Multiagent and Grid Systems – An International Journal 5 (2009) 93–108
DOI 10.3233/MGS-2009-0121
IOS Press

The agent environment in multi-agent systems: A middleware perspective

Danny Weyns[a,*], Alexander Helleboogh[a], Tom Holvoet[a] and Michael Schumacher[b]
[a]*DistriNet Labs, Katholieke Universiteit Leuven, Belgium*
[b]*Institute of Business Information Systems, University of Applied Sciences Western Switzerland, Sierre*

Abstract. Interaction is at the core of multi-agent systems. We use *agent environment* as a general term to denote the medium for agent interaction. Over the last years, the agent environment has been subject of active research. In this paper, we reflect on the role of the agent environment in multi-agent systems from a middleware perspective. Our study yields the following observations: (1) multi-agent system engineers consider distributed middleware (RMI, CORBA, etc.) as the basic platform for developing multi-agent systems, (2) common middleware services (security, persistency, etc.) are only minimally considered in multi-agent systems, (3) domain-specific middleware for multi-agent systems such as communication services and support for stigmergic coordination are typically developed as stand-alone services and as such difficult to compose with other services.

From these observations, we derive a number of challenges for research on environments in multi-agent systems: (1) to amplify reuse, application-specific services should be further consolidated into domain-specific services, (2) the problem of integration must be tackled, i.e. horizontal integration among domain-specific services for multi-agent systems, and vertical integration of domain-specific services upwards with the agents, and downwards with the common middleware services and the underlying distributed platform, (3) to support dynamic changing requirements of the system at hand, flexible composition and dynamic adaptation of services must be supported by the agent environment.

Keywords: Multi-agent systems, agent environment, middleware

1. Introduction

Over the last years, the role of the *agent environment* in multi-agent systems has been subject of active research [31,47–49]. In this paper, we take a step back and reflect on the notion of agent environment in multi-agent systems from a middleware perspective. We observe a strong connection between the environment in multi-agent systems and middleware in mainstream software engineering, both with respect to their role and evolution.

Middleware is the software layer that lies between the operating system and the applications on each node of the system [20]. Middleware shields software developers from low-level tedious and error-prone platform details by means of a consistent set of higher-level abstractions and services. Similarly, the agent environment in multi-agent systems is considered as a layer that mediates both the interaction among agents and the access to resources. The agent environment provides a set of services shielding agent developers from the low-level details of the underlying platform.

Whereas middleware was initially considered as a blackbox providing a set of higher-level programming abstractions and services, today middleware supports application-specific composition of components and services and dynamic adaptation according to the requirements of the system at hand [21,

*Corresponding author. E-mail: danny.weyns@cs.kuleuven.be.

ISSN 1574-1702/09/$17.00

38]. Similarly, the agent environment in multi-agent systems is traditionally considered as reusable communication or coordination infrastructure. Recent research puts forward the agent environment as a design abstraction that comprises an integrated set of application-specific components and services.

The observation of this connection has incited us to study the relationship between the notion of agent environment in multi-agent systems and state-of-the-art middleware.

Overview.

Environment is an overloaded term in software engineering. The term is used for development environment, execution environment, the part of the world extern to a system, etc. In multi-agent systems, the environment is often associated with the infrastructure in which agents are deployed. To clarify the meaning of environment used in this paper, we start in Section 2 by explaining the relation between a software system and its environment. Then, we zoom in on the system itself and elaborate on the role of middleware in system development. Section 3 zooms in multi-agent systems as a particular family of software systems and explains the notion of *agent environment* in multi-agent systems. In Section 4, we reflect on the connection between middleware and agent environment. From this reflection, we derive a number of challenges for future research on environments in multi-agent systems. Finally, we draw conclusions in Section 5.

2. Systems in mainstream software development

2.1. The system in its environment

Software systems are designed to satisfy particular functional and quality requirements. These requirements are issued by the group of stakeholders involved. For systems that interact with the external world via sensors and actuators, these requirements do not directly concern the software system. The requirements concern the environment in which the system will be installed [15]. The task of a system is to ensure that particular functionalities are achieved in the environment.

Jackson defines the environment as the part of the external world with which the system interacts, and in which the effects of the system will be observed and evaluated [19]. The distinction between the environment and the system is partly a distinction between what is given and what is to be constructed.

Figure 1 depicts the relation between a system and its environment. The system interacts with the environment by means of shared phenomena that are directly accessible via sensors and actuators. However, influencing phenomena that are private to the environment can only be done in an indirect manner: using sensors and actuators, the system tries to bring about causal chains to observe and affect private phenomena in the environment. Note that environment may refer to a given physical/hardware environment as well as to a given software environment, or a combination of both.

As an example, consider a car's cruise control system. The environment of the system consists of the car, its driver, the atmospheric conditions, the road the car drives on, etc. The cruise control system interacts with its environment through a sensor that can be used to observe the car's speed and an actuator that can be used to adjust the car's throttle. The requirements of the cruise control system are expressed in terms of phenomena in its environment. For example, the cruise control system should ensure that the car drives at a constant speed across the road. The cruise control system can only affect the car's speed indirectly, i.e. by relying on a causal chain between manipulations of the throttle actuator and alterations in the speed of the car.

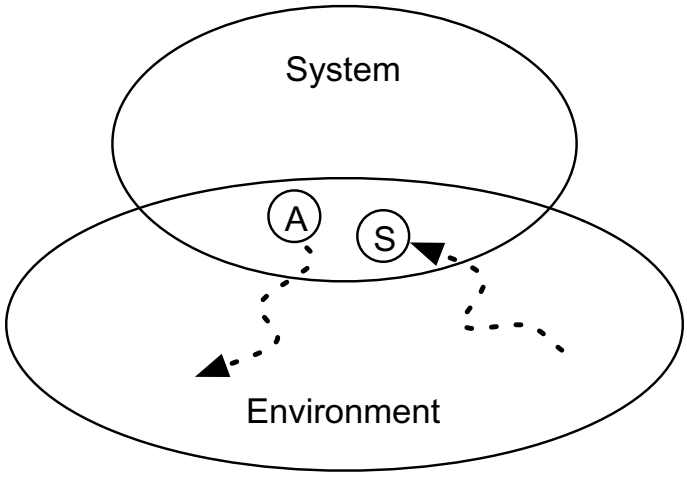


Fig. 1. The system and its environment [19].

