Environments for Multiagent Systems State-of-the-Art and Research Challenges

Danny Weyns[1], H. Van Dyke Parunak[2], Fabien Michel[3], Tom Holvoet[1], and Jacques Ferber[3]

[1] AgentWise, DistriNet, K.U.Leuven, B-3001 Leuven, Belgium
{danny.weyns, tom.holvoet}@cs.kuleuven.ac.be
[2] Altarum Institute, Ann Arbor, MI 48105-1579, USA
van.parunak@altarum.org
[3] LIRMM, CNRS, Montpellier, 34392 Montpellier Cedex 5, France
{fmichel, ferber}@lirmm.fr

Abstract. It is generally accepted that the environment is an essential compound of multiagent systems (MASs). Yet the environment is typically assigned limited responsibilities, or even neglected entirely, overlooking a rich potential for the paradigm of MASs.

Opportunities that environments offer, have mostly been researched in the domain of situated MASs. However, the complex principles behind the concepts and responsibilities of the environment and the interplay between agents and environment are not yet fully clarified.

In this paper, we first give an overview of the state-of-the-art on environments in MASs. The survey discusses relevant research tracks on environments that have been explored so far. Each track is illustrated with a number of representative contributions by the research community. Based on this study and the results of our own research, we identify a set of core concerns for environments that can be divided in two classes: concerns related to the structure of the environment, and concerns related to the activity in the environment. To conclude, we list a number of research challenges that, in our opinion, are important for further research on environments for MAS.

1 Introduction

There is a general agreement in the multiagent research community that environments are essential for multiagent systems (MASs). Yet most researchers neglect to integrate the environment as a primary abstraction in models and tools for MASs, or minimize its responsibilities. As a consequence, a rich potential of applications and techniques that can be developed using MASs is overlooked.

Popular frameworks such as Jade [9], Jack [44], Retsina [79] or Zeus [58] reduce the environment to a message transport system or broker infrastructure. Well-known methodologies such as Message [25], Prometheus [66] or Tropos [12] offer support for some basic elements of the environment, however they fail to consider the environment as a first-class entity. Standard literature on MASs

D. Weyns et al. (Eds.): E4MAS 2004, LNAI 3374, pp. 1–47, 2005.

used for education, including [73, 93, 45], only deals very briefly with the topic of environments. Even in the FIPA [34] specifications it is hard to find any functionality for the environment beyond message transport or broker systems. Restricting interaction to inter-agent communication neglects a rich potential of possibilities for the paradigm of MASs.

Researchers working in the domain of situated MASs traditionally integrate the environment as a first-class entity in a MAS. In situated MASs, the environment is an active entity with its own processes that can change its own state, independent of the activity of the embedded agents. Inspired by biological systems, several researchers have shown that the environment can serve as a robust, self-revising, shared memory for agents. This can unburden the individual agents from continuously keeping track of their knowledge about the system. Moreover, it enables the agents to use their environment as an excellent medium for indirect coordination. Gradient fields and evaporating marks in the environment can guide agents in their local context and as such facilitate the coordination in a community of agents in a decentralized fashion. Several practical applications have shown how the environment can contribute to manage complex problems. There are examples in domains such as supply chain systems, network support, peer-to-peer systems, manufacturing control, multiagent simulation etc. Since the exploitation of the environment in MASs results in better manageable solutions, it is a promising paradigm to deal with the increasing complexity and dynamism of future system infrastructure and more advanced problem domains, e.g. ad hoc networks or ubiquitous computing.

Despite the large amount of work in the domain of situated MASs, we are just at the very beginning of understanding the complex principles behind the concepts related to the environment and the interplay between agents and the environment. This paper aims to contribute in three ways. First we give an overview of the state-of-the-art on environments for MASs. Based on this study as well as the results of our own research, we identify a set of core concerns for environments, as a second contribution. Third, we outline a number of research challenges that, in our opinion, are important for the future development of environments for MASs.

2 Organization of the Paper

In Sect. 3, we start with an overview of the state-of-the-art on environments for MASs. Studying MAS literature with a focus on environments is a tough job. During our study, we encountered two types of difficulties: (1) the term *environment* has several different meanings, causing a lot of confusion, (2) the functionalities associated with the environment are often treated implicitly, or integrated in the MAS in an ad-hoc manner.

The confusion on what the environment comprises is mainly caused by mixing up concepts and infrastructure. Sometimes, researchers refer to the environment as the logical entity of a MAS in which the agents and other objects/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.

The fact that functionalities of the environment are often treated implicitly, or in an ad-hoc manner, indicates that in general, the MAS research community fails to treat the environment as a *first-class entity*. [36] defines a first-class module as: "a program building block, an independent piece of software which [...] provides an abstraction or information hiding mechanism so that a module's implementation can be changed without requiring any change to other modules." Thus, the environment is in general not treated as an independent building block that encapsulates its own clear-cut responsibilities in the MAS, irrespective of the agents.

Starting from this perspective, the overview of the state-of-the-art on environments for MASs we discuss in Sect. 3 is not just a summary of representative papers on the topic of environments for MASs. In fact, the number of research papers that are devoted to the environment is very limited. The overview is rather a reflection on MAS research literature in which we have put the spotlight on models and concepts associated with the environment. The survey is structured as follows:

3.1 General models for environments (Russell and Norvig, Ferber, Odell et al., Environments for mobile agents)
3.2 Inter-agent facilities
- Communication infrastructure (Huhns & Stephens, FIPA, Jade, Retsina)
- Models for indirect interaction
  - Classical blackboard communication
  - Tuple-based interaction models (JavaSpaces, Lime)
  - Stigmergy (Synthetic ecosystems, Network routing)
  - Interaction models related to space in MASs (MMASS)
- Environment as an organizational layer (AGR)

3.3 Agent-Environment interaction
- Perception of the environment (Robocup Soccer Server, Model for active perception)
- Dealing with actions in the environment (Synchronous model for action, Action model with regional synchronization)
- Task-environments (Wooldridge, TAEMS)

3.4 Environments in agent-oriented methodologies (Gaia)

For each track we selected a number of relevant contributions from the research community, specified in brackets. It is not a primary goal of the survey to be complete, but rather to give an overview of the wide range of different conceptions associated with the environment for MASs and its various uses.

In Sect. 4, we extract, from the listed research tracks, a set of core concerns for environments in MASs. We have divided the concerns in two classes:

4.1 Concerns related to the structure of the environment (Structuring, Resources, Ontology)
4.2 Concerns related to the activity in the environment (Communication, Actions, Perception, Environmental processes)

Each concern represents a *logical functionality* for which the environment may have a *natural responsibility*. Our goal is to make the logical functionalities *explicit*, i.e. as concerns of environments as first-class entities. We want to underline that the proposed list of concerns is not intended to be complete. Our aim is to give an initial impetus to explore the rich potential of environments for MASs.

Next in Sect. 5 we outline a number of research challenges that, in our opinion, are important for the further development of environments for MASs. We have divided the list in three categories:

5.1 Definition and scope of environments
5.2 Agent-environment interrelationship
5.3 Engineering environments

Each category discusses a number of applicable research challenges. These challenges may serve as a source of inspiration for future exploration of environments for MASs.

Finally, in Sect. 6 we draw conclusions.

Conventions. In the remainder of the paper, we use the following style conventions:

- Quotations are put in "quotation marks."
- Specific terms used in literature are marked in `teletype`.
- Terms of concepts we want to emphasize are marked in *italic*.

3 Environments for MASs: A Survey of the State-of-the-Art

In this section we give an overview of a number of important research tracks that, in one way or another, include some notion of environment. We start with discussing a couple of general models for environments that have been proposed in literature. Then we zoom in on various concepts and functionalities related to inter-agent facilities in the environment and agent-environment interaction. We conclude the section by discussing the position of environments in agent-oriented software engineering. Each track is illustrated with a number of relevant contributions from the research community.

3.1 General Models for Environments

We start our study on environments for MASs with a number of representative models for environments that have been proposed in the research community.

Russell and Norvig. In chapter 2 of [73], S. Russell and P. Norvig discuss how an intelligent agent relates to its environment: "An agent is anything that can be viewed as perceiving its environment through sensors and acting upon the environment through effectors." This generally acknowledged relationship between an agent and its environment is schematically depicted in Fig. 1.

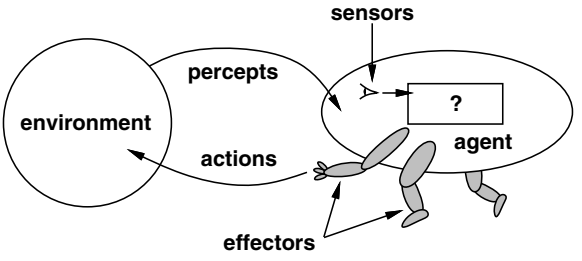

Fig. 1. Agent interaction with the environment [73]

Russell and Norvig discuss a number of key properties of environments that are now adopted by most researchers in the domain:

- Accessible versus inaccessible: indicates whether the agents have access to the complete state of the environment or not.
- Deterministic versus nondeterministic: indicates whether a state change of the environment is uniquely determined by its current state and the actions selected by the agents or not.
- Static versus dynamic: indicates whether the environment can change while an agent deliberates or not.
- Discrete versus continuous: indicates whether the number of percepts and actions are limited or not.

The most complex class of environments are those that are inaccessible, nondeterministic, dynamic and continuous. The first three properties of this list are properties typically occurring in MASs.

Russell and Norvig also define a "generic environment program", see Fig. 2. The program periodically gives the agents percepts and receives back their ac-

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

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

Fig. 2. A generic environment program [73]

tions. Next, the program updates the state of the environment based on the actions of the agents and of possibly other dynamic processes in the environment that are not considered as agents. This simple program for environments clearly illustrates the basic relationship between agents and their environment.

Ferber. In [28], J. Ferber discusses the modelling of environments for MAS at length. According to Ferber, an environment can either be represented as a single monolithic system, i.e. a centralized environment, or as a set of cells assembled in a network, i.e. a distributed environment. In a centralized environment, all agents have access to the same structure. In a distributed environment, each cell behaves like a centralized environment in miniature. However, a cell of a distributed environment differs in a number of ways from a centralized environment: (1) the state of a cell in a distributed environment depends on the surrounding cells, (2) the perception of agents in a distributed environment typically goes beyond one cell, (3) when agents move from cell to cell, the agent's link with the cells has to be managed and (4) the propagation of signals over the network of cells has to be managed. Orthogonal to the difference between a centralized or a distributed representation of environment, Ferber distinguishes between "generalized" and "specialized" models for environments. A generalized model is independent of the kind of actions that can be performed by agents. A specialized model is characterized by a well-defined set of actions. Ferber further distinguishes between purely communicative MASs (in which agents can only communicate by message transfer), purely situated MASs (in which agents can only act in the environment) and the combination of communicating and situated MASs.

Central to Ferber's model of an environment is the way actions are modelled. The action model of Ferber distinguishes between `influences` and `reactions` to influences. Influences come from inside the agents and are attempts to modify the course of events in the world. Reactions, which result in state changes, are produced by the environment by combining influences of all agents, given the local state of the environment and the laws of the world. This clear distinction between the products of the agents' behavior and the reaction of the environment provides a way to handle simultaneous activity in the MAS.

Ferber uses the BRIC formalism (Block-like Representation of Interactive Components) to model a MAS as a set of interconnected components that can exchange messages via the links. BRIC components encapsulate their own behavior and can be composed hierarchically. Fig. 3 depicts a model for a combined communicating and situated MAS in BRIC notation. In the BRIC model depicted in Fig. 3, the activity cycle of the MAS starts when the environment sends "perceptions" to the agents. As soon as the Synchronizer sends "synchronization of perceptions" signals to the agents, the agents are triggered to interpret the perceptions. Then, each agent produces an influence in the environment and possibly transmits a message to another agent. Next, the agent informs the Synchronizer it has finished its action by sending an "synchronization of actions" message. When all agents have sent their "synchronization of actions" messages, the Synchronizer sends a "synchronization of reactions" message to the Environ-

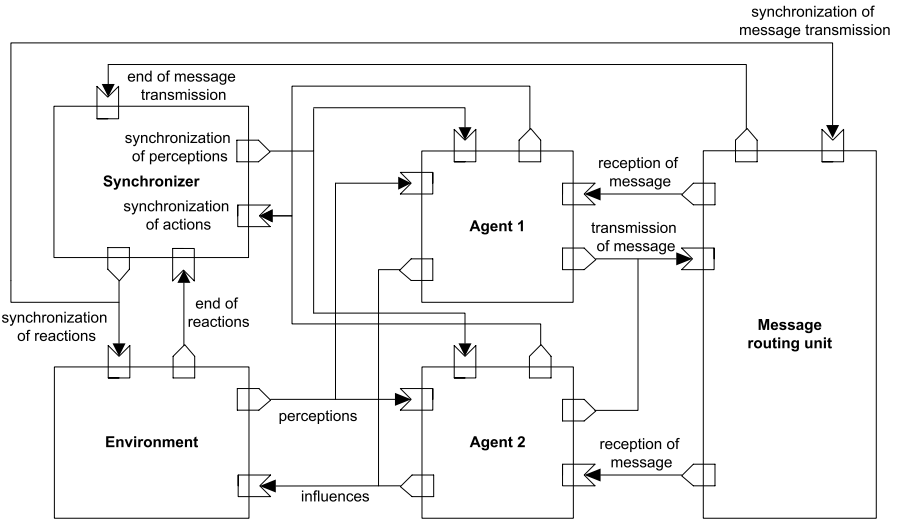

Fig. 3. BRIC model of communicating and situated MAS [28]

ment and simultaneously it sends a "synchronization of message transmission" to the Message routing unit. As a consequence, the Environment calculates the reactions to the collected influences, i.e. state changes of the Environment, and the Message routing unit delivers the messages. When the reactions are calculated, the Environment sends an "end of reactions" message to the Synchronizer. Analogously, the Message routing unit sends an "end of message transmission" when all messages are delivered. After that, the Environment sends the next perceptions to the agents and the whole cycle repeats. In the MAS model of Fig. 3, messages are synchronized with actions, i.e the messages are transmitted at the same time as the influences. A variant to this model is discussed in [87].

