# Environments for Situated Multi-agent Systems: Beyond Infrastructure

Danny Weyns[1], Giuseppe Vizzari[2], and Tom Holvoet[1]

[1] AgentWise, DistriNet, Katholieke Universiteit Leuven, Belgium
{danny.weyns, tom.holvoet}@cs.kuleuven.be
[2] DISCo, Università degli Studi di Milano–Bicocca, Italy
vizzari@disco.unimib.it

**Abstract.** There is a lot of confusion on what the environment of a multi-agent system (MAS) comprises. Sometimes, researchers refer to the environment as the logical entity of a MAS in which the agents and other resources are embedded. Sometimes, the notion of environment is used to refer to the software infrastructure on which the MAS is executed. Sometimes, environment even refers to the underlying hardware infrastructure on which the MAS runs.

Our research focuses on situated MASs, i.e. MASs in which agents have an explicit position in the environment. In this paper, we propose a three-layer model for situated MASs that considers agents as well as the environment as first-order abstractions. The aim of this model is to clarify the confusion between the concept of the environment and the infrastructure on which the MAS is deployed. The top layer of the model consists of the MAS application logic, the middle layer contains the software execution platform, and the physical infrastructure is located in the bottom layer. Starting from this model, we propose a classification of situated MASs based on the physical infrastructure of the MAS. We illustrate the different classes with examples from the research community and our own practice. We apply the three-layer model to each example. The models show that agents and the environment are abstractions that crosscut the three layers of the model.

## 1 Introduction

Despite most multi-agent system (MAS) definitions include the term environment (see, e.g., [1, 2]), in general, the environment is not considered as an independent building block in MASs. Typically, the environment is conceived as communication infrastructure, implementing a specific message transfer infrastructure and mechanisms for the management of agent discovery and acquaintance. Sometimes, the notion of environment is used to refer to the software infrastructure on which the MAS is executed. Sometimes, environment even refers to the underlying hardware infrastructure on which the MAS runs. Generally, the environment is only considered as infrastructure and not as a relevant entity at the application level. At the application level, however, several aspects of MASs that conceptually do not belong to the agents themselves should not be assigned to, or hosted inside agents. Examples are the topology of a spatial domain, specification and access management of domain specific resources, or support for indirect coordination. These (and other) aspects should be dealt with explicitly and the

D. Weyns, H. Van Dyke Parunak, and F. Michel (Eds.): E4MAS 2005, LNAI 3830, pp. 1–17, 2006.

environment is the natural candidate to encapsulate these aspects. In practice however, such aspects are typically integrated implicitly in MASs, or implemented in an ad-hoc manner. This indicates that in general, the MAS research community fails to treat the environment as a *first-order abstraction*, i.e. the environment is not considered as an independent building block that encapsulates its own, clearly defined responsibilities within the MAS, irrespective of the agents [3].

The importance of the environment as a first-order abstraction is particularly apparent for situated MASs. Situated MASs are characterized by the presence of an explicit spatial structure in which agents are placed. Generally, situated MASs are also characterized by specific perception and interaction mechanisms based on contextual properties, such as agents' relative positions. Situated MASs typically provide a means for indirect coordination, e.g., with digital pheromones [4] or gradient fields [5]. The domain specific stipulation of environmental markers, the management of the coordination infrastructure, and the actual implementation of these mechanisms should not be delegated to agents, but are instead typical responsibilities of the environment.

In this paper, we introduce a three-layer model for situated MASs that considers agents as well as the environment as first-order abstractions. The main goal of this model is to analyze relationships among agents, the environment, and the MAS deployment infrastructure, aiming to bring clarity in the confusion between the concept of the environment and MAS infrastructure.

This paper is structured as follows. Section 2 discusses a three-layer model for situated MASs that considers agents and the environment as first-order abstractions. We use this model to propose a classification of situated MASs based on the physical infrastructure the MAS is built upon. Section 3 illustrates the different classes with practical examples. Finally, in Sect. 4 we draw conclusions.

## 2 A Three-Layer Model for Situated MASs

The term environment is generally included in most agent and MAS definitions, but there is much confusion on relationships between the concept of environment and the deployment infrastructure of a MAS. In this section, we describe a three-layer model for situated MAS that aims to bring clarity in this confusion. Starting from this model we then propose a classification of situated MASs based on the physical infrastructure on which the MAS is deployed. We conclude with a discussion of related work. In the next section we apply the three-layer model to three applications that belong to different classes.

### 2.1 Three-Layer Model

The proposed model is a standard deployment model for distributed applications (see, e.g., [6]) applied to situated MAS-based applications. The model for situated MAS is depicted in Fig. 1.

The model is made up of the following three layers:

- The *multiagent system (MAS) application* layer at the top (i.e., the application logic and the MAS framework);