Systems are increasingly expected to cope with dynamics in the environments in which they are deployed [18]. A dynamic environment is an environment that changes frequently. In a dynamic environment, the operating conditions of a system are continuously changing. For example, the environment of the cruise control system is dynamic: the road may go uphill or downhill, turbulence or wind may arise that cause additional or reduced drag. These phenomena affect the causal chains by means of which the cruise control system affects the environment. For example, in case the road goes uphill, changing the throttle will affect the car's speed in a different manner compared to the case that the road goes downhill.

In mainstream software engineering, it is generally considered good practice to capture properties and assumptions regarding the environment in an explicit model. Such a model of the environment includes assumptions about the frequency and nature of the changes in the environment, the accuracy and latency of sensors and actuators, the assumptions about the causal chain from activation of an actuator to the changes in the actual environment, etc. Such a model describes essential characteristics and assumptions that must be checked for proper deployment of the system [19].

2.2. The role of middleware in system development

We now focus on the structure of distributed software systems. Over the last decade, the development of software systems increasingly emphasizes the reuse of software components. There is an ongoing trend away from programming applications from scratch to integrating them by configuring and customizing reusable components and frameworks [38]. Requirements for greater reuse in developing distributed software systems motivate the use of middleware-based architectures. Middleware is software that resides between the application and the underlying operating systems, network and hardware. Middleware shields software developers from low-level tedious and error-prone platform details. It provides software developers with a consistent set of higher-level abstractions and services closer to the application requirements. Figure 2 shows the multiple layers of middleware in distributed software systems [38].

`Host Infrastructure Middleware` encapsulates communication with the operating system. Widely used examples are the Java Virual Machine and the .NET platform. `Distributed Middleware` defines higher-level distributed programming models with reusable APIs and components that help programming distributed applications. Examples are Java Remote Method Invocation,

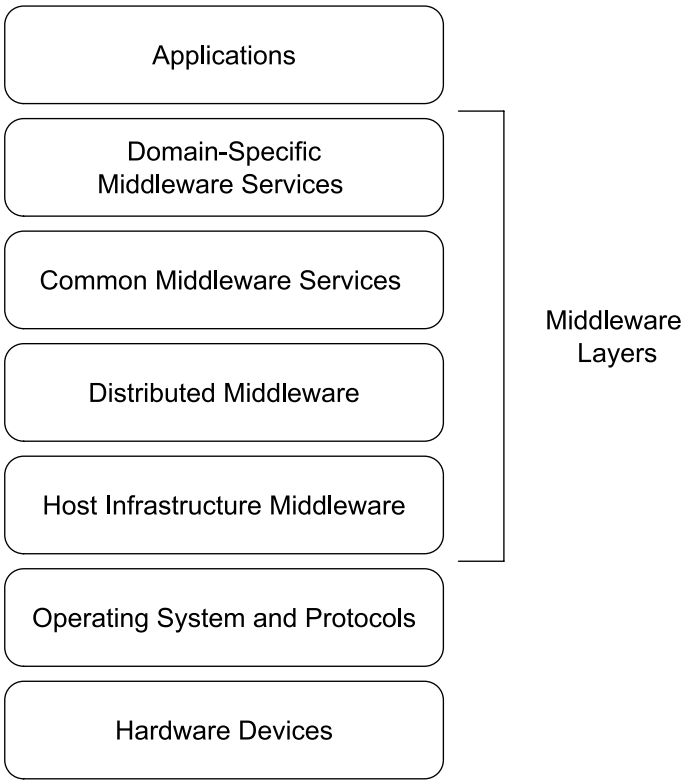


Fig. 2. Middleware layers with surrounding context [38].

Common Object Request Broker Architecture (CORBA), and SOAP that provides a simple XML-based protocol allowing applications to exchange structured information on the Web. `Common Middleware Services` define higher-level domain-independent services that support programming of application logic such as transactional behavior, security, and database access. An example technology is Enterprise Java Beans that enable software developers to link pre-defined services ("beans") without having to write much code from scratch. Finally, `Domain-Specific Middleware Services` are tailored to the requirements of particular domains. Examples are dedicated middleware services for electronic commerce and medical applications. Today, domain-specific middleware services tend to be less mature partly due to the lack of common middleware standards which are needed to provide a stable basis to create domain-specific services [38].

From programming towards flexible composition

The environments in which distributed software applications are deployed are very dynamic and heterogeneous. As a result, software must be dynamically composed and even be adapted at runtime. A major trend in middleware is to combine domain-specific middleware functionality with specific component frameworks (e.g. J2EE, .NET etc.). This approach enables the construction of applications from independently developed third party components and integrate built-in services covering non-functional requirements of a distributed application such as persistency and security. A typical example are service-oriented architectures [1] where the major part of application development boils down to assembling domain-specific services that comply to a set of declaratively specified policies. The

