On the Role of Environments in Multiagent Systems

Danny Weyns and Tom Holvoet
AgentWise, DistriNet, Katholieke Universiteit Leuven
Celestijnenlaan 200 A, B-3001 Leuven, Belgium
E-mail: {danny.weyns, tom.holvoet}@cs.kuleuven.be

Keywords: multiagent systems, environment, engineering environments

Received: June 30, 2005

For a long time, the role of the environment has been underestimated in multiagent systems research. Originating from research on behavior-based agents 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 elaborate on the role of environments in multiagent systems. We present a model for multiagent systems that puts forward agents and the environment as first-order abstractions. Starting from this model, we elaborate on the logical functionalities of the environment. Competence in engineering environments is a prerequisite to apply environments in practical multiagent system applications. We briefly discuss how current agent-oriented methodologies deal with the environment, and we discuss an approach for engineering environments that puts forward artifacts as building blocks for environments. After that we present the concern-based approach for engineering environments developed in our research group. This approach models the environment as a set of modules that represent different functional concerns of the environment. We illustrate how we have applied this approach in a real-world multiagent system application. The paper concludes with a number of research challenges that are important for the further exploration of environments for multiagent systems.

Povzetek: Opisuje vlogo okolij v multiagentnih sistemih.

1 Introduction

Multiagent systems are an approach to build 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: *an agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives* [56]. This definition stresses the importance of the environment, an agent is not an isolated entity but *exists in* an environment in which it senses and acts. In spite of the fundamental role of the environment in agent systems, most researchers neglect to integrate the environment as a primary abstraction in models and tools for multiagent systems, or minimize its responsibilities [49]. Typically, the responsibilities of the environment are reduced to a message transport system or broker infrastructure. Restricting interaction to inter-agent communication neglects a rich potential of possibilities for the paradigm of multiagent systems.

Opportunities that environments offer have been demonstrated in the domain of behavior-based agents and situated multiagent systems. In behavior-based agent systems, interaction in the environment has been considered as an essential feature for intelligent behavior for a long time [9, 25, 1]. 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 [29, 10, 26, 47]. 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 [49]. Several practical applications have shown how indirect interaction trough 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 [41], network support [6], peer-to-peer (P2P) systems [2], manufacturing control [37], and support for automatic logistic services [55].

Originating from research on behavior-based agents and situated multiagent systems, the importance of the environment in multiagent systems is now gradually being accepted in the multiagent system community in general. For example, [8] argues that the multiagent research community should not only focus attention on making agents smarter but also on making the environment more capable of managing and protecting the conditions in which agents have to operate. Recently, the environment has begun to emerge as the focus of research in its own right [14, 48, 34].

This paper is structured as follows. In Sect. 2, we present a model for multiagent systems that puts forward agents

and the environment as first-order abstractions. Starting from this model, we discuss logical functionalities of the environment in Sect. 3. Section 4 discusses engineering issues of environments and in Sect. 5 we illustrate a real-world application in which the environment plays a central role. Finally, in Sect. 7 we draw conclusions and list a number of research challenges for the further exploration of environments for multiagent systems.

2 The Environment Abstraction

In line with [49], we put forward agents and the environment as *first-order abstractions* in multiagent systems. This allows 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]. Just as the agents, 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 include 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 and coordination, 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.

One problem with the specification of environments 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 model for multiagent-based applications that describes the position of agents and the environment at three levels [54], see figure 1:

- the *multiagent system (MAS) 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.).

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

Below we elaborate on each layer and illustrate that the abstraction of the environment as well as the agents, crosscut the three layers in the model. Before that we introduce a simple file searching system in a P2P network that we use as a running example to illustrate the three-layer model [53]. The idea of this application is to let mobile agents act on behalf of users and browse a shared distributed file system to find requested files. Each user is situated in a particular node (its base). Users can offer files at their base and can send out agents to find files for them. Agents can observe the environment, however, to avoid network overload, agents can perceive the environment only to a limited extent, e.g. two hops from the agent's current position. An agent can perceive nodes and connecting links, bases on nodes, and files available on nodes. Agents can also sense signals. Each base emits such a signal. The intensity of the signal decreases with every hop. Sensing the signal of its base enables an agent to "climb up" the gradient, i.e. move towards its base or alternatively "climb down", i.e. move away from it. Finally, agents can sense pheromones. An agent can drop a file-specific pheromone in the environment when it returns back to its base with a copy of a file. Such a pheromone trail can not only help the agent later on when it needs a new copy of the file, it can also help other agents to find their way to that file. Pheromones evaporate, thereby limiting their influence over time. This is an important property to avoid that agents are misled when a file disappears from a certain node.

2.1 Multiagent System Application Layer

The Multiagent System Application layer contains the *Application Specific Logic*, i.e. Application Agents (AAs) and the Application Environment (AE) of the multiagent system. The AAs are the autonomous entities in the multiagent system, the AE offers a domain specific abstraction to AAs, hiding the complexity of resource access, interaction handling, and consistency management. The AE imposes the rules that regulate domain dynamics. Section 3 elaborates on responsibilities of the AE.