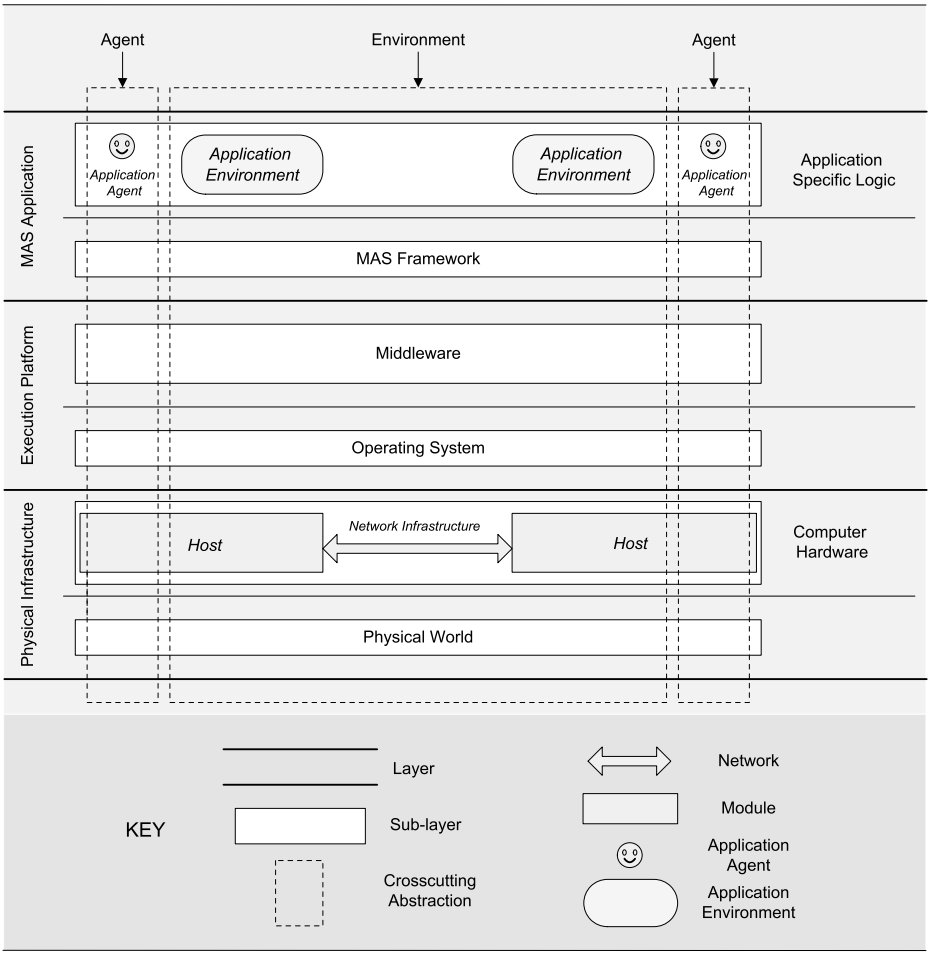


**Fig. 1.** Three-layer model for situated MASs

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

Below we elaborate on each layer and illustrate that the abstraction of the environment as well as the agents, crosscut the three layers in the model. Before that, we introduce a simple file searching system in a peer-to-peer (P2P) network [7] that we use as a running example to illustrate the different layers of the three-layer model. The idea of this application is to let mobile agents act on behalf of users and browse a shared distributed file system to find requested files. Each user is situated in a particular node (its base). Users can offer files at their base and can send out agents to find files for

them. Agents can observe the environment, however, to avoid network overload, agents can perceive the environment only to a limited extend, e.g. 2 hops from the agent's current position. An agent can perceive nodes and connecting links, bases on nodes, and files available on nodes. Agents can also sense signals. Each base emits such a signal. The intensity of the signal decreases with every hop. Sensing the signal of its base enables an agent to "climb up" the gradient, i.e. move towards its base or alternatively "climb down", i.e. move away from it. Finally, agents can sense pheromones. An agent can drop a file-specific pheromone in the environment when it returns back to its base with a copy of a file. Such a pheromone trail can not only help the agent later on when it needs a new copy of the file, it can also help other agents to find their way to that file. Pheromones evaporate, thereby limiting their influence over time. This is an important property to avoid that agents are misled when a file disappears from a certain node.

We now zoom in on each layer of the three-layer model.

**Multiagent System Application Layer.** The *MAS Application* layer consists of two sub-layers:

- The *Application Specific Logic* layer, which comprises the *Application Agents* and the *Application Environment* of the MAS, which represent the solution for the specific problem context. The Application Agents are the autonomous entities in the MAS, the Application Environment provides an application specific representation of the domain to Application Agents. The Application Environment enables Application Agents to interact with domain resources and with other Application Agents. The Application Environment offers a domain specific abstraction to Application Agents, hiding the complexity of resource access, interaction handling and consistency management. The Application Agents in the P2P file searching system are the logical entities that are created by the users to search for files in the network. The Application Environment is the logical entity that represents the space in which the Application Agents perform their job. The Application Environment offers a representation to the Application Agents of the neighboring nodes and connecting links of the network. The Application Environment also represents the available files, the gradient fields emitted by the bases, and the file-specific pheromones dropped by the agents.
- The *MAS Framework* layer: the Application Specific Logic is typically deployed on top of a *MAS Framework*. The latter supplies predefined MAS abstractions, such as a particular engine for agent's decision making, support for communication, a model for action, etc. These abstractions can be reused over different applications. In the P2P file searching system, the MAS framework layer likely provides a pheromone infrastructure and infrastructure for gradient fields. Another example is support for mobility of the agents.

**Execution Platform.** The Execution Platform is in turn subdivided in two sublayers:

- A *middleware* layer which serves as the glue between (distributed) components. It provides support for remote procedure calls, threading, transactions, persistence,

load balancing, generative communication, etc. In general, middleware offers a software platform on which distributed applications can be executed. An example of middleware support in the P2P file searching system is a distributed tuple-space infrastructure that provides a basic substrate for the pheromone and gradient field infrastructure.
- An *Operating System* layer that enables the execution of the application on the physical hardware and it offers basic functionality to applications, hiding low-level details of the underlying physical platform. The Operating System manages memory usage and offers transparent access to lower level resources such as files, it provides network facilities, it handles the intervention of the users, it provides basic support for timing, etc. The operating system provides many basic functions, one example is the file system.

**Physical Infrastructure.** The Execution Platform runs on top of the Physical Infrastructure, which can is generally divided in two parts:

- The *Computer Hardware*, which contains the *Hosts* with processors and the connecting *Network Infrastructure*. In the P2P file sharing system, the physical infrastructure consists of a computer machine on each node and a connecting network. Each machine is a possible access point to the system for a user.
- The *Physical World*, which refers to the physical parts of the MAS, if present in the application. In the P2P file sharing system this aspect is not relevant, and thus this layer is empty.