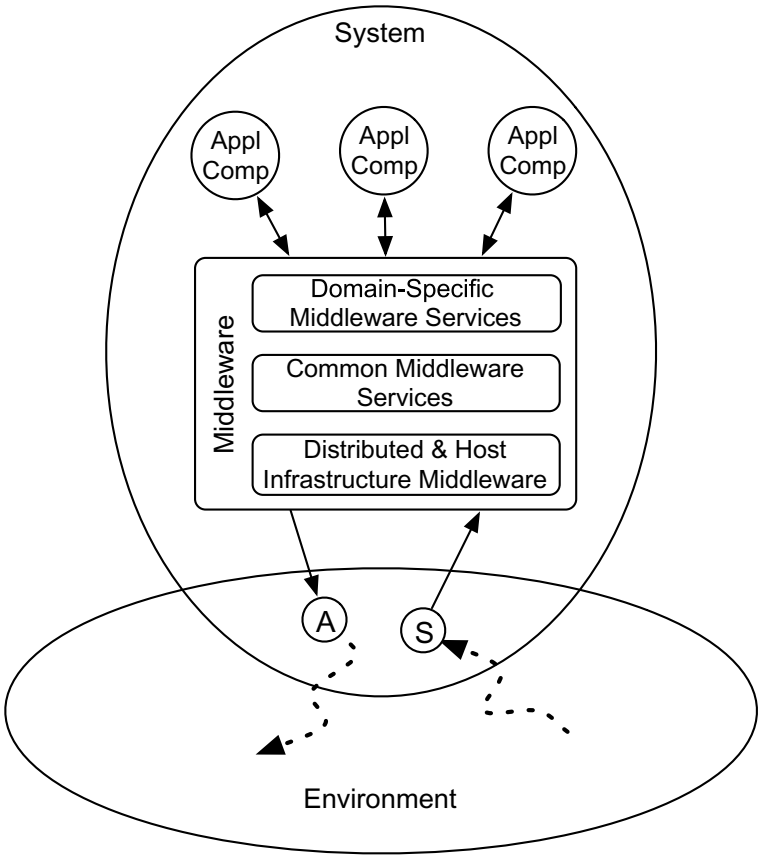


Fig. 3. A middleware-based system in its environment.

complexity of flexible composition and runtime adaptation of services in the face of the crosscutting nature of functionality is subject of active research [4,21].

2.3. Summary

Figure 3 shows a system in its environment. This figure refines the system depicted in Fig. 1 with the layered middleware architecture of Fig. 2. In analogy with Fig. 2, the system comprises application components supported by a middleware. Only the top-level middleware layers are shown in Fig. 3, i.e. the `Common Middleware Services` layer and the `Domain-Specific Middleware Services` layer. The lower-level middleware layers of Fig. 2 are depicted as the `Distributed & Host Infrastructure Middleware` layer in Fig. 3.

3. Multi-agent systems

Multi-agent systems (MAS) are a particular family of systems. In a multi-agent system, control is decentralized, i.e. none of the system components has global control over the system or global knowledge about the system [22,54]. The application components of a multi-agent system are agents. Agents are reflective software components that are capable of performing autonomous actions to meet their design

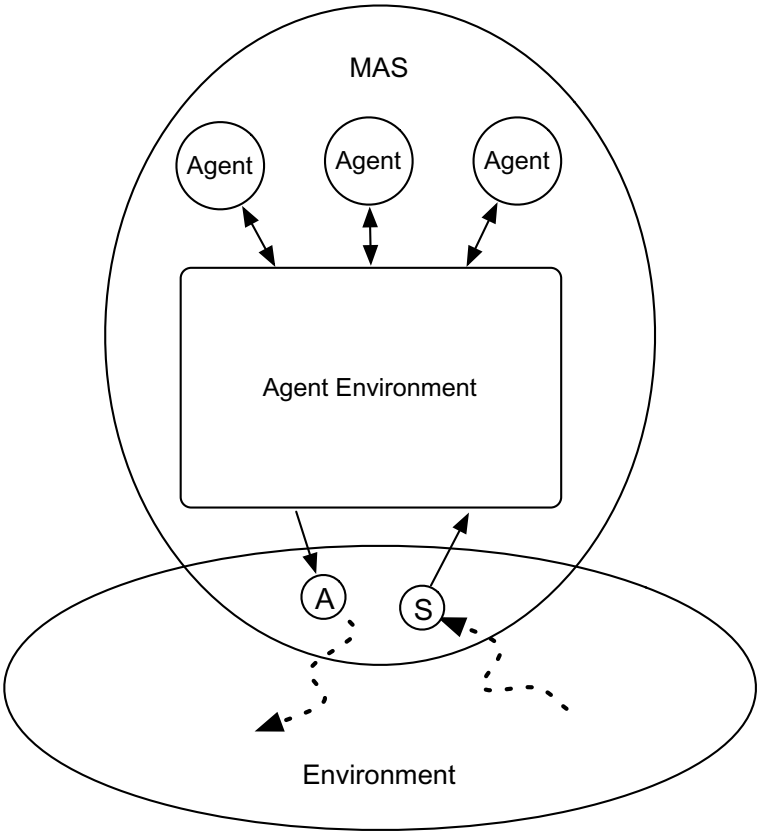


Fig. 4. A multi-agent system situated in its environment.

objectives. Decentralization of control implies that agents have to interact and coordinate their behavior in order to provide the overall system functionality. We use *agent environment* as a general term to denote the medium for agent interaction. As such, a multi-agent system consists of a set of agents and an agent environment that enables the agents to act in the environment and interact with each other. Figure 4 gives an overview of a multi-agent system situated in its environment.

We now zoom in on the role of the agent environment in multi-agent systems. We start with traditional approaches of agent environment. Then we elaborate on the agent environment as first-class design abstraction.

3.1. Traditional approaches for agent environment

Traditional approaches consider agent communication as the prior means for coordination [17]. Typically, dedicated communication infrastructure supports agent interaction with a directory facilitator acting as a yellow pages service for the agents to advertise and discover services, a management system that enables agents to register and locate one another (i.e. a white pages service), and a message transport system, i.e. a communication service for local and inter-node message exchange. A popular FIPA [10] compliant platform for agent communication is Jade [3].

Coordination infrastructures are a specific type of agent environment that provide an alternative for direct message exchange. Classical blackboard systems were the first type of coordination infrastructures

proposed by AI researchers [9]. A blackboard is an intermediary data repository that enables cooperating software modules to communicate indirectly and anonymously. In contrast to blackboard systems, tuple-based technologies use associative access to a shared dataspace for communication and synchronization purposes [13,25,28]. Coordination artifacts [27] generalize over different coordination models and languages. Research on computational institutions such as electronic institutions [26], logic-based institutions [42], and normative multi-agent systems [7] has developed a specific line of regulating infrastructures.

In stigmergic agent systems, agents coordinate their behavior through the manipulation of marks in the agent environment [14,29]. A classic example of stigmergic coordination are digital pheromones [5, 6] which software agents use to coordinate their behavior in a similar way as social ants coordinate. A digital pheromone is a dynamic structure in the agent environment that aggregates with additional pheromone that is dropped, diffuses in space and evaporates over time. Agents can use pheromones to dynamically form paths to locations of interest. Another well-established approach of stigmergic coordination are computational fields [8,23]. In this approach, the movements of agents are driven by abstract force fields that are spread in the agent environment (by agents or the agent environment itself). Agents coordinate their behavior by following the shape of the fields. Dynamics in the external world and movements of the agents induce changes in the surface of the fields, realizing a feedback cycle that influences agents' behavior. This feedback cycle enables the system to self-organize.

