Auton Agent Multi-Agent Syst (2007) 14:49–60
DOI 10.1007/s10458-006-9001-6

Infrastructures for the environment of multiagent systems

Mirko Viroli · Tom Holvoet · Alessandro Ricci · Kurt Schelfthout · Franco Zambonelli

Published online: 1 August 2006
Springer Science+Business Media, LLC 2006

Abstract The notion of environment is receiving an increasing attention in the development of multiagent applications. This is witnessed by the emergence of a number of infrastructures providing agent designers with useful means to develop the agent environment, and thus to structure an effective multiagent application. In this paper we analyse the role and features of such infrastructures, and survey some relevant examples. We endorse a general viewpoint where the environment of a multiagent system is seen as a set of basic bricks we call *environment abstractions*, which (i) provide agents with services useful for achieving individual and social goals, and (ii) are supported by some underlying software infrastructure managing their creation and exploitation. Accordingly, we focus the survey on the opportunities that environment infrastructures provide to system designers when developing multiagent applications.

Keywords Multiagent systems · Environment · Software infrastructures

M. Viroli (✉) · A. Ricci
DEIS, Alma Mater Studiorum – Università di Bologna, Cesena, Italy
e-mail: mirko.viroli@unibo.it

A. Ricci
e-mail: a.ricci@unibo.it

F. Zambonelli
DSMI, Università di Modena e Reggio Emilia, Italy
e-mail: franco.zambonelli@unimore.it

T. Holvoet · K. Schelfthout
AgentWise, DistriNet, Katholieke Universiteit Leuven, Leuven, Belgium
e-mail: tom.holvoet@cs.kuleuven.be

K. Schelfthout
e-mail: kurt.schelfthout@cs.kuleuven.be

1 Introduction

The MASs (multiagent systems) approach to software development is based on the idea that the emerging characteristics of modern, complex software systems can be suitably tackled by modelling software as a composition of autonomous entities, i.e. *agents*, whose behaviour can be understood and designed in terms of "achieving a goal" [10, 33]. Then, for agents to form a multiagent system they should be situated in a common computational and/or physical *environment*, where agent interactions with each other and with resources is enabled.

Traditionally, the environment of a multiagent system is however considered simply as the *deployment context* where agents are immersed in—which includes e.g. the communication infrastructure, the network topology, the physical resources available. In this case, the environment of a multiagent system is basically considered as an output of system analysis, and designers are passively subject to it. On the other hand, it is now increasingly recognised that the environment is a true design dimension of multiagent applications. It can encapsulate a significant portion of the system's complexity, in terms of services, mechanisms and responsibilities that the agents can fruitfully be freed of [19]. Therefore, as argued in detail in Weyns et al. [27], it is crucial to promote the environment as a *first-class abstraction* in the design of multiagent systems, impacting nearly all stages of multiagent system development—design, implementation, and run-time.

In fact, in the recent years we witnessed the development of a number of software infrastructures used to support a well-engineered environment to multiagent applications. Their structure and objectives are typically very diverse: they can be conceived for specific applications or to tackle some general scenarios; they can be best applied to either wired, nomadic, or ad-hoc networks; or can provide services related to e.g. coordination or organization; and so on. The development and exploitation of such infrastructures represents a significant part of the existing experience in developing environments for multiagent systems. It is therefore the basis for understanding the current and future perspectives of environments in multiagent systems.

The goal of this paper is to discuss the role of infrastructures for environment, and to survey some relevant examples of them. We elaborate on this issue through a model inspired by the three-layer architecture in Weyns et al. [30], which we exploit as a general viewpoint over seemingly diverse approaches. We see the environment of a specific multiagent system as a set of basic bricks we call *environment abstractions*. These are entities that an agent can perceive and interact with in order to exploit their function and services, to be used for individual as well as social goals. A middleware software layer for environments is therefore understood as an infrastructure providing some class of environment abstractions at run-time, making them available to engineers for developing the multiagent application at hand.

In Sect. 2 we explain in detail the notion of environment abstraction and its relation with environment infrastructures and multiagent applications. Section 3 surveys relevant examples of infrastructures focussing on their environment abstractions, and finally, Sect. 4 concludes discussing future research directions in agent-oriented software engineering.

