et al*., 2003). Specific different properties can be defined separately for each space, such as positions, locality, groups or roles.

- *Managing resources and services*. The environment acts as a container for resources and services to be accessed. Resources are objects with a specific state. Services are considered as reactive entities that encapsulate functionality. The environment is in charge of enabling and controlling the access to resources and services.
- *Maintaining environmental processes*. Besides the activity of the agents, the environment can assign particular activities to resources as well. Examples are aggregation, diffusion and evaporation of digital pheromones, a rolling ball that moves on, the local temperature that evolves over time, or garbage collection of outdated data. Maintaining such dynamics is an important functionality of the environment.
- *Enabling communication*. The environment defines concrete means for agents to communicate. The most used scheme is a message-passing style from one agent to the other. In generative or indirect communication, agents produce communication objects within the environment and consume it to read them. Issues such as anonymity and synchronization specify more concretely the semantics. Another important approach of communication is based on stigmergy (Parunak, 1997).
- *Ruling the multiagent system*. The environment can define different types of rules or laws on all entities in the multiagent system. Rules may restrict access to specific resources or services to particular types of agents, or determine the outcome of agents' interactions. Environment rules can become a very powerful tool to express the capabilities an environment needs to ensure consistency in the system. For example, in electronic institutions (Noriega & Sierra, 2002), agents have to follow well-defined communication protocols. Agent actions in an institution may have consequences that either limit or enlarge its subsequent acting possibilities. The environment can define and enforce the rules imposed on the interactions of agents in an electronic institution.
- *Providing observability*. Contrary to agents, the environment must be observable, agents must have access to resources and services. Agents may even be able to observe the actions of other agents; see, e.g., Tummolini *et al*. (2005) or Platon *et al*. (2005). Related to observability is the semantic description of resources and services. In an open system, it would be useful for agents to be able to understand at run-time a new environment they are discovering (Chang *et al*., 2005). This can be done by defining an environment ontology that describes the environment, its resources and services, and possibly the regulating laws.

The presented list of responsibilities is not intended to be complete, and neither are all the presented responsibilities applicable to every multiagent system. It is up to the designer to decide which responsibilities should be integrated in the multiagent application.

2.2.2 A coordination point of view

The issues presented above can be described taking the point of view of *coordination*. Coordination is defined as the *management of dependencies between activities* (Malone & Crowston, 1994). Whenever different agents come into play with one another, interdependencies rise. Thus generic dependencies should be identified and managed. Two types of dependencies and corresponding coordination can be distinguished; see Schumacher (2001) and Omicini & Ossowski (2003).

- Agents have *subjective dependencies* which refer to intra-agent dependencies towards other agents. The management of these subjective dependencies refers to *subjective coordination*, such as negotiation techniques.
- A multiagent system is also built by *objective dependencies* which refer to inter-agent dependencies, namely the configuration of the system in terms of the basic interaction means, agent generation/destruction, organization of spaces, etc. The management of these dependencies refers to *objective coordination*, because they are external to the agents.

Thus, objective coordination is essentially concerned with the environment. It can be handled with coordination models, which have been defined in the area of concurrent and distributed computing (Papadopoulos & Arbab, 1998). Coordination models are very useful, because of their clear semantics. A coordination model is described by a *coordination medium* (the means through which coordination takes place) and its *coordination laws*. Laws define the semantics of the coordination mechanisms. In some models, the coordination laws can even be changed at runtime; they are therefore an excellent candidate for supporting environment laws.

3 Engineering environments

Environment engineering is still in its infancy. In this section we briefly look at how state-of-the-art agent-oriented methodologies deal with the environment. After that, we discuss two approaches to model the environment. The first approach is inspired by social science and models the environment as a set of mediating artifacts that agents can use. The second approach decomposes the environment into a number of modules that correspond to different functional concerns of the environment, such as communication, perception, and actions.

3.1 Environments in agent-oriented software methodologies

Popular methodologies such as Prometheus (Padgham & Winikoff, 2003), Tropos (Giunchiglia *et al*., 2002), or Adelfe (Bernon *et al*., 2003) offer support for some basic elements of the environment; however, they do not consider the environment as a first-order abstraction. Two methodologies that explicitly cope with the environment are SODA (Omicini, 2001) and GAIA v.2. (Zambonelli *et al*., 2003).

3.1.1 SODA

SODA takes the environment into account and provides specific abstractions and procedures for the design of agent infrastructures. In SODA, the environment is the space in which agents operate and interact. SODA provides a resource model that models the application environment in terms of the available services, associated with abstract resources. The environmental model maps resources onto infrastructure classes. An infrastructure class is characterized by the services, the access modes, the permissions granted to roles and groups, and the interaction protocols associated to its resources. Infrastructure classes can be further characterized in terms of other features: their cardinality (the number of infrastructure components belonging to that class), their location (with respect to topological abstractions), and their owner (which may or may not be the same as the one of the agent system, given the assumption of decentralized control).