3.2. The agent environment as a first-class design abstraction

In the approaches discussed in the previous section, engineers consider the agent environment essentially as domain-specific infrastructure for agents [12]. Such infrastructures provide reusable solutions that can be applied over many applications. Yet, this perspective has a number of limitations. An infrastructure typically accounts for a predefined set of responsibilities. For a particular application, all responsibilities that are not managed by the infrastructure remain to be addressed either by the agents, or by additional system components. Assigning these responsibilities to agents often leads to heavy-weight agents. Integrating the infrastructure with additional system components raises the problem of composing functionality, which is often cumbersome since domain-specific infrastructures for agents have little flexibility and are not developed with such a composition in mind.

One of the main results of recent research on environments in multi-agent systems is the recognition of the agent environment as a first-class design abstraction in multi-agent systems [31,43,50]. Rather than a specific communication or coordination infrastructure, the agent environment provides a design space for multi-agent system engineers. On the one hand, engineers can use agents as well as the agent environment to make a well-considered assignment of responsibilities according to the system requirements at hand. On the other hand, considering the agent environment as a part of multi-agent system design compels engineers to deal explicitly with the configuration and assembling of the various system functionalities assigned to the agent environment.

Different types of functionalities can be provided by the agent environment [46]:

- *Virtualization.* The agent environment can provide an abstract representation of resources in the external world and keep the state of this virtualization in synchrony with the state of the actual resources. Examples of virtualizations are a representation of the local topology of nodes in a dynamic network [24] and a map of the area in which a robot is situated with its actual position and the positions of neighboring robots [52]. Virtualization provides a means to bridge the conceptual gap between the agents and the low-level details of accessing resources.

- *Virtual Resources.* The agent environment can serve as a container of virtual resources that provide functionalities and services that can be used by agents. A typical example is the approach based on the notion of "artifacts" that can represent stand-alone services providing specific functionalities to agents [36].
- *Mediated Interaction.* The agent environment can support mediated interaction to regulate the access to shared resources and to govern interactions between agents. Particular laws or rules can be defined that regulate interactions both among agents and with resources. Examples of mediated interaction are governed interaction [40,55], electronic institutions [2], tag-based coordination and reputation mechanisms [32], digital pheromones [30], and computational fields [44].
- *Reflection.* The agent environment can provide a reflective mechanism that enables the composition and function of the agent environment to be modified at runtime. Support for reflection is little studied in research on environments in multi-agent systems. The work on coordination artifacts is a promising approach in this direction [35].

Example application of the agent environment

We illustrate the use of the agent environment as a design abstraction in an industrial automated transportation system. In this application, automatic guided vehicles (AGVs) have to transport loads in a warehouse. To improve the flexibility of the system, a multi-agent control system was applied [45]. Each task that enters the system is represented by an agent and each AGV is controlled by an agent. Since the warehouse environment is very constrained, it restricts how agents can exploit the environment. Agents can only steer the AGVs along well-defined paths in the environment, they can command the machines to manipulate loads, and communicate with each other via wireless communication. An obvious approach to coordinate agents' behavior is to use negotiation protocols. However, due to the highly dynamic nature of the system (irregular stream of tasks, changing traffic loads, AGVs that leave for maintenance and afterwards re-enter the system, etc.), this results in a very complex solution. Therefore a *virtual environment* is employed that embodies a concrete instantiation of an agent environment. Since the agents are distributed over different machines, an instance of the virtual environment is provided on each machine. Dedicated middleware supports the synchronization of the state of the virtual environment on different machines when needed [39].

The virtual environment provides a virtualization of the underlying physical world. It contains a map that represents an abstraction of the warehouse layout. This map is used as a substrate to support different types of mediated interaction.

First, the virtual environment supports the assignment of transportation tasks to AGVs by means of a *field-based* coordination approach [44]. In this approach, tasks emit fields that attract idle AGVs. As the virtual environment is responsible for maintaining the fields, agents are alleviated from performing complex task assignment protocols: the agents of idle AGVs just steer their vehicles along the gradient of the combined field, guiding the AGVs towards loads that have to be transported.

Second, to avoid collisions, the virtual environment supports an approach inspired by traffic lights [51]. Agents place *hulls* in the virtual environment to demarcate the path they intend to drive. The virtual environment is responsible for detecting conflicts, and changing the hulls' color accordingly. Without a conflict, the environment changes the hull's color to *green*, which is the sign for the agent to drive on. In case of a conflict, the virtual environment resolves the problem and changes the color of the vehicle with the highest priority to green, indicating that it is safe to drive on. The color of the hulls of the other involved AGVs is changed to red, indication that they have to wait.

Because the virtual environment is assigned the responsibility to deal with the coordination of vehicle movements, the agents are relieved from the burden of performing complex negotiation protocols. This