2 Supporting the MAS environment with infrastructures

Following the definition provided in Weyns et al. [27], we shall see the environment as the part of the multiagent system gluing agents into a working system, mediating agent interaction and supporting agent access to resources. On the one hand, an agent infrastructure—like JADE [9], RETSINA [25], or JACK [13]—provides the designer with the agent concept. This is used as an abstraction useful to model and develop the entities that autonomously achieve goals. By analogy, we can say that environment infrastructures—like TOTA [11], TuCSoN [16], AMELI [6], and others described in the following—provide the designer with bricks to form the environment. These are abstractions useful to model and develop the environment of the multiagent system at hand, for they provide useful services related to interaction mediation and access to resources.

2.1 Environment abstractions

The portion of the system functionality that is assigned to the environment can quickly result in a quite complex software structure. This calls for modelling the software system realising the environment as a set of loosely coupled constituents. The reference model for environment described in Weyns et al. [27] specifies the main functions of the environment and their relationships: an environment design maps such functions to the software elements in which the environment is decomposed. In this paper, in particular, we are interested in a decomposition that reflects the agent perception of the environment structure and behaviour, emphasising the role of the environment in supporting the achievement of the multiagent system goals.

We first observe that the environment is not made by entities which are autonomous and goal-oriented like agents. Rather, they are “function-oriented”, in the sense that their main scope in the multiagent system is to provide a function to agents. Agents exploit this function by perceiving such entities and by acting upon them. In other words, the environment constituents are used by the agents and *never* the opposite. This calls for a dual characterisation with respect to the agent one: features like autonomy, proactivity, social attitude and goal-driven behaviour do not apply to the environment. Instead, it should feature reactivity, function-oriented design and behaviour, partial (or sometimes complete) controllability, and so on.

Accordingly, we see the environment as decomposed in building blocks we call *environment abstractions*. An environment abstraction is an entity of the environment encapsulating some function or service for the agents. An agent might perceive the existence of such abstractions in the environment, have a (possibly implicit) awareness of the opportunities they provide, and accordingly interact with them in order to achieve individual as well as social goals. From the development viewpoint, environment abstractions are seen as loci where the designer can enforce rules, norms, and functions, regulating the agent social behaviour.

2.2 Infrastructures for environment

Many classes of environment abstractions can be useful for developing the multiagent system at hand, and the same class of environment abstractions can be useful in many different multiagent systems. Building them from scratch each time is clearly not a viable approach, especially when reuse and effectiveness (correctness,

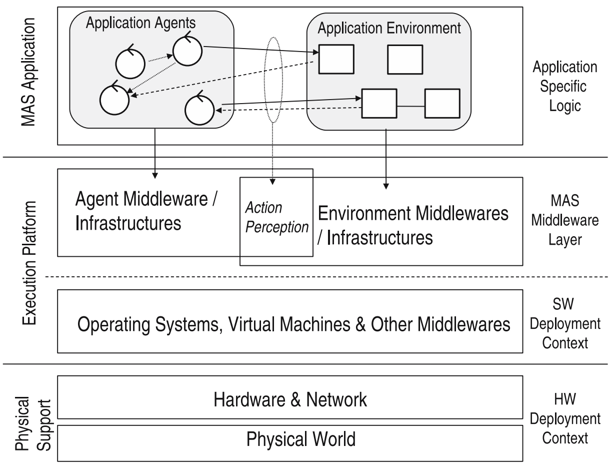

Fig. 1 MAS layers with environment-based supports. Rectangles represent S/W and H/W tiers of the application at different levels. Agents are expressed as circles, environment abstractions as boxes. Arrows from agents to environment abstractions represent actions, dashed arrows in the opposite direction represent perceptions. Arrows between agents represent direct agent communications, while arrows between environment abstractions represent intra-environment interactions. Vertical lines represent the infrastructure supporting a concept at the MAS application level. (Next figures will adopt similar conventions.)

robustness) are quality attributes required. Rather, it is almost mandatory to develop a middleware, that is a software layer handling the life-cycle of environment abstractions and their interaction with agents. So, by analogy with agent infrastructures, a middleware of this kind is seen as an *infrastructure* of services [8, 15]: its main role is to provide environment abstractions to agents for creation, access and manipulation.