3.1.2 GAIA v.2.

According to GAIA v.2. (hereafter GAIA), 'modelling the environment involves determining all the entities and resources that the multiagent system can exploit, control or consume when it is working towards the achievement of the organizational goal' (Zambonelli *et al*., 2003, p. 12).

In GAIA, the identification of the environmental model is part of the analysis phase and is intended to yield an abstract, computational representation of the environment in which the multiagent system will be situated. During the subsequent architectural design phases, the output of the environmental model (together with a primary role model, a preliminary interactions model, and a set of organizational rules) is integrated into the system's organizational structure that includes the real-world organization (if any) in which the multiagent system is situated. The organizational structure is then used to complete the preliminary role and interaction models. During the detailed (and final) design phase, the definition of the agent model and services model are derived from the completed role and interaction models. GAIA does not commit itself to

specific techniques for modeling roles, environment and interactions, etc. The outcome of the GAIA process is a technology-neutral specification that should be easily implemented using an appropriate agent-programming framework or an object or a component-based framework. With respect to the development of the environmental model, Zambonelli *et al*. (2003, p. 23) state 'it is difficult to provide general modeling abstractions and general techniques because the environments for different applications can be very different in nature and also because they are somehow related to the underlying technology'. Therefore a 'reasonable general approach is proposed (without the ambition to be universal), that describes the environment in terms of *abstract computational resources*, such as variables or tuples, made available to the agents for sensing (e.g. reading their values), for affecting (e.g. changing their values) and for consuming (e.g. extracting them from the environment)'. As such, the environmental model is represented as a list of resources, each associated with a symbolic name, characterized by the type of actions that the agent can perform on it and possibly associated with additional textual comments and descriptions. Zambonelli *et al*. (2003) confirm that in realistic development scenarios, the analyst would choose to provide a more detailed and structured view of environmental resources.

3.1.3 Summary

Although SODA and GAIA explicitly deal with the environment in the methodological process, the interpretation of what the environment comprises is meagre. Design support is limited to the representation of resources and simple access control to the resources.

3.2 Artifacts: building blocks for engineering multiagent system environments

A key point when engineering multiagent system environments is the level of abstraction. Adopted abstractions should make it possible to model agent–environment interactions at the agent level, i.e. in terms of actions and perceptions, abstracting from low-level mechanisms and protocols.

Inspired by Activity Theory (Ricci *et al*., 2003), and building upon the work on coordination artifacts (Omicini *et al*., 2004), the notion of *artifact* has been proposed as an abstract building block for modeling and engineering environments (Viroli *et al*., 2005). Contrary to an agent that is basically an autonomous, goal-oriented entity with social abilities, an artifact is a software entity designed to provide some kind of function or service that agents can use to achieve their goals (Ricci *et al*., 2005). Whereas an agent is typically located at a single node of a network, an artifact can be distributed over the network, i.e. an artifact can cover multiple nodes. This characterization fits the basic distinction made in Distributed Artificial Intelligence between goal-oriented entities (agents) which proactively interact, and function-oriented entities (artifacts) designed with a clear interface and working modalities to be used by goal-oriented entities to achieve their objectives (Conte & Castelfranchi, 1995). An artifact can be specified by (1) its *function*, i.e. *what* services the artifact provides; (2) its *usage interface*, i.e. the set of the operations which agents can invoke to use the artifact and exploit its function; and (3) a set of *operating instructions*, i.e. descriptions that explain *how* the artifact can be used to exploit its functionality.

Artifacts can be useful from two different perspectives: (1) analytical, i.e. as a way to describe, discuss, compare existing environment models and approaches keeping a certain level of abstraction and uniformity; and (2) from an engineering perspective, i.e. as a concrete way to design and build multiagent system. As a first rough classification, artifacts can be classified into three categories. The first class represents *resource artifacts*. A resource artifact mediates the access to a specific resource, or directly represents a resource in the multiagent environment. Resource artifacts provide a representation of computational or physical entities (from objects to Web services) at the abstraction level of the agents. The second class includes *coordination artifacts*. A coordination artifact provides a coordinating function or service; it can be used by agents as a tool for communication, coordination and, more generally, it supports social activities in the multiagent system; see Ricci *et al*. (2003) and Omicini *et al*. (2004). Finally, the

