Auton Agent Multi-Agent Syst (2007) 14:5–30
DOI 10.1007/s10458-006-0012-0

Environment as a first class abstraction in multiagent systems

Danny Weyns · Andrea Omicini · James Odell

Published online: 24 July 2006
Springer Science+Business Media, LLC 2006

Abstract The current practice in multiagent systems typically associates the environment with resources that are external to agents and their communication infrastructure. Advanced uses of the environment include infrastructures for indirect coordination, such as digital pheromones, or support for governed interaction in electronic institutions. Yet, in general, the notion of environment is not well defined. Functionalities of the environment are often dealt with implicitly or in an ad hoc manner. This is not only poor engineering practice, it also hinders engineers to exploit the full potential of the environment in multiagent systems.

In this paper, we put forward the environment as an explicit part of multiagent systems. We give a definition stating that the environment in a multiagent system is a first-class abstraction with dual roles: (1) the environment provides the surrounding conditions for agents to exist, which implies that the environment is *an essential part* of every multiagent system, and (2) the environment provides an *exploitable design abstraction* for building multiagent system applications. We discuss the responsibilities of such an environment in multiagent systems and we present a reference model for the environment that can serve as a basis for environment engineering. To illustrate the power of the environment as a design abstraction, we show how the environment is successfully exploited in a real world application. Considering the environment as a first-class abstraction in multiagent systems opens up new horizons for research and development in multiagent systems.

Keywords Environment in multiagent systems · Definition, responsibilities, reference model of the environment

D. Weyns (✉)
Katholieke Universiteit Leuven, Leuven, Belgium
e-mail: danny.weyns@cs.kuleuven.be

A. Omicini
Università di Bologna, Cesena, Italy
e-mail: andrea.omicini@unibo.it

J. Odell
Intelligent Automation Inc., Ann Arbor, MI, USA
e-mail: email@jamesodell.com

1 Introduction

All non-agent elements of a multiagent system (MAS) are typically considered to be part of the MAS environment. Such elements can include databases, Web services, communication infrastructures, and the topology of a spatial domain. Additionally, several classes of MAS use the environment as a means for agents to share information and coordinate their behavior. Examples of this include infrastructures that employ indirect coordination using digital pheromones and support governed interaction in electronic institutions.

Yet, while the notion of *environment* is understood to some degree in the MAS community, it is not well defined. Researchers and engineers associate the environment with an amalgam of resources, services, infrastructure, and so on. For the most part, the environment is an implicit part of MAS that is often dealt with in an ad hoc way. Since the environment accounts for a variety of responsibilities in MAS, we claim that the environment should be considered as an explicit part of MAS. This both results in better engineering practice and opens perspectives to exploit the environment in the design of MAS.

In this paper, we contend that the environment is an explicit part of MAS. We present a definition of the environment stating that the environment is a *first-class abstraction* in MAS with a dual role that provides: (1) the surrounding conditions for agents to exist (which implies that the environment is an essential part of every MAS), and (2) an exploitable design abstraction to build MAS applications. As such, the environment becomes a building block for MAS that engineers can use creatively in the design of a MAS. Distinguishing clearly between the responsibilities of agent and environment both supports separation of concerns in MAS and helps to manage the huge complexity of engineering real-world applications.

This paper is structured as follows. In Sect. 2, we give an overview of the role of the environment in MAS. From this overview, we derive three levels of support that can be provided by the environment in MAS. Section 3 introduces the definition of the environment and discusses important responsibilities of the environment in MAS. We describe a reference model for the environment that provides a common frame to discuss environment issues and serves as a basis for environment engineering. Section 4 illustrates how the environment is successfully exploited as a design abstraction in a real-world application. Finally, we draw conclusions and look at challenges for future research on environments in Sect. 5.

2 Role of the environment in MAS: An overview

In this section, we discuss the evolution of the role of environment in MAS and identify three levels of support that can be provided by the environment in MAS.

2.1 Evolution of the role of the environment in agent systems

Different perspectives exist on the roles which the environment plays in agent systems. We examine these different perspectives within two separate contexts: situated agent systems and cognitive agent systems.

2.1.1 Role of the environment in situated agent systems

With situated agent systems, we refer to the class of systems in which agents perform *situated actions*. The term situated actions emphasizes the interrelationship between an action and its context of performance. The term was first introduced by L. Suchman in [72] and later generally adopted in agent systems, see e.g. [5, 26, 39, 81, 90].

The environment as the "external world". Researchers and developers of situated agent systems have always devoted pertinent attention to the role of environment. Historically, situated agency originates from reactive robotics that emerged in the mid-1980s as an approach to build autonomous robots that could act efficiently in dynamic environments. The first generation of agent architectures coupled perception directly to action, enabling real-time reaction in the environment, see e.g. [10, 64]. Later, behavior-based architectures were developed that support runtime arbitration between parallel executing behaviors, allowing the agents to act efficiently in more complex environments. Classic examples of this are [2, 63, 39]. The initial reactive agent systems stressed the importance of *environmental dynamics*. The environment was considered as *external* to the system, i.e., the environment was not an explicit part of the models or architectures.

Since the time reactive agent research began, there has been an ongoing discussion about the exploitation of internal world models in agent architectures. Brooks argues against the need for any kind of world model at all [10]. Steels [71], however, states that "autonomous agents without an internal model of the environment will always be severely limited." Arkin [3] argues that "despite the assumptions of early work in reactive control, representational knowledge *is important* for robot navigation." Architectures for hybrid agents [40] integrate cognition (reasoning over internal representations of the world and planning) with reactivity (real-time reaction to stimuli) aiming to combine the advantages of planning and quick responsiveness. Today, hybrid architectures are a common approach in the robotics domain [4].

The environment as a medium for coordination. Since the early 1990s, researchers of situated agency have been investigating systems in which multiple agents work together to realize the system's functionality. In these systems, the agents exploit the environment to share information and coordinate their behavior. In its basic form, agents are driven by what they perceive in the environment. Each agent follows a set of simple behavioral rules, resulting in a *collective reactive behavior*, see e.g. [44, 61]. In such systems, the aggregate behavior of the MAS emerges from the local behavior of agents. For example, Zeghal and Ferber [93] use vector fields to control the landing and movements of a large group of aircrafts in a simulation. In this approach, each agent is guided by a potential field that it constructs based on attracting and repulsing forces which result from goals and obstacles, respectively.

In *stigmergic agent systems*, the environment serves as a medium for coordination. Stigmergic agents coordinate their behavior through the manipulation of marks in the environment in a similar way as social ants coordinate [32]. The environment is an active entity that maintains processes independent of the activity of the agents [56]. A classic example of stigmergic coordination are digital pheromones [9, 11, 57] which software agents use to coordinate their behavior. A digital pheromone is a dynamic structure in the environment that aggregates with additional pheromone that is dropped, diffuses into space, and evaporates over time. Agents can use pheromones to dynamically form paths to locations of interest. Digital pheromones combine reinforcement of interesting information (by the agents) with decay of information over time (truth maintenance by the environment). Another well-established approach of stigmergic coordination is using computational fields [19, 42, 80]. In this approach, the movements of agents are driven by abstract force fields that are spread in the environment (by agents or the environment itself). Agents coordinate their behavior by following the shape of the fields. Environment dynamics and movements of the agents induce changes in the surface of the fields, realizing a feedback cycle that influences the agents' movement. This feedback cycle enables the system (agents and environment) to self-organize.

While in collective reactive behavior, the environment provides the context that drives the agents; in stigmergic agent systems, however, the environment is an active medium that enables and constrains the interaction among agents.

Environment architecture. Since the mid-1990s, a family of MAS that is known as *situated multiagent systems* has been the subject of active research. Situated MAS emphasize the importance of architecture for agents and the environment, see e.g., [5, 26, 81]. Advanced types of situated agents support social behavior enabling them to set up explicit collaborations [88, 70]. The environment architecture in situated MAS includes: functionality for perception management, message delivering, action handling, and maintenance processes that manage state in the environment independently of agents. Agent interactions in the environment are subject to laws. Laws can represent domain-specific constraints, for example bandwidth limitations of the network. However, laws can also be used as a design instrument to impose rules in the MAS, for instance an interaction law may impose a policy on the access of a shared resource. In situated MAS, the environment is an explicit architectural abstraction with specific responsibilities that differ from agent responsibilities.

2.1.2 Perspective on the environment in cognitive agent systems

The role of the environment in cognitive agency is mainly concerned with the agent's cognitive model of the environment, the agent's action over the environment, and the practical reasoning over these actions. From the point of view of an individual agent, it is common to consider everything outside the agent—including the other agents—as the environment. Cognitive agents typically have subjective dependencies which refer to intra-agent dependencies towards other agents [68, 52]. The management of these subjective dependencies refers to subjective coordination, such as negotiation techniques. Yet, cognitive MAS are also built by objective dependencies which refer to inter-agent dependencies, i.e., the configuration of the system in terms of the basic means for interaction, organization of spaces, etc. The management of these dependencies refers to objective coordination, because they are external to the agents. Objective coordination is essentially concerned with the environment.

Here we consider five different perspectives on the role of the environment in cognitive agent systems: (1) the environment as a container and a means for communication, (2) the environment as an organizational layer, (3) the environment as a coordination infrastructure for cognitive agents, (4) Markovian environments, and finally (5) task environments. Successively, we touch upon each of these perspectives.