**Agent and Environment Crosscutting Abstractions.** Situated MAS applications typically comprise all three layers, although some sub-layers may be empty, e.g. when the MAS application is built from scratch, the MAS Framework layer is empty. Agents and the environment span the three layers of the three-layer model, this is graphically depicted in Fig. 1 with the dashed vertical rectangles. An agent, first of all, is composed of an application specific part, i.e. the Application Agent located in the MAS Application layer. The realization of this Application Agent may be based on a generic MAS Framework that, in turn, exploit an underlying Middleware and the Operating System services. Finally, the agent software is hosted and executes on a physical system that is part of the Physical Infrastructure layer. Analogously, the environment consists of an application specific part that corresponds to the Application Environment, located in the MAS Application layer. The Application Environment is typically built on top of a MAS Framework that is supported by generic Middleware and Operating System services. As for the agents, the environment software executes on a physical system, that includes Hosts provided with processors and an interconnecting Network Infrastructure.

The proposed three-layer model promotes the environment to a first-order abstraction, on the same level as the agents. Considering the environment as a first-order abstraction urges researchers to deal with responsibilities of the environment explicitly, and also promotes the modelling, design and implementation of concepts like perception, action handling, and locality in comprehensive ways. In particular, it must be noted that one of the responsibilities of the environment is to provide situated agents with proper perceptions, which may include information of the physical world obtained

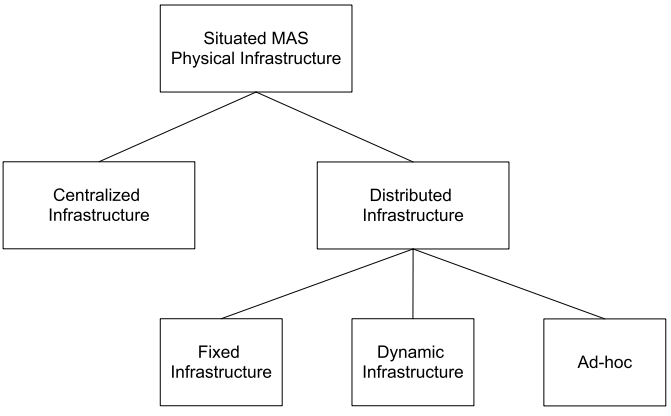


**Fig. 2.** A classification of situated MASs based on the physical infrastructure

through sensors. On the other hand, agents' actions may have effects which extend over the software environment and may cause modification on the physical world via actuators. The three-layer model also stresses the need to consider the "vertical" relationships among different layers.

## 2.2 Classification of Situated MASs

Starting from the three-layer model, we now present a classification of situated MASs based on the physical infrastructure on which the MAS is deployed, see Fig. 2. The goal of this classification is to further clarify the relationship between the agent and environment abstractions and the MAS infrastructure. The structural aspects of the environment are particularly relevant for situated MASs, as agents are deeply influenced by their position, which generally determines their perceptions and (inter)actions.

The physical MAS infrastructure can be centralized (*Centralized Infrastructure*), i.e., deployed on a single computer, or distributed, i.e., deployed on a set of computers that are connected through a computer network (*Distributed Infrastructure*). Distribution can be a constraint of the application, or a well-considered architectural decision. In a centralized MAS infrastructure, the environment is an encapsulated software entity deployed on a single computer. In a distributed MAS infrastructure, the distributed environment is a logical entity that physically consists of a set of software entities deployed on different nodes that are connected through a network. In a centralized setting, agents experience the environment as one shared entity that is locally accessible. In a distributed setting, agents can be aware of distribution or distribution may be transparent to agents. Distribution of an environment is supported by generic middleware infrastructure.

Besides the distinction between centralization or distribution, we further distinguish between the dynamics of the distributed infrastructure. The distributed infrastructure of a MAS can be static or change dynamically. In a static infrastructure, the number of computers and the layout of the connecting network does not change over time (*Fixed Infrastructure*). A topology of a dynamic infrastructure changes over time due to newly

added nodes or nodes that disappear (*Dynamic Infrastructure*). In other cases, the environment is not based on a predefined notion of adjacency, but generates ad-hoc spatial relationships reflecting for instance the positions of physical agents in an actual environment (*Ad-hoc*).

The complexity related to the design and implementation of hardware/software infrastructures related to agents and the environment grows in distributed scenarios, especially in cases that do not provide a fixed predefined infrastructure for the arrangement of agents and the environment.

### 2.3 Related Work

To our best knowledge, no deployment models for MASs were previously proposed that explicitly discusses the position of agents and the environment. However, several layered models for MAS infrastructure are discussed in literature. Here we look at three representative examples: Retsina, Jade and TOTA.

Retsina (Reusable Environment for Task-Structured Intelligent Network Agents) is a well-known MAS infrastructure [8]. Retsina is an open MAS infrastructure that supports communities of heterogeneous agents. The Retsina MAS infrastructure is built up in several layers. The bottom layer contains the operating environment that provides the platform on which the infrastructure components and the agents run. Retsina supports a broad range of execution platforms and it automatically handles different types of network transport layers. The operating environment corresponds to the Physical Infrastructure layer in the three-layer model presented in this paper. On top of the operating environment, Retsina defines eight different layers. The communication infrastructure layer provides communication channels for message transfer between peers, and multicast that is used for a discovery process to let the agents find infrastructural components. The ACL infrastructure layer provides an ontology and a protocol engine with a protocol language. The MAS management services layer offers tool support to monitor the activity of the agents and to launch the applications. The security layer supports agent authentication, secure communication and integrity of the Retsina infrastructure components. The ANS (Agent Name Services) layer provides a means to abstract away from physical locations by mapping agent identifiers to network addresses. The Matchmakers layer provides a mapping between agents and services. Service providers can advertise their services at the matchmakers and agents can request the matchmakers to get contact information of relevant providers. Finally, the Retsina-OAA InterOperator on top of the Retsina MAS infrastructure bridges the Retsina MAS infrastructure with the OOA platform (Open Agent Architecture). These eight layers provide middleware and MAS specific services that conceptually belongs to the Middleware layer and the MAS Framework layer in the three-layer model presented in this paper.