Figure 1 shows a logical view of how an infrastructure for environment abstractions ties with typical system layers in a multiagent application—the main structure of this picture is inspired by the work in Weyns et al. [30], though it is here adapted to exacerbate the role of environment infrastructures. The whole system is split in three main layers, concerning the physical support (i.e. the physical deployment context), an execution platform (the software layer over which the MAS runs), and the MAS application. The execution platform is itself divided into the software deployment context—including operating systems and other standard (non MAS-related) middlewares—and the MAS middlewares.

There can be two kinds of MAS middleware: (i) an infrastructure for agents (e.g. JADE) providing agent life-cycle, management, and often some other core services like direct communication, and (ii) one (or more) environment infrastructures, each providing some class of environment abstractions (e.g. tuple spaces, stigmergic fields, and so on). It is worth noting that agent and environment infrastructures cannot be completely isolated. An intersection should exist that handles agent to environment interactions, that is, agent actions and perceptions over environment abstractions.

On top of the layers, the MAS application is formed by agents living over the agent infrastructure, and abstractions provided by the environment infrastructures. Interactions between agents and environment abstractions (actions/perceptions) necessarily cross the agents and environment boxes. On the other hand, interactions between environment abstractions can occur as well, as happens e.g. in the diffusion mechanism of co-fields in TOTA [11].

Note that existing agent infrastructures typically include environment responsibilities. For instance, infrastructures like JADE, RETSINA and JACK already provide agent-to-agent direct communication—this is why in Fig. 1 the "application agents" area includes arrows for message exchange. This spread of responsibilities could be avoided if agent infrastructures provide primitive mechanisms to agent–environment interaction, and agent direct communication is supported by an environment infrastructure. In this case, we would model communication as an interaction mediated by an environment abstraction, which would represent a medium for the exchange of messages. Although mostly unexplored, such an approach would improve separation of concerns in agent-oriented development.

3 A survey of MAS infrastructures for environment

We present a survey of some relevant examples of environment infrastructures, which provide services related to coordination and organization. In each example, we will emphasise the environment abstractions supported, as well as the opportunities provided to designers for solving the application goals.

3.1 Tuple centres in TuCSoN

Many infrastructures have been developed based on the adoption of tuple spaces as basic support for the environment abstractions [22]. In their general form, tuple spaces are seen as shared repositories where agents can insert and retrieve information chunks (tuples), thus allowing for an indirect communication mechanism.

TuCSoN is an example of such infrastructure, providing agents with the *tuple centre* abstraction [16]. Tuple centres are spaces of logic tuples (first order logic terms), and can be programmed to react to certain interaction events (e.g. insertion/removal of certain tuples) and accordingly modify the state of the space through a chain of internal transitions. Such programs—written in the logic-based language called ReSpecT—are used to make tuple centres encapsulating coordination laws. TuCSoN infrastructure supports the creation, access and manipulation of tuple centres, distributed among the nodes of the multiagent system, independently of the underlying network being wireless, monadic, or ad-hoc.

An application example over TuCSoN is given by an open, dynamic, and distributed Workflow Management System [20]. There, several workflow engine components embed and enact the workflow rules—defined by designers—for the coordination of the tasks assigned to participants. This application can be engineered as a multiagent system where agents play the role of the participants, and workflow engines are designed and built as tuple centres, embedding and enacting the workflow rules. In the same application, other tuple centres can be used as repositories of various data, including e.g. a participants registry. Figure 2 shows a Workflow Management System as a MAS application, with TuCSoN used at the middleware layer. Agents can be built on any middleware, and only require the TuCSoN API (written in Java) in order to access tuple centres.