The AAs in the P2P file searching system are the logical entities that are created by the users to search for files in the network. The AE is the logical entity that represents the space in which the AAs perform their job. The AE offers a representation to the AAs of the neighboring nodes and connecting links of the network. The AE also represents the available files, the gradient fields emitted by the bases, and the file-specific pheromones dropped by the agents.

The application logic is typically deployed on top of a *Multiagent System Framework*. The multiagent system framework supports predefined multiagent systems abstractions, such as a particular engine for agent's decision making, support for communication, a model for action, etc. These abstractions can be reused over different applications. In the P2P file searching system, the multiagent system framework layer should provide a pheromone in-

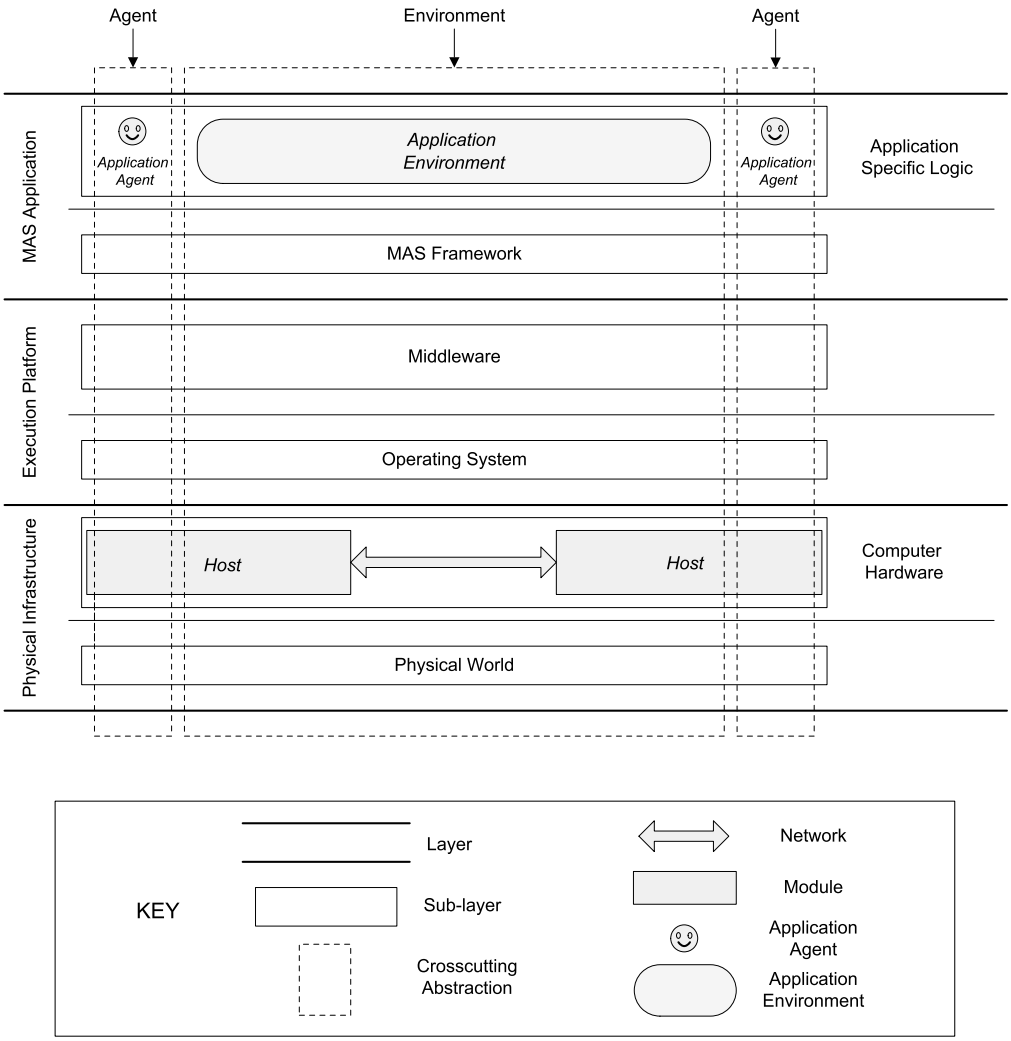

Figure 1: Three-Layer Model for Multiagent Systems.

frastructure and infrastructure for gradient fields. Another example is support for mobility of the agents.

2.2 Execution Platform

The Execution Platform is composed by a *Middleware* on top of an *Operating System*. Middleware serves as the glue between (distributed) components. It provides support for remote procedure calls, threading, transactions, persistence, load balancing, generative communication, etc. In general, middleware offers a software platform on which distributed applications can be executed. 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.

An example of middleware support in the P2P file searching system is a distributed tuple-space infrastructure that provides a basic substrate for the pheromone and gradient field infrastructure. The operating system provides many basic functions, one example is the file system.

2.3 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. In the P2P file sharing system, the physical infrastructure consists of a computer machine on each node and a connecting network. Each machine is an access point to the system for a user.

We refer the interested reader to [54] in which the three-layer model for multiagent systems is applied to several other practical applications.