Environment as a container and a means for communication. The majority of researchers on cognitive agent systems consider the environment as a means for agent communication (i.e., message exchange) and a container for agents and resources. According to Huhns and Stephens [35], MAS environments provide an infrastructure specifying communication and interaction protocols and a container for agents that may be self-interested or cooperative. The key building block of an environment in the FIPA reference model [27] is the *agent platform* that consists of: (1) a directory facilitator acting as a yellow pages service for the agents to advertise and discover services, (2) an agent management system that enables agents to register on the platform and to locate one another (i.e., a white pages service) and that controls resource usage, and (3) a message transport system, i.e., a communication service for local and inter-platform message exchange. Jade (1999) is a FIPA compliant Java platform that is widely used in academics and industrial projects. The Jade platform provides a layer that shields agents from the complexity of the underlying execution system. Jade includes a distributed naming service, a yellow pages service, and a distributed message transfer system.

Environment as an organizational layer. Several researchers associate the environment with organizational concepts in MAS, such as organizations, groups, roles. Examples are [18, 30, 36, 49, 51, 92]. From an organizational perspective, a MAS can naturally be considered and designed as a *computational organization* [91] that defines a framework for agent activities. That is, the organization imposes a set of constraints on the behavior of agents, and offers a set of facilities and services that agents may use. Ferber et al. [24] make a distinction between agent-centered MAS and organization-centered MAS. In organization-centered MAS, the organization acts as (1) a "dynamic framework" where agents may enter and leave organizations at will, and (2) an environment for resources, services, communications and tasks, through the concepts of both groups and roles. In [25], Ferber et al. consider an organization as a special kind of environmental zone, called an area. Actions are associated with organizations, i.e., communicating, entering a group or leaving it, playing a role, and creating a group.

Environment as a coordination infrastructure for cognitive agents. Coordination infrastructures for cognitive agents are investigated for a long time. Classical blackboard systems were the first type of mediated interaction models proposed by AI researchers [16, 21]. A blackboard is an intermediary data repository that enables cooperating software modules to communicate indirectly and anonymously. In contrast to blackboard systems, tuple-based technologies use associative access to a shared dataspace for communication and synchronization purposes. Tuplespaces were first introduced in Linda [31]. Agents in Linda communicate by putting tuples in, and removing them from a shared space, i.e., the tuplespace. Throughout the years, variants for distributed computing appeared, such as Sun's JavaSpaces [28], TuCSoN [55], MARS [12], and LIME [46].

Software infrastructures [53] provide reusable solutions for coordination of cognitive agents. Software infrastructures rule MAS by defining the laws that govern the observable behavior of agents and their mutual interaction, and providing MAS engineers with the abstractions to encapsulate them. Infrastructure laws may be implicit for agents, such as laws that determine the number of agents that can enter a MAS and participate in its activity. Laws may also be explicit and regulate the interactions in agent societies via rules and norms.

Coordination artifacts [54] generalize over different coordination models and languages. They embody and enact the laws of MAS coordination and can be used by engineers to rule MAS behavior and drive the system towards the achievement of its global goals. The behavior of coordination artifacts can be adapted at runtime, cognitive agents can act over the artifacts to affect and suitably change the MAS behavior [62]. This latter *reflective usage* of coordination artifacts provides a potential means for self-organizing MAS. Recently, the work on coordination artifacts has been generalized towards artifacts. Artifacts encapsulate the environment responsibilities to support individual and social activities within a MAS organization [78].

Research on computational institutions such as electronic institutions [48], logic-based institutions [77], or normative MAS [8, 13] has developed a specific line of regulating infrastructures. Computational institutions allow MAS engineers to superimpose laws and norms to MAS agents. Norms can be enacted in prescriptive way (only admissible behaviors are possible), or unruly behaviors can be simply detected and sanctioned by the infrastructure. In computational institutions, the laws and norms governing the activities of agents in the structured environment can either be made explicitly available for agent reasoning, or be induced by agent interpretation, as in [8].

Markovian environments. Markov decision processes (MDP) are a popular approach to model and solve computational problems. An MDP model consists of four components: a set of states, a set of actions, the effects of the actions, and the immediate rewards of the actions. There are a number of algorithms that can automatically solve the decision problem of an MDP. A partially observable MDP (POMDP) adds partial observability to an MDP, which is a property of many problem domains. Partially observable Markovian environments are defined as a tuple $\langle S, s_o, A, T, O, \Omega \rangle$ [37]. S is the set of all possible environment states and s_o is the initial state. A is the set of all possible actions applicable in the environment and T is the probabilistic transition function that specifies how each of the actions change the state of the environment. O is the set of observations, and Ω is the probabilistic observation function that describes the probability that an observer will observe that the environment has moved from one state to another under a particular action. The work of Nair and Tambe [47] on team formation is one interesting example that uses POMDP.

Task environments. A task environment defines both the characteristics of the environment in which the agents must operate, together with a set of tasks that the agent must carry out in the environment [90]. The most common types of tasks are *achievement tasks* that are of the form "achieve a state of affairs," and *maintenance tasks* of the form "maintain a state of affairs." An achievement task is specified by a number of goal states. The agent is required to bring about one of these goal states. An example of a simple achievement task environment is the blocks world. A maintenance task environment is a task environment in which an agent is required to keep (or avoid) some state of affairs. A simple example is a software agent whose task it is to maintain the set of available services in a particular context. Complex tasks can be specified by combinations of achievement and maintenance tasks.

A well-known model for task environments is the TAEMS framework (Task Analysis, Environment Modelling, and Simulation), developed by Decker and Lesser. TAEMS can be used to specify, reason about, analyze, and simulate task environments. TAEMS is claimed to be independent of the agent architecture and the modelled domain [34].

2.2 Levels of support provided by the environment

From the various roles of the environment in MAS discussed in the previous section, we now derive three levels of support that can be provided by the environment.

- *Basic level.* At the basic level, the environment enables agents to access the deployment context. With the deployment context, we refer to the given hardware and software and external resources with which the MAS interacts (sensors and actuators, a printer, a network, a database, a Web service, etc.). Providing access to the deployment context to agents is an essential functionality of the environment in agent systems.
- *Abstraction level.* The abstraction level bridges the conceptual gap between the agent abstraction and low-level details of the deployment context. The abstraction level shields low-level details of the deployment context and possible other resources in the system to the agents. An abstraction level of the environment is common in agent systems and is supported in most agent platforms.
- *Interaction-mediation level.* The interaction-mediation level offers support to: (1) regulate the access to shared resources, and (2) mediate interaction between agents. With an interaction-mediation level, the environment becomes an active entity in the MAS. Support for interaction mediation enables agents to exploit the environment to coordinate their behavior.

The three levels of support represent different degrees of functionality provided by the environment that agents can use to achieve their goals. An interesting additional level of support could be a "reflective level" that provides a reflective interface to the functionality supported by the environment. Such reflective interface enables cognitive agents to modify the functional behavior of the environment. The work on artifacts is a promising approach that proposes support at the reflective level as a means for self-organizing MAS [62].

In the following three subsections, we illustrate each level of support provided by the environment with examples. We start with a "naked" environment in which agents directly access the deployment context. Subsequently we add an abstraction level and interaction-mediation level.

2.2.1 Basic level: Direct access to the deployment context

The environment as the deployment context represents the most elementary perspective on the environment in an agent system. Figure 1 depicts example scenarios in which agents directly access the deployment context.

In the example, the agent on the left side inserts two values into a table of a database. The agent in the middle opens a socket on a particular port number to contact another agent. The two agents on the right side access a shared printer. From these examples it becomes clear that direct access to the deployment context compels agents to deal with low-level details of the network, resources, and so on. Especially in dynamic and unpredictable deployment contexts such as ad hoc networks the agents tasks become arduous.

2.2.2 An abstraction level

The environment can provide agents with an abstraction level that shields low-level details of the deployment context—as well as other resources in the system. This perspective on the environment is common in agent systems, examples are Jade [7], FIPA platform [27], and Retsina [73]. Figure 2 depicts example scenarios of an environment containing an abstraction layer.

Agents now access abstractions of the resources they are interested in. The Agenda repository allows agents to interact with the database at a higher level of abstraction. In the example, the agent adds an appointment in the agenda. The abstraction level takes the burden of transforming the agents commands into SQL instructions to interact with the actual database. The network abstraction in the middle of Fig. 2 provides a communication infrastructure to

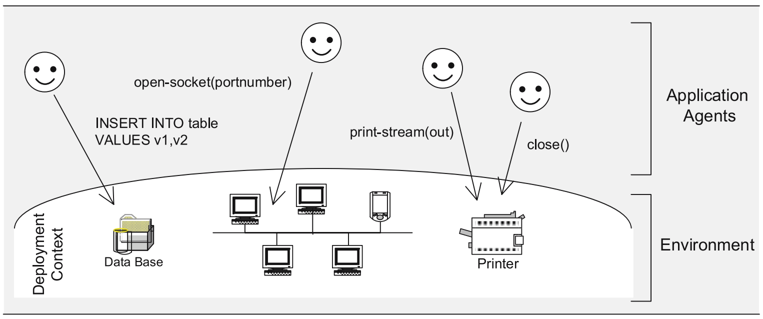

Fig. 1 Agents directly access the deployment context

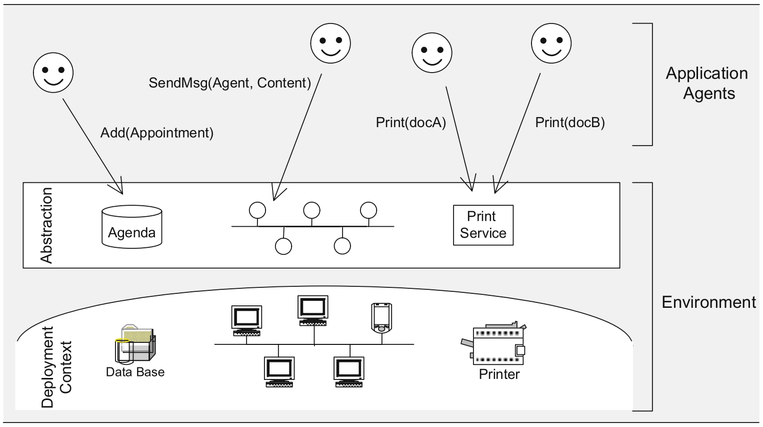

Fig. 2 Abstraction level that shields agents from the details of the deployment context

agents to send and receive messages using agent names instead of sockets with ports or IP numbers, etc. In a similar manner, the Print Service provides an abstraction of the printer that allows agents to instruct the service to print a document instead of sending streams of bytes to the output port, etc. Some other examples of functionality that can be supported by the abstraction level are mobility or fusion of sensor data.