Odell et al. A classic paper on environment modelling for MAS is [61]. According to J. Odell and his colleagues, "an 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. An example of a law in the agent system is that two agents are not allowed to occupy the same place at the same time. In accordance with [68], an environment is defined as a tuple $< State, Process >$. *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, e.g. the aggregation, diffusion and evaporation of pheromones that ant-like agents

use to coordinate. Odell and his colleagues argue for a "common processing platform [...] that would provide a foundation upon which agent applications could build to leverage their own specific environmental requirements." However, they conclude, "In spite of the acronym, the FIPA (Agent Platform) architecture focusses almost entirely on the electronic environment, and does not address the physical environment. As such, it does not address the real potential of an active environment [...] to get more powerful interaction."

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. Basically, communication is the conveyance of information from one entity to another. A difference exists between transmission and communication. Communication requires that the information transmitted by one agent results in a state change of another, i.e. an act of sensing and deciding (although the latter may simply choose to do nothing). An interesting point of view related to this issue is discussed in [82]. L. Tummolini and his colleagues propose the notion of `Behavioral Implicit Communication` (BIC) as a parasitical form of communication that exploits both environmental properties and the agents' capacity to interpret each other's actions. To enable BIC, the environment needs to support the observability of the actions of the agents.

Odell and his colleagues define an agent's `social environment` as "a communication environment in which the agents interact in a coordinated manner". The social environment consists of (1) the social units (groups) in which the agent participates, (2) the roles that are employed for social interaction and (3) all the other members who play roles in these social groups. A group can be empty if no agents participate in the group; its collection can also contain a single participating agent or multiple agents. Groups have a unique identity in the overall system. As such, groups can become social actors, e.g. a business organization that interacts with sector groups in industry. The authors define a role as an abstract representation of an agent's function, service or identification within a group. Roles determine the patterns of dependencies and interactions among agents.

Environments for Mobile Agents. Since the mid nineties, mobile agents have been an active area for research and development communities. Mobile agents have the ability to migrate autonomously across a network, based on the principle of code mobility. A mobile agent is capable to suspend its execution at one node (at an arbitrary moment or at particular points in its life time), to move along with its code and its execution state to another node, and to resume its execution seamlessly. As such, a mobile agent is not bound to the network host where it begins execution. This permits a mobile agent to move to a destination node that contains the resources or services with which it wants to interact. As such mobile agents provide flexibility inside a distributed network to reduce network load and optimize service performance. Support for mobil-

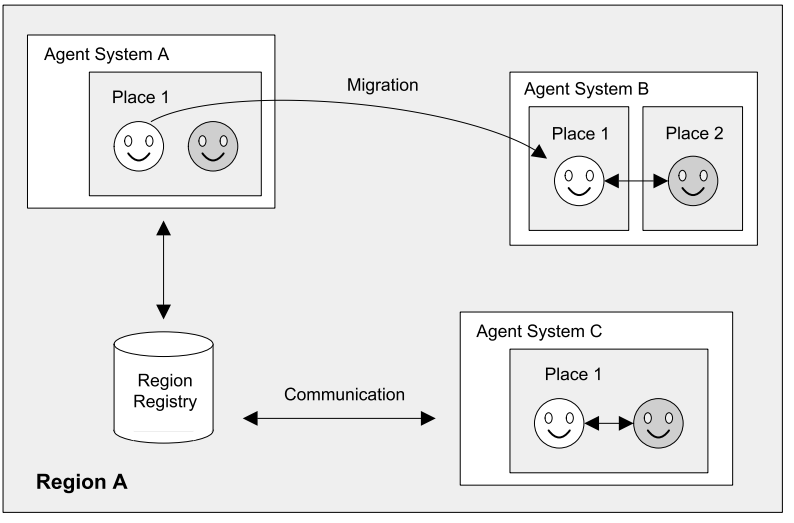

Fig. 4. Structure of a Distributed Agent Environment [65]

ity introduces additional requirements for the multiagent platform. During the last decade, many `platforms for mobile agents` have been developed. Some representative examples are Aglets from IBM [1], Voyager from Objectspace [83], Grasshopper from IKV++ [39], Ajanta from University of Minnesota [2] or SOMA developed at the University of Bologna [76].

Mobile agent platforms realize a distributed processing environment that is usually referred to as `Distributed Agent Environment` (DAE). DAEs typically support a hierarchy of locality abstractions to model physical network resources. Fig. 4 depicts an abstract overview of a DAE. The white agents symbolize mobile agents, the gray symbolize stationary agents.

On each host, at least one `agent system` has to run to support the execution of agents. Each agent system provides one or more `places`. A place is an executing context that offers specific services. An example is a trading place where agents can offer or buy information and service access. A `region` groups a number of agent systems (typically in a local area network). Each region has a `region registry` that maintains information about all registered agent systems, places and the hosted agents. The current location of mobile agents is updated in the corresponding region registry after each migration. The terminology used in Fig. 4 (region, place and agent system) is standardized by the OMG MASIF standard [63]. [65] enumerates a number of common capabilities for mobile agent platforms:

1. Agent execution: basic provisions to put incoming agents into execution, taking into account the binding to local resources.
2. Transport: mobility support to facilitate the network transport of agent code and execution state.

3. Unique identification: support for the generation of unique agent identifiers, even in the scope of the entire Internet.
4. Communication: support to enable agents to communicate with one another and with platform services.
5. Security: support for security issues such as authentication, access control of resources and integrity guarantees for code/state during the transfer over an untrusted network.
6. Management: enable system administrators to interact with the system, e.g. to monitor agents or to interrupt the execution of an agent task.

An important issue for mobile agent systems is interoperability. Interoperability permits the integration of heterogeneous agent systems and legacy systems. To obtain interoperability, most platforms for mobile agents therefore comply to one of the two main standards, the OMG MASIF standard [63] or the FIPA standard [34].

3.2 Inter-agent Facilities

In this section, we zoom in on various concepts and functionalities related to inter-agent facilities in the environment. We have organized the material in line with the taxonomy of agent interaction mechanisms proposed in [69]. We start with studying traditional infrastructure for direct message transfer between agents. The most commonly used form of direct message flow are peer-to-peer conversations, but also a distinguished agent that commands a subordinate is an example. Next, we discuss several models for indirect interaction, including blackboard systems, tuple-based interaction models and stigmergy. To conclude we look at models in which the environment serves as an organizational layer.

Communication Infrastructure. Communication is without any doubt a basic aspect of any MAS. In this section, we focus on communication infrastructure for message transfer between agents. We start with some general reflections on agent communication from Huhns and Stephens. Then we look at the FIPA agent platform for communication. In connection we discuss two concrete architectures for communication: the FIPA compliant middleware platform Jade, and the Retsina MAS infrastructure.

Huhns and Stephens. In the 2nd chapter of [45], M. Huhns and L. Stephens discuss characteristics and concerns of multiagent environments. The authors list the following characteristics:

1. Multiagent environments provide an infrastructure specifying communication and interaction protocols
2. Multiagent environments are typically open and have no single centralized designer
3. Multiagent environments contain agents that are autonomous and distributed and may be self-interested or cooperative

Next the authors list a brief summary of a number of concerns of multiagent "execution environments":

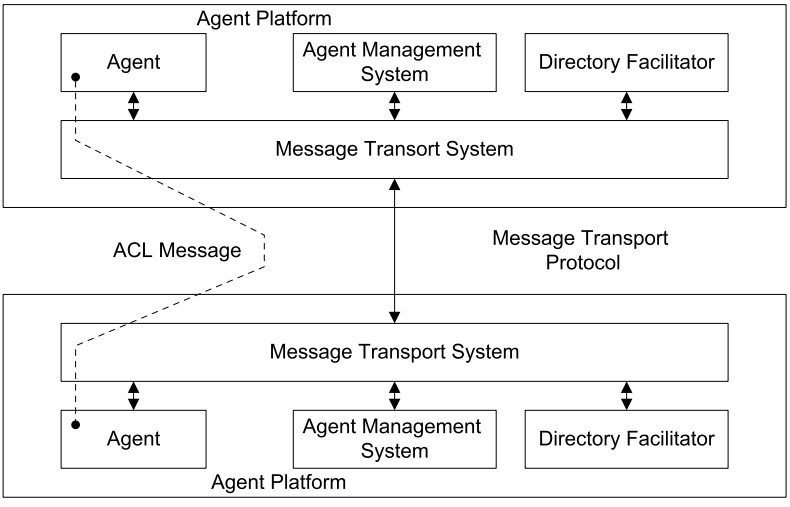

Fig. 5. FIPA agent platform reference model [34]

1. Design autonomy: relates to the platform, interaction protocols and agent architecture
2. Communication infrastructure: relates to type of communication medium and the type of connection
3. Directory service: white or yellow pages
4. Message protocol: refers to language (e.g. KQLM) and technology (e.g. CORBA)
5. Mediating and Security services: e.g., support needed for transactions or authentication
6. Operations support: refers e.g. to archiving

Hunhns and Stephens look at the environment as a *computational infrastructure* that enables agents to *communicate* with one another.

FIPA. The FIPA (Foundation for Intelligent Physical Agents) agent platform reference model [34] illustrates a typical communication infrastructure for direct message exchange, see Fig. 5.

The key building block of an environment in FIPA is the `agent platform`. An agent platform includes a "run-time environment" that defines the life cycle of the agent system, and executes e.g. on a Java virtual machine. The building blocks of the agent platform are: (1) a `directory facilitator` acting as a yellow pages service for the agents to advertise and discover services offerings, (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. The message transport system is specified in great detail. It specifies transport protocols (low level details for wired and

wireless transfer of messages between interfaces on different agent platforms) and message transport envelopes (encoding of metadata required for message forwarding over individual transport protocols). Lastly, the message transport system also includes specifications of several ACL message representations that define the syntax to be used when sending messages. Besides a standard for message transport, FIPA also provides standards for interagent communication, i.e. it defines the precise semantics of the exchanged bits. These specifications are divided in four sections: (1) the message structure specification that defines the structure of FIPA-ACL (FIPA Agent Communication Language) messages, (2) a library of performatives, defining the semantics of different communicative acts, (3) a number of protocols, i.e. message sequences applicable in agent systems and (4) a content language for FIPA messages, called FIPA-CL (FIPA Content Language). Note that FIPA does not define an ontology language to express domain knowledge. An increasing number of agent platforms comply with the FIPA standard, including Jack [44], Jade [9] and Zeus [58].

JADE. Fig. 6 depicts the Jade (Java Agent Development Environment) architecture [9]. Jade 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 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.

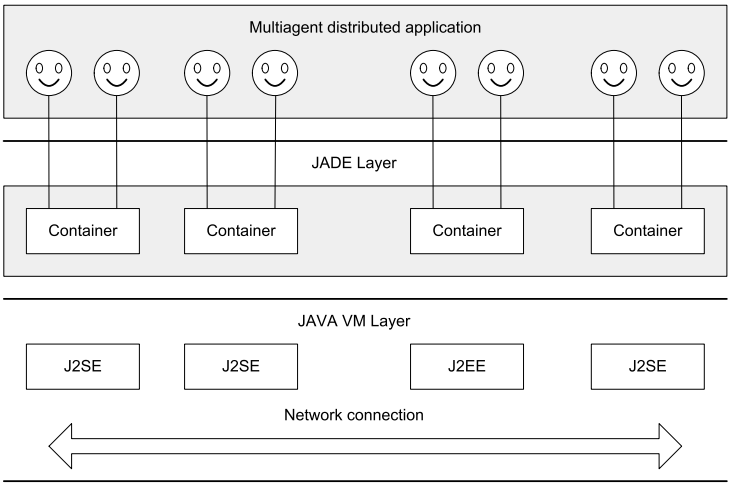

Fig. 6. The Jade architecture [9]

Agents can dynamically discover each other and communicate by exchanging asynchronous messages. The structure of the messages complies with the FIPA-ACL language definition. Jade provides a set of skeletons of typical interaction protocols. The platform also supports mobility of code and execution state (exclusive the data on the JVM -Java Virtual Machine- stack). This enables agents to stop running on a host, migrate to a different remote host and restart execution from the point they stopped. Jade is widely used in the academic community and several companies are using Jade for their internal projects, including Telecom Italy [81], Whitestein Technologies AG [90] and Rockwell Automation [72].