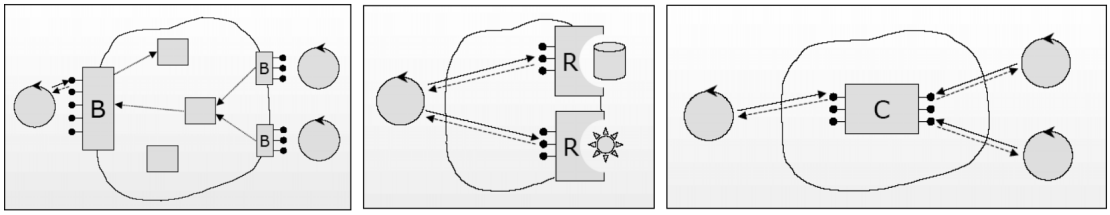

Figure 2 Basic types of artifacts: boundary artifacts (B), resource artifacts (R), coordination artifacts (C)

third class of artifacts are *organization artifacts*, which have an organizational or security function. An example of an organization artifact is a boundary artifact. A boundary artifact can be used to characterize and control the presence of an agent in an organization context, reifying and enacting a *contract* between the agent and the organization. For example, boundary artifacts can be used as 'filters', allowing only agent actions that satisfy the contract for the specific role(s) the agent plays in an organization. A picture with various kinds of artifacts is depicted in Figure 2.

Concrete examples of artifacts in the context of the general purpose coordination infrastructure TuCSoN (Omicini& Zambonelli, 1999) are *tuple centres* (Omicini & Denti, 2001) and *agent coordination contexts* (Omicini, 2002; Ricci *et al*., 2004). A tuple center is an example of a coordination artifact whose coordinating behavior can be specified dynamically in a language called ReSpecT. An agent coordination context is an example of a boundary artifact. An agent coordination context enables (and filters) agent actions (and patterns of actions) according to (1) the role(s) the agent plays, and (2) the organizational rules of the organization context where the agent is situated.

3.2.1 Issues in engineering environments with artifacts

As the basic notion of artifact appears in be a useful building block in engineering the environment of multiagent systems, a number of issues naturally arise concerning its ability to cover the various responsibilities of environments.

One issue is related to inter-artifact interactions: how can services provided by artifacts exploit each other in a compositional way? To scale up with the complexity of an environment, it is necessary to compose artifacts, e.g. to build a service incrementally on top of another, by making a new artifact realizing its service by interacting with other artifacts. To this end, artifacts are *linkable*: one artifact can invoke the operations of another artifact. The reply to that invocation will be transmitted by the receiver through the invocation of another operation in the sender. The linkability property of artifacts supports an incremental design of the environment at subsequent levels of abstractions.

Another issue of interest is the ability to associate artifacts with topological aspects: how can artifacts take into account the topology of a multiagent system environment? As mentioned previously, contrary to an agent, which is typically seen as a point-like abstraction conceptually located to a single node of the network, artifacts can also be distributed. A single artifact can in principle be used to model a distributed service, accessible from more nodes of the net. Using linkability, a distributed artifact can then be conceived as a composition of linked artifacts, scattered over a number of different physical locations. Conceptually, the composite of artifacts can be seen as a single distributed artifact. The interplay between artifacts and topological aspects appears to be a very interesting research topic in the context of multiagent system environments.

Finally, life-cycle aspects should also be considered: how can artifacts actually be created, discovered, and destroyed? Key issues here include the engineering of basic infrastructures for multiagent system environments, as well as standardization.

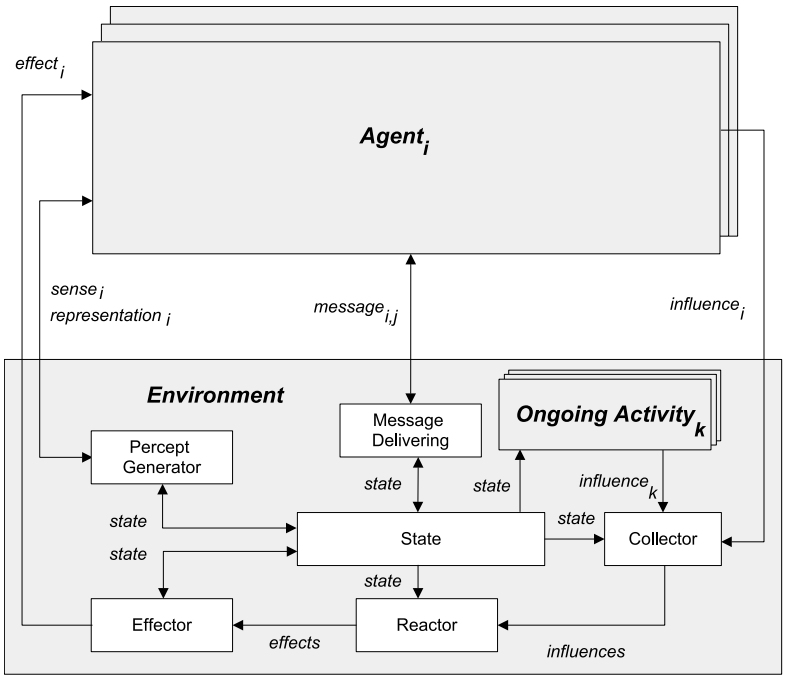

Figure 3 Concern-based modularization of the environment

3.3 Concern-based environment engineering

The second approach models the environment as a set of modules that represent different functional concerns of the environment; see Weyns & Holvoet (2004) and Weyns *et al.* (2005c). Figure 3 depicts a high-level module view of the environment architecture.