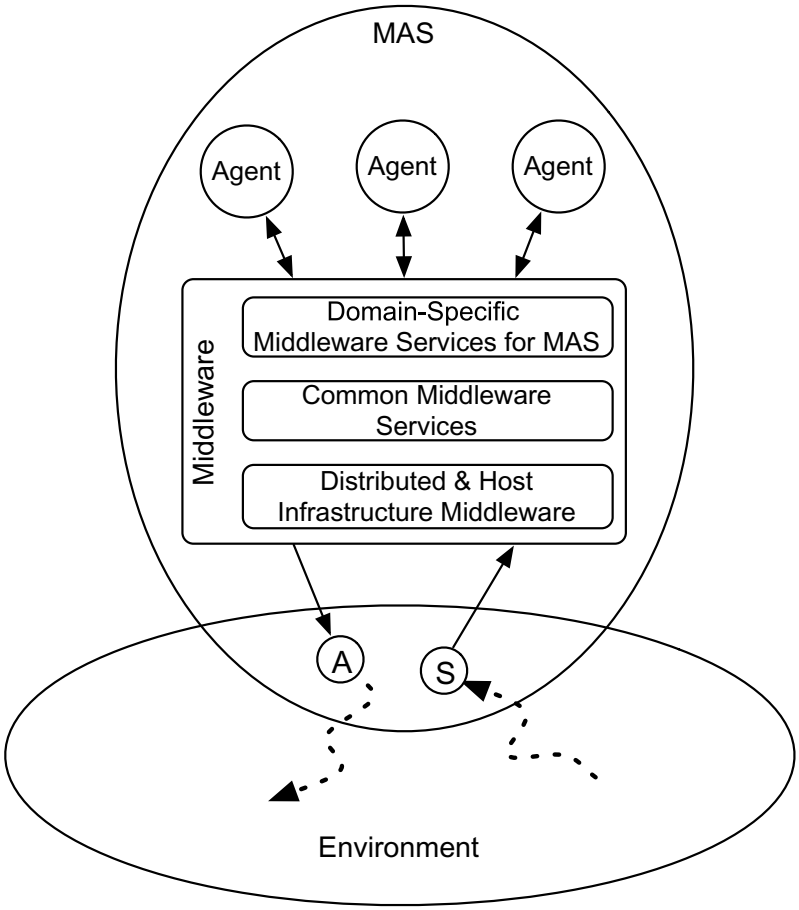


Fig. 5. A multi-agent system with the agent environment mapped onto a layered middleware architecture.

example illustrates how the agent environment can be used as an customizable medium that facilitates the development of a multi-agent system application.

4. The agent environment and middleware

In this section, we discuss the connection between the agent environment and middleware. We reflect on their relationship, and we discuss a number of challenges for future research.

4.1. Connection between agent environment and middleware

Mapping Fig. 4 to Fig. 3 yields Fig. 5. The mapping shows that the agent environment can be considered as middleware specifically targeted at multi-agent systems.

When considering the agent environment as middleware, the question arises how the agent environment relates to the different middleware layers. We take a closer look at each layer:

– `Distributed & Host Infrastructure Middleware.` Multi-agent system engineers generally consider distributed middleware services (RMI, CORBA, SOAP, etc.) as a basic platform

to build multi-agent systems. The services provided by the bottom layer are not the main focus of research on agent environments, but are typically considered as given infrastructure.

– `Common Middleware Services.` In multi-agent system development, common middleware services such as security, persistency, transactions, etc. are only considered minimally. For lab prototypes, there is a tendency not to consider these domain-independent services. Since the proportion of real-world multi-agent systems is rather limited, little experience exists with integrating common middleware services in multi-agent systems.
  A number of agent-based platforms integrate particular common middleware services. Examples are Retsina [41] developed at Carnegie Mellon University that includes basic services for security, performance monitoring, logging, and failure monitoring, and the more recently developed Living Systems of Whitestein Technologies [53] that is integrated with J2EE and provides support for data management with transactions, persistency, client access through Web services, etc.
  In some cases, support for domain-independent concerns is reinvented and developed from scratch at the agent application level. In general, however, it is not common practice to consider these concerns in multi-agent system development as services at a middleware level.
– `Domain-Specific Middleware Services.` Support for agent interaction such as communication services for message exchange and infrastructures for coordination are part of the domain-specific middleware services layer. These infrastructures are built on top of the distributed middleware platform and comprise programming abstractions and services that can be reused across multi-agent system applications. Almost all agent platforms offer domain-specific middleware services. The types of support are very different and include support for distributed message communication such as FIPA-OS [11], electronic institutions [2], artifacts [34], pheromone infrastructure [6], and distributed tuples [23]. Examples of more specific approaches are delegate multi-agent systems [16], tag-based interaction [32], cognitive stigmergy [33], and communication filters [37].
  Specific middleware services for multi-agent systems are also the main focus of research on environments in multi-agent systems. Still, the agent environment is often considered to be a part of the application (i.e. a first-class design abstraction at application level), rather than middleware. This is nevertheless a logical step in the typical evolution of middleware: first building many concrete applications and afterwards consolidating common features shared by these applications in reusable middleware services.

4.2. Research challenges

Reflecting on the connection between the agent environment and middleware, we list a number of research challenges that, in our opinion, are important for the further research on agent environments.

Consolidation

We observe two common approaches for developing the agent environment today:

1. Multi-agent system developers use a particular communication or coordination infrastructure to support the application at hand. However, all functionality of the agent environment that is not provided by this infrastructure is then integrated with this infrastructure in an ad-hoc manner.
2. Developers build the agent environment from scratch. This approach yields custom solutions for each application.

Current practice in agent environment design is not particularly targeted to increase reuse of services across applications. An ongoing research challenge will be to consolidate prevalent application-specific services in domain-specific services in order to amplify reuse. This includes defining reusable programming abstractions to support application developers and developing accompanying runtime infrastructure.

Integration

Applications require middleware support that is tailored to their needs. However, stabilized services provided by the agent environment, such as communication and coordination services, are typically developed as stand-alone services and as such difficult to compose with other services. Supporting flexible integration of different services poses severe research challenges. Figure 6 zooms in on Fig. 5 and illustrates the integration challenges. We distinguish between challenges with respect to horizontal integration and vertical integration:

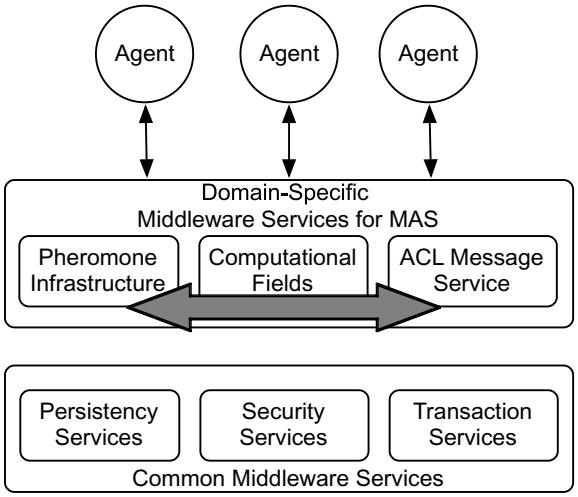
- *Horizontal integration of domain-specific services for multi-agent systems.* Horizontal integration, as illustrated in Fig. 6(a), comprises support for integrating several services that reside at the same level of abstraction. For example, a particular multi-agent system can rely on several coordination mechanisms, e.g. a combination of direct message exchange, and indirect coordination through computational fields and digital pheromones. Such applications require domain-specific middleware that offers suitable services and supports a flexibly composition of these services according to the application requirements at hand.
- *Vertical integration of domain-specific services upwards with the agents layer.* This integration, illustrated in Fig. 6(b), mainly concerns facilitating the use of domain-specific services for the agent application developer. Establishing standards for describing how agents interact with domain-specific services is a core issue with respect to interoperability.
- *Vertical integration of domain-specific services downwards with the common middleware services.* This integration is illustrated in Fig. 6(c). In mainstream software engineering, domain-specific middleware services are underpinned by common middleware services. However, in research on agent environments, the use of common middleware services is currently understated, and provides a significant research challenge. Moreover, underpinning the agent environment with common middleware services prevents reinventing the wheel and emphasizes that research on agent environment fits well within mainstream middleware.