RETSINA. Retsina (Reusable Environment for Task-Structured Intelligent Network Agents) [79] is a well-known MAS infrastructure, see Fig. 7. Retsina is an open MAS infrastructure that supports communities of heterogeneous agents. The Retsina MAS infrastructure is build up in several layers. The operating environment 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 communication infrastructure provides two types of communication channels: one for message transfer between peers, the other for multicast that is

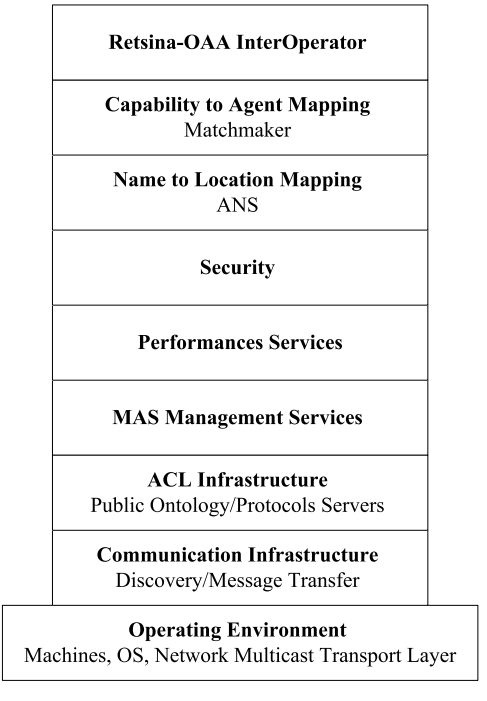

Fig. 7. The Retsina MAS infrastructure [79]

used for a discovery process to let the agents find infrastructural components. The ACL used in Retsina is KQML (Knowledge Query and Manipulation Language) [33]. Retsina provides an ontology derived from the Wordnet Ontology [27] and a protocol engine with a protocol language. The MAS management services offer tool support to monitor the activity of the agents and to debug and launch the applications. Retsina provides a service for performance monitoring in simulation. The security module supports agent authentication, secure communication and integrity of the Retsina infrastructure components. A first basic high-level infrastructural support is offered through ANSs (Agent Name Services). An ANS provides a means to abstract away from physical locations by mapping agent identifiers to network addresses. ANSs do not participate in the transactions between agents, they only provide the agents with addresses that they can cache, removing the need for unnecessary lookups. A second level of infrastructural support is offered by middle agents, i.e. matchmakers. Matchmakers provide 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. Advertisement and requests have to be formulated in a special language called LARKS (Language for Advertisement and Request for Knowledge Sharing) [78]. The Retsina-OAA InterOperator on top of the Retsina MAS architecture bridges the Retsina MAS infrastructure with the OAA platform (Open Agent Architecture) [18]. Due to fundamental differences in the architectures, not all inter-agent interactions can be translated.

Models for Indirect Interaction. In this section we discuss interaction models in which entities interact indirectly through some kind of communication abstraction. Indirect (or mediated) interaction is characterized by a number of fundamental properties, such as name uncoupling, space uncoupling and time uncoupling. In order to communicate, interacting entities do not have to know each other explicitly, nor do they have to be at the same place, they do not even have to co-exist at he same time. Especially in open, highly dynamic, distributed systems, these properties enable flexible and robust interaction among the cooperating entities. An interesting attempt to define a unified framework for indirect interaction is the work on `coordination artifacts` of A. Omicini, A. Ricci and M. Viroli [64].

Classical Blackboard Communication Infrastructure. Blackboard systems were the first type of mediated interaction models proposed by AI researchers [24][20]. A blackboard is an intermediary data repository that enables cooperating software modules to communicate indirectly and anonymously. A classic blackboard system consists of three main components [20], see Fig. 8:

1. The `knowledge sources` are independent computational modules that together contain the expertise to solve the problem.
2. The `blackboard` is a system-wide data repository containing the shared data; interaction between knowledge sources only happens via the blackboard.

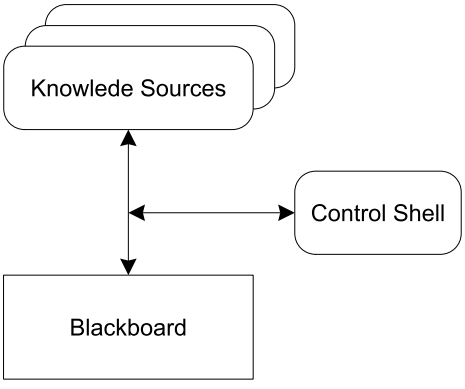

Fig. 8. Components of a classical blackboard system

3. A `control component` makes runtime decisions about the course of problem solving. When the currently executing knowledge source completes, the control component selects the most appropriate pending knowledge source for execution. To guide its selection, knowledge sources provide the control component with the necessary control knowledge.

Traditional MASs contrast with blackboard systems since they emphasize autonomy of agents, coordinated interaction between the agents, distribution (thus no central data repository) and organization as an emergent global phenomenon. As such, MASs and blackboard systems are two technologies with different application domains. Traditional blackboard systems are most appropriate for closely collaborating problem solving, while the focus of MASs is on solving large-scale distributed problems.

Tuple-based Interaction Models. 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 [16]. Linda is a coordination language, where coordination is defined in the spirit of separation of concerns: computation, i.e. the internal behavior of the active entities in the system, and coordination, i.e. the management of the interdependent active entities, especially their communication and synchronization, should be separated as much as possible. Linda attains this by providing a coordination language that enables communication between agents. Agents in Linda communicate by putting tuples in, and removing them from a shared space, i.e. the tuplespace. The Linda language is in essence composed out of three primitives: *in*, allows to take a tuple out of the tuplespace that matches with a given template; *out*, allows to put a tuple in the tuplespace; and *rd* that allows to non-destructively read a tuple based on a template. Throughout the years variants for distributed computing appeared, such as MARS [15], Sun's JavaSpaces [77] and LIME [57]. We take a closer look at the latter two.

JavaSpaces. [77] is a tuplespace model developed as part of (and as base of) Sun's Jini [35]. JavaSpaces is a fairly straightforward translation of the original Linda model to a distributed setting. JavaSpaces offers the possibility of several remotely accessible tuplespaces. Since it was developed in the context of the Java programming language, not tuples but objects are put in the tuplespace. JavaSpaces adds the possibility of distributed transactions on the tuplespace. The fact that this is a hard problem was raised by N. Busi [14]. Busi showed that the serializability of transactions is not always guaranteed by the JavaSpaces system. JavaSpaces remains important as it is supported by Sun and used as discovery mechanism for the Jini system.

LIME. (Linda In a Mobile Environment) [57] is a middleware system that allows communication between agents in a similar way as Linda does. However, it is built to operate in a mobile environment, as opposed to Linda which is conceived for parallel computing. Instead of communicating through one centralized tuplespace, in Lime each agent carries its own tuplespace. The traditional tuplespace operations are available, augmented with other operations such as the *non-blocking read* and *non-blocking in* operations. The originality of the approach is that, when agents reside on the same or a connected host, their tuplespaces are merged transparently, i.e. agents have the illusion of a locally shared tuplespace. The Lime middleware can be used for applications where both the agents are mobile (i.e. moving from host to host) and the hosts are mobile (i.e. physically moving). In order to make this possible a location parameter is added to the operations, so that agents can select the tuplespace they wish to interact with. Also, to cope with the dynamic environment, `reactions` can be defined, i.e. code that is executed by the tuplespace when specific tuples are inserted in the tuplespace.

In recent years, a number of tuple-based systems were proposed for ad hoc and mobile computing. ObjectPlaces [74], EgoSpaces [47] and TOTA [49] add mechanisms for sharing tuples across tuplespaces. ObjectPlaces maintains an agent defined view on a host's surroundings. A view is an up-to-date representation of the state of tuplespaces on neighboring nodes in the network, and this representation is maintained as the network and the contents of the tuplespaces change. This can be done efficiently since the interface to the tuplespaces in ObjectPlaces is asynchronous (i.e. operations do not block, but their result is returned when it is available), as opposed to the synchronous interface common in other tuplespace-like systems. In the EgoSpaces system, a view is similarly a description of neighboring hosts in the network, and the system allows agents to execute Linda-like operations on the tuplespaces gathered from the view specification. EgoSpaces is built upon the Lime system. TOTA takes a different approach. The TOTA middleware maintains distributed tuples: a distributed tuple can for example represent a gradient field that decays as it is propagated on the network. This tuple is thus spread out over different distributed tuplespaces, and the TOTA middleware maintains the tuple as the network topology changes.

Stigmergy. The term stigmergy is coined by Grassé [38] to explain nest construction in termite colonies. The concept indicates that individual entities interact indirectly through a shared environment: one individual modifies the environment and others respond to the modification, and modify it in turn. [68] discusses several uses of stigmergy for MAS.

A popular means for such indirect interaction is through pheromones. A pheromone is a chemical substance (or a software counterpart) deposited in the environment. A pheromone has three interesting properties: (1) it aggregates, i.e. newly dropped pheromone merges with/reinforces already existing pheromone, (2) it diffuses, meaning it propagates in its local environment, and (3) it evaporates, meaning it decays over time. A pheromone is thus a representation of shared agent knowledge: it spreads to other nearby agents, allowing a local information transfer; it can be reinforced by other agents, allowing the MAS to incrementally build a solution; and disappears over time, which is a natural way to cope with dynamism in the environment.

Some applications using stigmergy include solving constraint problems, used by Dorigo's Ant Colony Optimization [23]; routing calls through telecommunication networks [11]; manufacturing control [13] and peer to peer systems [56]. For more application examples and more in-depth technical discussion, we refer to [10]. Here we take a closer look at two representative uses of stigmergy. First we zoom in on synthetic ecosystems presented in [13], than we look at the telecommunication network routing infrastructure presented in [11].

Synthetic Ecosystem. In [13], S. Brueckner considers a synthetic ecosystem where on the one hand agents control physical entities in the real world, but on the other hand, agents act among each other in a software environment. To enable indirect coordination among software agents in the same way social ants coordinate, the software environment emulates the "services" provided by the real world of ants. The part of the software environment realizing the services is called the `pheromone infrastructure`.

The pheromone infrastructure models a discrete spatial dimension. It comprises a finite set of places and a topological structure linking the places. A link connecting two places has a downstream and an upstream direction. Thus, for each place there is a set of downstream and a set of upstream neighbor places that are directly linked to it. Each agent in a synthetic ecosystem is mapped to a place, i.e. the current location of the agent, which may change over time. The pheromone infrastructure models a finite set of pheromone types. A pheromone type is a specification of a software object comprising a strength-slot (real number) and other data-slots. For each pheromone type, a propagation direction (downstream or upstream) is specified.

The pheromone infrastructure handles a finite set of software pheromones for each pheromone type. Every data-slot, except the strength-slot, is assigned a value of a finite domain to form one pheromone (type, direction etc.) The strength value (i.e. the value in the strength-slot) is interpreted as a specific amount of the pheromone. Different pheromones of a synthetic ecosystem may be stored in each place.

An agent may perform the following activities at its current place in the pheromone infrastructure:

- Access the references to all agents located at a place.
- Perceive the neighbor places of a place.
- Sample the local strength values of a specified set of pheromones.
- Initiate a change in the local strength of a specified pheromone by a specified value.

The pheromone infrastructure manipulates the values in the strength-slot of the pheromones at each place in the following way:

1. External input (aggregation): based on a request by an agent, the strength of the specified pheromone is changed by the specified value.
2. Internal propagation (propagation/diffusion): Assume an external input of strength s into a pheromone g at a place p. The input event is immediately propagated to the neighbors of p in the propagation direction of g. There, the local strength of g is changed by an input weaker than s. An even weaker input propagates to the following neighbors. The stepwise weakening of the input is influenced by g's propagation parameter.
3. Without taking changes caused by external input or propagation into account, the strength of each pheromone is constantly reduced in its absolute value (evaporation). The reduction is influenced by the evaporation parameter of the pheromone.

There is a major difference between the algorithms realized in the pheromone infrastructure and those observed in nature. After an ant deposits pheromones on the ground, evaporation disperses it. Particle by particle the pheromone moves through the continuous space driven by Brownian motion. At the initial location the amount of pheromones is reduced, while it builds up somewhere else or vanishes completely. In the discrete space of the pheromone infrastructure, propagated pheromones have only specific locations on which to "settle down". Furthermore, the structure of the space is not homogeneous. At some places, pheromones may be propagated to many places, while at other places no further propagation is possible. As a consequence, the mechanisms of evaporation and propagation of pheromones are modelled separately. Instead of continuously exchanging particles among places, there is one "wave" of input events running along the links, which is triggered by the original input of the agent.

The pheromone infrastructure realizes an application-independent support for synthetic ecosystems designed according to a number of design principles, such as decentralization, locality, parallelism, indirect communication, information sharing, feedback, randomization and forgetting. In [13], the principles of synthetic ecosystems and the proposed pheromone infrastructure are applied to manufacturing control systems. V. Parunak and his colleagues have applied digital pheromones in several practical applications, for an overview we refer to [67].

Network Routing. In [11], E. Bonabeau and his colleagues present an ant-like mechanism for routing and load balancing in telecommunication networks

that builds upon work of R. Schoonderwoerd [75] and S. Guérin [40]. Routing allows calls to be transmitted from a source to a destination through a sequence of intermediate switching nodes. The pathway of a message must be as short as possible, taking into account fluctuations of user traffic and changes of the network structure (e.g. link or switch failures.) To provide fault tolerance and spreading the computational load, the routing functionality should be implemented in a decentralized way. Social insects exhibit flexibility and robustness, solving difficult problems in a highly distributed way. The authors exploit this knowledge to tackle the routing problem in telecommunication networks. In the original routing algorithm of Schoonderwoerd [75], a node N_i (of a network with n nodes), with $k(i)$ neighbors (links being bidirectional) is characterized by a routing table $R_i = [r^i_{l,m}]_{n-1,k(i)}$ that has $n-1$ rows and k columns: each row corresponds to a destination node and each column to the next node. $r^i_{l,m}$ gives the probability that a given message, the destination of which is node N_l, be routed from node N_i to node N_m.

Agents go from their source node to their destination node by moving from node to node. The next node an agent will move to is selected according to the routing table of its current node. Agents update routing tables of nodes viewing their node of origin as a destination node, i.e. agents use certain knowledge about the portion of the network they come from to modify routing tables. For its part, this modification will influence the routing of messages and agents that have this portion of the network as destination. This approach avoids requiring agents to go back all the way to their node of origin to update the intermediate routing tables.

More precisely, an agent modifies the row corresponding to its source node, which is viewed as its destination node. With N_s the source node of an agent, N_m the node it just came from, and N_i its current node at time t, the entry $r^i_{s,m}(t)$ is reinforced while other entries $r^i_{s,l}(t)$ in the same row decay. The modification is determined by a reinforcement parameter δr that depends on the agent's characteristics. The influence of δr of a given agent must depend on how well this agent is performing, e.g. aging can be used to modulate δr. If an agent has been waiting a long time along its route to its destination node, it means that the nodes it has visited and links it has used are congested, so that δr should decrease with the agent's age.