In summary, the abstraction level provides an appropriate interface to the agents, shielding low-level details of the network, legacy systems, etc. In dynamic or unpredictable deployment contexts such as ad hoc networks, the abstraction level—typically supported by appropriate middleware—can shield the complexity of the deployment context (mobility, nodes that come and go, etc.) from the agents.

2.2.3 Interaction-mediation level

In addition to the abstraction level, an environment can provide an interaction-mediation level to support mediated interaction in the environment. Figure 3 depicts example scenarios of interaction mediation.

The agent in the middle of the figure interacts with a pheromone infrastructure that is deployed on top of the network topology. The agents on the right side participate in an electronic institution. One agent aims to enter a scene (i.e., an agent group meeting in the institution), the other agent is setting the price for a particular good. The normative system of the electronic market ensures that agents' interactions conform to the shared conventions of the electronic institution. Agents that violate the norms of the institution will be sanctioned, which in turn will affect their future acting possibilities. For some resources, the interaction-mediation layer will be transparent (in the example, this is the case for the Agenda abstraction).

Examples of infrastructures for mediated interaction are coordination infrastructures [55], infrastructures for digital pheromones [11], law-governed interaction [45], computational fields [41, 43], infrastructures for electronic institutions [22], or tag-based coordination and reputation mechanisms [15, 33, 59].

With an interaction-mediation level, the environment becomes an active entity in the system. The environment regulates particular activity in the system. Typically, the environment assigns activities to resources independent of agent activities such as:digital pheromone

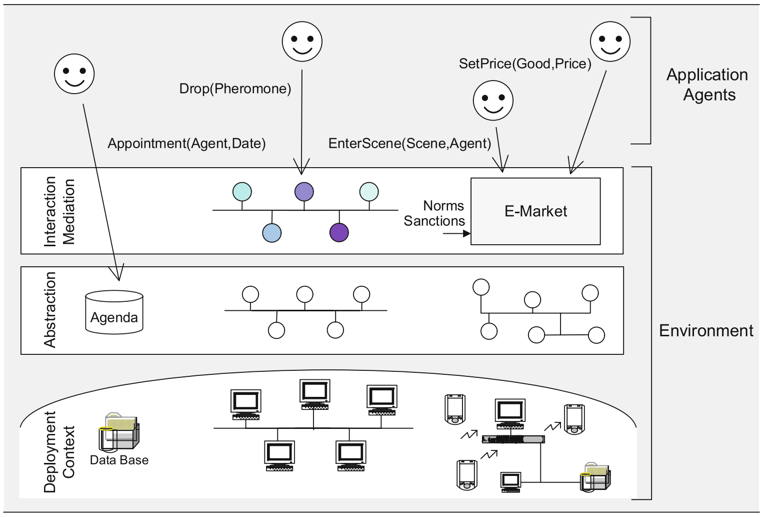

Fig. 3 The environment mediates the interaction with resources and among agents

aggregation, diffusion, and evaporation, computation field maintenance in a mobile network, or normative state maintenance in an electronic institution.

3 Defining environment as a first-class abstraction in MAS

In the previous section, we gave an overview of the environment's role in MAS. Current practice in MAS considers the environment essentially as "infrastructure" for agents. This perspective on the environment, however, does not exploit the full potential of the environment in MAS.

An infrastructure typically accounts for a specific set of responsibilities in the system. This hampers flexible assignment of responsibilities among agents and the environment. For a particular application, all responsibilities that are not managed by the infrastructure remain to be addressed by the agents, often leading to agents that are more complex than necessary. Moreover, infrastructures are typically confined to a particular kind of approach (pheromones, fields, norms, etc.). However, a solution may benefit from integrating different kinds of approaches according to the requirements of the system at hand. MAS infrastructures are usually not developed with this kind of integration in mind. Finally, infrastructures typically focus on one set of responsibilities in the system. Communication infrastructures provide support for messages transfer, pheromone infrastructures provide support for a indirect coordination with digital pheromones, and so on. The remaining functionalities of the environment often remain implicit or are dealt with in an ad hoc manner—or even worse, agents are used to provide functionalities and services that are not appropriate for them.

In this section, we put forward the notion of environment as a first-class design abstraction in MAS. In other words, the environment is a building block that is considered explicitly and can be exploited creatively when building MAS applications. First, we give a definition of the environment. Second, we explain important responsibilities that can be assigned to the MAS environment. Last, we introduce a reference model for the environment. This reference model provides a common frame to discuss environment issues and it can serve as a basis for environment engineering.

3.1 Definition of environment

Before providing our definition of environment, a short overview of previous definitions described in literature is presented.

3.1.1 Environment definitions in literature

MAS literature gives several definitions of the environment in MAS. We give a number of representative examples.

Russell and Norvig [65] define a "generic environment program." This simple program gives the agents percepts and receives back their actions. The program then updates the state of the environment based on the actions of the agents and possibly other dynamic processes in the environment that are not considered to be agents. Russell and Norvig's environment program illustrates the basic relationship between agents and their environment.

Rao et al. [60] specify characteristics of the environment for a broad class of agent system application domains: (1) at any instant in time there are potentially many different ways in which the environment can evolve; (2) at any instant in time there are potentially many different actions possible; (3) different objectives may not be simultaneously achievable; (4) the actions that best achieve the various objectives are dependent on the state of the environment (context); (5) the environment can only be sensed locally; (6) the rate at which computations and actions can be carried out is within reasonable bounds to the rate at which the environment evolves. Rao and his colleagues describe the typical characteristics of the external world in which agent systems are deployed and with which the agent systems interact.

Parunak [56] defines an environment as a tuple $\langle State, Process\rangle$. *State* is a set of values that completely define the environment, including the agents and objects within the environment. *Process* indicates that the environment itself is an active entity. It has its own process that can change its state, independently of the actions of the embedded agents. The primary purpose of *Process* is to implement dynamism in the environment, such as maintenance processes of digital pheromones. Parunak's definition of environment underlines the active nature of the environment.

Ferber [23] defines an environment as "a space E, which generally has a volume." Objects, including agents, are situated in E. That is, at a given moment, any object has a position in E. Objects are related to one another and agents are able to perceive objects and to manipulate (passive) objects in E. Agents' actions are subject to the "laws of the universe" that determine the effects of the actions in the environment. Ferber's definition underlines the container function of the environment and emphasizes the separation of agent actions (as attempts to modify the course of events in the environment) from the reaction to those actions in the environment (i.e., the outcome of the actions).

Demazeau [17] considers four essential building blocks for agent systems: agents (i.e., the processing entities), interactions (i.e., the elements for structuring internal interactions between entities), organizations (i.e., elements for structuring sets of entities within the MAS), and finally the environment that is defined as "the domain-dependent elements for structuring external interactions between entities." The environment in Demazeau's perspective emphasizes the structuring qualities of the elements external to the agent system.

Odell et al. [50] define an environment as follows: "The environment provides the conditions under which an entity (agent or object) exists". The authors distinguish between the physical environment and the communication environment. The physical environment provides the laws, rules, constraints, and policies that govern and support the physical existence of agents and objects. The communication environment provides (1) the principles and processes that govern and support the exchange of ideas, knowledge and information, and (2) the functions and structures that are commonly employed to exchange communication, such as roles, groups, and the interaction protocols between roles and groups. Odell's definition of environment underlines the different structures of the environment (physical, communicative, social) and the mediating nature of the environment.

3.1.2 Our definition

Based on insights that we have derived from recent research on environments in MAS, we introduce the following definition of the environment[1]:

> The environment is a first-class abstraction that provides the surrounding conditions for agents to exist and that mediates both the interaction among agents and the access to resources.

First of all, this definition states that the environment is *a first-class abstraction*. This stresses the fact that the environment is an independent building block in the MAS that encapsulates its own clear-cut responsibilities, irrespective of the agents. Second, the environment *provides the surrounding conditions for agents to exist*. This implies that the environment is an essential part of every MAS. The environment is the part of the world with which the agents interact, in which the effects of the agents will be observed and evaluated. Moreover, on their own, agents are just individual loci of control. To build a useful system out of individual agents, agents must be able to interact. The environment provides the glue that connects agents into a working system. Third, the environment *mediates both the interaction among agents and the access to resources*. This states that the environment can be an active entity with specific responsibilities in the MAS. The environment provides a medium for sharing information and mediating coordination among agents. As a mediator, the environment not only enables interaction, it also constrains it. As such, the environment provides a design space that can be exploited by the designer. Distinguishing between agent and environment responsibilities supports separation of concerns in MAS, and helps to manage the huge complexity of engineering complex real-world MAS applications. Experiences with designing concrete environments in a particular domain will yield reusable mechanisms and patterns that will further help to manage complexity.

3.2 Responsibilities of the environment

Building upon [85], we now discuss a number of core responsibilities that can be assigned to the environment. For each responsibility, we briefly touch upon the level of support provided by the environment (as discussed in Sect. 2.2).