The *PerceptGenerator* module is responsible for perception (Weyns *et al.*, 2004). When an $agent_i$ is interested in perceiving its neighborhood, it invokes a $sense_i$ command on the environment. Such a sense command contains one or more *foci* that expresses the agent's current interests of perception. The *PerceptGenerator* then composes a $representation_i$ based on the foci, the current *state* of the environment and a set of *perceptual laws*. A perceptual law constrains the composition of a representation according to the requirements of the modeled domain. An example is a perceptual law that specifies how an area behind an obstacle is out of scope of a perceiving agent.

The *MessageDelivering* module is responsible for message transfer. When a message arrives, the *MessageDelivering* module passes the message to the list of addressees indicated in the message. It is possible to provide *communication laws* that are applied when messages are transferred. An example is a communication law that specifies the maximal distance that messages can be delivered. Communication laws are interesting for simulation purposes, but can also be a useful instrument for designers, e.g. to regulate the message transfer.

The *collector–reactor–effector* modules take care of action handling. The action model is based on the influence–reaction model of Ferber & Müller (1996). According to this model, agents produce influences in the environment and subsequently the environment reacts by combining the influences to deduce a new state of the world from them. The reification of actions as influences enables the environment to combine simultaneously performed activities in the system. The *collector* module collects the influences of all simultaneously performed activities in the multiagent system and passes them to the Reactor module. The simultaneity of activity can be based on transactional semantics, or it can be determined by a synchronization mechanism; see, e.g., Ferber (1999) and Weyns & Holvoet (2004). The collector passed the influences to the *reactor* module that calculates, according to a set of domain-specific *interaction laws*, the reaction, i.e. state changes in the environment and effects for the agents. An example of an effect is an agent that receives a packet that it has picked up. An example of an interaction law is a law that determines the effects of two

RoboCup football players that kick the ball simultaneously. The reactor finally passes the effects to the *effector* module which applies the outcome of the interaction, i.e. it updates the state of the environment and passes effects to the applicable agents.

Ongoing activities provides an abstraction for environmental processes as discussed in Section 2.2. An ongoing activity is defined by an *operation* that produces influences in the environment according to the state of the world. Examples of ongoing activities are a moving ball, an evaporating pheromone, a self-managing gradient field, or an automatic garbage collector for objects.

It is important to note that the module view of the environment architecture, as depicted in Figure 3, abstracts from distribution. For a practical application, the state of the environment, the delivering of messages, ongoing activities, etc. will be implemented according to the domain at hand, i.e. centralized or distributed. Another important remark is that the presented model also abstracts from real-word resources, external to the multiagent system. The *state* of the environment may represent external resources. Support to keep the state of the representation consistent with external resources is not covered by the presented model. In the next section, we discuss an example in which the state of the environment represents resources in the physical world.

4 Applying the environment in applications

In this section, we illustrate how the approach of concern-based engineering of environments is applied to an automated transportation system for warehouse logistics. This real-world application uses a multiagent system in the control of an automatic guided vehicle (AGV) transportation system (Weyns *et al*., 2005e).

4.1 Automated guided vehicles coordination

The AGV transportation system we discussed was developed in a joint R&D project between the AgentWise research group and Egemin, a manufacturer of automating logistics services in warehouses and manufactories (Egemin Modular Controls Concept, 2005). An AGV transportation system uses unmanned vehicles to transport loads through a warehouse. Typical applications are repackaging and distributing incoming goods to various branches, or distributing manufactured products to storage locations. An AGV is provided with a battery as its energy source. AGVs can move through a warehouse, following fixed paths on the factory floor, typically guided by a laser navigation system, or by magnets or cables that are fixed in the floor. The low-level control of the AGVs in terms of sensors and actuators, staying on track on a path, turning, and determining the current position, etc., is handled by the AGV control software.

Figure 4 depicts a high-level model of the situated multiagent system. The situated multiagent system consists of two kinds of agents, *transport agents* and *AGV agents*. Transport agents are located at *transport bases*. AGV agents are located in AGVs that are situated on the factory floor. The communication infrastructure provides a wireless network that enables mobile AGVs to communicate with each other and with transport agents on transport bases.