Based on an idea of Guérin [40], Bonabeau and his colleagues propose to update not only the row that corresponds to an agent's source node, but all rows corresponding to all the intermediate nodes visited by the agent. Thereby the reinforcement of an entry associated with a given name is discounted by a factor that depends on the agent's age relative to the time it visited that node. [11] shows that the extended approach yields significantly better performance results. The authors however, point to the simplifications of previously used models and state that realistic tests in complex network models are needed. Therefore a deeper understanding of the limits and constraints of communication networks is necessary.

Interaction Models Related to Space in MASs. The ancestors of agent models providing an explicit representation of the spatial structure of the environment are Cellular Automata (CA) [91][92]. The CA model provides a regular lattice of automata, characterized by a homogeneous state and transition rule. The related structure is naturally suited to represent an abstraction of a physical environment, and CA have been widely used to model problems in which spatial features can play an important role. Some approaches providing the integration of CA and agent systems have been proposed, see e.g. [22]. Several platforms for MAS-based simulation, developed in line with Swarm [55], implement a spatial structure of the environment in terms of regular grids.

The Multilayered Multi Agent Situated System (MMASS) [6] is a MAS model providing an explicit representation of the agents environment and an interaction model strongly related to the agents context. The environment is modelled as a multi-layered structure, where each layer is represented as a connected graph of sites. Layers may represent abstractions of a physical environment, but can also represent logical aspects, e.g. the organizational structure of a company. Between the layers specific connections (interfaces) can be defined that are used to specify that information generated in one of these layers, may propagate into a different one. In MMASS, agents can (1) interact through a reaction among adjacent entities, (2) emit fields that are diffused in the environment, and (3) can be perceived by other agents. After experiments for the simulation of complex systems, the MMASS model has been recently proposed for applications in the ubiquitous computing scenario [50]. This type of application requires software architectures and tools based on models comprising some notion of space. Among other approaches sharing this viewpoint, it is important to mention Co-Fields [48] (Computational Fields) of M. Mamei, L. Leonardo and F.Zambonelli. Co-Fields supports the coordination of agents in an environment by means of distributed data structures (i.e. the co-fields) that can be spread either by the agents themselves or by other elements of the environment. Agents can sense the intensity of co-fields and are constantly guided by them, e.g. by moving towards local minima.

Environment as an Organizational Layer. Recently a particular interest has been given to organizational concepts within MAS such as "organizations", "groups", "communities", "roles" etc. [21, 29, 46, 37, 95, 59]. From an organizational perspective, a MAS can naturally be considered and designed as a *computational organization* [94] that defines a framework for agent activities, i.e. the organization imposes a set of constrains for the behavior of agents, and offers a set of facilities and services that agents may use. In [30] J. Ferber and his colleagues make a distinction between `ACMAS` or agent-centered MAS and `OCMAS` or organizational-centered MAS. In OCMAS, the organization acts (1) as a "dynamic framework" where agents may enter and leave organizations at will, and (2) as an environment for resources, services, communications and tasks, through the concepts of both groups and roles.

Thinking in terms of organizational design differs from the agent-centered approach that has been dominant during many years. When building an OC-

MAS, the designer first concentrates on the organizational level by specifying the structures and pattern of activities among agents, based on abstractions such as groups, roles, interaction protocols, authority constraints between roles, etc. At this stage, no mental issues such as beliefs or goals are considered. It is only when the organization has been specified that the MAS developer focusses on the agent's internal architecture.

Several models of OCMAS have been proposed [4, 30, 60]. Here, we briefly examine the AGR model (previously called Aalaadin) [29, 30] which is a very simple organizational model.

AGR. The AGR model is based on three primitive concepts: *Agent*, *Group* and *Role*. In the AGR model, agents play roles within groups. An agent may play multiple roles at the same time and may be a member of several groups. A group, as a part of an organization, is used as a context for patterns of activity. Agents are only allowed to communicate with agents of the same group. Suppose that an agent a of group G wants to communicate with an agent b of group H, but a does not belong to H and b does not belong to G. Communication can only be established when agent a joins group H, or agent b joins group G, or an agent c exists that is member of both groups G and H, and that can act as a mediator for this communication. This restriction on the scope of communication supports the creation of well-defined organizational structures such as hierarchies.

Groups act as environments for agents. An agent may enter or leave a group as a human may enter or leave a house or a social structure such as a firm or a lab. Within a group, agents provide services and facilities that the other agents of the group may use. Partitioning a society of agents into several groups enables a designer to build secure systems where secured groups of agents protect themselves by requesting authorization to be joined.

AGR provides a set of diagrams to describe organizations [30]. In the "cheeseboard diagram", a group is represented as an oval that imitates a board. Agents are represented by skittles that are positioned on a board and cut across a board when they belong to several groups.

A role is represented as a hexagon and a line links the role to agents. Fig. 9 illustrates a concrete organization using the cheeseboard diagram. In this exam-

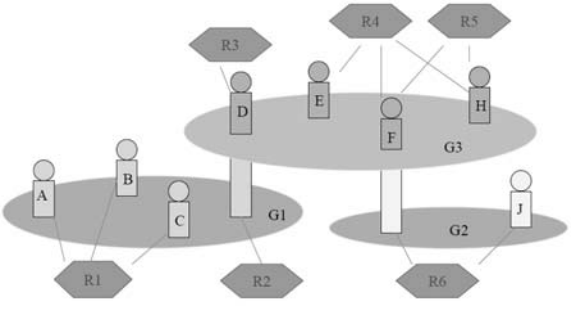

Fig. 9. The "cheeseboard" diagram in AGR for describing concrete organizations

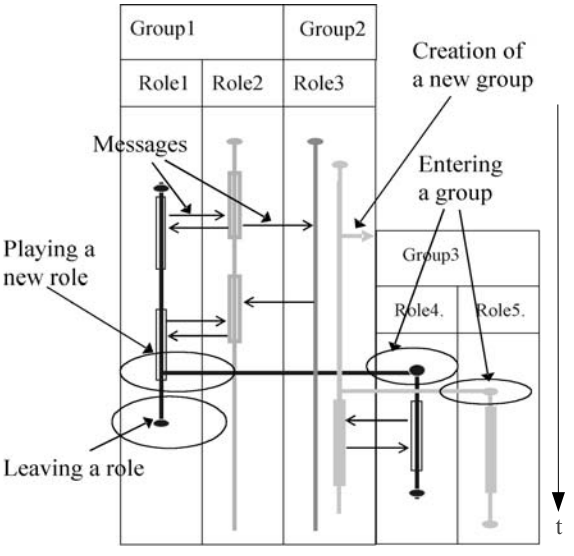

Fig. 10. The organizational sequence diagram in AGR

ple, the agent F is a member of both groups, $G2$ and $G3$, and the agent plays roles $R4$ and $R5$ in group $G2$, and role $R6$ in group $G3$.

The "organizational sequence diagram" describes the dynamics of organizations, i.e. the temporal relationships between organizational events, such as the creation of a group, an agent that enters a group or leaves it, or the acquisition of a role. The organizational sequence diagram can be seen as an extension of UML sequence diagram that incorporates the dynamics of roles and groups.

Contrary to an AUML sequence diagram where the life-line of an agent is represented by a single vertical line, in an organizational sequence diagram the life-line of an agent may consists of several (possibly parallel) segments. Each segment describes the *life* of an agent playing a specific role in a specific group. Parallel segments represent the fact that an agent plays several roles simultaneously. Fig. 10 depicts an example of a organizational sequence diagram.

MadKit [41, 52] is a multiagent platform, that has been designed according to the AGR model. In MadKit, groups and roles are used as core mechanisms for building, launching, deploying, simulating and observing multiagent programs. Several practical applications have proven the usefulness of MadKit and the underlying AGR model.

Extensions of AGR. In AGR, organizations do not encompass the notions of situatedness and action. To integrate the notion of situatedness in AGR, a spatial relationship could be added to a group. However, this extension would raise many difficult problems: e.g. what is the semantics of "distance" in relation to roles, is a role representing a "social location" as coordinates represent spatial locations? This approach has not been followed so far. To include the notion of action in

AGR, it is necessary to reify the concept of environment and to integrate it with the organizational concepts.

In [70], V. Parunak and J. Odell propose an extension of AGR by reifying the environment. In this model, an agent is both a member of (possibly several) groups, and an element of an environment. This work is interesting, but needs further exploration. In [32], J. Ferber and F. Michel propose another approach and 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.

In summary, the main idea of the research track on AGR is to offer an organizational-centered approach to build MASs. In AGR, the designer first considers the organization of the MAS as an accessible organizational structure in which agents have to behave, i.e. the designer builds the agent system according to the roles the agents play in the organization. Afterwards, the designer can focuss on the agent internal architectural details.

3.3 Agent-Environment Interactions

In this section, we discuss different models related to agent-environment interaction. First we look at agents' perception of the environment. Then we zoom in on a couple of models for actions. The section concludes with a brief discussion of the notion of task-environments.

Perception of the Environment. Perception is the ability of an agent to observe its neighborhood, resulting in a percept of the environment. Percepts describe the sensed environment in the form of expressions that can be understood by the agent. Agents use percepts to update their knowledge about the world or use it directly for decision making. In the case of an agent situated in the physical world, perception can be implemented in hardware: for example, it might be a video camera or a laser sensor on a mobile agent. For software agents situated in a virtual environment, perception must be implemented in software. Although perception is very common for any MAS, relatively little research work has been done in this area. Most of the research on perception can be found in robotics and cognitive science. For virtual environments, where all aspects of perception must be modelled explicitly, only a couple of theories and generic models for perception have been proposed. First, we illustrate perception in the RoboCup Soccer Server, then we discuss a domain independent model for active perception.

RoboCup Soccer Server. The RoboCup Soccer Server [71] supports three kinds of sensors in its sensor model: the aural sensor, the visual sensor and the body sensor. The aural sensor detects messages sent by the referee, the coaches and the other players. All messages are received immediately. The format of an aural sensor message is:

(*hear Time Sender Message*)

Time indicates the current time, *Sender* refers to the sender and *Message* to the content of the received message. Several server parameters affect the aural sensor. E.g., a player can only hear a message if the player's hear capacity is at least *hear_decay*, since the hear capacity of the player is decreased by that number when a message is heard. Every cycle, the hear capacity is increased with *hear_inc*, but is limited to *hear_max*. Players can receive more than one message at the same time. A message of a player is transmitted only to the players within *audio_cut_dist* meters from that player.

The visual sensor reports objects currently seen by the player. The information is automatically sent to the player every *sense_step*, a fixed period of time. Visual information arrives from the server in the following format:

(*see ObjName Distance Direction DistChng DirChng BodyDir HeadDir*)

ObjName refers to the name of the observed object, *Distance* and *Direction* are self-explaining. *DistChng* and *DirChng* refers to information about the relative velocity of the target object. *BodyDir* and *HeadDir* are only included if the observed object is another player and indicate the head and body direction of the other player relative to the observing player. The visible sector of a player is dependent on several parameters such as *sense_step*, which determines the basic time step between received visible information, *visible_angle*, i.e. the player's view cone, and *visible_distance* being the number of meters a player is able to see an object. If an object is within the distance but not in the view cone, then the player can only perceive the type of the object (ball, player, goal etc.) but not the exact name of the object. The player itself can influence the frequency and quality of the information by changing *ViewWidth* and *ViewQuality*.

Finally, the body sensor reports the current "physical" status of the player. This information is automatically sent to the player every *sense_body_step*. The transmitted information contains different kinds of player-specific information, such as: *AmountOfSpeed*, i.e. an approximation of the player's current speed, *HeadDirection*, i.e. the relative direction of the player's head, and *MoveCount*, i.e. a counter that indicates the number of move commands the player has executed so far.

The Robocup Soccer Server supports a rich and flexible model for perception, however the model is confined to the Robocup domain.

Model for Active Perception. In [89], D. Weyns, E. Steegmans and T. Holvoet propose a generic model for active perception that focusses on software agents situated in a virtual environment. Active perception enables an agent to direct its perception at the most relevant aspects of the environment according to its current task, facilitating better situation awareness and helping to keep processing of perceived data under control. The authors state that their model is generic in the sense that (1) it is independent of any domain or specific topology of the environment; (2) it offers reusable core abstractions for active perception in situated MASs, and (3) it offers support to model domain specific properties of perception. Fig. 11 gives a high level overview of the model. The model de-

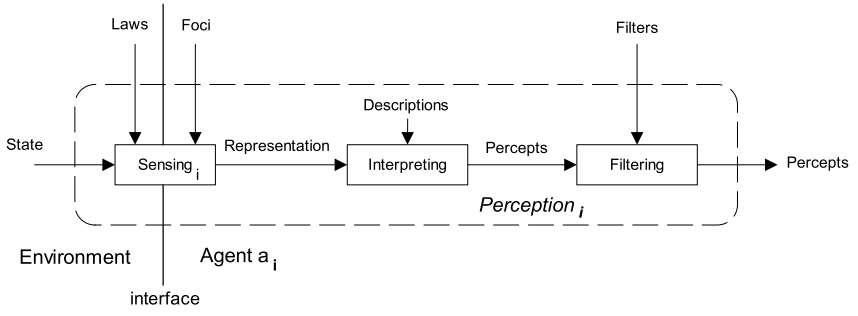

Fig. 11. Model for active perception

composes active perception in three functional modules: `sensing`, `interpreting` and `filtering`.