Jade (Java Agent Development Environment) [9] is a pure Java, middleware platform intended for the development of distributed multiagent applications based on peer-to-peer communication. Jade includes Java classes to support the development of application agents and the "run-time environment" that provides the basic services for agents to execute. An instance of the Jade run-time is called a container, and the set of all containers is called the platform. The platform provides a middleware layer that hides

from agents the complexity of the underlying execution system. Jade includes a naming service ensuring that each agent has a unique name, and a yellow pages service that can be distributed across multiple hosts. Agents can dynamically discover each other and communicate by exchanging asynchronous messages. Jade provides a set of skeletons of typical interaction protocols. The Jade platform also supports mobility of code, enabling agents to stop running on a host, migrate to a different remote host and restart execution from the point they stopped. The Jade middleware layer corresponds to the MAS Framework layer in the three-layer model presented in this paper. The Jade layer executes on top of a Java Virtual Machine layer that provides generic middleware support for Web services, distributed communication, threading, transaction management, security, etc.

In [10], Mamei and Zambonelli introduce the notion of "spatial computing stack" and apply it to the TOTA (Tuples On The Air) middleware. The spatial computing stack defines a framework for spatial computing mechanisms at four levels: the physical level at the bottom, the structure level above it, then follows the navigation level, and finally the application level at the top. The "physical level" deals with how components find each other and start communication with each other. In the case of TOTA, a node detects in-range nodes via one-hop message broadcast. The "structure level" is the level at which a spatial structure is built and maintained by components in the physical network. In TOTA, a tuple can be injected from a node. A TOTA tuple is defined in terms of a content and a propagation rule. The content represents the information carried on by the tuple and the propagation rule determines how the tuple should be propagated across the network. Once a tuple is injected it propagates and creates a centered spatial structure in the network representing some spatial feature relative to the source. At the "navigation level" components exploit basic mechanisms to orient their activities in the spatial structure and to sense and affect the local properties of space. TOTA defines an API to allow application components to sense TOTA tuples in their one-hop neighborhood and to locally perceive the space defined by them. Navigation in the space consists of agents acting on the basis of the local shape of specific tuples. At the "application level", navigation mechanisms are exploited by application components to interact and organize their activities. TOTA enables complex coordination tasks in a robust and flexible way. An example is a group of agents that coordinate their respective movements by following locally perceived tuples downhill or uphill resulting in specific formations. The spatial computing stack model extends over the three layers of the model presented in this paper. The physical level is situated in the Physical infrastructure, the structure and navigation level are situated in the Middleware layer, and the application level finally is situated in the MAS Application layer.

## 3 Applying the Three-Layer Model for Situated MASs

In this section, we apply the three-layer model for situated MASs to three MAS applications with different physical infrastructures. First we look at a multiagent-based simulation application that is deployed on a centralized infrastructure. Then we look at a MAS-based control system that is deployed on an ad-hoc infrastructure. Finally, we zoom in on a mobile MAS application that is deployed on a dynamic infrastructure.

### 3.1 MMASS-Based Crowd Simulation

In this section, we discuss an application that simulates a crowd adopting a situated MAS [11] approach. In particular, the application supports the modelling and study of crowds and pedestrian behaviour in large rooms, e.g., lecture halls. According to the classification proposed in Sect. 2.2, this application is classified as *centralized infrastructure*. The goal of the simulation is to support architects and designers of large rooms in their decision making activities, for instance to determine the number and positions of emergency exits. Given a design and specific starting conditions, the architect can then obtain an indication of the behaviour of a crowd, for example in an evacuation situation.

There are several approaches to this problem which adopt a Cellular Automata (CA) based model (see, e.g., [12]), but they generally relax the basic CA model and allow action–at–a–distance. Moreover these approaches typically model only homogeneous behaviour of the simulated entities by means of cell states and transition rules that also include the local state of the environment and the laws that regulate its dynamics. In this way, the environment and the embedded entities are mixed up, causing large cell states and very complex transition rules. The MAS approach instead provides a clean separation between the environment and the entities which inhabit it, and also allows to model heterogeneity of agents in a more convenient way.

For the application, the Multilayered Multi Agent Situated System (MMASS) [13] model was adopted to support the design and development of the crowd simulation application. MMASS provides an explicit spatial representation of the environment and an interaction model strongly related to the agents' context. In MMASS, agents can (1) interact through a reaction with adjacent entities, (2) emit fields that are diffused in the environment, and (3) can be perceived by other agents. Fig. 3 depicts the three-layer model for situated MASs, applied to the MMASS simulation application.

*MAS Application Layer.* The crowd simulation system is composed of a set of Pedestrian Agents situated in a Lecture Hall (i.e. a virtual environment that represents a bidimensional abstraction of a physical space). Pedestrians (and other relevant elements of the environment) generate fields that can be perceived by pedestrians according to specific diffusion and perception mechanisms. The perception of these fields, and their local state, influences pedestrian behaviors. In critical situations, the pedestrians can use the perceived fields to move towards emergency exits.

All the application specific elements are built on top of the MMASS Framework. The MMASS Framework offers a set of basic components for applications that use the MMASS model. In particular the framework supports: the definition of the *spatial structure* of the environment by means of basic elements (i.e. nodes and edges); *interaction* among agents by means of field based mechanisms and though the reaction among adjacent agents; and support the definition of domain specific agents through *agent templates*, which define the fundamental elements common to all MMASS agents.