2.4 Related Models

To our best knowledge, no deployment models for multiagent systems were previously proposed that explicitly discusses the position of agents and the environment. However, several layered models for multiagent system infrastructure are discussed in literature, prominent examples are Retsina [44] and JADE [4]. Here we look at two other examples, the spatial computing stack model applied to TOTA [26], and a model with multiple environments for multiagent systems proposed in [23].

TOTA. In [26], Mamei and Zambonelli introduce the notion of "spatial computing stack" and apply it to the TOTA middleware (Tuples On The Air). The spatial computing stack defines a framework for spatial computing mechanisms at four levels: the physical level at the bottom, the structure level above it, then follows the navigation level, and finally the application level at the top. The "physical level" deals with how components find each other and start communication with each other. In the case of TOTA, a node detects in-range nodes via one-hop message broadcast. The "structure level" is the level at which a spatial structure is built and maintained by components in the physical network. In TOTA, a tuple can be injected from a node. A TOTA tuple is defined in terms of a content and a propagation rule. The content represents the information carried on by the tuple and the propagation rule determines how the tuple should be propagated across the network. Once a tuple is injected it propagates and creates a centered spatial structure in the network representing some spatial feature relative to the source. At the "navigation level" components exploit basic mechanisms to orient their activities in the spatial structure and to sense and affect the local properties of space. TOTA defines an API to allow application components to sense TOTA tuples in their one-hop neighborhood and to locally perceive the space defined by them. Navigation in the space consists of agents acting on the basis of the local shape of specific tuples. At the "application level", navigation mechanisms are exploited by application components to interact and organize their activities. TOTA enables complex coordination tasks in a robust and flexible way. An example is a group of agents that coordinate their respective movements by following locally perceived tuples downhill or uphill resulting in specific formations.

The spatial computing stack model extends over the three layers of the model presented in this paper. The physical level is situated in the Physical infrastructure, the structure and navigation level are situated in the Middleware layer, and the application level finally is situated in the multiagent system Application layer.

Multiple Environments. In [23], Gouaich and Michel make a statement to model different "aspects" of the environment with different environments. Essentially, the authors consider different instances of an environment within a single multiagent system. As an illustrative example they refer to the three-layer model described in this paper and associate with each layer in this model a separate environment. The authors state that considering different environments for different aspects improves modularity and extensibility of multiagent systems. Another example discussed in the paper is the AGRE [18] model. AGRE considers a spatial, a temporal, and an organizational aspect in one environment abstraction. Gouaich and Michel state that when a new aspect is identified and must be integrated in the AGRE model, the entire model must be revised.

Unfortunately, the authors do not explain what the added value is of considering different environments for different aspects instead of dealing with different aspects *in an disciplined manner* in one environment abstraction. The authors also keep silence on crosscutting issues related to different aspects, and how the approach with multiple environments deals with this problem.

Explicitly dealing with different *concerns* of a software system is good software engineering practice. However, it is unclear whether it is useful to associate *separate* environments with different environmental concerns in a multiagent system. One way to match the approach of different environments with the three-layer model is to consider the environment as a multidimensional entity with different dimensions for different aspects/concerns, rather than a separate environment for each aspect/concern.

3 The Role of Environments in Multiagent Systems

Having clarified how the agents and the environment are first-order abstractions that span the application logic, the execution platform and the physical infrastructure, we now elaborate on the logical functionalities of the environment.

The functionalities of the environment we discuss in this section are located in the multiagent system Application layer, i.e. the top layer in Fig. 1. Several functionalities may seem quite natural responsibilities of environments. We want to stress, however, that in practice the functionalities we put forward are often dealt with in an implicit or ad hoc way. Our goal is to make the logical functionalities *explicit*, i.e. as concerns of environments as first-order abstractions. Not every functionality we discuss is relevant for every possible environment. In practice, it is up to the designer to decide which functionalities should be integrated in the environment model for the domain at hand. Finally, we want to underline that the proposed list of functionalities is not intended to be complete but rather serves as a start to explore the many-sided role of environments in multiagent systems.

3.1 Structuring

The environment is first of all a shared "space" for the agents, resources and services, which structures the whole

system. Resources are objects with a specific state. Services are considered as reactive entities that encapsulate functionality. The agents as well as resources and services are dynamically interrelated to each other. It is the role of the environment to define the rules which these relationships have to comply to. As such the environment acts as a *structuring* entity for the multiagent system. This structuring can take different forms: it can be spatial, see e.g. [10][3], but also organizational, e.g. [17][57], or the environment can be structured as a mediating entity as e.g. in [20][24]. Specific properties can be defined separately for each space, such as positions, locality, groups or roles. Structuring is a fundamental functionality of the environment. The structure of the environment is a design choice that depends upon the requirements of the domain at hand, and the designer should deal with it explicitly.

3.2 Managing Resources and Services

Besides structuring, the environment is also in charge of enabling and controlling the access to resources and services. In general, resources can be read/perceived, written/modified or consumed by agents. Services on the other hand provide functionality to the agents on their request.