A transport agent represents a transport that needs to be handled by an AGV. AGV agents are responsible for executing the assigned transports. AGVs are situated in a physical environment, however, this environment is very constrained: AGVs cannot manipulate the environment, except by picking and dropping loads. This restricts how AGV agents can exploit their environment. Therefore, a virtual environment was introduced for agents to live in. This virtual environment offers a medium that agents can use to exchange information and coordinate their behavior. Besides, the virtual environment serves as a suitable abstraction that shields the AGV agents form low-level issues, such as the physical control of the AGV. The AGV control software that deals with

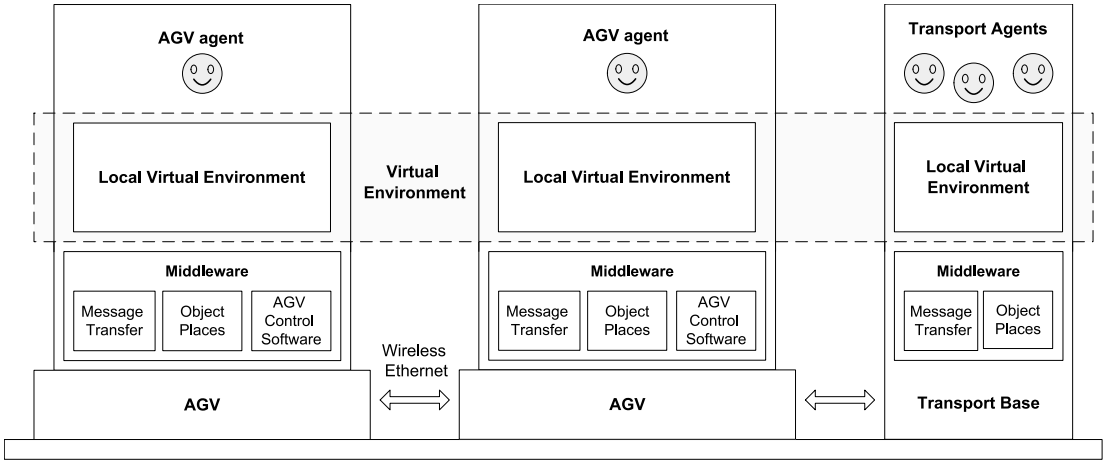

Figure 4 High-level model of the AGV transportation system

the low-level control of the AGVs is fully reused. As such, the AGV agents control the movement and actions of AGVs on a fairly high level.

In the AGV application, the only physical infrastructure available to the AGVs is a wireless network for communication. In other words, the virtual environment is necessarily distributed over the AGVs and transport bases. In effect, each AGV and each transport base maintains a *local virtual environment*, which is a local manifestation of the virtual environment. Local virtual environments are merged with other local virtual environments opportunistically, as the need arises. In other words, *the* virtual environment as a software entity does not exist; rather, there are as many local virtual environments as there are AGVs and transport bases. Some of these local virtual environments may recently be synchronized with each other, while others may not. From the agent perspective, the virtual environment appears as one entity. The synchronization of state of neighboring local virtual environments is supported by the ObjectPlaces middleware (Schelfthout & Holvoet, 2005).

4.2 Exploiting the virtual environment

We illustrate the use of the virtual environment with a couple of examples.

4.2.1 Routing

For routing purposes, the virtual environment has a static map of the paths through the warehouse. This graph-like map corresponds to the layout used by low-level AGV control software. To allow agents to find their way through the warehouse efficiently, the virtual environment provides signs on the map that the agents use to find their way to a given destination. These signs can be compared to traffic signs by the road that provide directions to drivers. At each node in the map, a sign in the virtual environment represents the cost to a given destination for each outgoing segment. The cost of the path is the sum of the static costs of the segments in the path. The cost per segment is based on the average time it takes for an AGV to drive over the segment. The agent perceives the signs in its environment, and uses them to determine which segment it will take next.

4.2.2 Traffic information

Besides the static routing cost associated with each segment, the cost is also dependent on dynamic factors, such as congestion of a segment. To warn other agents that certain paths are blocked or have a long waiting time, agents mark segments with a dynamic cost on a *traffic map* in the virtual environment. Agents mark the traffic map by dropping pheromones on the applicable segments. When AGVs enter each others' neighborhood, the information of the traffic maps is exchanged and

merged to provide up-to-date information to the AGV agents. Since pheromones evaporate over time, outdated information automatically vanishes over time. AGV agents take the information on the traffic map into account when they decide how to drive through the warehouse.

4.2.3 Collision avoidance

AGV agents avoid collisions by coordinating with other agents through the virtual environment. AGV agents mark the path they are going to follow in their environment using *hulls*. The hull of an AGV is the physical area the AGV occupies. A series of hulls then describes the physical area an AGV occupies along a certain path. If the area is not marked by other hulls (the AGV's own hulls do not intersect with others), the AGV can move along and actually drive over the reserved path. Afterwards, the AGV removes the markings in the virtual environment. Weyns *et al*. (2005d) discuss collision avoidance through the virtual environment in detail.

In summary, the virtual environment serves as a flexible coordination medium, which hides much of the distribution of the system from the agents: agents coordinate by putting marks in the environment, and observing marks from other agents. The virtual environment creates opportunities beyond a physical environment that situated AGV agents can exploit.