*Execution Platform.* The MMASS framework, that is based on Java technology, exploits some basic library for XML file access for configuration matters. Facultatively,

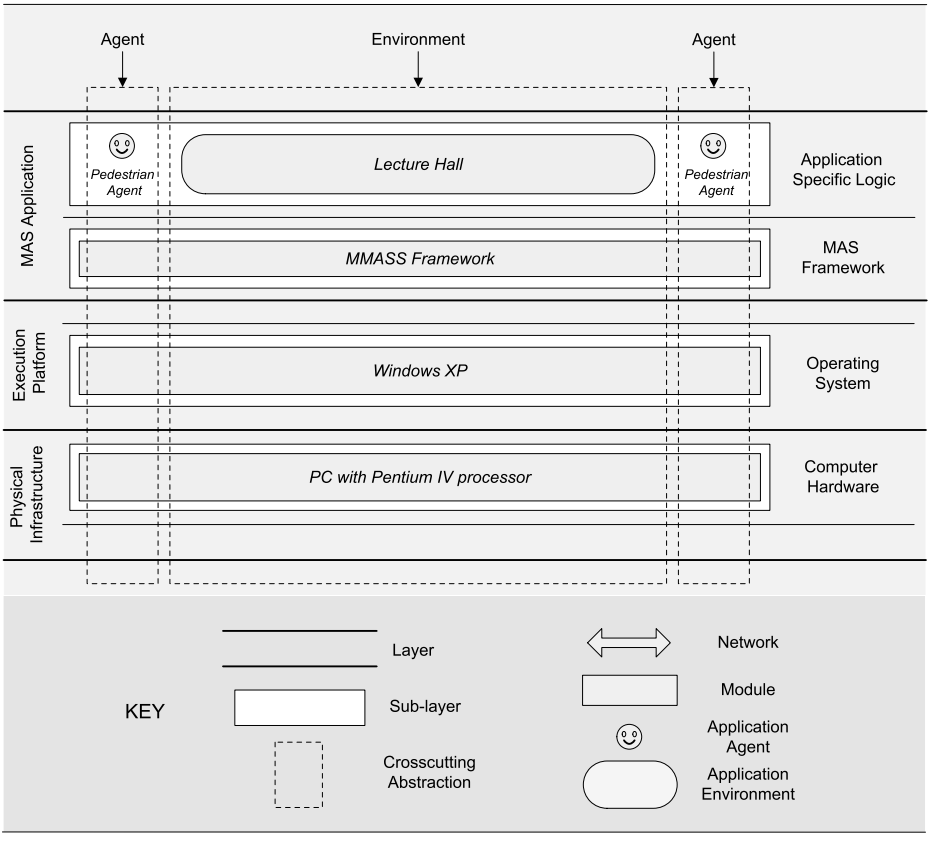


**Fig. 3.** The three-layer model for the MMASS based simulation case

the simulator is able to generate a 3D visualization by means of 3D Studio Max, a commercial 3D modelling and visualization tool [1]. For reasons of performance, the application can be distributed, however, in this specific case, a centralized approach is used, so there is no need for supporting middleware.

The crowd simulation runs on the Windows XP operating system.

*Physical Infrastructure.* The physical infrastructure of the application consists of a single PC provided with a Pentium IV processor.

### 3.2 Automated Guided Vehicles Coordination

In this section, we apply the three-layer model to a real-world application that uses a situated MAS to the control of an automatic guided vehicle (AGV) transportation system [14]. This application is classified as *ad-hoc* infrastructure in the classification

[1] http://www4.discreet.com/3dsmax/

proposed in Sect. 2.2. The application is developed in the context of a R&D project between the AgentWise research group and Egemin[2], a manufacturer of industrial automated logistic service systems. Traditionally, AGV transportation systems use a central server that controls the system. Although efficient, the centralized architecture lacks flexibility. In the project we investigate the feasibility of a decentralized architecture aiming to improve flexibility.

An AGV transportation system uses unmanned vehicles to transport loads through a warehouse. Typical applications are repackaging and distributing incoming goods to various branches, or distributing manufactured products to storage locations. AGVs can move through a warehouse, guided by a laser navigation system or by magnets or cables that are fixed in the floor. AGVs are provided with a battery as energy source.

The main functionalities of an AGV transportation system are: (1) perform transports: transports are generated by client systems (warehouse management system, operator, etc.) and have to be assigned to AGVs that can execute them; (2) collision avoidance and deadlock prevention; (3) when an AGV is idle it has to park at a free park location; (4) when an AGV runs out of energy, it has to charge its battery at one of the charging stations. The low-level control of the AGVs in terms of sensors and actuators (staying on track on a segment, turning, and determining the current position, etc.), is handled by the low-level AGV control software called E'nsor[3].

Fig. 4 depicts the three-layer model for the AGV transportation system.

*MAS Application Layer.* The situated MAS consists of an environment and two kinds of agents, *Transport Agents* and *AGV Agents*. Transport Agents are located at transport bases, a transport base may host one or more Transport Agents. A transport base is a computer system that is in charge to manage the transports of a particular area in the warehouse. Together, the transport bases, connected through a wired network, cover the whole layout of the warehouse. AGV Agents are located on mobile AGV machines that are situated on the factory floor. With each AGV there is one AGV Agent associated.

Transport bases receive transport requests from client systems, i.e. typically a warehouse management system, but it can also be another logistic machine or even an operator. For each new transport request, a new Transport Agent is created that is responsible to assign the transport to an AGV and to ensure that the transport is completed correctly. The Transport Agent also determines the priority of the transport. The priority of a transport depends on the kind of transport, the pending time since its creation, and the nature of other transports in the system. Transport agents interact with other related transport agents to determine the correct priority over time. AGV Agents are responsible for executing the assigned transports.

Since the physical environment of a factory is very constrained, it restricts how agents can use their environment. Therefore a Virtual Environment has been introduced for the agents to live in. This Virtual Environment offers an application specific medium that Application Agents can use to exchange information and coordinate their behavior. One example of the use of the Virtual Environment are road signs. The Virtual Environment provides a logical map consisting of nodes and segments that corresponds with

[2] http://www.egemin.com/

[3] E'nsor ® is an acronym for Egemin Navigation System On Robot.