The environment structures the multiagent system. The environment is first of all a shared "space" for the agents, resources, and services which structures the whole system. Here, resources have a specific state that can be manipulated by agents, and services are considered as reactive entities that provide functionality to the agents. The agents—as well as resources and services—are dynamically interrelated to each other. It is the environment's

[1] This definition integrates and refines preliminary versions that were discussed at E4MAS (2005a, b) and the AL3-TF (2005).

role to define the rules to which these relationships must comply. As such, the environment acts as a *structuring* entity for the MAS. In general, different forms of structuring can be distinguished:

- *Physical structure* refers to spatial structure, topology, and possibly distribution, see e.g., [5, 11].
- *Communication structure* refers to infrastructure for message transfer [7, 73], infrastructure for stigmergy [11, 42], or support for implicit communication [58, 74].
- *Social structure* refers to the organizational structure of the environment in terms of roles, groups, societies, etc., see e.g. [25, 50, 51, 76, 91].

Structuring is a fundamental functionality of the environment. Structures may be constrained by the domain at hand, or they may be carefully considered as design choices. Physical and communication structures are part of the deployment context and should be supported by an appropriate level of abstraction. Social structure is typically supported at the interaction-mediation level.

The environment embeds resources and services. An important functionality of the environment is to embed resources and services. Resources and services are typically situated in a physical structure. The environment should provide support at the abstraction level shielding low-level details of resources and services to the agents.

The environment can maintain dynamics. Besides the activity of the agents, the environment can have processes of its own, independent of agents. A typical example of dynamics in the environment is the evaporation, aggregation, and diffusion of digital pheromones. Another example of an environmental activity is a self-managing field in a network. When the topology of the physical network changes, the environment maintains the consistency of its field. The environment may also provide support for maintaining agent-related state, for example, the normative state of an electronic institution or tags for reputation mechanisms. Such dynamics are an important functionality of the environment.

Depending on the nature of the dynamics, their maintenance can be supported at the abstraction level (e.g., maintenance of fused sensor data) or at the interaction-mediation level (e.g., maintenance of coordination infrastructures).

The environment is locally observable to agents. Contrary to agents, the environment must be observable. Agents should be able to inspect the different structures of the environment, as well as the resources, services, and possibly external state of other agents. Observation of a structure is typically limited to the current context (spatial context, communication context, and social context) in which the agent finds itself. In general, agents should be able to inspect the environment according to their current tasks. Weyns et al. [89] discuss an example of selective perception where "foci" are proposed to enable agents to perceive their environment selectively, according to their current tasks.

Related to observability is the semantic description of the domain, which can be defined by an environment ontology, see e.g. [14]. The ontology must cover the different structures of the environment as well as the observable characteristics of resources, services and agents, and possibly the regulating laws. For symbolically oriented agents, an explicit ontology should be available to the agents so that they can interpret their environment and reason about it. For non-reasoning agents, the designer/developer applies the ontology to encode the agent's internal structures. As such, these kinds of agents have an implicit ontology that enables them to make decisions and to interact.

Agents should be able to observe the environment at the appropriate level of abstraction. Support for observability of the social context is situated at the interaction-mediation level.

The environment is locally accessible to agents. Agents must be able to access the different structures of the environment, as well as its resources, services, and possibly external state of other agents.

As for observability, accessing a structure is limited to the current context in which the agent finds itself. Access to spatial structure refers to support for metrics, mobility, and so on. Access to communication infrastructure refers to support for direct communication (message transfer), support for indirect communication (marks, fields, etc.), or support for implicit communication (over-hearing, over-sensing, etc.). Access to social structures refers to the ability to interact with social units, such as organizations, group membership, and other normative social structures.

In general, resources can be perceived, modified, generated, 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, and the (current) interrelationships with other resources, services, or agents.

Similar to observability, agents should be able to access the environment at the right level of abstraction. Support for access of the social context is situated at the interaction-mediation level.

The environment can define rules for the multiagent system. The environment can define different types of rules on all entities in the MAS. Rules may refer to constraints imposed by the domain at hand (e.g., mobility in a network), or to laws imposed by the designer (e.g., limitation of access to neighboring nodes in a network for reasons of performance). Rules may restrict the access of specific resources or services to particular types of agents, or determine the outcome of agent interactions. By imposing rules on an agent's activity, the environment acts as an arbitrator that attempts to preserve the agent system in a consistent state according to the properties and requirements of the application domain.

In electronic institutions [48], agents interact through agent group meetings that are called *scenes*. Interactions in a scene 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 interactions of agents in an electronic institution.

Rules in the environment govern the shared access to resources by agents, as well as the constraints on the interaction among agents. As such, the support for rules is situated at the interaction-mediation level.

3.3 Reference model for the environment

We now introduce a reference model for the environment. In particular, it describes a functional decomposition of the application environment. By the term *application environment*, we refer to the part of the environment that has to be designed for the application at hand, i.e., the functionality supported at both the abstraction and interaction-mediation levels as described in Sect. 2.2. Figure 4 provides an overview of the reference model.

At the top, the application environment is accessed by the agents. The agents are the domain-specific entities that autonomously make decisions and act in the environment. At the bottom, the application environment interfaces with the deployment context.

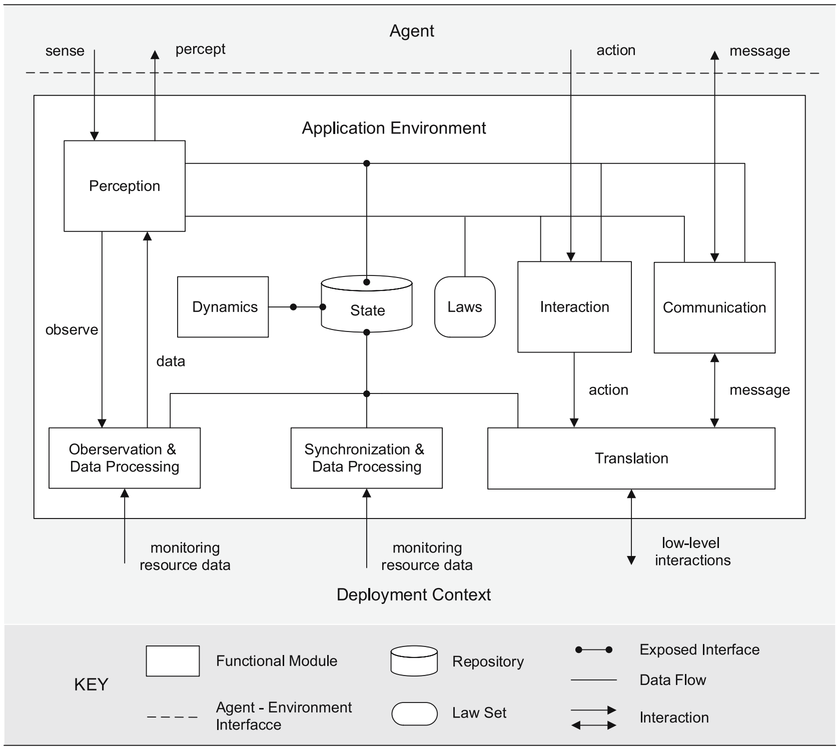

Fig. 4 A reference model for the environment in MAS

The reference model consists of a set of modules that represent core functionalities of the environment and the flows between these modules. Inevitably, the decomposition is biased by our own interpretation of the "essential functions" of an environment. However, we have kept the model fairly general that we believe fits well with most models and interpretations of environment considered in literature.

The proposed model covers those environmental responsibilities discussed in the previous section. The decomposition is primarily driven by the way agents interact with the environment. An agent can *sense* the environment to obtain a *percept* (i.e., a representation of its vicinity), an agent can perform an *action* in the environment (i.e., attempt to modify the state of affairs in the environment), and it can exchange *messages* with other agents. Whereas action is concerned with direct manipulation of the state of affairs in the environment, communication is not. Communicative interaction occurs in a sequential manner and concerns the coordination of actions among agents. Communicative interaction enables agents to resolve conflicts, request each other for services, establish a future cooperation, and so on. Employing perception, action, and communication as distinct ways to access the environment is shared by many researchers in the community, see e.g. [23, 50, 91]. In the next section, we examine the reference model modules in more detail.

3.3.1 Modules of the reference model

State. The State module represents the actual state of the application environment. The environment's state typically includes an abstraction of the deployment context—possibly extended with other states related to the MAS environment. Examples of deployment context-related state include abstract representations of the network topologies, or maps of a physical environment. Other examples can include infrastructure representations of digital pheromones that are deployed on top of a network or social structures such as an electronic institution. The environment's state may also include agent-specific data, such as tags of normative state associated with the agents. The State module exposes a number of interfaces that enable other modules to read and modify the state of the environment.

Synchronization & Data Processing. The Synchronization & Data Processing module monitors domain-specific parts of the deployment context and keeps the corresponding representation in the State module up to date. Rather than providing only a reflection of the monitored state of recourses in the deployment context, the synchronization module can provide additional functions to accommodate the sensors used in real-world applications. Functions might include sorting of data, sensor calibration, data correction, or data interpolation. An example of functionality provided by the Synchronization & Data Processing module is a dynamic network where changes are reflected in a network abstraction maintained in the state of the environment.

Dynamics. The Dynamics module manages the environmental dynamics that occur independently of the agents or the deployment context. A typical example is the maintenance of a digital pheromone.

Laws. Laws represent application-specific constraints on the interactions of agents in the environment. Laws can impose restrictions on agent perception, interaction, and communication. We discuss examples of different types of laws, below.

Perception. The Perception module provides the functionality for agents to perceive the environment. When an agent senses the environment, the Perception module generates percepts according to the current state of the application environment and possibly data observed from the deployment context. Agent perception is subject to perception laws. Perception laws provide a means to constrain perception. For example, in an electronic institution, an agent will only be able to monitor the scenes for which it has successfully been registered. Perception laws are also useful for dealing with technological issues, for example a designer can introduce default limits for perception in order to restrain the amount of information that has to be processed, or to limit an occupied bandwidth.