The extent to which agents are able to access a particular resource or service may depend on several factors such as the nature of the resource or service, the capabilities of the agent, the (current) interrelationships with other resources, services or agents, etc. In general, the access to the resources and services can be described by a set of laws defined by the domain at hand, see e.g. [19][47].

3.3 Providing Observability

Contrary to agents, the environment must be observable, i.e. agents must be able to inspect their neighborhood. Besides the observation of resources and services, agents may even be able to observe the actions of other agents [45]. In general, agents should be able to inspect the environment according to their current preferences. Examples of selective perception are [53] where "foci" are proposed to enable agents to perceive their environment related to their current tasks, and [24, 42] where "views" are proposed as selector for perception. Perception is constrained not only by agents' capabilities, but also by environmental properties (which in fact reflect properties of the problem domain). In [53] the environmental constraints are made explicit in the form of "perceptual laws".

Related to observability is the semantic description of the domain. This can be done by defining an environment ontology, see e.g. [12]. The ontology must cover the structure of the environment as well as the observable characteristics of resources, services and agents, their interrelationships, and possibly the regulating laws. In an open system, it would be useful for agents to be able to understand at run-time a new environment they are discovering. For symbolically-oriented agents, an explicit ontology should be available to the agents to enable them to interpret their environment and reason about it. For reactive/behavior-based/stigmergic agents, the designer/developer applies the ontology to encode the agents' internal structures. As such, these kinds of agents have an implicit ontology that enables them to make decisions.

3.4 Enabling Communication

Communication is inextricably bound up with multiagent systems. The environment defines concrete means for agents to communicate. Communication can take different forms. The most used scheme is a message-passing style from one agent to the other. In generative or indirect communication, agents produce communication objects in the environment and consume them to read them. Well-known properties of generative communication are name, space and time decoupling. [22] extends this list of properties with locality and non-intentionality. An important other approach of communication is based on stigmergy [36]. Each of these types of communication has its own pros and cons. Designers should be aware of the potency as well as the impact of each type of communication for their solution. Selecting a particular type of communication should be an architectural choice, determined by the requirements of the problem domain at hand.

3.5 Maintaining Environmental Processes

Besides the activity of the agents, the environment can assign particular activities to resources as well. A digital pheromone, for example, is a dynamic structure as it aggregates with additional pheromone that is dropped, it diffuses in space and it evaporates over time. Other examples are a rolling ball that moves on, or the local temperature that evolves over time. Maintaining such dynamics is an important functionality of the environment, it is useful for self-organization, see e.g. [10, 43].

3.6 Ruling the Multiagent System

The environment can define different types of rules or laws on all entities in the multiagent system. Environment rules are a powerful tool to express the capabilities an environment needs to ensure consistency in the system. Rules may restrict access to specific resources or services to particular types of agents, or determine the outcome of agents' interactions.

Dealing with interactions in multiagent systems in general is a very complex matter. In [27], Minsky and Ungureanu point out the difficulties to control the activities of agents operating in distributed systems and propose coordination policies to deal with control. According to the authors, coordination policies need to be formulated explicitly rather than being implicit in the code of the agents involved and they should be enforced by means of a generic,

broad spectrum mechanism. The environment is the natural candidate to embed such control mechanism.

In electronic institutions [28], agents interact through agent group meetings that are called scenes. Interactions in a scene have to follow a well-defined communication protocol. Scenes can be composed in a performative structure. The specification of a performative structure contains a description of how the different roles can legally move from scene to scene. Agents within a performative structure may participate in different scenes at the same time with different roles. Agent actions in the context of an institution may have consequences that either limit or enlarge its subsequent acting possibilities. Such consequences will impose obligations to the agents and affect its possible paths within the performative structure. The environment can define and enforce the rules imposed on the movements and interactions of agents in an electronic institution.

A particular problem is the regulation of simultaneous actions. If we allow multiple agents to act in the environment in parallel, we need explicit models to deal with actions that range far beyond the scope of state changes based on simple individual manipulation of objects. [19] and [47] discuss models for simultaneous actions. Central to these models are (1) the distinction between the products of the agents' behavior on the one hand and the reaction of the environment on the other hand, and (2) a set of explicitly defined laws that govern the effects of the actions of the agents. These models resolve a number of fundamental issues with respect to actions in multiagent systems, however, dealing with actions in multiagent systems needs extensive further research to grow into full maturity.

4 Engineering Environments

An important condition to apply environments in practical multiagent system applications is competence in the engineering environments. Disciplined design practices for agents in general are in their infancy, and extending these techniques to environments greatly increases the scope of work to be done [49]. In this section, we first give a brief overview how current agent oriented methodologies deal with the environment. After that we discuss two proposals for engineering environments.

4.1 Environments in Agent-Oriented Software Methodologies

Popular methodologies such as Prometheus [35], Tropos [21] or Adelfe [11] 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 [30] and GAIA v.2. [57].

4.1.1 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 be or not the same as the one of the agent system, given the assumption of decentralized control).