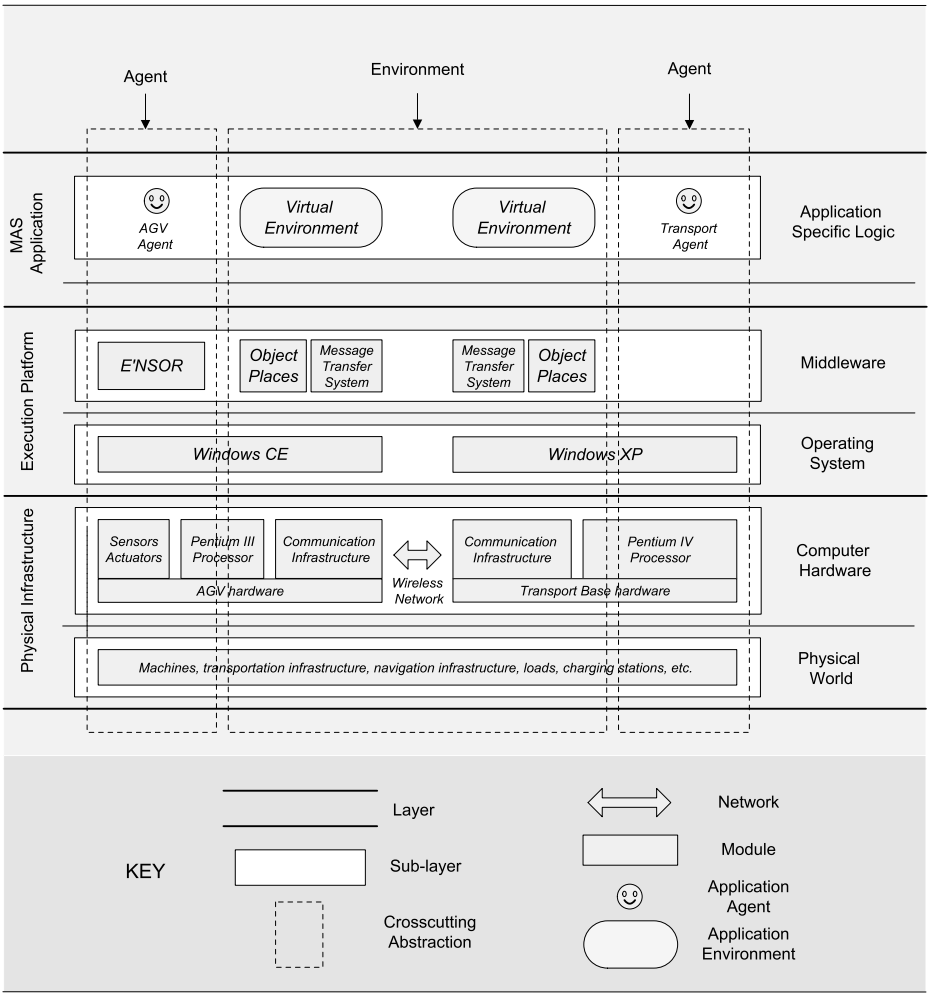


**Fig. 4.** Three-layer model applied to the AGV transportation application

the physical layout of the factory floor. 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 per segment is based on the average time it takes for an AGV to drive over the segment. This cost has a static part that depends on the length and the properties of the segment, and a dynamic part that depends on the recent traffic load on the segment. The Virtual Environment maintains the dynamic part of the cost of a segment according to the time AGVs are delayed on the segment. The AGV Agent perceives the signs in the Virtual Environment, and uses them to determine which segment it will take next. Transport Agents use the Virtual Environment to find AGV agents to assign the transports, and to follow the progress of the assigned transports. To assign the transport, the Transport Agent negotiates with AGV Agents of idle AGVs near to

the location of the load. Once the transport is assigned, the awarded AGV handles the transport.

*Execution Platform.* The *Message Transfer System* enables agents to send messages to each other. The E'nsor software that deals with the low-level control of the AGVs is fully reused. As such the AGV Agents control the movements of the AGVs on a fairly high level.

Since the only physical infrastructure available to the agents is a wireless network to communicate, the Virtual Environment is necessarily distributed. In effect, each AGV and transport base in the system maintains a local Virtual Environment, which is a local manifestation of the Virtual Environment. Synchronization of the state of the local Virtual Environment with local Virtual Environments of neighboring AGVs and transport bases is supported by the ObjectPlaces [15] middleware. The local Virtual Environment uses the ObjectPlaces middleware by sharing objects in a tuplespace-like container, called an *objectplace*. Each AGV and each transport base has one objectplace locally available. Objects in objectplaces on remote AGVs and transport bases can be gathered using a *view*. A view specifies (1) which objectplaces need to be included in the view (e.g. the objectplaces of all AGVs within a specific range), and (2) what objects need to be included in the view (e.g. positions of AGVs).

The AGV software runs on Windows CE, the Transport Base software runs on Windows XP.

*Physical Infrastructure.* The AGV machines are equipped with a Pentium III processor. AGVs can interact with the physical infrastructure via sensors, actuators and communication infrastructure. Transport bases are equipped with a Pentium IV processor and provides communication infrastructure for Transport Agents to communicate. Communication between AGVs and transport bases happens via a wireless communication network. The factory floor consists of navigation infrastructure for the AGVs, the transportation system infrastructure, the loads that AGVs have to transport, etc.

### 3.3 TOTA: A Mobile Computing Application

As a final example, we discuss an application that supports visitors of a museum to retrieve information about art pieces, to orientate in the museum, and to meet each other in case of organized groups [5]. This mobile computing application is deployed on top of the TOTA [16] middleware. The application is classified as *dynamic infrastructure* in the classification we have proposed in Sect. 2.2.