Sensing maps the state of the environment onto a `representation`. A representation is defined as a structured assembly of symbols that refers back to something in the environment, i.e. external to the agent. The mapping of state to representation depends on two factors. First the agent can select a set of `foci`. Each focus is characterized by its sensibility, but may have other properties too, such as an operating range, a resolution etc. Focus selection enables an agent to direct its perception, it allows it to sense the environment only for specific types of information. E.g., in an ant-like MAS, one agent may be interested in a "visible" perception of his environment, while another agent may be interested in "smelling" pheromones. To sense the desired type of information, both agents have to select a different appropriate focus. Second, the representation of the state is composed according to a set of `perceptual laws`. A perceptual law is an expression that constrains the composition of a representation according to the requirements of the modelled domain. As such, perceptual laws are an instrument for the designer to model domain specific constraints on perception. Contrary to physical sensing that incorporates such constraints naturally, such constraints have to be modelled explicitly in software MASs. Examples are a perceptual law that specifies how an area behind an obstacle is out of the scope of a perceiving agent or a law that under certain conditions adds some noise to perception. Besides the modelling of domain specific sensing, perceptual laws also permit the designer to introduce "synthetic" constraints on perception. E.g., for reasons of efficiency one could introduce default limits for perception in order to restrain the amount of information the agents have to process. It is important to notice that the model supports parallel sensing of the environment. Since agents can select different foci simultaneously, sensing typically results in a compound representation of the environment. This property is important to enable agents to sense their environment in a multi-mode manner.

The second functionality of active perception is interpretation. Interpretation maps a representation to a `percept`. To interpret a representation, agents use

`descriptions`. Descriptions are blueprints that enable agents to extract percepts from representations. Percepts describe the sensed environment in the form of expressions that can be understood by the internal machinery of the agent. E.g., consider a representation that contains a number of similar objects in a certain area. The agent that interprets this representation may use one description to interpret the distinguished objects and another description to interpret the group of objects as a cluster.

The third and final functionality of active perception is filtering. By selecting a set of `filters` an agent is able to select only those data items of a percept that match specific selection criteria. Each filter imposes conditions on the elements of a percept. These conditions determine whether the elements of a percept can pass the filter or not. E.g., an agent that has selected a focus to perceive its environment visually and is currently interested in only the agents within its perceptual range can select an appropriate filter that matches only agents for its percept.

An important remark concerns dynamism of perception. In the context of open systems, it is important that the components of the perception system can change (or be changed) dynamically, by the engineer or by the agents themselves. According to the authors, in the model for active perception, the agent can change the set of selected foci and filters dynamically according to its ongoing tasks. On the other hand, perceptual laws are pre-defined by the designer. As such, perceptual laws can not cope with unpredictable changes in the environment. To deal with run-time changes of domain specific constraints on perception, the model can be extended with infrastructure to adapt the set of perceptual laws (according to the changes in the system) or to replace laws dynamically.

Models for Actions. The classical approach to deal with actions is based on the (environmental) transformation of states, i.e. an action is defined as a transition of state, that is, as an operator whose execution produces a new state. From an observational point of view, the result of the behavior of an agent -its action- is directly modelled by modifying the environmental state variables. Whereas this approach suffices in classical AI where only one agent is acting, it fails for MASs were several agents are acting concurrently on a shared environment.

In [28], J. Ferber indicates a number of weaknesses of the classical approach to action. A first limitation is that only very elementary actions can be described. Complex composite actions can hardly be described. A harder problem is the static nature of actions as state transformations. Dynamic processes in the environment, such as the evaporation of pheromones, cannot be described, only the transformation obtained by the application of actions can. Another drawback is the lack of (flexible) support for simultaneous actions. E.g., to deal with a possible collision, explicit tests (as well as their possible consequences) must be added to the action code to verify whether two agents step to the same location or not. While the MAS community always assumes that the actions of different agents are carried out in parallel, the classical models for action do not offer

an adequate formal basis to represent collective actions. Finally, the approach does not distinguish between the actions themselves (what the agents do) and the consequence of the actions. Thus, traditional approaches to actions leave the description of how actions happen aside and only take into account the results of the actions.

Hereafter, we zoom in on two models for action that deal with these drawbacks. First, we zoom in on the action model for situated MASs of Ferber and Müller, described in [31]. Next, we look at the action model of Weyns and Holvoet, described in [86], that builds upon the Ferber-Müller model. From the scarce other work that is done on explicit models for action, we point to the work of F. Okuyama, R. Bordini and A. da Rocha Costa [62], which presents an XML based description language for actions and its effects, called ELMS (Environment Description Language for Multi-Agent Simulations). For more background information on action models we refer to [28].

Synchronous Model for Action. The action model of J. Ferber and J.P. Müller is based on three main principles. First, it distinguishes between influences and reactions to influences. Influences come from inside the agents and are attempts to modify the course of events in the world. Reactions, which result in state changes, are produced by the environment by combining influences of all agents, given the local state of the environment and the laws of the world. This clear distinction between the products of the agents' behavior and the reaction of the environment enables the handling of simultaneous actions. Second, the model decomposes the system dynamics in two parts, the dynamics of the environment and the dynamics of the agents situated in the environment. Third, the model describes the different dynamics of the MAS by means of abstract state machines.

Contrary to classical theories that only use the state of the world to describe evolution in a MAS, in Ferber and Müller's model evolution is described as the transformation of what they call `dynamical state`. Such a dynamical state is defined as a tuple consisting of the state of the environment and the set of influences simultaneously produced in the environment. The evolution of the MAS is defined as a function called *Cycle*, that in each step transfers the dynamical state to the next dynamical state. The *Cycle* function is further split in two sub-functions, *React* and *Exec*. A set of laws of the world describe how the next state of the world is computed given the previous state and the set of influences. In addition, a set of operators is defined for the agents that allow them to produce influences in the environment. The *React* function takes the influences and according to the current state of the world and its laws, produces the next state of the world. The *Exec* function produces the influences in the next dynamical state.

To describe the overall dynamics of the system, Ferber and Müller integrate the *React* and *Exec* functions in the *Cycle* function. The *Cycle* function then expresses the evolution of a MAS with n agents, i.e. in each cycle the function produces (1) a new state of the environment as reaction of the environment to the set of produced influences and (2) a new set of influences produced by the agents and the dynamics of the environment. A global synchronizer is responsible for the

synchronization of the cyclic evolution of the MAS. This synchronizer ensures that, "at a given moment, all the agents are treated as acting simultaneously, and that the environment reacts only subsequently, before handing over to the agents in an endless loop" [28].

The Ferber-Müller model deals with complex interactions in the environment and between agents, solving the fundamental problem of simultaneous actions in an elegant way. Besides, the model is applicable to purely reactive agents as well as to agents with memory. The model is basically restricted to synchronous MAS evolution, i.e. the MAS evolves at one global pace. Because the influences of *all* agents are treated as if they happened together, each influence can potentially interfere with any other influence.

Action Model with Regional Synchronization. As an alternative to the centralized synchronization model of Ferber-Müller, D. Weyns and T. Holvoet [86] propose an action model based on regional synchronization. With regional synchronization, there is no longer one global synchronizer, but instead each agent is equipped with his own local synchronizer. Each synchronizer is responsible to handle all synchronization issues for its associated agent. Before each action, the agent's synchronizer synchronizes with the other synchronizers in its neighborhood. The result of the synchronization algorithm is the formation of a group of agents, called a `region`. Agents of the same region act simultaneously, but independent of the other agents of the MAS. An algorithm for regional synchronization is discussed in detail in [85, 88].

The action model that integrates regional synchronization, describes the dynamics of a MAS composed of a set of agents that exist in an environment and act simultaneously based on their locality. Besides the actions invoked by the agents, other activities may be going on in the environment too. In [86], such activities are denoted as `ongoing activities`. Examples of ongoing activities are a moving object or, in the context of ant-like systems, an evaporating pheromone. Weyns and Holvoet use a different notion of dynamical state than the Ferber-Müller model. Dynamical state is defined as a tuple consisting of the state of the environment, and a set of `consumptions`. A consumption [1] is an effect from the environment reserved for a particular agent. Such consumption results from the reaction of the environment to the most recently produced influences for that agent. When an agent "consumes" a consumption, the consumption can be absorbed by the agent (e.g. food that is turned into energy), the agent may simply hold an element (e.g. an object he has picked up) or the consumption may affect the agent's state (e.g. the arm of a robot is wrenched through an external force).

The dynamics of the system is defined as the *Cycle* function that maps a dynamical state to the next dynamical state. To clarify the activities invoked by the agents A and the ongoing activities D in the environment on the one hand and the reaction of the environment upon both activities on the other hand, the *Cycle* function is split up in two parts. The first part is composed of two sub-functions: $Exec^A$ and $Apply^D$. The second part is a single function *React*

[1] The notion of consumption is introduced by Ferber in [28].

that represents the reaction of the environment to the simultaneously performed activity of agents and ongoing activities.

$Exec^A$ represents the activities invoked by the agents. The subset of simultaneous acting agents consume their consumptions and produce a set of influences through the application of operators. The effects of the ongoing activities in the environment are induced by the $Apply^D$ function. Depending on the state of the environment, the set of ongoing activities produce a set of influences in the environment through the application of a set of parallel composed operators that are associated with the ongoing activities.

Since the activities invoked by the agents and the ongoing activities in the environment happen simultaneously, the influences resulting from $Apply^D$ and $Exec^A$ have to be combined. The reaction of the environment to the simultaneously performed activity of the agents and the ongoing activities in the environment is described by the *React* function, i.e. in the state of the environment, for the united sets of influences, and according to the set of parallel composed laws, *React* determines the next state of the environment and produces a new set of consumptions.

The evolution of the dynamical system is then defined as a sequence of cycles. In each cycle the *Cycle* function transfers the dynamical state into the next dynamical state, i.e. it produces (1) a new state of the environment and (2) a new set of consumptions. This twofold transfer is the result of the *reaction* of the environment to the *execution* of a set of operators invoked by a subset of agents that exist in the system together with the application of a set of operators resulting from the ongoing activities in the environment, given the state of the environment, a set of consumptions, and a set of laws of the world.

Comparison. In [86], Weyns and Holvoet compare the two discussed models for action. Two obvious differences between the models are the definition of dynamical state and the granularity of the groups of synchronized agents. The models are compared from the perspective of the typical execution-reaction cycle for situated MASs, graphically depicted in Fig. 12.

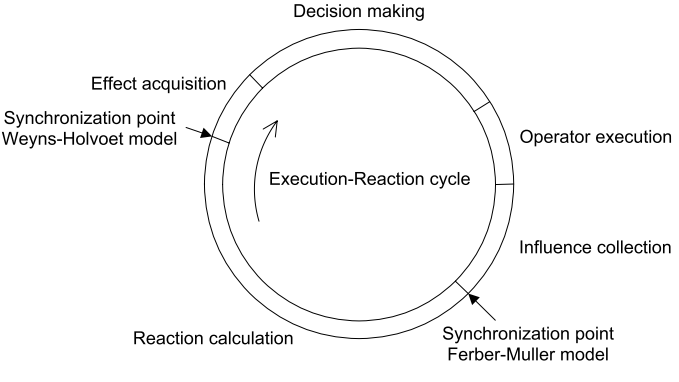

Fig. 12. Comparison of the two model for action based on the execution-reaction cycle

In the Ferber-Müller model, dynamical state is composed of state and *influences*. As such, the dynamics of the MAS can be expressed as the reaction of the environment to the set of influences and subsequently the production of a new set of influences, given the state of the environment and the laws of the world. So, the execution-reaction cycle runs from the point where the influences are collected to the next point where influences are collected, indicated by the "Synchronization point Ferber-Müller model" in Fig. 12. The start of the cycle is initiated by the environment and as such the model takes an *environment-centered* view on MAS evolution. The granularity of synchronously acting agents in the Ferber-Müller is the whole group of agents in the MAS. All agents act at one global pace, i.e. the influences of all agents in each cycle are considered as happening simultaneously. Thus, in the Ferber-Müller model, all agents synchronize in each passage of the execution-reaction cycle at the "Synchronization point Ferber-Müller model" and act simultaneously.

In the regional synchronized model for action, dynamical state is composed with state and *consumptions*. The dynamics of the MAS can be expressed as the consummation of a subset of consumptions and the production of a set of influences to which the environment subsequently reacts (according to the applicable laws) by updating its state and producing a new set of consumptions. So in the Weyns-Holvoet model the execution-reaction cycle runs from the point where the reactions are calculated to the next point where the reactions are calculated, indicated by the "Synchronization point Weyns-Holvoet model" in Fig. 12. Here the agents (on a per region basis) take the initiative to start their cycles, and as such the model takes an *agent-centered* view of MAS evolution. In this model agents of different regions can consume their consumptions independently and run asynchronously through the execution-reaction cycle. In the Weyns-Holvoet model, the granularity of synchronous acting agents are regions of synchronized agents. Influences of agents within a region are considered as happening simultaneously, however different regions can act asynchronously. Thus, in the Weyns-Holvoet model, in each passage of the execution-reaction cycle agents synchronize at the "Synchronization point Weyns-Holvoet model" and form regions that act simultaneously.

Task Environments. In [93], M. Wooldridge defines a `task environment` as a tuple $< Env, \Psi >$. An environment Env is a triple $Env = < E, e_0, \tau >$, where E is a set of environment states, $e_0 \in E$ is an initial state, and τ is a state transformer function. $\Psi : R \rightarrow \{0, 1\}$ is a predicate over runs. A run $r \in R$ of an agent in an environment is a sequence of interleaved environment states and actions, i.e.:

$$r : e_0 \overset{\alpha_0}{\rightarrow} e_1 \overset{\alpha_1}{\rightarrow} e_2 \overset{\alpha_2}{\rightarrow} \ldots \overset{\alpha_{u-1}}{\rightarrow} e_u$$

with $e_i \in E$ the set of environment states and $\alpha_j \in Ac$ the set of actions available to the agents. A task environment thus specifies:

- The properties of the system the agent will inhabit, i..e. the environment Env.
- The criteria by which an agent either failed or succeeded in its task, i.e. the specification Ψ.

According to Wooldridge, the most common types of tasks are `achievement tasks`, those 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. A well-known achievement task environment is the blocks world, see e.g. [73]. 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 which task it is to maintain the set of available services in a particular context. Complex tasks might 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 K. Decker and V. Lesser [43]. TAEMS can be used to specify, reason about, analyze, and simulate task environments. TAEMS is independent of the agent architecture and the inherent nature of the modelled domain. For details, we refer to [80].