4.1.2 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" [57] pp. 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 in 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 modelling 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, [57] pp.23 states "it is difficult to provide general modelling 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 con-

suming (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. The authors of [57] confirm that in realistic development scenarios, the analyst would choose to provide a more detailed and structured view of environmental resources.

4.2 Summary

Although SODA and GAIA explicitly put forward the environment as a first-order abstraction 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.

4.3 Engineering Approaches for Environments

In this section, we zoom in on two approaches to engineer environments. 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 models the environment as a composition of modules that represent different functional concerns of the environment, such as communication, perception, actions and interaction.

4.3.1 Artifacts as Building Blocks for Engineering Environments

Inspired by Activity Theory [33], and building upon the work on coordination artifacts [32, 39], the notion of *artifact* has been proposed as an abstract building block for modeling and engineering environments [34, 46]. 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. This characterization fits the basic distinction made in Distributed Artificial Intelligence [13] between goal-oriented entities (agents) which pro-actively 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. 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 systems. As a first rough classification, artifacts can be classified in three categories. A first class are *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 services, such as a web service) at the abstraction level of the agents. A second class are *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 support social activities in the multiagent system [33, 32]. Finally, a 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. E.g., 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.

Concrete examples of artifacts in the context of the general purpose coordination infrastructure TuCSoN [26] are a *Tuple Centre* [21] and a *Agent Coordination Context* [40]. A tuple centre is an example of a coordination artifact which 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.

4.3.2 Concern-Based Engineering of Environments

The second approach models the environment as a set of modules that represent different functional concerns of the environment [47, 50]. Fig. 2 depicts a high-level module view of the environment architecture.

The *PerceptGenerator* module is responsible for perception [53]. 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 modelled 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 examples is a communication law that specifies

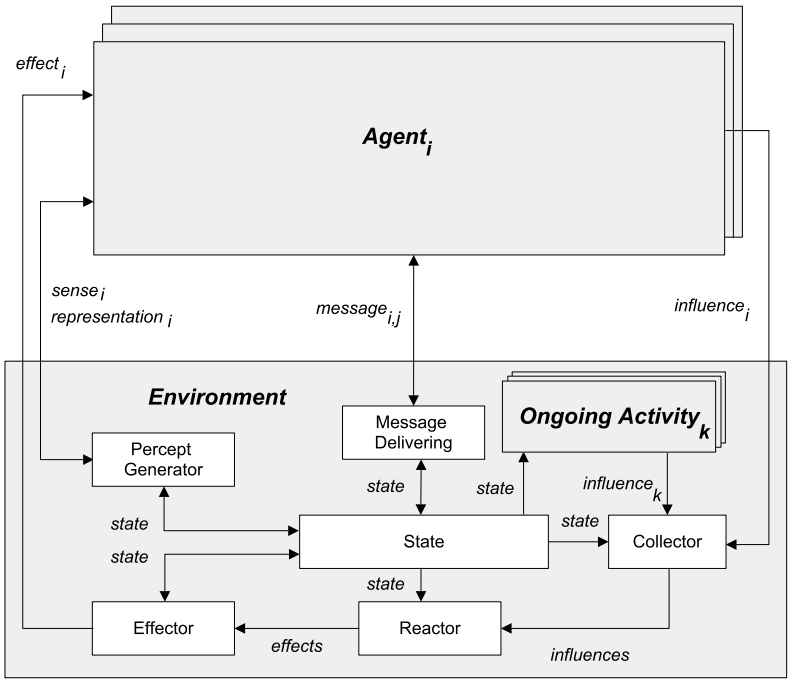

Figure 2: Concern-based modularization of the environment.

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 J. Ferber and J.P. Müller [19]. According to this model, agents produce influences into 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 activity in the system. The *Collector* module collects the influences of all simultaneously performed activity in the multi-agent 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 [16, 47]. 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 that applies the outcome of the interaction, i.e. it updates the state of the environment and passes the effects to the applicable agents.

Ongoing Activities correspond to environmental processes as discussed in Sect. 3. 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 notice that the module view of the environment architecture as depicted in Fig. 2 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 where the state of the environment represents resources in the physical world.

5 Applying the Environment in a Real-World Application

In this section, we illustrate how we have applied the approach of concern-based engineering of environments to an automated transportation system for warehouse logistics. This real-world application is developed in a joint R&D project between the AgentWise research group and Egemin, a manufacturer of automating logistics services in warehouses and manufactories [15, 52].