Observation & Data Processing. The Observation & Data Processing module provides the Perception module with the required functionality to observe the deployment context. Data obtained from the observation of elements in the deployment context are passed to the Perception module, possibly after some processing. The functionality to process resource data is similar to the functionality discussed in the Synchronization & Data Processing module.

Interaction. The Interaction module deals with agents' actions in the environment. The Interaction module encapsulates an action model that describes how the various—concurrently executed—action commands are executed. An example of an action model is Ferber's influence-reaction model [23] that separates what an agent wants to perform from what actually happens. Agents produce influences in the environment and subsequently the environment reacts to the influences resulting in a new state of the world. Agent actions can be divided in two classes: actions that attempt to modify state of the application environment and actions that attempt to modify elements of the deployment context. An example of the former is an agent that drops a digital pheromone in the environment. An example of the latter is an agent that writes data in an external database. Agent actions are subject to interaction laws. In case several agents attempt to access an external resource simultaneously, an interaction law may impose a policy on the access of that resource. In case an agent attempts to modify the state of the application environment, the outcome of the action may be subject to a law.

An example of the latter case is an agent that violates a norm in an electronic institution and is sanctioned for that. Actions related to the deployment context are passed to the Translation module that converts the high-level actions of agents into low-level interactions in the deployment context.

Communication. The Communication module collects messages and delivers them to the appropriate agents. Similar to the Interaction module, the Communication module may regulate the messages exchanged between agents based on the current state of the application environment and a set of laws. Communication laws can just regulate the message stream, or they can impose application-specific regulations on exchanged messages. An example of this latter are a set of laws that impose agents to follow the prescribed steps of a particular protocol. The Translation module converts the high-level message descriptions (in an Agent Communication Language) into low-level communication primitives of the deployment context.

3.3.2 Levels of support and the reference model

In Sect. 2.2, we have identified three levels of support that can be provided by the environment: the basic level, the abstraction level, and the interaction-mediation level. The three levels and the reference model provide two different views on the structure and basic functions of the environment. Figure 5 shows how these two views are related to one another.

The vertical decomposition of the environment consists of two main parts, the application environment and the deployment context.

The application environment is divided in two rows. The top row deals with the activities of agents in the application environment (perception, communication, and action) and dynamics in the application environment that happen independent of agents. This row also defines the various laws that constrain the activity of agents in the environment. The functionality of the top row corresponds to the abstraction and interaction-mediation level support provided by the environment.

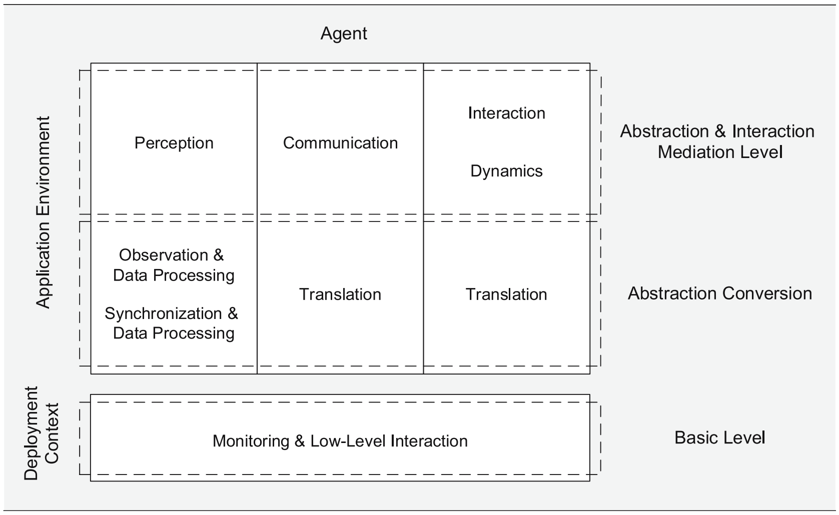

Fig. 5 Levels of support provided in the reference model of the environment

The bottom row of the application environment deals with the interaction with the deployment context. This row provides support for (1) the translation of high-level activity related to agents to low-level interactions related to the deployment context and vice versa, and (2) the pre-processing of resource data to transfer it to a higher-level representation useful to agents. The bottom row enables the conversation between the abstraction level of the top row and the basic level of the deployment context.

The deployment context provides support for monitoring and low-level interaction with resources external to the multiagent system. Access to the deployment context corresponds to the basic level support provided by the environment.

3.3.3 Discussion

Distribution. The application environment in the reference model (Fig. 4) abstracts from distribution of the multiagent system. For a distributed application, the deployment context consists of multiple processors deployed on different nodes that are connected through a network. For such applications, the application environment typically has to be distributed over the processors of the application nodes. For some applications, the same functionalities of the application environment are deployed on each node. For other applications, specific functionalities are deployed on different nodes (e.g., when different types of agents are deployed on different nodes). Some functionalities provided by the application environment may be limited to the local context (e.g., observation of the deployment context may be limited to resources of the local deployment context); other functionalities may be integrated (e.g., neighboring nodes may share state). Integration of functionality among nodes typically requires additional support. Such support may be provided by appropriate middleware. Examples are support for message transfer in a distributed setting (e.g. [7]), support for a distributed pheromone infrastructure (e.g., [11]), or support for mobility (e.g., [38]).

Crosscutting Concerns. It is important to notice that the reference model of the application environment abstracts from various complex concerns such as security or monitoring. Such concerns are very difficult to capture in one module and usually crosscut several parts of the system functionality. Aspect-oriented software development (AOSD) is a recently developed software engineering discipline that aims to integrate crosscutting concerns in an application in a non-invasive manner. The issues with crosscutting concerns in multiagent systems are hardly explored and are open research problems. An example of early research in this direction is [29].

Human–Computer Interaction. The reference model of the application environment does not explicitly handle human–computer interaction. Depending on the application domain, the role of humans in multiagent systems can be very diverse. In some applications, humans can play the role of agents and interact directly—or via some kind of intermediate wrapper—with the application environment. In other applications, humans can be part of the deployment context with which the MAS application interacts. Industrial applications typically require system monitoring. Functionality for monitoring the application environment can be integrated orthogonally into the application's functionality by means of AOSD techniques.

3.3.4 Reference model, a basis for environment engineering

The reference model provides a common frame for researchers and developers to discuss environmental issues. To build a MAS—and an environment, in particular—an appropriate *software architecture* must be defined. A software architecture defines the structures of the system, which comprise architectural elements (software modules, processes, deployment units, etc.) and the relationships among the elements [6]. A software architecture maps functionality to a system decomposition. The reference model provides an abstract functional decomposition of the environment that can help software architects to build concrete software architectures for environments. Software architectures differ in the way this mapping is applied, aiming to satisfy the specific systems requirements of the application at hand. Experience from building concrete software architectures in an application area may yield reusable architectural approaches, such as architectural patterns [69] or a reference architecture. A reference architecture maps the functional decomposition of the reference model to an abstract software architecture for a family of applications and provides a blueprint to design concrete software architectures for applications of that family. One example of a reference architectures for MAS are PROSA [75, 76] that is tailored to the domain of manufacturing control. Another example is the reference architecture for situated MAS presented in Weyns and Holvoet [82] that has proven to be valuable for distributed applications operating in highly dynamic operating circumstances and where global control is hard to achieve. This latter reference architecture was applied in the automated logistic transportation system we discuss in the next section.

4 Exploiting the environment in a real world application

In this section, we illustrate how the environment is successfully exploited as a design abstraction in a real world application. This application is an automated transportation system for warehouse logistics that has been developed in a joint R&D project between the DistriNet research group and Egemin, a manufacturer of automating logistics services in warehouses and manufactories [20, 87]. The transportation system uses automatic guided vehicles (AGVs) to transport loads through a warehouse. Typical applications include distributing incoming goods to various branches, and distributing manufactured products to storage locations. AGVs are battery-powered vehicles that can move through a warehouse. AGVs use a navigation system to follow predefined paths on the factory 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 is handled by the AGV control software.

4.1 Multiagent system for the AGV transportation system

Figure 6 shows a high-level model of the application. The MAS of the transportation system consists of two kinds of agents: *transport agents* and *AGV agents*. Transport agents represent transports that need to be handled by an AGV and are located at *transport bases*, i.e., stationary computer systems. AGV agents are responsible for executing transports and are located in mobile vehicles that are situated on the factory floor. The communication infrastructure provides a wireless network that enables AGV agents at vehicles to communicate with each other and with transport agents on transport bases.

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 inhabit. This virtual environment provides an interaction-mediation level that agents can use as a medium to exchange information and coordinate their behavior. In addition, the virtual environment provides a suitable abstraction level that

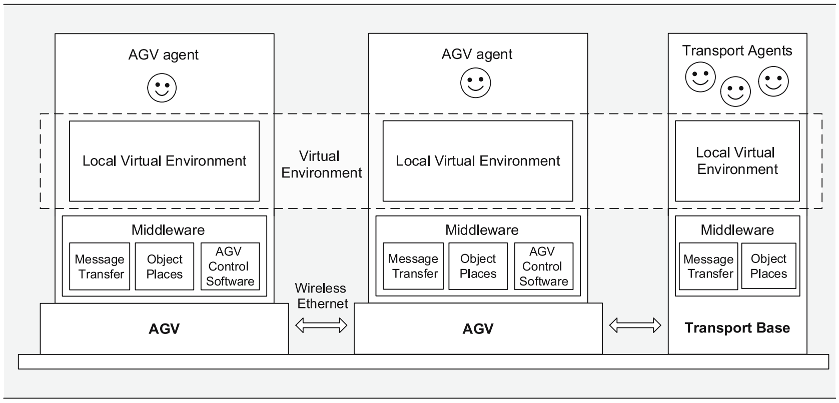

Fig. 6 A virtual environment as a design abstraction in an AGV transportation system

shields the AGV agents from 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 reusable. 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 [66, 67].