5 Conclusions

There is a growing awareness in the multiagent research community that the environment plays a crucial role in multiagent systems. In this paper, we have put forward the environment as a first-order abstraction in multiagent systems. Important responsibilities of the environment are (1) the environment structures the multiagent system as a whole; (2) the environment is in charge of managing resources and services; (3) contrary to agents, the environment must be observable; (4) the environment must define concrete means for the agents to communicate; (5) the environment is responsible for maintaining ongoing processes in the system; and finally (6) the environment can define different types of rules on all the entities in the multiagent system.

The research track on environments is still young and many issues are open for future research; we have just started to explore the possible responsibilities of environments in multiagent systems. The term 'environment' is vague and ill-defined in relation to multiagent systems. An ongoing research challenge will be developing a clearer understanding of what we mean by an 'environment'. In this paper we have discussed an initial model for multiagent systems that considers agents and the environment as first-order abstractions. These abstractions span the application logic, the execution platform and the physical infrastructure of the multiagent system. However, the exact nature of the relationship between the agent software, the environment software, and the software and hardware that make up the computational substrate needs further clarification. Recent initiatives tackle these and related research questions; see E4MAS (2005) and the AgentLink-III (2005) Technical Forum.

Environment engineering is still in its infancy. In this paper, we discussed two initial models for engineering environments: artifacts and concern-based modularization. Study of agent-oriented methodologies shows that current methodologies offer little support for designing environments; a whole domain of work is waiting to be tackled. From a methodological point of view, the environment should be considered as a first-order abstraction in design models and description languages. Agent-oriented programming has led to the proliferation of frameworks and development platforms for agents. Recognition of the importance of environments will stimulate extensions to these tools, or even the development of new tools that can support environments within which agents from different platforms can interact.

Besides tackling the research challenges, we have to apply environments in real-world multiagent system applications. In this paper, we discussed a practical application and showed how a virtual environment creates opportunities for agents to exchange information and coordinate their behavior in a way that would be impossible in the physical environment.

Encountering the complexity of real applications will urge us to invent new ways to exploit environments.

Acknowledgements

We would like to express our appreciation to the attendees of the workshops on Environments for Multiagent Systems in New York 2004 and Utrecht 2005, and the AgentLink III Technical Forum in Ljubljana, 2005 for the inspiring discussions that have considerably contributed to the work presented in this paper.

References

AgentLink-III 2005, Technical Forum Group on Environments for Multiagent Systems, `http://www.cs.kuleuven.ac.be/~distrinet/events/e4mas/tfg2005/`.

Arkin, R, 1998, *Behavior-based Robotics*. Cambridge, MA: MIT Press.

Babaoglu, O *et al*., 2002, Anthill: a framework for the development of agent-based peer-to-peer systems. In *Proceedings of the 22nd International Conference on Distributed Computing Systems*, Vienna, Austria. Los Alamitos, CA: IEEE Computer Society Press, pp. 15–22.

Bandini, S *et al*., 2005, A spatially dependent communication model for ubiquitous systems. In Weyns, D *et al*. (eds.), *Proceedings of the 1st International Workshop on Environments for Multi-agent Systems* (*Lecture Notes in Computer Science, 3374*). Berlin: Springer, pp. 74–90.

Bernon, C, *et al*., 2003, ADELFE: a methodology for adaptive multiagent systems engineering. In Petta, P, Tolksdorf, R and Zambonelli, F (eds.), *Engineering Societies in the Agents World III* (*Lecture Notes in Computer Science, 2577*), Madrid, Spain. Berlin: Springer, pp. 156–169.

Bonabeau, E, *et al*., 1998, Routing in telecommunications networks with ant-like agents. In *Proceedings of the 2nd International Workshop on Intelligent Agents for Telecommunication a Applications*, Paris, France. London: Springer, pp. 60–71.

Brooks, R, 1991, Intelligence without representation. *Artificial Intelligence* **47**, 139–159.

Chang, P *et al*., 2005, From reality to mind: a cognitive middle layer of environment concepts for believable agents. In Weyns, D *et al*. (eds.), *Proceedings of the 1st International Workshop on Environments for Multi-agent Systems* (*Lecture Notes in Computer Science, 3374*). Berlin: Springer, pp. 57–73.

Conte, R and Castelfranchi, C (eds.), 1995, *Cognitive and Social Action*. London: UCL Press.

E4MAS, 2005, International workshop series on Environments for Multiagent Systems, `http://www.cs.kuleuven.ac.be/~distrinet/events/e4mas/`.

Egemin Modular Controls Concept 2005, EMC2 project, Flemish Institute for the Advancement of Scientific-Technological Research in the Industry, IWT, Belgium, `http://emc2.egemin.com/`.

Ferber, J, 1999, *An Introduction to Distributed Artificial Intelligence*. London, UK: Addison-Wesley.