The automated transportation system uses automatic guided vehicles (AGVs) to transport loads through a warehouse. Typical applications are distributing incoming goods to various branches, or distributing manufactured

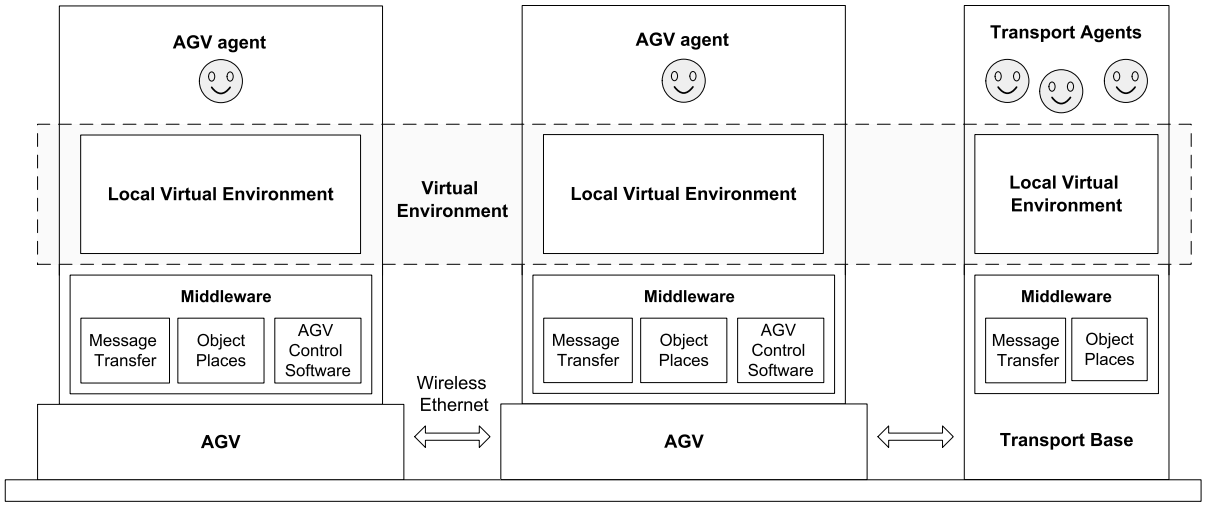

Figure 3: High-level model of the AGV transportation system.

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 (such as staying on track on a path, turning, and determining the current position, etc.), is handled by the AGV control software. Fig. 3 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 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 have been synchronized recently with each other, while others may not. From the agent perspective, the virtual environment appears as one entity. The synchronization of the state of neighboring local virtual environments is supported by the ObjectPlaces middleware [42].

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

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.

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 come in 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.

Collision Avoidance. AGV agents avoid collisions by coordinating with other agents through the virtual environment. AGV agents mark the path they are going to drive 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. [51] discusses 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.

6 Conclusions and Challenges

There is a growing awareness in the multiagent research community that the environment plays a crucial role in multiagent systems. In this paper, we discussed the role of environments 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 to 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 to maintain 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 mutiagent 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 [14, 34].

The engineering of environments 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. Initial work in that direction has been conducted, e.g. [5]. 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. Exploring work in that direction is on its way, see e.g. [7].

Besides the research work, 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.

Acknowledgement

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. A word of appreciation also goes to the anonymous reviewers for their usefully comments to improve this paper.

References

[1] R.C. Arkin. *Behavior-based robotics*. Massachusetts Institute of Technology, MIT Press, Cambridge, MA, USA, 1998.

[2] O. Babaoglu, H. Meling, and A. Montresor. Anthill: A framework for the development of agent-based Peer-to-Peer systems. In *Proceedings of the 22nd International Conference on Distributed Computing Systems*, pages 15–22, Vienna, Austria, 2002. IEEE Computer Society, Digital Library.

[3] S. Bandini, S. Manzoni, and G. Vizzari. A spatially dependent communication model for ubiquitous systems. In Weyns et al. [48], pages 74–90.

[4] F. Bellifemine, A. Poggi, and G. Rimassa. Developing multi-agent systems with a FIPA-compliant agent framework. *Software - Practice and Experience*, 31(2):103–128, 2001.

[5] C. Bernon, M. Cossentino, and J. Pavón. An Overview of Current Trends in European AOSE Research. *In this volume*.

[6] E. Bonabeau, F. Henaux, S. Guérin, D. Snyers, P. Kuntz, and G. Theraulaz. Routing in telecommunications networks with ant-like agents. In *Proceedings of the 2nd International Workshop on Intelligent Agents for Telecommunication a Applications*, pages 60–71, Paris, France, 1998. Springer, London, UK.

[7] R. Bordini, L. Braubach, A. El Fallah-Seghrouchni, M. Dastani, J. Gomez-Sanz, J. Leite, G. O'Hare, A. Pokahr, and A. Ricci. A survey on languages and platforms for MAS implementation. *In this volume*.

[8] J. M. Bradshaw, N. Suri, A. Ca nas, R. Davis, K. Ford, R. Hoffman, R. Jeffers, and T. Reichherzer. Terraforming Cyberspace. *Computer*, 34(7):48–56, 2001.

[9] R. Brooks. Intelligence without representation. *Artificial Intelligence*, 47:139Ű–159, 1991.

[10] S. Brueckner. *Return from the ant, Synthetic ecosystems for manufacturing control*. Ph.D Dissertation, Humboldt University, Berlin, Germany, 2000.

[11] S. Peyruqueou G. Picard C. Bernon, M. P. Gleizes. ADELFE: A methodology for adaptive multiagent systems engineering. In P. Petta, R. Tolksdorf, and F. Zambonelli, editors, *Engineering Societies in the Agents World III*, volume 2577 of *Lecture Notes in Computer Science*, pages 156–169, Madrid, Spain, 2003. Springer, Berlin, Heidelberg, Germany.