4.2 Perception, action, and communication in the AGV local virtual environment

We now zoom in on the local virtual environment of the AGVs. Figure 7 depicts a high-level module view of the software architecture of the AGV local virtual environment.

The module view of the local virtual environment maps fairly well with the functional decomposition of the reference model discussed in the previous section. The local virtual environment offers perception, action, and communication capabilities to the AGV agents that are dealt with by the perception, action, and communication managers, respectively. The perception manager only interacts with the state repository; the Observation module in the reference model is absent in the AGV application. The functionality of the Translation module in the reference model is integrated in both the communication manager and action manager. Furthermore, the Laws in the reference model are not explicitly modeled in the virtual environment but are integrated in the functionality provided by different modules.

Perception in the virtual environment is handled by the perception manager. The perception manager's task is straightforward: when the agent requests a percept (for example, the current positions of neighboring AGVs), the perception manager queries the necessary information from the state repository of the local virtual environment and returns the percept to the agent. Actions are handled by the action handler. One kind of action concerns the

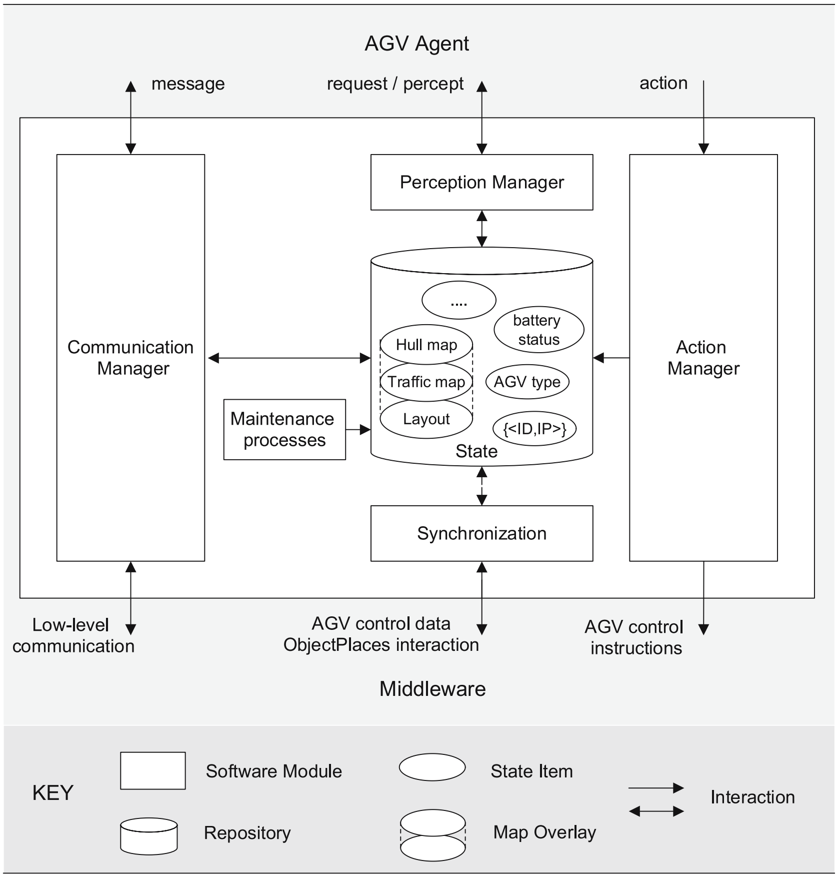

Fig. 7 High-level view of the software architecture of the AGV local virtual environment

physical actions of the AGV, for example moving over some distance or picking up a load. These actions are handled fairly easily by passing them on to the AGV control software. The action manager translates agent actions to low-level AGV control instructions. Other kinds of actions do not actually have an effect on the behavior of the AGV, but manipulate the state of the virtual environment. Marking hulls is one example of this, which we describe in the following section. Communication is handled by the communication manager, enabling agents to communicate directly with other agents through the environment. A typical example is an AGV agent that informs a transport agent about the status of the transport. The communication manager translates the high-level messages to low-level communication instructions that can be sent through the network (resolving agent names to IP numbers, etc.).

The synchronization module maintains consistency between the status of the AGV and ObjectPlaces on the one hand and the state of the local virtual environment on the other. To synchronize state among local virtual environments, the interaction between the synchronization module and the middleware is bidirectional. Finally, the maintenance processes are responsible to maintain dynamics in the local virtual environment, such as evaporation of pheromones or the maintenance of fields. We illustrate this in the next section.

In summary, the local virtual environment offers high-level primitives to the AGV agent to act in the environment, perceive the environment, and communicate with other agents. The virtual environment provides an abstraction level that shields the agent from lower level issues. An overview of the software architecture of the AGV application is described in Weyns et al. [87], details of the software architecture of the environment are described in Weyns et al. [86].

4.3 Mediated interaction through the virtual environment

We now illustrate how the virtual environment provides different functionalities at the interaction-mediation level to enable agents to coordinate their behavior.

Transport assignment. Due to the highly dynamic nature of transport creation, the assignment of transports to AGVs is complex. To cope with the continuously changing circumstances in the environment a field-based approach is used to assign transports to AGVs [80]. In this approach, transport agents emit local fields into the environment that attract idle AGVs. To avoid multiple AGVs driving to the same transport, AGV agents emit repulsive fields. AGV agents combine received fields and follow the gradient of the combined field that guide the AGVs towards pick locations of transports. The AGV agents continuously reconsider the situation of the environment and transport assignment is delayed until the load is finally picked—which benefits the flexibility of the system.

Routing. For routing purposes, the virtual environment has a static layout 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 the 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 (via the ObjectPlaces middleware) and merged by the synchronization module to provide up-to-date information to the AGV agents. Since pheromones evaporate over time (which is managed by a maintenance process), 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. In case of a conflict, the priorities of the transported loads and the vehicles determine which AGV can move on. Afterwards, the AGV removes the markings in the virtual environment. Weyns et al. [86] discuss collision avoidance through the virtual environment in detail.

5 Conclusions

A study of the environment's role in MAS have taught us that the environment is an implicit part of MAS that is often dealt with in an ad hoc manner. This leads to poor engineering practice and hinders engineers to exploit the full potential of the environment in MAS.

In this paper, we put forward the environment as an explicit part of MAS. Furthermore, we defined environment as a first-class abstraction in MAS that has a dual role. First, the environment is an essential part of every MAS. The environment is the part of the world with which the agents interact, in which the effects of the agents will be observed and evaluated. It is the glue that connects agents into a working system. Second, the environment provides a design space that can be exploited by the designer. Distinguishing between agent and environment responsibilities supports separation of concerns in MAS, and helps to manage the enormous difficulties involved in engineering complex real-world MAS applications.

We have identified three levels of support that can be provided by the environment: (1) direct access to the deployment context, (2) an abstraction level that shields agents from the low-level details of deployment, and (3) mediated interaction between agents.

Important responsibilities of the environment include: (1) the environment provides structure for the MAS as a whole; (2) the environment embeds resources and services; (3) the environment can be responsible for maintaining dynamics in the system that happen independent of agents; (4) contrary to agents, the environment is observable; (5) the environment is locally accessible to agents; and finally (6) the environment can define different types of rules on all the entities in the MAS.

We presented a reference model for the environment in MAS. This reference model provides a common frame to discuss environmental issues. The reference model can also serve as a basis for environment engineering. In particular, the reference model provides a functional decomposition of the environment that can help software architects when building concrete software architectures for environments.

Finally, we showed how the environment is exploited in a real world application. The virtual environment in this application serves as a flexible coordination medium, which hides much of the complexity of the system (distribution, mobility, etc.) 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 the agents can exploit.

Many issues are open for future research on environments in MAS. Weyns et al. [85] give an extensive overview of challenges in the domain. One major challenge for future research is environment engineering. A key step in environment engineering will be to define a suitable software architecture for the environment. Providing the environment with an appropriate level of support to agents, dealing with environment responsibilities explicitly, and mapping core functionalities of the environment to architectural elements will be crucial aspects for architectural design of environments. An important challenge for research on environments will be the development of architectural approaches for the design of environments, such as architectural patterns and reference architectures. Such architectural approaches embody knowledge and experiences acquired from building concrete software architectures in specific application domains and can provide a solid basis for large-scale reuse.

The environment offers opportunities for all types of agency, and as such provides a challenging area for synergetic research in the interest of MAS in general. Considering the environment as a first-class abstraction in MAS opens up new horizons for research and development in MAS.

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 and Budapest, 2005 for the inspiring discussions that have considerably contributed to the work presented in this paper. We are grateful to Tom Holvoet, Mirko Viroli, Paul Valckenaers, Eric Platon, Alexander Helleboogh, and Alessandro Ricci for their critical feedback on early drafts of the paper. A final word of appreciation goes to the anonymous reviewers for their accurate remarks on the initial version of the paper. By taking their critical comments into account we were able to improve the paper significantly.