Visitors are provided with PDAs, and further it is assumed that the museum is provided with a dense distributed network of computer-based devices, associated with rooms, art pieces, alarm systems, climate conditioning systems, etc. The topology of this network dynamically changes when visitors enter, leave or move through the museum, or also when art pieces are moved, e.g., for special exhibitions. The activities of visitors are typically contextual, i.e., related to the environmental setting (rooms, types of art pieces, members of a group, etc.).

The museum application is build on top of the TOTA middleware (see also Sect. 2.3). TOTA enables the interaction among a network of possibly mobile nodes, each running a local version of TOTA. Each node holds a reference to a limited set of neighboring nodes. The structure of the network is automatically updated by the nodes to support dynamic changes (nodes that enter, move or fail). Entities that live in this dynamic space are able to inject tuples on each node. A TOTA tuple is defined in terms of a content and a propagation rule. Tuples injected in a node are spread by the middleware according to the propagation rule. This rule can also defines how the content of the tuple changes during propagation. In this way it is possible to implement spacial related coordination mechanisms, such as fields, removing the burden of coordination from the agents. A detailed study of TOTA can be found in [16].

Fig. 5 depicts the three-layer model applied for the museum application.

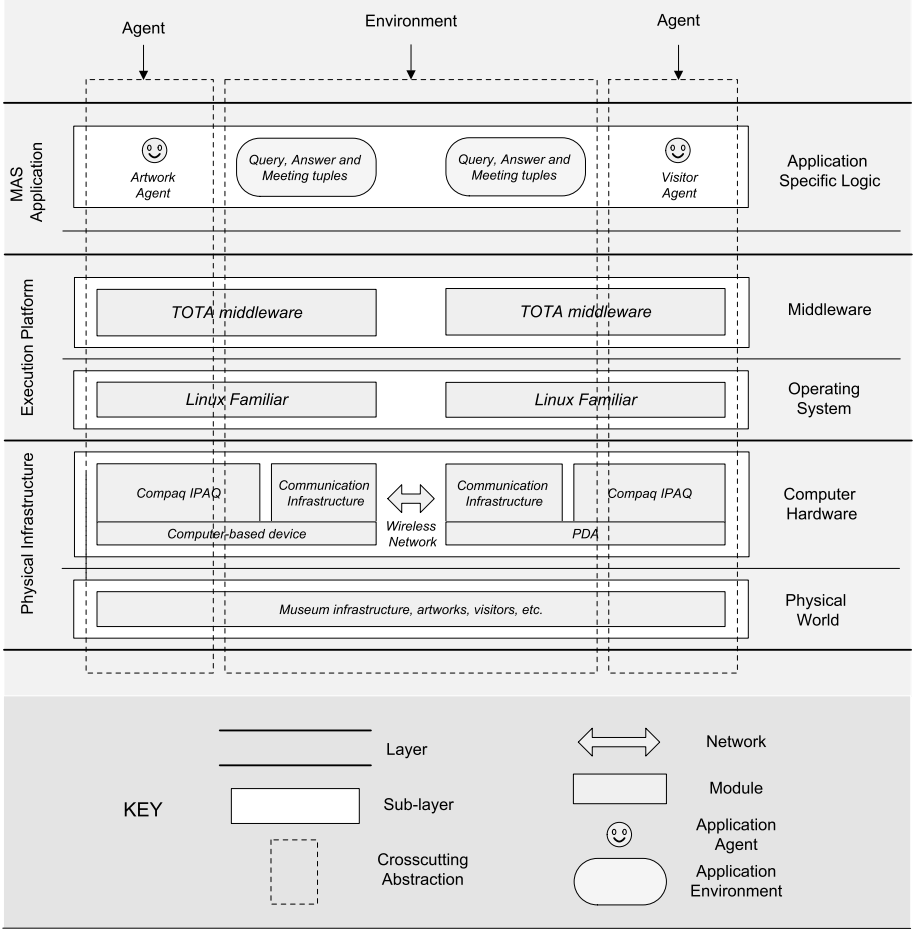


**Fig. 5.** The three-layer model applied to the museum application

*MAS Application Layer.* The situated MAS consists of two basic types of agents, Artwork Agents and Visitor Agents that are associated with active entities in the museum. Each agent is able to inject and perceive application specific tuples in their environment. Besides, the MAS consists of system nodes that provide support for tuple propagation (not depicted in Fig. 5). A Visitor Agent can inject a Query tuple in the TOTA infrastructure to indicate his/her interest for a particular art piece. Such a tuple creates a gradient field leading to the queried art piece. The corresponding Artwork Agent react to the Query tuple by injecting an Answer tuple. This Answer tuple can reach a tourist even while he/she is moving. Thus, the gradient fields guide the interested visitors towards the source of interest. If a visitor is interested in locating a specific artwork, its Visitor Agent senses the field generated by that artwork, that guides him toward the artwork [4].

Visitor Agents can also express their interest for a group meeting. Therefore the tourists inject Meeting tuples in the TOTA infrastructure. Tourists then have to follow downhill the gradient field generated by the farther other tourist in the group. This way tourists will move toward each other, to meet in their barycenter room.

Artwork and Visitor Agents are examples of Application Agents, while Query, Answer and Meeting tuples and their corresponding gradient fields are domain specific objects that are part of the Application Environment.

*Execution Platform.* TOTA is a generic middleware infrastructure that supports mechanisms for the management of field diffusion (i.e. transmission of fields among TOTA peers) and the management of dynamism in the structure of the TOTA network. TOTA offers support to develop application specific tuples as well as agents. For example, the Meeting tuple in the museum application is based on the generic Gradient tuple and Downhill tuple defined by TOTA, and the Artwork Agent is based on the generic AgentInterface also provided by the TOTA middleware.

The museum application runs on the Familar distribution of Linux.

*Physical Infrastructure.* The museum application is hosted on Compaq IPAQ PDAs, equipped with 802.11b wireless network devices. A similar kind of equipment must be associated with the other nodes of the network, including the artworks that host Artwork Agents.