Dynamic composition and adaptation

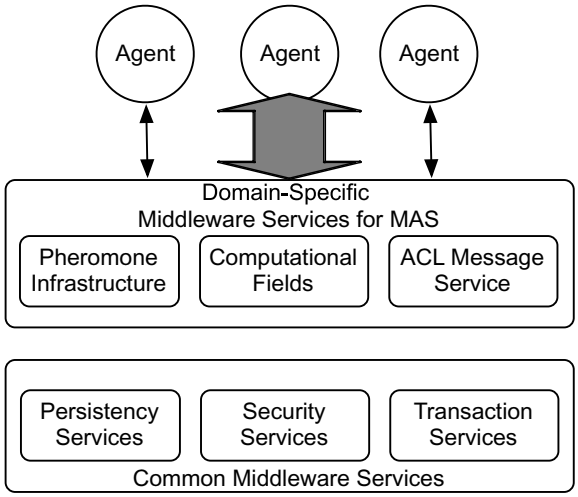
To deal with increasing dynamic operating conditions of multi-agent systems, flexible composition and dynamic adaptation of services must be supported by the agent environment. Current research on environments in multi-agent systems focuses on application-specific design of the agent environment. However, there is little attention for dynamic composition and adaptation of services. Two possible approaches to provide support for dynamic composition and adaptation of services are:

- The services of the agent environment provide a dedicate interface that allow the agents of the application layer to monitor the changing operating conditions and adapt the behavior and/or compositions of the services.
- A dedicated infrastructure monitors the dynamics in the system and its environment and adapts the behavior and/or composition of the middleware services according to the requirements of the system at hand.

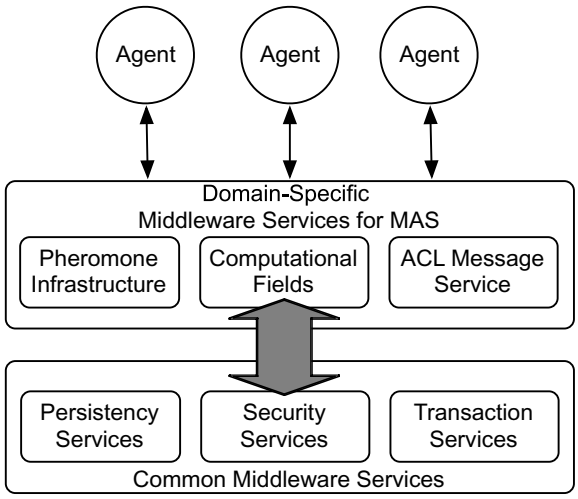
In these approaches, aspect-orientation is a promising approach to modularize the self-adaptive behavior of the system.



(a) Horizontal integration of domain-specific services for multi-agent systems.



(b) Vertical integration of domain-specific services upwards with the agent application.



(c) Vertical integration of domain-specific services downwards with the common middleware services.

Fig. 6. Integration challenges when considering the agent environment as domain-specific middleware.

5. Conclusions

In this paper, we have placed the notion of agent environment in a broader setting of mainstream software engineering. This reflection learns us that the agent environment fits well in the general

perspective of middleware.

Over the years, a number of domain-specific services for agent interaction have emerged. Examples are communication services for message exchange, electronic institutions, and support for stigmergic coordination. Typically, these services are developed as stand-alone services and as such are difficult to compose. Middleware recognizes the need for flexible composition of services as one of its main research challenges. One of the main merits of recent research on environments in multi-agent systems is the recognition of the agent environment as a first-class design abstraction. From a middleware perspective, agent environment design corresponds to flexible composition of domain-specific services. Given the relatively new field of research on agent environments and the limited engineering practice, we are only at the beginning of consolidating domain-specific services for multi-agent systems.

Putting the agent environment in a middleware perspective yields a number of challenges for future research on environments in multi-agent systems. An ongoing challenge will be consolidating application-specific services in domain-specific services in order to amplify reuse. An important problem to tackle is integration, i.e. horizontal integration among domain-specific services for multi-agent systems, and vertical integration of domain-specific services upwards with the agent layer, and downwards with the common middleware services and the underlying distributed platform. Another crucial challenge is to develop proper models and support for dynamic composition and adaptation of services in the agent environment.

Acknowledgments

The research described in this paper is sponsored by the DiCoMAS project that is funded by Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT). Danny Weyns is funded by the Research Foundation Flanders (FWO). We thank Alessandro Ricci and Mirko Viroli for the inspiring discussions on the subject of this paper. Finally, we are grateful to the anonymous reviewers for the very useful feedback on early versions of the paper.