Ferber, J *et al*., 2003, From agents to organizations: an organizational view of multi-agent systems. In Giorgini, P, Müller, JP and Odell, J (eds.), *Agent-Oriented Software Engineering IV* (*Lecture Notes in Computer Science, 2935*), Melbourne, Australia. Berlin: Springer, pp. 214–230.

Ferber, J and Müller, JP, 1996, Influences and reaction: a model of situated multiagent systems. In Tokoro, M (ed.), *Proceedings of the 2nd International Conference on Multi-agent Systems*, Kyoto, Japan. Menlo Park, CA: AAAI Press, pp. 72–80.

Franklin, S and Graesser, A, 1996 Is it an agent or just a program? A taxonomy for autonomous agents. In Muller, J, Wooldridge, M and Jennings, N (eds.), *Intelligent Agents III. Agent Theories, Architectures, and Languages* (*Lectures Notes in Artificial Intelligence, 1193*). Berlin: Springer, pp. 21–35.

Giunchiglia, F *et al*., 2002, The TROPOS software development methodology. Processes, models and diagrams. In Castelfranchi, C and Johnson, WL (eds.), *Proceedings of the 1st Joint Conference on Autonomous Agents and Multiagent Systems*, Bologna, Italy. New York: ACM Press, pp. 35–36.

Helin, H *et al*., 2005, Context-aware business application service co-ordination in mobile computing environments. In *Proceedings of the International Workshop on Ambient Intelligence*, Utrecht, The Netherlands, pp. 1–12.

Maes, P, 1994, Modeling adaptive autonomous agents. *Artificial Life* **1**(1–2), 135–162.

Malone, T and Crowston, K, 1994, The interdisciplinary study of coordination. *ACM Computing Surveys* **26**(1), 87–119.

Mamei, M and Zambonelli, F, 2004, Programming pervasive and mobile computing applications with the TOTA middleware. In *Proceedings of the 2nd International Conference on Pervasive Computing and Communications*, Orlando, FL. Washington, DC: IEEE Computer Society Press, pp. 263–276.

Noriega, P and Sierra, C, 2002, Electronic institutions: future trends and challenges. In *Proceedings of the 6th International Workshop on Cooperative Information Agents* (*Lecture Notes in Computer Science, 2446*), London: Springer, pp. 14–17.

Odell, J *et al*., 2003, Modeling agents and their environment. In Giunchiglia, F, Odell, J and Weiß, G (eds.), *Agent- Oriented Software Engineering III* (*Lecture Notes in Computer Science, 2585*), Bologna, Italy. Berlin: Springer, pp. 16–31.

Omicini, A, 2001, SODA: societies and infrastructures in the analysis and design of agent-based systems. In Ciancarini, P and Wooldridge, M (eds.), *Agent-Oriented Software Engineering* (*Lecture Notes in Computer Science, 1937*), Limerick, Ireland. Berlin: Springer, pp. 185–193.

Omicini, A, 2002, Towards a notion of agent coordination context. In Marinescu, D and Lee, C (eds.), *Process Coordination and Ubiquitous Computing*. CRC Press, pp. 187–200.

Omicini, A and Denti, E, 2001, From tuple spaces to tuple centres. *Science of Computer Programming* **41**(3), 277–294.

Omicini, A and Ossowski, S, 2003, Objective versus subjective coordination in the engineering of agent systems. In Klusch, M, Bergamaschi, S, Edwards, P and Petta, P (eds.), *Intelligent Information Agents: An AgentLink Perspective* (*Lecture Notes in Computer Science, 2586*). Berlin: Springer, pp. 179–202.

Omicini, A *et al*., 2004, Coordination artifacts: environment-based coordination for intelligent agents. In Jennings, N, Tambe, M, Sierra, C, Sonenberg, L, Parsons, S and Sklar, E (eds.), *Proceedings of the 3rd Joint Conference on Autonomous Agents and Multiagent Systems*, New York, USA. Washington, DC: IEEE Computer Society Press, pp. 286–293.

Omicini, A. and Zambonelli, F, 1999, Coordination for Internet application development. *Autonomous Agents and Multiagent Systems* **2**(3), 251–269.

Padgham, L and Winikoff, M, 2003, Prometheus: a methodology for developing intelligent agents. In Giunchiglia, F, Odell, J and Weiß, G (eds.), *Agent-Oriented Software Engineering III* (*Lecture Notes in Computer Science, 2585*), Bologna, Italy. Berlin: Springer.

Papadopoulos, G and Arbab, F, 1998, Coordination models and languages. In Zelkowitz, M (ed.), *Advances in Computers, The Engineering of Large Systems*, vol. 46. Elsevier, Academic Press.

Parunak, V, 1997, Go to the ant: engineering principles from natural agent systems. *Annals of Operations Research* 75, 69–101.

Parunak, V, 2001, The AARIA Agent architecture: from manufacturing requirements to agent-based system design. *Integrated Computer-Aided Engineering* **8**(1).