References

1. AL3-TF: 2005, AgentLink Technical Forum Group on Environments for Multiagent Systems. http://www.cs.kuleuven.ac.be/∼distrinet/events/e4mas/tfg2005/.
2. Arkin, R. (1989). Motor schema-based mobile robot navigation. *International Journal of Robotics Research, 8*(4), 92–112.
3. Arkin, R. (1990). Integrating behavioral, perceptual, and world knowledge in reactive navigation. *Robotics and Autonomous Systems, 6*(1–2), 105–122.
4. Arkin, R. (1998). *Behavior-based robotics*. Cambridge, MA: MIT Press.
5. Bandini, S., Manzoni, S., & Simone, C. (2002). Dealing with space in multiagent systems: A model for situated multiagent systems. In C. Castelfranchi & W. L. Johnson (Eds.), *First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2002)*. Bologna, Italy, ACM Press.
6. Bass, L., Clements, P., & Kazman, R. (2003). *Software architecture in practice*. Addison Wesley Publishing Comp.
7. Bellifemine, F., Poggi, A., & Rimassa, G. (1999). Jade, A FIPA-compliant agent framework. In *Fourth International Conference on Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM 1999)* (pp. 97–108). London, UK.
8. Boella, G., & L. van der Torre, (2003). Attributing mental attitudes to normative systems. In J. S. Rosenschein, M. J. Wooldridge, T. Sandholm, & M. Yokoo (Eds.), *Second International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2003)*. Melbourne, Australia: ACM Press.
9. Bonabeau, E., Henaux, F., Guérin, S., Snyers, D., Kuntz, P., & Theraulaz, G. (1998). Routing in telecommunications networks with ant-like agents. In *Second International Workshop on Intelligent Agents for Telecommunication a Applications (IATA 1998)*. Paris, France: Springer-Verlag.
10. Brooks, R. (1986). Achieving artificial intelligence through building robots. *AI Memo 899, MIT Lab*.
11. Brueckner, S. (2000), *Return from the ant, synthetic ecosystems for manufacturing control*. Ph.D Dissertation, Humboldt University, Berlin, Germany.
12. Cabri, G., Leonardi, L., & Zambonelli, F. (2000). MARS: A programmable coordination architecture for mobile agents. *IEEE Internet Computing, 4*(4), 26–35.
13. Castelfranchi, C., Dignum, F., Jonker, C., & Treur, J. (2000). Deliberative normative agents: principles and architecture. In *Sixth International Workshop on Intelligent Agents, Agent Theories, Architectures, and Languages (ATAL 2000)*. London, UK, Springer-Verlag.
14. Chang, P., Chen, K., Chien, Y., Kao, E., & Soo, V. (2005). From reality to mind: A cognitive middle layer of environment concepts for believable agents. In Weyns et al. (2005a), Springer-Verlag.
15. Conte, R., & Castelfranchi, C. (1995). Understanding the functions of norms in social groups through simulation. *Artificial societies, the computer simulation of social life* pp. 252–267.
16. Corkill, D. (2003). Collaborating software: Blackboard and multi-agent systems and the future. In *International Lisp Conference*. New York, NY, USA.
17. Demazeau, Y. (2003). Multi-Agent Systems Methodology. In *Second Franco-Mexican School on Cooperative and Distributed Systems (LAFMI 2003)* http://lafmi.lania.mx/escuelas/esd03/ponencias/Demazeau.pdf.
18. Demazeau, Y., & Costa, A. C. R. (1996). Populations and organizations in open multi-agent systems. In *Symposium on parallel and distributed artificial intelligence*. Hyderabad, India.
19. Drogoul, A., & Ferber, J. (1994). Multiagent simulation as a tool for studying emergent behavior processes in societies. In *Simulating societies: computer simulation of social phenomena*. UCL Press.
20. EMC^2 (2005). Egemin modular controls concept project. Supported by IWT, Belgium. http://emc2.egemin.com/.

21. Englemore, R. S., & Morgan, A. (1988). *Blackboard systems*. Addison-Wesley.
22. Esteva, M., Rosell, B., Rodriguez-Aguilar, J. & Arcos, J. (2004). AMELI: An agent-based middleware for electronic institutions. In N. Jennings, C. Sierra, L. Sonenberg, & M. Tambe (Eds.), *Third joint conference on autonomous agents & multi-agent systems (AAMAS 2003)*. New York, NY, USA.
23. Ferber, J. (1999). *An introduction to distributed artificial intelligence*. Addison-Wesley.
24. Ferber, J., Gutknecht, O., & Michel, F. (2003). From agents to organizations: An organizational view of multi-agent systems. In F. Giunchiglia, J. Odell, & G. Weiß (Eds.), *Agent-oriented software engineering III, third international workshop (AOSE 2002), Bologna, Italy, Revised Papers and Invited Contributions*, Vol. 2585 of *Lecture Notes in Computer Science*. Springer-Verlag, Betrlin, Heidelberg, New York.
25. Ferber, J., Michel, F., & Baez, J. (2005). AGRE: Integrating environments with organizations. In Weyns et al. (2005a). Springer-Verlag.
26. Ferber, J., & Müller, J. P. (1996). Influences and reaction: A model of situated multiagent systems. In M. Tokoro (Ed.), *Second international conference on multi-agent systems (ICMAS 1996)*. Kyoto, Japan, AAAI Press, Menlo Park, California, USA.
27. FIPA (2002). Foundation for intelligent physical agents, FIPA abstract architecture specification. http://www.fipa.org/repository/bysubject.html.
28. Freeman, E., Hupfer, S., & Arnold, K. (1997). JavaSpaces: Principles, patterns, and practice. Addison-Wesley.
29. Garcia, A., Kulesza, U., & Lucena, C. (2005). Aspectizing multi-agent systems: from architecture to implementation. In R. Choren, A. Garcia, C. Lucena, & A. Romanovsky (Eds.), *Software engineering for multi-agent systems III (SELMAS 2004)*, Vol. 3390 of *Lecture Notes in Computer Science*. Springer.
30. Gasser, L. (2001). Perspectives on organizations in multi-agent systems. In *Multi-agent systems and applications: Ninth ECCAI advanced course ACAI 2001 and agent link's 3rd European agent systems summer school (EASSS 2001)*, Vol. 2086 of *Lecture Notes in Computer Science*. Springer-Verlag.
31. Gelernter, D., & Carrierro, D. (1992). Coordination Languages and their significance. *Communications of the ACM, 35*(2), 97–107.
32. Grassé, P. P. (1959). La reconstruction du nid et les coordinations inter-individuelles chez bellicositermes natalensis et cubitermes sp. la theorie de la stigmergie. Essai d'interpretation du comportement des termites constructeurs. *Insectes Sociaux, 6*, 41–81.
33. Hales, D. (2002). Group reputation supports beneficent norms. *Journal of Artificial Societies and Social Simulation, 5*(4).
34. Horling, B., Lesser, V., Vincent, R., Wagner, T., Raja, A., Zhang, S., Decker, K., & Garvey, A. (2005). *The TAEMS White Paper, Multi-Agent Systems Lab University of Massachusetts*.
35. Huhns and Stephen (1998). Multiagent Systems and Societies of Agents. In Weiss, G. (Ed.), Multiagent systems, a modern approach to distributed artificial intelligence. Cambridge, MA, USA: MIT Press.
36. Jennings, N. R. (2000). On agent-based software engineering. *Artificial Intelligence, 117*(2).
37. Kaelbling, L. P., Littman, M. L., & Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domain. *Artificial Intelligence, 101*(1–2), 99–124.
38. Kotz, D., & Gray, R. (1999). Mobile agents and the future of the internet. *ACM Operating Systems Review, 33*(3), 3–17.
39. Maes, P. (1990). Situated agents can have goals. *Robotics and Autonomous Systems, 6*(1–2), 49–70.
40. Malcolm, C., & Smithers, T. (1990). Symbol grounding via a hybrid architecture in an autonomous assembly system. *Robotics and Autonomous Systems, 6*(1–2), 123–145.
41. Mamei, M., & Zambonelli, F. (2004). Co-fields: A physically inspired approach to distributed motion coordination. *IEEE Pervasive Computing, 3*(2), 51.
42. Mamei, M., & Zambonelli, F. (2006). *Field-based coordination for pervasive multiagent systems*, Springer Series on Agent Technology. Springer-Verlag.
43. Mamei, M., Zambonelli, F., & Leonardi, L. (2003). Tuples On the air: A middleware for context-aware computing in dynamic networks. In *Second international workshop on mobile computing middleware (MCM 2003)*. IEEE CS Press.
44. Mataric, M. (1994). Leaning to behave socially. In *Third international conference on simulation of adaptive behavior*. Brighton, UK: MIT Press.
45. Minsky, N., & Ungureanu, V. (2000). Law-governed interaction: A coordination and control mechanism for heterogeneous distributed systems. *ACM Transactions on Software Engineering Methodologies, 9*(3), 273–305.
46. Murphy, A., Picco, G., & Roman, G. (2001). LIME: A middleware for physical and logical mobility. In *Twenty-First international conference on distributed computing systems (ICDCS 2001)* (p. 254). Washington, DC, USA, IEEE Computer Society.
47. Nair, R., & Tambe, M. (2005). Hybrid BDI-POMDP framework for multiagent teaming. *Journal of AI Research, 23*, 367–420.