[12] P. Chang, K. Chen, Y. Chien, E. Kao, and V. Soo. From reality to mind: A cognitive middle layer of environment concepts for believable agents. In Weyns et al. [48], pages 57–73.

[13] R. Conte and C. Castelfranchi, editors. *Cognitive and social action*. UCL Press, University College, London, UK, 1995.

[14] E4MAS. International workshop series on Environments for Multiagent Systems. *http://www.cs.kuleuven.ac.be/∼distrinet/events/e4mas/*, 8/2005.

[15] Egemin Modular Controls Concept. EMC^2 project, Flemish Institute for the Advancement of Scientific-Technological Research in the Industry, IWT, Belgium. *http://emc2.egemin.com/*, 8/2005.

[16] J. Ferber. *An introduction to distributed artificial intelligence*. Addison-Wesley, London, UK, 1999.

[17] J. Ferber, O. Gutknecht, and F. Michel. From agents to organizations: An organizational view of multiagent systems. In P. Giorgini, J. P. Müller, and J. Odell, editors, *Agent-Oriented Software Engineering IV*, volume 2935 of *Lecture Notes in Computer Science*, pages 214–230, Melbourne, Australia, 2003. Springer, Berlin, Heidelberg, Germany.

[18] J. Ferber, F. Michel, and J. Baez. AGRE: Integrating environments with organizations. In Weyns et al. [48], pages 48–56.

[19] J. Ferber and J. P. Müller. Influences and reaction: A model of situated multiagent systems. In M. Tokoro, editor, *Proceedings of the 2th International Conference on Multi-agent Systems*, pages 72–80, Kyoto, Japan, 1996. American Association for Artificial Intelligence, AAAI Press, Menlo Park, California, USA.

[20] D. Gelernter and D. Carrierro. Coordination languages and their significance. *Communications of the ACM*, 35(2), 1992.

[21] F. Giunchiglia, J. Mylopoulos, and A. Perini. The TROPOS software development methodology: Processes, models and diagrams. In C. Castelfranchi and W. L. Johnson, editors, *Proceedings of the 1st Joint Conference on Autonomous Agents and Multiagent Systems*, pages 35–36, Bologna, Italy, 2002. ACM Press, New York, NY, USA.

[22] D. Goldin and D. Keil. Toward domain-independent formalization of indirect interaction. In *Proceedings of the 13th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 393–394, Modena, Italy, 2004. IEEE Computer Society, Digital Library.

[23] A. Gouaich and F. Michel. Towards a unified view of environment(s) within multiagent systems. *In this volume*.

[24] C. Julien and G. C. Roman. Egocentric context-aware programming in ad hoc mobile environments. In *Proceedings of the 10th Symposium on Foundations of Software Engineering*, pages 21–30, Charleston, South Carolina, USA, 2002. ACM Press, New York, NY, USA.

[25] P. Maes. Modeling adaptive autonomous agents. *Artificial Life*, 1(1-2):135–162, 1994.

[26] M. Mamei and F. Zambonelli. Programming pervasive and mobile computing applications with the TOTA middleware. In *Proceedings of the 2nd International Conference on Pervasive Computing and Communications*, pages 263–276, Orlando, Florida, 2004. IEEE Computer Society, Washington, DC, USA.

[27] N. Minsky and V. Ungureanu. Law-governed interaction: a coordination and control mechanism for heterogeneous distributed systems. *ACM Transactions on Software Engineering Methodologies*, 9(3):273–305, 2000.

[28] P. Noriega and C. Sierra. Electronic institutions: Future trends and challenges. In *Proceedings of the 6th International Workshop on Cooperative Information Agents*, volume 2446 of *Lecture Notes in Computer Science*, pages 14–17. Springer-Verlag, London, UK, 2002.

[29] J. Odell, V. Parunak, M. Fleischer, and S. Breuckner. Modeling agents and their environment. In F. Giunchiglia, J. Odell, and G. Weiß, editors, *Agent-Oriented Software Engineering III*, volume 2585 of *Lecture Notes in Computer Science*, pages 16–31, Bologna, Italy, 2003. Springer, Berlin, Heidelberg, Germany.

[30] A. Omicini. SODA: Societies and infrastructures in the analysis and design of agent-based systems. In P. Ciancarini and M. Wooldridge, editors, *Agent-Oriented Software Engineering*, volume 1957 of *Lecture Notes in Computer Science*, pages 185–193, Limerick, Ireland, 2001. Springer, Berlin, Heidelberg, Germany.

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

[32] A. Omicini, A. Ricci, M. Viroli, C. Cristiano, and L. Tummolini. Coordination artifacts: Environment-based coordination for intelligent agents. In N. Jennings, M. Tambe, C. Sierra, L. Sonenberg, S. Parsons, and E. Sklar, editors, *3rd Joint Conference on Autonomous Agents and Multiagent Systems*, pages 286–293, New York, NY, USA, 2004. IEEE Computer Society, USA.