References

[1] G. Alonso, F. Casati, H. Kuno and V. Machiraju, *Web Services – Concepts, Architectures and Applications*, Springer, 2004.
[2] J. Arcos, P. Noriega, J. Rodriguez-Aguilar and C. Sierra, E4MAS Through Electronic Institutions. In Weyns et al. [49].
[3] F. Bellifemine, A. Poggi, and G. Rimassa, Jade, A FIPA-compliant Agent Framework, in: *Fourth International Conference on Practical Application of Intelligent Agents and Multi-Agent Technology*, London, UK, 1999.
[4] G. Blair, G. Coulson, A. Andersen, L. Blair, M. Clarke, F. Costa, H. Duran-Limon, T. Fitzpatrick, L. Johnston, R. Moreira, N. Parlavantzas and K. Saikoski, The Design and Implementation of OpenORB 2, *Distributed Systems Online* **2**(6), 2001.
[5] E. Bonabeau, F. Henaux, S. Guérin, D. Snyers, P. Kuntz and G. Theraulaz, Routing in Telecommunications Networks with Ant-Like Agents, in: *Second International Workshop on Intelligent Agents for Telecommunication and Applications*, volume 1437 of *Lecture Notes in Computer Science*, S. Albayrak and F. Garijo, eds, Paris, France, Springer-Verlag, 1998.
[6] S. Brueckner, *Return from the Ant, Synthetic Ecosystems for Manufacturing Control*, Ph.D Dissertation, Humboldt University, Berlin, Germany, 2000.
[7] C. Castelfranchi, F. Dignum, C. Jonker and J. Treur, Deliberative Normative Agents: Principles and Architecture, in: *Sixth International Workshop on Intelligent Agents, Agent Theories, Architectures, and Languages* (*ATAL 2000*), London, UK, Springer-Verlag, 2000.
[8] A. Drogoul and J. Ferber, Multiagent Simulation as a Tool for Studying Emergent Behavior Processes in Societies, in: *Simulating Societies: Computer Simulation of Social Phenomena*, UCL Press, 1994.
[9] R. Englemore and A Morgan, *Blackboard Systems*, Addison-Wesley, 1988.

[10] FIPA, Foundation for Intelligent Physical Agents, http://www.fipa.org/, 10/2007.
[11] FIPA-OS, Agent Toolkit, http://sourceforge.net/projects/fipa-os/, 10/2007.
[12] L. Gasser, Mas infrastructure: Definitions, needs and prospects, in: *International Workshop on Infrastructure for Multi-Agent Systems*, London, UK, Springer-Verlag, 2001, pp. 1–11.
[13] D. Gelernter and D. Carrierro, Coordination languages and their significance, *Communications of the ACM* **35**(2) (1992).
[14] P.P. Grassé, La Reconstruction du nid et les Coordinations Inter-Individuelles chez Bellicositermes Natalensis et Cubitermes sp. La theorie de la Stigmergie. Essai d'interpretation du Comportement des Termites Constructeurs, *Insectes Sociaux* **6** (1959).
[15] I. Hayes, M. Jackson and C. Jones, Determining the specification of a control system from that of its environment, in: *Formal Methods*, volume 2805 of *Lecture Notes in Computer Science*, K. Araki, S. Gnesi and D. Mandrioli, eds, Springer, 2003.
[16] T. Holvoet and P. Valckenaers, Environment Support for Coordinating Agents Intentions. In Weyns et al. [49].
[17] M. Huhns and L.M. Stephens, Multiagent Systems and Societies of Agents, in: *Multiagent Systems, A Modern Approach to Distributed Artificial Intelligence*, MIT Press, 2000.
[18] V. Issarny, M. Caporuscio and N. Georgantas, A Perspective on the Future of Middleware-Based Software Engineering, in: *Future of Software Engineering, 29th International Conference on Software Engineering*, Minneapolis, USA, 2007.
[19] M. Jackson, The Meaning of Requirements, *Annals of Software Engineering* **3** (1997), 5–21.
[20] S. Krakowiak, What is Middleware? http://middleware.objectweb.org/, 2003.
[21] B. Lagaisse and W. Joosen, True and Transparent Distributed Composition of Aspect-Components, in: *Middleware*, volume 4290 of *Lecture Notes in Computer Science*, M. van Steen and M. Henning, eds, Springer, 2006.
[22] P. Maes, *Artificial Life Meets Entertainment: Life like Autonomous Agents* **38**(11) 108–114.
[23] M. Mamei and F. Zambonelli, *Field-Based Coordination for Pervasive Multiagent Systems*, Springer Series on Agent Technology, Springer-Verlag, 2006.
[24] M. Mamei, F. Zambonelli and L. Leonardi, Tuples On The Air: A Middleware for Context-Aware Computing in Dynamic Networks, in: *Second International Workshop on Mobile Computing Middleware* (*MCM 2003*), IEEE CS Press, 2003.
[25] A. Murphy, G.P. Picco and G.C. Roman, LIME: A Middleware for Physical and Logical Mobility, in: *Twenty-First International Conference on Distributed Computing Systems* (*ICDCS 2001*), Washington, DC, USA, IEEE Computer Society, 2001, p. 524.
[26] P. Noriega and C. Sierra, Electronic Institutions: Future Trends and Challenges, in: *Sixth International Workshop on Cooperative Information Agents*, volume 2446 of *Lecture Notes in Computer Science*, M. Klusch, S. Ossowski and O. Shehory, eds, Springer-Verlag, 2002.
[27] A. Omicini, A. Ricci, M. Viroli, C. Castelfranchi and L. Tummolini, Coordination artifacts: Environment-based coordination for intelligent agents, in: *3rd international Joint Conference on Autonomous Agents and Multiagent Systems*, N.R. Jennings, C. Sierra, L. Sonenberg and M. Tambe, eds, New York, USA, 2004. ACM.
[28] A. Omicini and F. Zambonelli, Coordination for Internet Application Development, *Autonomous Agents and Multi-Agent Systems* **2**(3) (1999), 251–269, Special Issue: Coordination Mechanisms for Web Agents.
[29] H.V.D. Parunak, Go to the Ant: Engineering Principles from Natural Agent Systems, *Annals of Operations Research*, 75, 1997.
[30] H.V.D. Parunak, S. Brueckner and J. Sauter, Digital Pheromones for Coordination of Unmanned Vehicles. In Weyns et al. [47].
[31] H.V.D. Parunak and D. Weyns, Special Issue on Environments for Multi-Agent Systems, *Autonomous Agents and Multi-Agent Systems* **14**(1) (2007).
[32] E. Platon, N. Sabouret and S. Honiden, Environmental Support for Tag Interactions. In Weyns et al. [49].
[33] A. Ricci, A. Omicini, M. Viroli, L. Gardelli and E. Oliva, Cognitive Stigmergy: Towards a Framework Based on Agents and Artifacts. In Weyns et al. [49].
[34] A. Ricci and M. Viroli A. Omicini, CArtAgO: An Infrastructure for Engineering Computational Environments in MAS. In Weyns et al. [49].
[35] A. Ricci, M. Viroli and A. Omicini, Environment-Based Coordination Through Coordination Artifacts. In Weyns et al. [47].
[36] A. Ricci, M. Viroli and A. Omicini, Give agents their artifacts: The a&a approach for engineering working environments in mas, in: *6th International Joint Conference on Autonomous Agents and Multiagent Systems*, Honolulu, Hawai'i, USA, 2007, pp. 613–615.
[37] J. Saunier, F. Balbo and F. Badeig, Environment as Active Support of Interaction. In Weyns et al. [49].
[38] R. Schantz and D. Schmidt, Middleware for Distributed Systems, in: *Encyclopedia of Computer Science and Engineering*, B. Wah, ed., 2007.
[39] K. Schelfthout, Supporting Coordination in Mobile Networks: A Middleware Approach, *Ph.D, Katholieke Universiteit Leuven*, 2006.
[40] M. Schumacher and S. Ossowski, The Governing Environment. In Weyns et al. [48].