48. Noriega, P., & Sierra, C. (2002). Electronic institutions: future trends and challenges. In M. Klusch, S. Ossowski & O. Shehory (Eds.), *Sixth international workshop on cooperative information agents (CIA 2002)*. Vol. 2446 of *lecture notes in computer science*. Springer-Verlag.
49. Odell, J., Parunak, V., Breuckner, S., & Fleischer, M. (2003a). Temporal aspects of dynamic role assignment. In F. Giunchiglia, J. Odell & Weiß, G. (Eds.), *Agent-oriented software engineering III, third international workshop (AOSE 2002), Bologna, Italy, Revised Papers and Invited Contributions*, Vol. 2585 of *Lecture Notes in Computer Science*. Springer-Verlag.
50. Odell, J., Parunak, V., Fleischer, M., & Breuckner, S. (2003b), Modeling agents and their environment. In *Agent-oriented software engineering III, third international workshop (AOSE 2002), Bologna, Italy, Revised Papers and Invited Contributions*, Vol. 2935 of *Lecture Notes in Computer Science*. Springer-Verlag.
51. Omicini, A. (2001). SODA: societies and infrastructures in the analysis and design of agent-based systems. In P. Ciancarini & M. J. Wooldridge (Eds.), *Agent-oriented software engineering (AOSE 2001)*, Vol. 1957 of *LNCS*. Springer-Verlag. First International Workshop, Limerick, Ireland. Revised Papers.
52. Omicini, A., & Ossowski, S. (2003). Objective versus subjective coordination in the engineering of agent systems. In M. Klusch, S. Bergamaschi, P. Edwards & P. Petta (Eds.), *Intelligent information agents: An agentlink perspective*, Vol. 2586 of *lecture notes computer science*. Springer-Verlag.
53. Omicini, A., Ossowski, S., & Ricci, A. (2004a). Coordination infrastructures in the engineering of multiagent systems. In F. Bergenti, M.-P. Gleizes & F. Zambonelli (Eds.), *Methodologies and software engineering for agent systems: The agent-oriented software engineering handbook*, Vol. 11 of *multiagent systems, artificial societies, and simulated organizations* (ch. 14, pp. 273–296). Kluwer Academic Publishers.
54. Omicini, A., Ricci, A., Viroli, M., Castelfranchi, C., & L. Tummolini (2004b). Coordination artifacts: Environment-based coordination for intelligent agents. In N. R. Jennings, C. Sierra, L. Sonenberg, & M. Tambe (Eds.), *Third international joint conference on autonomous agents and multiagent systems (AAMAS 2004)*. New York, USA, ACM.
55. Omicini, A., & Zambonelli, F. (1999). Coordination for internet application development. *Autonomous Agents and Multi-Agent Systems, 2*(3), 251–269. Special Issue: Coordination Mechanisms for Web Agents.
56. Parunak, V. (1997). Go to the ant: Engineering principles from natural agent systems. *Annals of Operations Research, 75* 69–101.
57. Parunak, V., Brueckner, S., & Sauter, J. (2005). Digital pheromones for coordination of unmanned vehicles. In Weyns et al. (2005a). Springer-Verlag.
58. Platon, E., Sabouret, N., & Honiden, S. (2005a). Oversensing with a softbody in the environment: Another dimension of observation. In *Modeling others from observation at international joint conference on artificial intelligence*. Edinburgh, Scotland.
59. Platon, E., Sabouret, N., & Honiden, S. (2005b). Tag interactions in multiagent systems: Environment support. In M.-P. Gleizes, G. Kaminka, A. Nowe, S. Ossowski, K. Tuyls & K. Verbeeck (Eds.), *Third European workshop on multiagent systems (EUMAS 2005)*. Brussels, Belgium.
60. Rao, A. S., Georgeff, M., & Sonenberg, E. A. (1992). Social plans: A preliminary report. *Decentralized A.I., 3*, 57–76.
61. Reynolds, C. (1987). Flocks, herds and schools: A distributed behavior model. *Computer Graphics, 21*(4), 25–34.
62. Ricci, A., Omicini, A., & Denti, E. (2003). Activity theory as a framework for MAS coordination. In P. Petta, R. Tolksdorf & F. Zambonelli (Eds.), *Engineering societies in the agents world III (ESAW 2002)*, Vol. 2577 of *lecture notes in computer science*. Springer-Verlag.
63. Rosenblatt, J. K., & Payton, D. W. (1989). A fine grained alternative to the subsumbtion architecture for mobile robot control. In *IEEE international joint conference on neural networks (IJCNN 1989)*. Washington, DC, IEEE Press.
64. Rosenschein, S. J., & Kaelbling, L. P. (1986). The synthesis of digital machines with provable epistemic properties. In *First conference on theoretical aspects of reasoning about knowledge* pp. 83–98. Monterey, CA.
65. Russell, S., & Norvig, P. (2003). *Artificial Intelligence: A modern approach*. Prentice Hall.
66. Schelfthout, K., Holvoet, T., & Berbers, Y. (2005a). Views: Customizable abstractions for context-aware applications in MANETs. In *Fourth international workshop on software engineering for large-scale multiagent systems (SELMAS 2005)*. St. Louis, Missouri, ACM Press.
67. Schelfthout, K., Weyns, D., & Holvoet, T. (2005b). Middleware for protocol-based coordination in dynamic networks. In *Third International workshop on middleware for pervasive and ad hoc computing (MPAC 2005)*. Grenoble, France, ACM Press.
68. Schumacher, M. (2001). *Objective coordination in multi-agent system engineering, design and implementation*, Vol. 2039 of *lecture notes in computer science*. Springer-Verlag.
69. Shaw, M., & Garlan, D. (1996). *Software architecture: Perspectives on an emerging discipline*. Prentice-Hall.

70. Steegmans, E., Weyns, D., Holvoet, T., & Berbers, Y. (2004). A Design Process for Adaptive Behavior of Situated Agents. In J. Odell, P. Giorgini & J. Müller (Eds.), *Agent-oriented software engineering V, fifth international workshop (AOSE 2003), New York, NY, USA, revised selected papers*, Vol. 3382 of *lecture notes in computer science*. Springer-Verlag.
71. Steels, L. (1990). Exploiting analogical representations. *Robotics and Autonomous Systems, 6*(1–2), 71–88.
72. Suchman, L. A. (1987). *Plans and situated actions: The problem of human-machine communication*. Cambridge University Press.
73. Sycara, K., Paolucci, M. Velsen, M. V., & Giampapa, J. (2003). The RETSINA MAS infrastructure. *Autonomous Agents and Multi-Agent Systems, 7*(1-2), 29–48.
74. Tummolini, L., Castelfranchi, C., Omicini, A., Ricci, A., & Viroli, M. (2005). “Exhibitionists” and “Voyeurs” do it Better: A shared environment for flexible coordination with tacit messages. In Weyns et al. (2005a), Springer-Verlag.
75. Valckenaers, P., & Holvoet, T. (2005). The environment: An essential abstraction for managing complexity in MAS-based manufacturing control. In Weyns et al. (2005a), Springer-Verlag.
76. Valckenaers, P., & Van Brussel, H. (2005). Holonic manufacturing execution systems. *CIRP Annals-Manufacturing Technology, 54*(1), 427–432.
77. Vasconcelos, W. (2004). Logic-based electronic institutions. In *Declarative agent languages and technologies: First international workshop (DALT 2003), Melbourne, Australia, July 15, 2003, revised selected and invited papers*, Vol. 2990 of *lecture notes in computer science*. Springer-Verlag.
78. Viroli, M., Omicini, A., & Ricci, A. (2005). Engineering MAS environment with artifacts. In D. Weyns, V. Parunak & F. Michel (Eds.), *Second international workshop environments for multi-agent systems (E4MAS 2005)*. AAMAS 2005, Utrecht, The Netherlands.
79. Weiss, G. (1998). *Multiagent systems, a modern approach to distributed artificial intelligence*. Cambridge, MA, USA: MIT Press.
80. Weyns, D., Boucke, N., Holvoet, T., & Schols, W. (2006). Gradient field based task assignment in an AGV transportation system. In *Fifth international joint conference on autonomous agents and multiagent systems (AAMAS 2006)*. Hakodate, Japan.
81. Weyns, D., & Holvoet, T. (2004). Formal model for situated multi-agent systems. *Fundamenta Informaticae, 63*(2), 125–158.
82. Weyns, D., & Holvoet, T. (2006). A reference architecture for situated multiagent systems. In D. Weyns, V. Parunak & F. Michel (Eds.), *Third international workshop on environments for multiagent systems (E4MAS 2006)*. Hakodate, Japan.
83. Weyns, D., Parunak, V., & Michel, F. (Eds.) (2005a). Environments for multiagent systems, first international workshop (E4MAS 2004), New York, USA, 2005. Revised Selected Papers, Vol. 3374 of *lecture notes in computer science*. Springer-Verlag.
84. Weyns, D., Parunak, V., & Michel, F. (Eds.) (2005b). Environments for multiagent systems II, second international workshop (E4MAS 2005), Utrecht, The Netherlands, 2005. revised papers and invited contributions, Vol. 3830 of *lecture notes in computer science*. Springer-Verlag.
85. Weyns, D., Parunak, V., Michel, F., Holvoet, T., & Ferber, J. (2005c). Environments for multiagent systems, state-of-the-art and research challenges. In Weyns et al. (2005a). Springer-Verlag.
86. Weyns, D., Schelfthout, K., & Holvoet, T. (2005d). Exploiting a virtual environment in a real-world application. In Weyns et al. (2005b). Springer-Verlag.
87. Weyns, D., Schelfthout, K., Holvoet, T., & Lefever, T. (2005e). Decentralized control of E’GV transportation systems. In M. Pechoucek, D. Steiner & S. Thompson (Eds.), *Fourth joint conference on autonomous agents and multiagent systems, industry track, Utrecht, The Netherlands (AAMAS 2005)*. ACM Press, New York, NY, USA.
88. Weyns, D., Steegmans, E., & Holvoet, T. (2004a). Protocol based communication for situated multi-agent systems. In N. R. Jennings, C. Sierra, L. Sonenberg & M. Tambe (Eds.), *Fourth international joint conference on autonomous agents and multiagent systems (AAMAS 2004)*. New York, USA, ACM.
89. Weyns, D., Steegmans, E., & Holvoet, T. (2004b). Towards active perception in situated multi-agent systems. *Applied Artificial Intelligence, 18*(8–9), 867–883.
90. Wooldridge, M. (2002). *An introduction to multiagent systems*. England: John Wiley and Sons, Ltd.
91. Zambonelli, F., Jennings, N., & Wooldridge, M. (2003). Developing multiagent systems: The Gaia methodology. *ACM Transactions on Software Engineering and Methodology, 12*(3) 417–470.
92. Zambonelli, F., & Parunak, V. (2002). From design to intention: Signs of a revolution. In C. Castelfranchi & W. L. Johnson (Eds.), *First international joint conference on autonomous agents and multi-agent systems (AAMAS 2002)*. Bologna, Italy.
93. Zeghal, K., & Ferber, J. (1993). CRAASH: A coordinated collision avoidance system. In *European Simulation Conference*. Lyon, France.