[33] A. Omicini and F. Zambonelli. Coordination for Internet application development. *Autonomous Agents and multiagent systems*, 2(3):251–269, 1999.

[34] AgentLink Technical Forum Group on Environments for Multiagent Systems. *http://www.cs.kuleuven.ac.be/∼distrinet/events/e4mas/tfg2005/*, 8/2005.

[35] L. Padgham and M. Winikoff. Prometheus: A methodology for developing intelligent agents. In F. Giunchiglia, J. Odell, and G. Weiß, editors, *Agent-Oriented Software Engineering III*, volume 2585 of *Lecture Notes in Computer Science*, Bologna, Italy, 2003. Springer, Berlin, Heidelberg, Germany.

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

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

[38] A. Ricci, A. Omicini, and E. Denti. Activity theory as a framework for MAS coordination. In P. Petta, R. Tolksdorf, and F. Zambonelli, editors, *Engineering Societies in the Agents World III*, volume 2577 of *Lecture Notes in Computer Science*, pages 96–110, Madrid, Spain, 2003. Springer, Berlin, Heidelberg, Germany.

[39] A. Ricci and M. Viroli. Coordination artifacts: A unifying abstraction for engineering environment-mediated coordination in MAS. *In this volume*.

[40] A. Ricci, M. Viroli, and A. Omicini. Agent coordination context: From theory to practice. *Cybernetics and Systems*, 2:618–623, 2004.

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

[42] K. Schelfthout and T. Holvoet. Views: Customizable abstractions for context-aware applications in MANETSs. In A. Garcia, R. Choren, C. Lucena, A. Romanovsky, T. Holvoet, and P. Giorgini, editors, *Software Engineering in Large-Scale Multi-agent Systems*, St. Louis, USA, 2005. ACM Press, Digital Library.

[43] G. Di Marzo Serugendo, M. P. Gleizes, and A. Karageorgos. Self-organisation and emergence in MAS: An overview. *In this volume*.

[44] K. Sycara, M. Paolucci, M van Velsen, and J. Giampapa. The Retsina MAS infrastructure. *Autonomous Agents and Multi-Agent Systems*, 7(1-2):29–48, 2003.

[45] L. Tummolini, C. Castelfranchi, A. Omicini, A. Ricci, and M. Viroli. "Exhibitionists" and "Voyeurs" do it better: A shared environment for flexible coordination with tacit messages. In Weyns et al. [48], pages 215–231.

[46] M. Viroli, A. Ricci, and A. Omicini. Engineering MAS environment with artifacts. In D. Weyns, V. Parunak, and F. Michel, editors, *Proceedings of 2nd International Workshop on Environments for Multiagent Systems*, pages 1–16, Utrecht, The Netherlands, 2005.

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

[48] D. Weyns, V. Parunak, and F. Michel, editors. *Proceedings of the 1st International Workshop on Environments for Multi-Agent Systems*, volume 3374 of *Lecture Notes in Computer Science*, Berlin, Heidelberg, Germany, 2005. Springer.

[49] D. Weyns, V. Parunak, F. Michel, T. Holvoet, and J. Ferber. Environments for multiagent systems, State-of-the-art and research challenges. In Weyns et al. [48], pages 1–47.

[50] D. Weyns, K. Schelfthout, and T. Holvoet. Architectural design of a distributed application with autonomic quality requirements. In D. Garlan, M. Litoiu, H. M´ller, J. Mylopoulos, D. Smith, and K. Wong, editors, *Design and Evolution of Autonomic Computing Software*, St. Louis, USA, 2005. ACM Press, Digital Library.

[51] D. Weyns, K. Schelfthout, and T. Holvoet. Exploiting a virtual environment in a real-world application. In D. Weyns, V. Parunak, and F. Michel, editors, *Proceedings of 2nd International Workshop on Environments for Multiagent Systems*, pages 1–18, Utrecht, The Netherlands, 2005.

[52] D. Weyns, K. Schelfthout, T. Holvoet, and T. Lefever. Decentralized control of E'GV transportation systems. In M. Pechoucek, D. Steiner, and S. Thompson, editors, *4th Joint Conference on Autonomous Agents and Multiagent Systems, Industry Track*, pages 67–75, Utrecht, The Netherlands, 2005. ACM Press, New York, NY, USA.

[53] D. Weyns, E. Steegmans, and T. Holvoet. Towards active perception in situated multiagent systems. *Applied Artificial Intelligence*, 18(9-10):867–883, 2004.

[54] D. Weyns, G.. Vizzari, and T. Holvoet. "Environments for multiagent systems: Beyond infrastructure. In D. Weyns, V. Parunak, and F. Michel, editors, *Proceedings of 2nd International Workshop on Environments for Multiagent Systems*, pages 1–17, Utrecht, The Netherlands, 2005.

[55] Whitestein Technologies, Living Systems. *http://www.whitestein.com/pages/index.html*, 8/2005.

[56] M. Wooldridge and N. Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.

[57] F. Zambonelli, N. Jennings, and M. Wooldridge. Developing multiagent systems: The GAIA methodology. *Transactions on Software Engineering and Methodology*, 12(3):317–370, 2003.