Platon, E *et al*., 2005, Oversensing with a softbody in the environment: another dimension of observation. In Kaminka, G, Pynadath, D and Geib, C (eds.), *Proceedings of Modeling Others from Observation at International Joint Conference on Artificial Intelligence*, Edinburgh, Scotland.

Ricci, A *et al*., 2003, Activity theory as a framework for MAS coordination. In Petta, P., Tolksdorf, R and Zambonelli, F (eds.), *Engineering Societies in the Agents World III* (*Lecture Notes in Computer Science, 2577*), Madrid, Spain. Berlin: Springer, pp. 96–110.

Ricci, A *et al*., 2004, Agent coordination context: from theory to practice. *Cybernetics and Systems* **2**, 618–623.

Ricci, A *et al*., 2005, Engineering MAS environment with artifacts. In Bordini, R, Dastani, M, Dix, J and Seghrouchni, AEF (eds.), *Proceedings of 3rd International Workshop on Programming Multiagent Systems*, Utrecht, The Netherlands, pp. 163–178.

Sauter, J and Parunak, V, 1999, ANTS in the supply chain. In *Proceedings of the Workshop on Agent-Based Decision Support Managing Internet-Enabled Supply Chain*, Seattle, WA, pp. 1–9.

Schelfthout, K and Holvoet, T, 2005, Views: customizable abstractions for context-aware applications in MANETSs. In Garcia, A, Choren, R, Lucena, C, Romanovsky, A, Holvoet, T and Giorgini, P (eds.), *Software Engineering in Large-Scale Multi-agent Systems*, St Louis, MO. New York: ACM Press.

Schumacher, M, 2001, *Objective Coordination in Multiagent System Engineering, Design and Implementation* (*Lecture Notes in Computer Science, 2039*). Berlin: Springer.

Tummolini, L *et al*., 2005, 'Exhibitionists' and 'Voyeurs' do it better: a shared environment for flexible coordination with tacit messages. In Weyns, D *et al*. (eds.), *Proceedings of the 1st International Workshop on Environments for Multi-agent Systems* (*Lecture Notes in Computer Science, 3374*). Berlin: Springer, pp. 215–231.

Viroli, M *et al*., 2005, Engineering MAS environment with artifacts. In Weyns, D, Parunak, V and Michel, F (eds.), *Proceedings of 2nd International Workshop on Environments for Multiagent Systems*, Utrecht, The Netherlands, pp. 1–16.

Weiss, G, 1998, *Multiagent Systems, A Modern Approach to Distributed Artificial Intelligence*. Cambridge, MA: MIT Press.

Weyns, D and Holvoet, T, 2004, Formal model for situated multiagent systems. *Fundamenta Informaticae* **63**(2–3), 125–158.

Weyns, D *et al*., 2004, Towards active perception in situated multiagent systems. *Applied Artificial Intelligence* **18**(9–10), 867–883.

Weyns, D *et al*. (eds.), 2005a, *Proceedings of the 1st International Workshop on Environments for Multi-agent Systems* (*Lecture Notes in Computer Science, 3374*). Berlin: Springer.

Weyns, D *et al*.. 2005b, Environments for multiagent systems, state-of-the-art and research challenges. In Weyns, D *et al*. (eds.), *Proceedings of the 1st International Workshop on Environments for Multi-agent Systems* (*Lecture Notes in Computer Science, 3374*). Berlin: Springer, pp. 1–47.

Weyns, D *et al*., 2005c, Architectural design of a distributed application with autonomic quality requirements. In Garlan, D, Litoiu, M, Müller, H, Mylopoulos, J, Smith, D and Wong, K (eds.), *Design and Evolution of Autonomic Computing Software*. St Louis, MO: ACM Press.

Weyns, D *et al*., 2005d, Exploiting a virtual environment in a real-world application. In Weyns, D, Parunak, V and Michel, F (eds.), *Proceedings of 2nd International Workshop on Environments for Multiagent Systems*, Utrecht, The Netherlands, pp. 1–18.

Weyns, D *et al*., 2005e, Decentralized control of E'GV transportation systems. In Pechoucek, M, Steiner, D and Thompson, S (eds.), *Proceedings of the 4th Joint Conference on Autonomous Agents and Multiagent Systems, Industry Track*, Utrecht, The Netherlands. New York: ACM Press, pp. 67–75.

Weyns, D *et al*., 2005f, Environments for multiagent systems: beyond infrastructure. In Weyns, D, Parunak, V and Michel, F (eds.), *Proceedings of 2nd International Workshop on Environments for Multiagent Systems*, Utrecht, The Netherlands, pp. 1–17.

Zambonelli, F *et al*., 2003, Developing multiagent systems: the GAIA methodology. *Transactions on Software Engineering and Methodology* **12**(3), 317–370.