[41] K. Sycara, M. Paolucci, M. Van Velsen and J. Giampapa, The RETSINA MAS Infrastructure, *Autonomous Agents and Multi-Agent Systems* **7**(1–2) (2003), 29–48.
[42] W. Vasconcelos, Logic-Based Electronic Institutions, in: *First International Workshop on Declarative Agent Languages and Technologies*, volume 2990 of *Lecture Notes in Computer Science*, J. Leite, A. Omicini, L. Sterling and P. Torroni, eds, 2004.
[43] M. Viroli, T. Holvoet, A. Ricci, K. Schelfthout and F. Zambonelli, Infrastructures for the environment of multiagent systems. *Autonomous Agents and Multiagent Systems* **14**(1) (2007), 49–60.
[44] D. Weyns, N. Boucké, T. Holvoet and W. Schols, Gradient Field-Based Task Assignment in an AGV Transportation System, in: *Fifth Joint Conference on Autonomous Agents and Multi-Agent Systems*, Future University-Hakodate, Japan, 2006.
[45] D. Weyns and T. Holvoet, Architectural Design of a Situated Multiagent System for Controlling Automatic Guided Vehicles, *Multi-Agent Systems and Software Architecture, Special Issue of the International Journal on Agent-Oriented Software Engineering*, (to appear).
[46] D. Weyns, A. Omicini and J. Odell, The Environment as a First-Class Abstraction in Multiagent Systems, *Autonomous Agents and Multiagent Systems* **14**(1) (2007), 5–30.
[47] D. Weyns, H.V.D. Parunak and M. Fabien, eds, *First International Workshop on Environments for Multiagent Systems*, volume 3374 of *Lecture Notes in Computer Science*, New York, NY, USA, 2004, Springer-Verlag.
[48] D. Weyns, H.V.D. Parunak and M. Fabien, eds, *Second International Workshop on Environments for Multi-Agent Systems*, volume 3830 of *Lecture Notes in Computer Science*, Utrecht, The Netherlands, 2005, Springer-Verlag.
[49] D. Weyns, H.V.D. Parunak and M. Fabien, eds, *Second International Workshop on Environments for Multi-Agent Systems*, volume 4389 of *Lecture Notes in Computer Science*, Utrecht, The Netherlands, 2006, Springer-Verlag.
[50] D. Weyns, H.V.D. Parunak, F. Michel, T. Holvoet and J. Ferber, Environments for Multiagent Systems, State-of-the-Art and Research Challenges. In Weyns et al. [47].
[51] D. Weyns, K. Schelfthout and T. Holvoet, Exploiting a Virtual Environment in a Real-World Application. In Weyns et al. [48].
[52] D. Weyns, K. Schelfthout, T. Holvoet and T. Lefever, Decentralized control of E'GV transportation systems, in: *Fourth Joint Conference on Autonomous Agents and Multiagent Systems, Industry Track, Utrecht, The Netherlands*, M. Pechoucek, D. Steiner and S. Thompson, eds, ACM Press, New York, NY, USA, 2005.
[53] Whitestein, Living Systems, www.whitestein.com/pages/solutions/ls_ts.html, 10/2007.
[54] M. Wooldridge and N.R. Jennings, Intelligent Agents: Theory and Practice, *The Knowledge Engineering Review* **10**(2) (1995), 115–152.
[55] W. Zang, C. Serban and J. Minsky, Establishing Global Properties of Multi-Agent Systems via Local Laws. In Weyns et al. [49].

Authors' Bios

Danny Weyns received a Ph.D in Computer Science in 2006 from the Katholieke Universiteit Leuven for work on multi-agent systems and software archtitecture. He has currently a post-doc position at the department of Computer Science, Katholieke Universiteit Leuven. Danny's main research interests are in software architecture, self-managing systems, multi-agent systems, and middleware for decentralized systems.

Alexander Helleboogh received a Ph.D in Computer Science in 2007 from the Katholieke Universiteit of Leuven for work on simulation of distributed control applications in dynamic environments. Currently, Alexander works as a post-doctoral researcher at the department of Computer Science, Katholieke Universiteit Leuven. His main research interests are multi-agent modeling and simulation, software architecture and middleware for decentralized systems, and traffic control systems.

Tom Holvoet is a professor in the Computer Science Department at the Katholieke Universiteit of Leuven, Belgium. He obtained a PhD in 1997 for work in open concurrent software development at the Katholieke Universiteit Leuven. Tom's research interests include coordination in decentralized systems, software architecture, middleware, and aspect-oriented software development. He leads a research group on multi-agent systems with seven researchers.

Michael Schumacher is a professor in the Institute of Business Information Systems from the University of Applied Sciences Western Switzerland. Previously, he was a research associate in the Artificial Intelligence Lab at the Swiss Federal Institute of Technology Lausanne (EPFL) in Switzerland, and a visiting researcher in the Intelligent Software Agents Group at Carnegie Mellon University in Pittsburgh, USA. He received his PhD in Computer Science from the University of Fribourg (Switzerland). His main research interests cover agent-oriented software engineering, semantic web service infrastructures and workflow systems. He is especially interested in real-world applications in ehealth, simulations and traffic management.