3.4 Environments in Agent-Oriented Methodologies

Popular methodologies such as Message [25], Prometheus [66] or Tropos [12] offer support for some basic elements of the environment, however they do not consider the environment as a first-class entity. To our knowledge, the only methodology for analysis and design of MASs that considers the environment as a primary abstraction is the extended version of Gaia described in [94]. F. Zambonelli and his colleagues state that "identifying and modelling the environment involves determining all the entities and resources that the MAS can exploit, control or consume when it is working towards the achievement of the organizational goal." The authors distinguish between computational (or virtual) environments (e.g. a Web site) and physical environments (e.g. a manufacturing pipeline). A list of issues is put forward that has to be taken into consideration during the environmental modelling phase:

1. What are the environmental resources that agents can effectively sense and effect? The environment model should distinguish between the existence and the (possibly constrained) accessibility of a resource.
2. How should the agent perceive the environment? This question refers to the representation of the environment according to given circumstances.
3. What of the existing scenario should be characterized as part of the environment? The distinction between the agents and the environment is not always clear cut. For dynamic environmental resources, the designer has to decide whether they should be modelled as agents or as dynamic resources in the environment.

In Gaia, the identification of the environmental model is part of the analysis phase and is intended to yield an abstract, computational representation of the environment in which the MAS will be situated. During the subsequent architectural design phase, the output of the environmental model (together with a primary role model, a preliminary interactions model and a set of organizational rules) is integrated in the system's organizational structure that includes the real-world organization (if any) in which the MAS is situated. The organizational structure is then used to complete the preliminary role and interaction models. During the detailed (and final) design phase, the definition of the agent model and services model are derived from the completed role and interaction models. According to the authors, Gaia does not commit itself to specific techniques for modelling roles, environment and interactions etc. The outcome of the Gaia process is a technology-neutral specification that should be easily implemented using an appropriate agent-programming framework of a modern object or a component-based framework. With respect to the development of the environmental model, the authors state "it is difficult to provide general modelling abstractions and general techniques because the environments for different applications can be very different in nature and also because they are somehow related to the underlying technology." Therefore the authors propose a "reasonable general approach (without the ambition to be universal), and treat the environment in terms of *abstract computational resources*, such as variables or tuples, made available to the agents for sensing (e.g. reading their values), for affecting (e.g. changing their values) and for consuming (e.g. extracting them from the environment)." As such the environmental model is represented as a list of resources, each associated with a symbolic name, characterized by the type of actions that the agent can perform on it and possibly associated with additional textual comments and descriptions. A notation is used that is based on the Fusion [19] notation, e.g. :

reads	$Var1$	*//readable resource of the environment*
	$Var2$	*//another readable resource*
changes	$Var3$	*//a variable that can also be changed by the agent*

The authors indicate that "in realistic development scenarios, the analyst would choose to provide a more detailed and structured view of environmental resources." In particular, specific issues related to the modelling of environmental resources may be required to enrich/complement the basic notation. Some examples are:

- the representation of the logical/physical relationships between the resources in the environment. A graphical schema may be of help to model and to identify how and from where a resource can be accessed.
- the dynamics of the environment. The authors propose additional annotations to the basic notation to deal with this issue.
- dealing with a priori unknown availability of resources. The authors suggest an associative access model as the Linda tuple space to suit this purpose.

To deal with active components (services and computer-based systems) as part of the *operational environment*, the authors give some general guidelines.

When the role of the active components is simply that of a data provider (e.g. a Web server or a computer-based sensor), they should be modelled in terms of resources. However, if the environment contains components that are capable to perform complex operations (e.g. active databases or active control systems), the components should not be treated as part of the environment but, instead, they should be agentified.

4 Concerns for Environments

The survey described in the previous section shows us that the environment includes a broad diversity of functionalities. In this section we extract a number of core concerns for environments from the survey. We focus on concerns that represent *logical functionalities* of the environment.

As we already mentioned in the introduction section, researchers have highly different views on the concept of environment, causing a lot of confusing what the environment comprises. As a start to disentangle the confusion, we propose a 3-layer model for MASs[2] as depicted in Fig. 13.

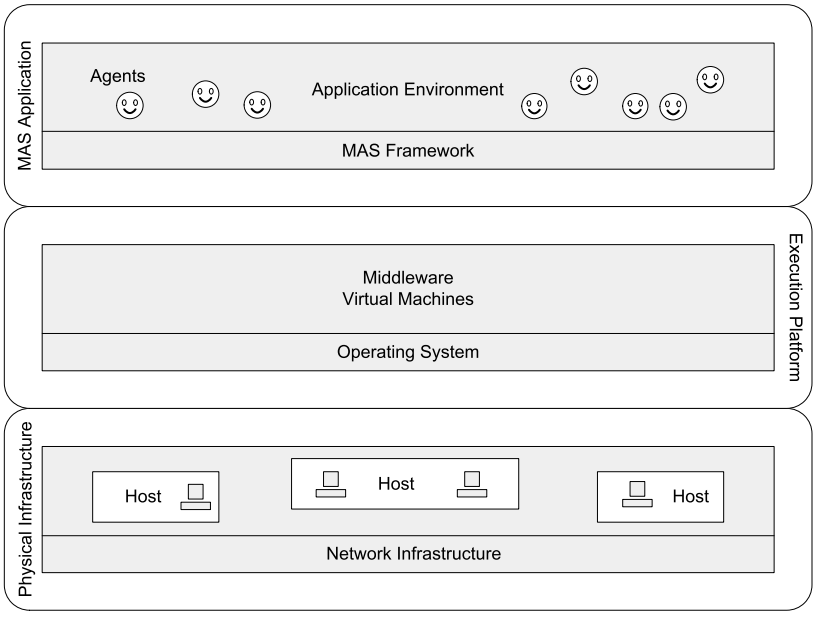

Fig. 13. 3-layer model for MASs

The MAS Application layer typically consists of two sub-layers: (1) the domain specific application logic containing the `Application Environment` with

[2] The focus of the proposed model is first of all on software agents.

the embedded Agents, and (2) a supporting MAS framework that offers high-level programming abstractions for the MAS developer such as support for communication or a pheromone infrastructure. The MAS Application runs on top of an "Execution Platform" that typically is composed of a generic (distributed) Middleware infrastructure and Virtual Machines that execute on top of an Operating System. The Execution Platform is mapped onto the "Physical Infrastructure" that comprises the hosts with processors and the connecting Network Infrastructure.

In [51], K. Mertens and his colleagues identify two levels of environments: the "application environment" and the "execution environment." According to the authors, the application environment provides the context for the agents to perform their actions, to communicate with one another and to acquire information about the problem they have to solve. The application environment is the translation of the problem situation, e.g. a grid world with tiles and holes for the Tileworld or a graph structure of locations with accessible files for a peer-to-peer file sharing system. The execution environment is the platform that is used to execute the MAS. The execution environment is mapped onto the physical layout of the hardware, e.g. a JVM that is mapped onto a single host or a number of JVMs mapped on different hosts. Whereas in Mertens' model of the environment, the agents are externally connected to the application environment, we have placed the agents *inside* the Application Environment emphasizing (1) that agents are inextricably bound up with their environment, and (2) that agents together with the Application Environment form an abstraction layer (and run) on top of an Execution *Platform* that maps onto a Physical Infrastructure.

The concerns of the environment we discuss in this section are located in the MAS Application layer, i.e. the top layer in Fig. 13. We distinguish between two classes of concerns: concerns related to the structure of the environment and concerns related to activity in the environment. Several concerns may seem to be quite natural functionalities for environments. We want to stress, however, that in practice the concerns we put forward are often dealt with in an implicit or ad hoc way. Our goal is to make the logical functionalities *explicit*, i.e. as concerns of environments as first-class entities. Not every concern we discuss is relevant for every possible MAS environment. The set of concerns should rather be viewed as a portfolio of logical functionalities for which the environment may have a "natural responsibility." In practice, it is up to the designer to decide which concerns should be integrated in the environment model for the domain at hand. As a final remark, we want to underline that the proposed list of concerns is not intended to be complete. Our goal is to give an initial impetus to explore the rich potential of environments for MASs. In the next section we discuss a number of research challenges that may serve as a source of inspiration for further exploration.

4.1 Concerns Related to the Structure of the Environment

We start with discussing a number of concerns related to structural features of the environment. Successively we look at structuring, resources and ontology.

Structuring. Agents and objects of a MAS share a common environment. The agents as well as the objects are dynamically interrelated to each other. It is the role of the environment to define the rules under which these relationships can exist and can evolve. As such the environment acts as a *structuring* entity for the MAS. This structuring can take different forms: it can be spatial, see e.g. [5, 13, 7, 54], but also organizational, e.g. [30, 94], or the environment can be structured as a mediating entity as e.g. in [16, 57, 74]. Structuring is a fundamental functionality of the environment. The structure of the environment is a design choice that depends upon the requirements of the domain at hand, and should be dealt with explicitly.

Resources. Besides the agents, an environment typically comprises different types of objects or (logical) resources. It is a responsibility of the environment to control the access to the resources. In general, resources can be read/perceived, writed/modified or consumed by agents. The extent to which agents are able to access a particular resource may depend on several factors such as the nature of the resource itself, the resource's current relationship to other resources or agents, the neighborhood of the agent to the resource, the capabilities of the agent etc. In general, the access to the resources can be described by a set of laws defined by the domain at hand, see e.g. [28, 86].

Ontology. In [17], P. Chang and his colleagues state: "agents must be able to understand their environment." Therefore, an environment must specify an ontology that provides a conceptual representation of the domain at hand. The ontology must cover the structure of the environment as well as the observable characteristics of objects, resources and agents, and their interrelationships. For symbolically-oriented agents, an explicit ontology should be available to the agents to enable them to interpret their environment and reason about it. For reactive/behavior-based/stigmergic agents, the designer/developer applies the ontology to encode the agents' internal structures. As such, these kinds of agents have an implicit ontology that enables them to make decisions.

4.2 Concerns Related to the Activity in the Environment

Next we discuss a number of concerns related to activity performed in the environment. First we look at concerns related to activity produced by the agents: communication, actions and perception. We conclude with the responsibilities of the environment related to activity produced by resources or objects.

Communication. As stated in the state-of-the-art overview, communication is inextricably bound up with MASs. Communication can take very different forms, ranging from direct message transfer over anonymous mediated communication via a shared space to indirect communication through stigmergy. Each of these types of communication has its own pros and cons. Designers should be aware of the potency as well as the impact of each type of communication for their solution. Selecting a particular type of communication should be an architectural choice, determined by the requirements of the problem domain at hand.

Actions. Dealing with actions in MASs in general is a very complex matter. If we allow multiple agents to act in the environment in parallel, we need explicit models to deal with actions that range far beyond the scope of state changes based on simple individual manipulation of objects. More than a decade ago, S. Hanks, M. Pollack and P. Cohen already raised in [42] the problem of "how the effects of simultaneous actions differ from the effects of those actions performed sequentially." In the state-of-the-art section we discussed a couple of models for action for MASs. Central to these models are (1) the distinction between the products of the agents' behavior on the one hand and the reaction of the environment on the other hand, and (2) a set of explicitly defined laws that govern the effects of the actions of the agents. These models resolve a number of fundamental issues with respect to actions in MASs, however, dealing with actions in MAS needs extensive further research to grow into full maturity.

Perception. Perception implies that the environment must be an *observable* entity. Agents must be able to inspect their neighborhood. In general, agents should be able to inspect the environment according to their current preferences. In the state-of-the-art overview, we discussed several examples of selective perception, such as "foci" proposed in [89] or "views" as proposed in [47] and [74]. Perception is constrained not only by agents' capabilities, but also by environmental properties (which in fact reflect properties of the problem domain). In [89] the environmental constraints are made explicit in the form of "perceptual laws". As for action models, there is still a wide (unexplored) field open to the concern of perception.

Environmental Processes. Besides the activity of the agents, resources or objects can produce activity in the environment too. A digital pheromone, for example, is a dynamic structure as it aggregates with additional pheromone that is dropped, it diffuses in space and it evaporates over time. Other examples are a rolling ball that moves on, or the local temperature that evolves over time. Maintaining such dynamics is an important functionality of the environment, see e.g. [13, 86].

5 Challenges

To conclude this paper, we list a number of research challenges that, in our opinion, are important for the further exploration of environments for MAS. We have divided the list in three categories: issues with respect to the definition and scope of environments, issues with respect to the interrelationship between agents and their environment, and finally issues concerning the engineering of environments for MASs.

5.1 Definition and Scope of Environments

In Sect. 2, we noted that the term "environment" is vague and ill-defined in relation to MASs. An ongoing research challenge will be developing a clearer

understanding of what we mean by an "environment." Defining anything requires relating it to other entities. In the case of environments, their definition requires relating them first to the agents that inhabit them, then to one another, and finally to different application domains.

Environments and Agents. What is the difference between the environment and the agents that inhabit it? A wide variety of distinctions have been proposed. Here are only a few examples:

- What the developer writes for a specific application are the agents. The software or hardware infrastructure on which the agents run is the environment.
- The environment provides the logical context for the agents to perform their actions, to communicate with one another and to acquire information about the problem they have to solve.
- Agents are autonomous, in that they proactively pursue objectives. The environment has no objectives.
- In a refinement of the previous suggestion, agents have achievement goals, while the environment can have only maintenance goals.
- The environment is extensive and unbounded, while the agents are bounded and localized in the environment.
- The environment embodies the given dynamics or "laws of physics" of a problem domain. The agents react to those laws in contingent ways.
- The environment is open to inspection. Individual agents are opaque. In other words, the environment implements what everyone is presumed to be able to see about the domain, while agents hide local decisions that should not be open to direct inspection or manipulation by others.