### 3.4 Discussion

The three example applications clearly illustrate how agents and the environment crosscut the three layers of the MAS model. In the MMASS application, the MAS Application (i.e., Application Agents and the Application Environment) runs on top of a dedicated MAS Framework, while in the AGV and the TOTA applicaton the MAS Application directly runs on top of generic middelware infrastructure. In general, applications of the class *distributed infrastructure* are candidates to be supported by generic middleware.

In the MMASS and the AGV application, the Application Agents experience the Application Environment as a common shared entity. In the TOTA example, the Application Agents are aware of the network topology, changes in the context are reflected in

[4] We have simplified the explanation of this example, for a detailed discussion see [5].

modifications of perceived gradient fields. An interesting research issue is the relationship between the way Application Agents experience the Application Environment and the underlying executing platform and the physical infrastructure.

All the discussed applications reify elements of the physical environment. All of them also augment this environment with additional elements (gradient fields, marks, etc.) to enable the situated agents to better exploit the environment. Such additional support for indirect interaction has consequences on different layers of the applications, typically the two top layers of the model. This additional support for indirect interaction illustrates how the environment, as a first-order abstraction, can be used creatively in the design and implementation of the problem solution.

## 4 Conclusion

Generally, the environment is not considered as a first-order abstraction in the MAS research community. Often, the environment in MASs is confused with the infrastructure on which the MAS is deployed. As a consequence, the functionality of the environment is mostly integrated in the MAS in an implicit or ad-hoc manner. To clarify the confusion between the concept of the environment and the infrastructure of the MAS, we have presented a three-layer model for situated MASs. The three-layer model promotes the agents as well as the environment as first-order abstractions. The MAS application logic is located on top, the middle layer consists of the software infrastructure, and the bottom layer of the model represents the physical infrastructure. Agents and the environment crosscut the three layers of the model.

Starting from this model, we have proposed a classification of situated MASs based on the physical infrastructure. We have applied the three-layer model to three applications that represent different classes of the classification.

The major conclusion are:

1. Environment and infrastructure are no synonyms; more than that, the Application Environment as well as the Application Agents exploit infrastructure of the MAS.
2. The Application Environment is a powerful instrument that can be used creatively in the design of a MAS solution, helping to manage the complexity of engineering real-world applications.

An interesting track for future research in to study the relationship between the structure of the environment at the MAS Application layer (as experienced by the Application Agents) and the underlying execution platform and physical infrastructure.

## Acknowledgements

This research is supported by the K.U.Leuven research council (AgCo2) and the Flemish Institute for Advancement of Research in Industry (EMC$^2$), and was partially funded by the Italian Ministry of University and Research within the FIRB project Multichannel Adaptive Information Systems.

## References


1. Wooldridge, M., Jennings, N.: Intelligent agents: Theory and practice. The Knowledge Engineering Review **10** (1995) 115–152
2. Ferber, J.: Multi–Agent Systems, An Introduction to Distributed Artificial Intelligence. Addison–Wesley (1999)
3. Weyns, D., Parunak, V., Michel, F., Holvoet, T., Ferber, J.: Environments for Multiagent Systems State-of-the-Art and Research Challenges. In: E4MAS. Volume 3374 of Lecture Notes in Computer Science., Springer (2005)
4. Brueckner, S.: Return from the Ant, PhD Dissertation. Humboldt-Universitat Berlin, Germany (2000)
5. Mamei, M., Zambonelli, F., Leonardi, L.: Programming Pervasive and Mobile Computing Applications with the TOTA Middleware. In: 2nd IEEE International Conference on Pervasive Computing and Communication (Percom2004), IEEE Computer Society (2004)
6. Couloris, G., Dollimore, J., Kindberg, T.: Distributed Systems: Concept and Design (3rd ed.). Addison Wesley (2001)
7. Weyns, D., Steegmans, E., Holvoet, T.: Towards Active Perception in Situated Multi-Agent Systems. Applied Artificial Intelligence **18** (2004) 867–883
8. Sycara, K., Paolucci, M., Velsen, M.V., Giampapa, J.: The RETSINA MAS Infrastructure. Autonomous Agents and Multi-Agent Systems **7** (2003) 29–48
9. Bellifemine, F., Poggi, A., Rimassa, G.: Jade, A FIPA-compliant Agent Framework. 4th International Conference on Practical Application of Intelligent Agents and Multi-Agent Technology (1999)
10. Mamei, M., Zambonelli, F.: Spatial Computing: the TOTA Approach. Self-* Approaches to Distributed Computing, Lecture Notes in Computer Science Hot Topics Series (2005)
11. Bandini, S., Manzoni, S., Vizzari, G.: Situated Cellular Agents: A Model to Simulate Crowding Dynamics. IEICE Transactions on Information and Systems: Special Issues on Cellular Automata **E87-D** (2004) 669–676
12. Schadschneider, A.: Cellular automaton approach to pedestrian dynamics. In: Pedestrian and Evacuation Dynamics. Springer-Verlag (2002) 75–98
13. Bandini, S., Manzoni, S., Simone, C.: Dealing with Space in Multi–Agent Systems: A Model for Situated MAS. In: Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems, ACM Press (2002) 1183–1190
14. Weyns, D., Schelfthout, K., Holvoet, T.: Design and evolution of autonomic application software, DEAS, St. Louis, USA, 2005. (http://www.cs.kuleven.ac.be/∼danny/deas_2005.pdf)
15. Schelfthout, K., Weyns, D., Holvoet, T.: Middleware for protocol-based coordination in dynamic networks. In: MPAC '05: Proceedings of the 3rd international workshop on Middleware for pervasive and ad-hoc computing, New York, NY, USA, ACM Press (2005)
16. Mamei, M., Zambonelli, F., Leonardi, L.: Tuples On The Air: A Middleware for Context Aware Computing in Dynamic Networks. In: International ICDCS Workshop on Mobile Computing, IEEE Computer Society (2003)