Each of these distinctions (and others that might be proposed) will yield different conclusions about the relative responsibilities of the agents and the environment, how they are mapped onto a given problem domain, and the life cycle of the design and implementation of a real system. These distinctions deserve formalization and analysis. Certainly, they are not orthogonal to one another. How many truly distinct views are there of the relation between agent and environment?

Taxonomy of Environments. With a formal understanding of the different ways that agents can be related to their environment, we can then classify environments with respect to one another. This level of understanding will enable researchers to be sure that they are talking about the same kind of environment in describing their systems and arguing for the relative merits of alternative approaches.

Environments and Domains. One reason that definitions of "environment" have proliferated is that MASs have been applied to a wide range of different applications domains, which impose differing constraints and afford different resources for interactions among agents. For example, it is natural for designers of a MAS intended to provide packet routing and quality of service management on a communications network to associate the environment with the existing

infrastructure of hardware and software that makes up the network and on which the agents will have to execute. In another domain, such as an agent-based simulation of an ecosystem, the environment as well as the agents will be custom-built for the application, and the distinction between agent and environment will be driven more by the differences between bounded vs. unbounded scope and given vs. contingent dynamics. In yet another domain, such as electronic commerce, the distinction between a transparent environment and opaque agents that can hide proprietary details of individual participants becomes paramount.

With a taxonomy of environments in hand, we can begin to develop systematic principles for relating a specific kind of environment to a particular domain. A number of research questions address the question of the relation between a taxonomy of environments and a taxonomy of domains. These include:

- Are there domains that do not need an explicit distinction between environment and agent?
- Are there domains that are particularly well suited to this distinction?
- What features of domains make them particularly amenable or hostile to this distinction?
- Are there particular functions of environments that are valuable regardless of the application domain?
- In general, how can a specification of a domain be mapped to a specification of a particular environment that will best support MASs serving that domain?

5.2 Agent-Environment Interrelationship

As our understanding of the space of possible environments becomes more refined, we need to explore in more detail the relation between agents and their environments. This relationship can be elaborated along at least three dimensions: architecture, protocol, and topology.

Architecture. In many applications, both agents and their environment will be software running on some physical computing system. What is the relationship between the agent software, the environment software, and the software and hardware that make up the computational substrate? In some cases, the agents may be completely dependent on the environment for their access to computational services, so that the environment (whatever other services it provides) becomes an "operating system" for the agents. In other cases, agents and the environment may have independent access to computational services (perhaps on different physical CPU's, as in robotic applications), and will interact with one another as computational peers. In this latter case, the question of how to distinguish the agents from the environment becomes particularly important, since at one level the environment is just another program executing architecturally at the same level as the agents.

These two cases are not mutually exclusive. One can imagine an architecture in which agents and environment execute on separate CPU's, but in which some services (such as inter-agent communications) are only available through the

shared environment. Still other configurations are possible, such as the case of a purely physical two-dimensional arena that provides the environment for soccer robots. The exploration of architectural alternatives for relating agents to environments offers ample scope for a new discipline within software engineering.

Protocol. By "protocol," we mean the set of conventions by which agents interact with the environment. The issue of protocol governs the degree to which an environment is open to heterogeneous agents, or to agents that are designed without advance knowledge of the environment. Protocols can vary along at least two dimensions: physical vs. digital and natural vs. arbitrary.

- In a physical protocol, agents must have physical sensors and effectors that can change and sense the state of the physical world. In a digital protocol, agents need only a way to read and write a register, such as a message mailbox or a communication channel.
- In a natural protocol, the structure of the interaction is constrained by the broader laws of physics, and any agent that complies with these laws can interact. In an arbitrary protocol, the signs exchanged by the agents are defined by some engineered language that each agent must understand in order to interact. The more natural the protocol, the more open the system will be to other agents that were constructed without detailed knowledge of the environment.

All four combinations of these dimensions are possible, illustrated in Tab. 1.

Table 1. Taxonomy of agent-environment protocols, with examples

	Physical	Digital
Natural	Agents try to push a ball toward their goal in a soccer arena.	An agent trying to communicate on a shared channel detects whether or not there is traffic on the channel (similar to Ethernet). It is unnecessary for the agent to understand the traffic that it detects.
Arbitrary	The goal is marked by a visual target of a designated color.	Agents exchange KQML messages through the environment.

This taxonomy of agent-environment protocols is very preliminary, and offers several directions for further research:

- What other dimensions distinguish the different ways in which agents interact with their environments?
- How do those dimensions impact the degree to which the system is open or closed?

- What responsibilities does the environment have, and what services can it provide, to increase its openness to heterogeneous agents?
- Alternatively, in applications that must be highly secure, how can environments ensure that only authorized agents have access to their services?

Topology. One characteristic of many environments is that they define a topology within which agents exist locally. A soccer agent is at only one location within the arena; a network agent lives on a specific router; an information retrieval agent visits only one database at a time. Such topological constraints can simplify agent reasoning (by restricting the range of information an agent must consider to that which is locally available) and system deployment (by restricting interaction to nearby entities and thus enabling the use of low-bandwidth communications), but may also pose challenges in achieving timely, long-range interactions. Research questions involving the relation between agents and the environment's topology include:

- What topologies does each kind of environment naturally induce? In robotics, environments naturally conform to low-dimensional manifolds, such as the surface of the earth or (for flying robots) the atmosphere. Computer networks cannot be mapped to such manifolds, but are typically power-law graphs [8] or more complex structures [3]. In some information retrieval applications, the topology may conform to the semantic structure of natural language [84].
- What does it mean for an agent to be "local" in an environment? In other words, how should the topology constrain agent actions? In some cases, agents may be able to access information only about their current location in the topology, but may be able to communicate with other agents that are remote from them. In other cases, direct inter-agent communications may be restricted to agents at the same location.
- What constraints do different topologies impose on agent interactions with the environment, and with one another through the environment? The existence of such phenomena as small-world shortcuts or highly varying node degrees can lead to interactions that vary widely from what our intuitions lead us to expect in low-dimensional manifolds or planar graphs.
- What is the environment's responsibility with regard to locality? Is it responsible, for instance, to enforce locality of agent interaction? Should all agents be equally localized, or can agents have different degrees of scope? For example, in a tree-structured environment, one can imagine that each agent can interact directly with all agents at its nodes and at lower-level nodes. How should agent scope be related to environment structure?

5.3 Engineering Environments

The previous two areas of research challenges provide ample scope for theoretical exploration, and will lead to many intriguing intellectual issues. The ultimate social benefit from these insights, though, requires their application to concrete problems, and the challenge of engineering environments for such problems will pose significant challenges in both the design and implementation of practical systems.

Design. Disciplined design practices for agents in general are in their infancy, and extending these techniques to environments greatly increases the scope of work to be done. A first step is gaining recognition for environments as first-class entities in MAS design. A number of the points discussed above directly impact design, such as how different sorts of domains map to different types of environments, and how agents are related to their environments. The results of studies in these areas need to be captured in tools such as description languages and other representational mechanisms that will enable engineers to communicate unambiguously about alternative designs. In many cases, the environment may incorporate physical as well as digital elements, and design tools need to support the integration of these domains.

Implementation. The growth of agent-oriented programming led to the proliferation of frameworks and development platforms for agents. Similarly, growing recognition of the importance of environments will stimulate extensions to these tools, or even the development of new tools that can support environments within which agents from different platforms can interact. Inevitably, embodying the services of an environment in a platform will collapse some of the dimensions we have explored in the previous paragraphs. For example, it is unlikely that a single platform will support all of the architectural options discussed above. The success of rival platforms in the market will itself be an important tool to assessing which of these dimensions are most important for practical use, and which can safely be removed from the developer's inventory. A critical question for implementation concerns the relationship between the logical and physical distribution of the environment. In some cases, it will make sense for an environment that represents a physically distributed topology to be distributed physically itself, while in other cases many of the benefits of the agent-environment distinction can be retained even if the environment is hosted on a single machine.

In sum, recognizing the distinction between agent and environment opens up new horizons for research and development comparable to those inaugurated by the development of the agent metaphor itself in the 1990's. These suggestions may help to inspire the next generation of researchers to explore directions that, at this point, seem the most promising. A measure of their success will be the degree to which the natural momentum of the research community overtakes them and leads to a self-sustaining body of science and technology that recognizes environments as first–class entities in the study of multi-agent systems.

6 Conclusions

Environments for MASs are too often assigned limited responsibilities, overlooking a rich potential for the paradigm of MASs. In this paper we have given a survey on the use of environments for MASs. From the study we learned that environments include a broad diversity of functionalities.

We used the insights from the survey to extract a first set of general concerns for environments, each concern representing a particular logical functionality for environments. A fundamental concern of the environment is that it *structures* the MAS. The environmental structuring can take different forms, such as spacial or organizational. Since agents must be able to understand their environment, an explicit ontology is required that covers the structure of the environment as well as the observable characteristics of objects, resources and agents, and their interrelationships. Besides structural aspects, we identified a set of concerns related to the *activity* in the environment. The most common activity of agents in the environment is communication. We discussed several types of communication for MASs. The designer should be aware of the potency as well as the impact of each form of communication and select the appropriate form according to the requirements of the problem domain. Next to communication, agents typically perform actions in the environment. It is the responsibility of the environment to *define the rules* for, and *enforce the effects* of, the agents' actions. Agents must also be able to perceive their environment. As such, the environment is an *observable* entity, contrary to the agents themselves. The environment should enable agents to observe their neighborhood according to their preferences, however, perception is constrained by environmental properties that reflect limitations in the modelled domain. Finally, we clarified that agents are not the only entities that can produce activity in the environment. Objects or resources too may be active in the environment (for example, a pheromone or a moving object). Maintaining such dynamics is an important responsibility of the environment.

The set of concerns we have proposed is not complete, but intended as a start to make the potential responsibilities for environments for MAS explicit. We listed a number of research challenges that may serve as a source of inspiration for further exploration.

We hope that this paper may contribute to extend the exploration of environments for MASs. Environments carry a rich potential for the paradigm of MASs. However, as long as researchers and software developers limit the functionality of environments, or deal with its responsibilities in an implicit or ad hoc manner, the full potential of environments will not be revealed. To discover and exploit the full potential of environments, we must treat environments as first-class entities. Recognizing environments as first-class entities opens up new horizons for research and development in MASs.

Acknowledgments

We would like to thank the attendees of the First International Workshop on Environments for Multiagent Systems [26] for the valuable discussions that have contribute to the work presented in this paper. A special word of appreciation also goes to Kurt Schelfthout, Alexander Helleboogh and Guiseppe Vizzari for their kind cooperation.

References

1. Aglets: http://www.trl.ibm.com/aglets/
2. Ajanta: http://www.cs.umn.edu/Ajanta/home.html
3. Alderson, D., Doyle, J., Govindan, R., Willinger, W.: Toward an Optimization-Driven Framework for Designing and Generating Realistic Internet Topologies. ACM SIGCOMM Computer Communications Review (2003)
4. Amiguet, M., Müller, J.P., Baez-Barranco, J.A., Nagy, A.: The MOCA Platform. Multi-Agent-Based Simulation II, Lecture Notes in Computer Science, Vol. 2581. Springer-Verlag, Berlin Heidelberg New York (2003)
5. Bandini, S., Manzoni S., Simone C.: Dealing with Space in Multi-Agent Systems: a Model for Situated MAS. Second International Joint Conference on Autonomous Agents and Multiagent Systems, ACM Press, Bologna, Italy (2002)
6. Bandini, S., Manzoni, S., Simone, C.: Heterogeneous agents situated in heterogeneous spaces. Applied Artificial Intelligence, Taylor & Francis 16(9-10) (2002)
7. Bandini, S., Manzoni, S., Vizzari, G.: A Spatially Dependant Communication Model for Ubiquitous Systems. First International Workshop on Environments for Multiagent Systems, New York (2004)
8. Barabasi, A., Albert, R.: Emergence of scaling in random networks. Science, 286(509) (1999)
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. Bonabeau, E., Dorigo, M., Theraulaz, G.: Swarm Intelligence: From Natural to Artificial Systems. SFI Studies in the Sciences of Complexity, Oxford University Press (1999)
11. Bonabeau, E., Henaux, F., Guérin, S., Snyers, D., Kuntz P., Theraulaz, G.: Routing in Telecommunications Networks with "Smart" Ant-Like Agents. Intelligent Agents for Telecommunications Applications (1998)
12. Bresciani, P., Giorgini, P., Giunchiglia, F., Mylopoulos, J., Perrini, A.: Tropos: an Agent-Oriented Software Development Methodology. Technical Report DIT-02-0015, University of Trento, Italy (2002)
13. Brueckner, S.: Return from the Ant. PhD Dissertation, Humboldt-Universität Berlin, Germany (2000)
14. Busi, N., Zavattaro, G.: On the Serializability of Transactions in JavaSpaces. Electronic Notes Theoretical Computer Science, Vol. 54 (2001)
15. Cabri, G., Leonardi L., Zambonelli, F.: MARS: a Programmable Coordination Architectue for Mobile Agents. IEEE Internet Computing (2000)
16. Gelernter, D., Carrierro, D.: Coordination Languages and their Significance. Communications of the ACM, 35(2) (1992)
17. Chang, P., Chen, K., Chien, Y., Kao, E., Soo, V.: From Reality to Mind: A Cognitive Middle Layer of Environment Concepts for Believable Agents. First International Workshop on Environments for Multiagent Systems, New York, 2004.
18. Cheyer, A., Martin, D.: The Open Agent Architecture. Journal of Autonomous Agents and Multi-Agent Systems, **4(1)** (2001)
19. Coleman, D., Arnold, P., Bodoff, S., Dollin, D., Hayes, H., Jeremas, P.: Object Oriented Development: the Fusion Method. Prentice-Hall International, Hemel Hampstead, UK (1994)
20. Corkill, D.: Collaborating Software. International Lisp Conference, New York (2003)

21. Demazeau, Y., Rocha Costa, A.C.: Populations and organizations in open multi-agent systems. 1st National Symposium on Parallel and Distributed AI (1996)
22. Dijkstra, J., Timmermans, H.J.P., Jessurun, A.J.: A Multi–Agent Cellular Automata System for Visualising Simulated Pedestrian Activity. 4th International Conference on Cellular Automata for Research and Industry (2001)
23. Dorigo, M., Maniezzo, V., Colorni, A.: The Ant System: Optimization by a Colony of Cooperating Agents. IEEE Transactions on Systems, Man, and Cybernetics-Part B, **26(1)** (1996)
24. Englemore, R.S., Morgan, A. (eds.): Blackboard Systems. Addison-Wesley (1988)
25. Evans, R., Kearney, P., Caire, G., Garijo, F., Gomez Sanz, J., Pavon, J., Leal, F., Chainho, P., Massonet, P.: MESSAGE: Methodology for Engineering Systems of Software Agents. EURESCOM, EDIN 0223-0907 (2001)
26. E4MAS: First International Workshop on Environments for Multiagent Systems. New York (2004) http://www.cs.kuleuven.ac.be/˜ distrinet/events/e4mas/
27. Fellbaum, C.: WordNet: An Electronic Lexical Database. MIT Press (1998)
28. Ferber, J.: Multi-Agent Systems, An Introduction to Distributed Artificial Intelligence. Addison-Wesley, ISBN 0-201-36048-9, Great Britain (1999)
29. Ferber, J., Gutknecht, O.: A Meta-Model for the Analysis and Design of Organizations in Multi-Agent Systems. 3rd International Conference on Multi Agent Systems, Paris, France (1998)
30. Ferber, J., Gutknecht, O., Michel, F.: From Agents to Organizations: an Organizational View of Multi-Agent Systems. Agent-Oriented Software Engineering IV. Springer-Verlag, Berlin Heidelberg New York (2003)
31. Ferber, J., Müller, J.P.: Influences and Reaction: a Model of Situated Multiagent Systems. 2th International Conference on Multi-agent Systems, Japan, AAAI Press (1996)
32. Ferber, J., Michel, F.: Integrating Environments with Organization-Centered Multiagent Systems, Environments for Multiagent Systems, Weyns, D., Parunak, H.V.D, Michel, F. (Eds.), Lecture Notes in Artificial Intelligence Vol. 3477, Berlin Heidelberg New York, Springer (2005)
33. Finin, T., Labrou, Y., Mayfield, J.: KQLM as an Agent Communication Language. Software Agents, MIT Press (1997)
34. FIPA: Foundation for Intelligent Physical Agents. http://www.fipa.org/
35. Freeman, E., Hupfer, S., Arnold, K.: JavaSpaces: Principles, Patterns an Practice. The Jini Technology Series, Addison-Wesley (1999)
36. Free On-Line Dictionary of Computing. http://foldoc.doc.ic.ac.uk/foldoc/index.html
37. Gasser, L.: Perspectives on Organizations in Multi-Agent Systems, Multi-Agent Systems and Applications: 9th ECCAI Advanced Course ACAI 2001 and Agent Link's 3rd European Agent Systems Summer School, EASSS. Luck, M., Mark, V., Stpnkov, O., Trappl, R. (Eds.), Lecture Notes in Artificial Intelligence, Vol. 2086. Berlin Heidelberg New York, Springer (2001)
38. Grassé, P.P.: 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, Vol. 6 (1959)
39. Grasshopper: http://www.grasshopper.de/
40. Guérin, S.: Optimisation multiagents en environment dynamic: application au routage dans les réseaux de télécommunications. Dissertation, University of Rennes I and Ecole Nationale Supérieure des Télécommunications de Bretange (1997)

41. Gutknecht, O., Ferber, J., Michel, F.: Integrating tools and infrastructures for generic multi-agent systems, 5th International Conference on Autonomous agents, Montreal, Quebec, Canada, ACM Press (2001)
42. Hanks, S., Pollack, M., Cohen, P.: Benchmarks, Testbeds, Controlled Experimentation, and the Design of Agent Architectures, AI Magazine 14(4) (1993)
43. Horling, B., Lesser, V., Vincent, R., Wagner, T., Raja, A., Zhang, S., Decker, K., Garvey, A.: The Taems White Paper, Multi-Agent Systems Lab University of Massachusetts.
44. Howden, W., Ronnquist, R., Hodgson, A., Lucas, A.: JACK Intelligent Agents, http://www.agent-software.com/shared/home/
45. Huhns, M.N., Stephens, L.M.: Multi-Agent Systems and Societies of Agents. G. Weiss (ed.), Multi-agent Systems, ISBN 0-262-23203-0, MIT press (1999)
46. Jennings, N.R.: On agent-based software engineering. Artificial Intelligence 117(2), Elsevier Science Publishers (2000)
47. Julien, C., Roman, G.C.: Egocentric Context-Aware Programming in Ad Hoc Mobile Environments. 10th International Symposium on the Foundations of Software Engineering, Charleston, USA (2002)
48. Mamei, M., Leonardi, L., Zambonelli, F.: Co-Fields: Towards a Unifying Approach to the Engineering of Swarm Intelligent Systems. Lecture Notes in Artificial Intelligence Vol. 2577. Springer-Verlag, Berling Heidelberg New York (2003)
49. Mamei, M., Zambonelli, F., Leonardi, L.: Tuples On The Air: A Middleware for Context-Aware Computing in Dynamic Networks. ICDCS Workshops (2003)
50. Manzoni, S., Nunnari, F., Vizzari, G.: Towards a Model for Ubiquitous and Mobile Computing. Theory And Practice of Open Computational Systems, TAPOCS. IEEE Computer Society (2004)
51. Mertens, K., Holvoet, T., Berbers, Y.: Adaptation in a Distributed Environment, First International Workshop on Environments for Multiagent Systems, New York (2004)
52. Michel, F., Ferber, J., Gutknecht, O.: Generic Simulation Tools Based on MAS Organization, 10th European Workshop on Modelling Autonomous Agents in a Multi Agent World MAMAAW'01, Annecy, France (2001)
53. Michel, F., Gouaich, A., Ferber, J.: Weak Interaction and Strong Interaction in Agent Based Simulations. 4th Workshop on Multi-Agent Based Simulation, MABS'03 at AAMAS 2003, Melbourne, Australia (2003)
54. Mili, R., Leask, G., Shakya, U., Steiner, R., Oladimeje, E.: Architectural Design of the DIVAS Environment. First International Workshop on Environments for Multiagent Systems, New York (2004)
55. Minar, N., Burkhart, R., Langton, C., Askenazi, M.: The swarm simulation system: A toolkit for building multi-agent simulations. Working Paper 96-06-042, Santa Fe Institute (1996)
56. Montresor, A.: Anthill: a Framework for the Design and Analysis of Peer-to-Peer Systems. 4th European Research Seminar on Advances in Distributed Systems, Bertinoro, Italy (2001)
57. Murphy, A., Picco, G.P., Roman, G.C.: LIME: a Middleware for Physical and Logical Mobility. 21th International Conference on Distributed Computing Systems (2001)
58. Nwana, S., Ndumu, D.T., Lee, L.C., Collis, J.C.: Zeus: A Toolkit for Building Distributed Multi-Agent Systems. 3th International Conference on Autonomous Agents, Seattle, WA, USA (1999)

59. Odell, J., Parunak, H.V.D., Breuckner, S., Fleischer, M.: Temporal Aspects of Dynamic Role Assignment. Agent-Oriented Software Engineering IV: 4th International Workshop, Melbourne, Australia. Springer-Verlag, Berlin Heidelberg New York (2003)
60. Odell, J., Parunak, H.V.D., Fleischer, M.: The Role of Roles in Designing Effective Agent Organizations. Software Engineering for Large-Scale Multi-Agent Systems, Lecture Notes in Computer Science Vol. 2603. Springer-Verlag, Berlin Heidelberg New York (2003)
61. Odell, J., Parunak, H.V.D., Fleischer, M., Breuckner, S.: Modeling Agents and their Environment. Agent-Oriented Software Engineering III, Giunchiglia, F., Odell, J., Weiss, G. (eds.) Lecture Notes in Computer Science, Vol. 2585. Springer-Verlag, Berlin Heidelberg New York (2002)
62. Okuyama, F., Bordini, R., da Rocha Costa, A.C.: ELMS: An Environment Description Language for Multiagent Simulation. First International Workshop on Environments for Multiagent Systems, New York (2004)
63. OMG MASIF: http://www.fokus.gmd.de/research/cc/ecco/masif/index.html
64. Omicini, A., Ricci, A., Viroli, R., Castelfranci, C., Tummolini, L.: Coordination Artifacts: Environment-based Coordination for Autonomous Agents, 3th Joint Conference on Autonomous Agents and Multi-agent Systems, ACM Press, New York (2004)
65. Omicini, A., Zambonelli, F., Klusch, M., Tolksdorf R., (eds.): Coordination of Internet Agents: Models, Technologies and Applications. Springer Verlag, Berlin Heidelberg New York (2001)
66. Padgham, L., Winikoff, M.: Prometheus: A methodology for Developing Intelligent Agents. 3th Agent-Oriented Software Engineering Workshop, Bologna, Italy (2002)
67. Parunak, H.V.D.: Altarum Institute, http://www.altarum.net/˜ vparunak/
68. Parunak, H.V.D.: Go to the Ant: Engineering Principles from Natural Agent Systems. Annals of Operations Research, Vol. 75 (1997)
69. Parunak, H.V.D., Brueckner, S., Fleischer, M., Odell, J.: A Design Taxonomy of Multi-Agent Interactions. Agent-Oriented Software Engineering IV, Melbourne. Springer-Verlag, Berlin Heidelberg New York (2003)
70. Parunak, H.V.D., Odell, J.: Representing social structures in UML, Agent-Oriented Software Engineering II, Wooldridge, M., Weiss, G., Ciancarini, P. (Eds.) Lectue Notes in Computer Science Vol. 2222, Berlin Hiedelberg New York, Springer (2002)
71. RoboCup: http://www.robocup.org/
72. Rockwell: http://www.rockwell.com/
73. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach, Prentice Hall (2003)
74. Schelfthout, K., Holvoet, T.: An Environment for Coordination of Situated Multi-Agent Systems. First International Workshop on Environments for Multiagent Systems, New York (2004)
75. Schoonderwoerd, R., Holland, O., Bruten, J., Rothkrantz, L.: Ant-based load balancing in telecommunication networks. Adaptive Behavior, Vol. 5 (1997)
76. SOMA: http://www-lia.deis.unibo.it/Research/SOMA/
77. Sun Microsystems, Inc.: The JavaSpaces v1.2.1 Specification (2002)
78. Sycara, K., Klusch, M., Widoff, S., Lu, J.: Dynamic Service Matchmaking Among agents in Open Environments. ACM SIGMOD Record 28(1) (1999)
79. Sycara, K., Paolucci, M., van Velsen, M., Giampapa, J.: The Retsina MAS Infrastructure, Kluwer Academic Publishers (2001)
80. TAEMS: http://dis.cs.umass.edu/research/taems/

81. Telecom Italia: http://www.telecomitalialab.com/
82. Tummolini, L., Castelfranchi, C., Omicini, A., Ricci, A., Viroli, M.: "Exhibitionists" and "Voyeurs" do it better: a Shared Environment for Flexible Coordination with Tacit Messages. First International Workshop on Environments for Multiagent Systems, New York (2004)
83. Voyager: http://www.recursionsw.com/voyager.htm
84. Weinstein, P., Parunak, H.V.D., Chiusano, P., Brueckner, S.: Agents Swarming in Semantic Spaces to Corroborate Hypotheses. Joint Conference on Autonomous Agents and Multiagent Systems, New York (2004) http://www.altarum.net/ vparunak/AAMAS04AntCAFE.pdf
85. Weyns, D., Holvoet, T.: Regional Synchronization for Situated Multi-agent Systems. 3rd International/Central and Eastern European Conference on Multi-Agent Systems, CEEMAS 2003, Prague, Czech Republic, Lecture Notes on Computer Science, Vol. 2691. Springer-Verlag, Berlin Heidelberg New York (2003)
86. Weyns, D., Holvoet, T.: A Formal Model for Situated Multi-agent Systems. Formal Approaches for Multi-Agent Systems, Special Issue of Fundamenta Informaticae, 63(2) (2004)
87. Weyns, D., Holvoet, T.: Look, Talk, Do: A Synchronization Scheme for Situated Multi-Agent Systems. UK Workshop on Multi-agent Systems, Liverpool (2002)
88. Weyns, D., Holvoet, Y.: A Colored Petri Net for Regional Synchronization in Situated Multiagent Systems. First International Workshop on Petri Nets and Coordination, PNC'04, Bologna, Italy (2004)
89. Weyns, D., Steegmans, E., Holvoet, T.: Towards Active Perception in Situated Multi-agent Systems. Journal on Applied Artificial Intelligence 18(9-10) (2004)
90. Whitestein: http://www.whitestein.com/pages/index.html
91. Wolfram, S.: Theory and Applications of Cellular Automata. World Press (1986)
92. Wolfram, S.: A New Kind of Science. Wolfram Media, ISBN 1-57955-008-8 (2002)
93. Wooldridge, M.: An Introduction to MultiAgent Systems. ISBN 0-471-49691-X. John Wiley and Sons, Ltd. England (2002)
94. Zambonelli, F., Jennings, N., Wooldridge, M.: Developing multiagent systems: The Gaia Methodology. Transactions on Software Engineering and Methodology, **3(12)**, ACM Press (2003)
95. Zambonelli, F., Parunak, H.V.D.: From design to intention: signs of a revolution. First International Joint Conference on Autonomous agents and Multiagent Systems, Bologna, Italy, ACM Press (2002)