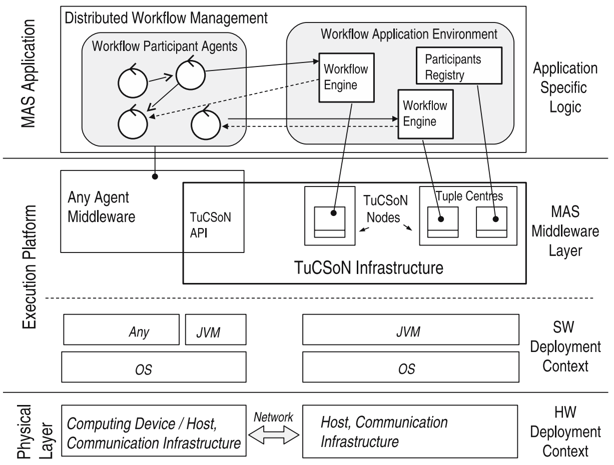

Fig. 2 The distributed workflow management application on top of TuCSoN

The environment abstractions provided to agents by TuCSoN are tuple centres, which in the Workflow Management System application implement workflow engines and registries. Designers of multiagent systems can mainly exploit tuple centres for encapsulating simple up to complex coordination rules, though they can be used also for knowledge sharing and indirect communication.

3.2 Co-fields in TOTA

Another example of tuple-based approaches is the TOTA middleware [11], typically deployed in distributed and dynamic network topologies, and providing mobile agents with a tuple space in each node. Rather than simply inserting and retrieving tuples, the set of tuple spaces in the network model a global, distributed space fabric where *gradient fields* are created and then perceived by agents. Each agent can inject in the net some tuples which encapsulate rules for their diffusion, aggregation and evaporation—rules making such tuples spreading to the neighbour nodes and changing in time until eventually disappearing. The distribution of such tuples in time hence really models a field, which can be perceived by agents and navigated to retrieve resources and/or other agents.

As an example we consider the "intelligent museum" application of field-based coordination. This is a system supporting the visitors of a museum in retrieving information about art pieces, to orientate inside the museum, and meet other people in the case of organised groups. The structure of a MAS application built on top of TOTA middleware, described in Weyns et al. [30], is similar to the one in Fig. 2. The TOTA middleware builds on top of a physical layer where mobile devices are carried by humans and interact through a wireless network. On top, the TOTA middleware provides agents with the ability of creating and perceiving co-fields.

Hence, TOTA provides as environment abstraction the TOTA tuple spaces spread over the net topology, which implement the distributed space supporting co-fields. This can be exploited by a designer in wireless and ad-hoc networks to implement services related to awareness and emergent coordination.

3.3 Mobile tuple spaces in Lime

A related infrastructure for mobile agents is Lime [18], where dynamically shared tuple spaces are supported. Agents have an individual interface tuple space (ITS) where they put and retrieve data, and which moves along with agents. When an agent resides in a host, its ITS actual content is not simply the result of its owner agent's interactions, but it is rather obtained by dynamically joining it with the ITSs of the other agents in the same host. Hence, Lime can primarily be considered as an infrastructure for knowledge sharing and for indirect communication. Furthermore, Lime provides also for a reactive programming mechanism, which can be used to inject some coordination law in the tuple spaces, similarly e.g. to TuCSoN.

The environment abstraction provided by Lime is hence a tuple space local to each agent, which contains private information as well as information global to the host. Typical application scenarios of Lime feature mobile agents roaming an unknown environment, getting partial observations, and dynamically joining their knowledge.

3.4 Virtual environments in the AGV application

A number of interesting infrastructures for the environment in multiagent systems emerged to tackle issues of specific industrial applications. Although conceived for specific scenarios, some aspects of such infrastructures are of quite general applicability, and are related to the infrastructures previously described.

As an example, we consider the AGV (automatic guided vehicles) application [28] supported by the ObjectPlaces [24] infrastructure, where unmanned vehicles controlled by agents transport various kinds of loads through a warehouse. A "virtual environment" (VE) is supported that keeps a consistent and updated map of the physical environment (including vehicles' location and status, such as whether they are executing a job). This is realised through a number of partial representations called *local VEs*: each AGV keeps a local VE used as the information to share. Besides AGVs, the system contains one or more transport bases, where orders are generated and assigned to AGV agents by a number of Transport Agents. Each Transport Agent is responsible for assigning one transport to an AGV, and for making sure it is executed. To achieve this goal, a Transport Agent uses a local VE to get an up to date view on the AGVs position and status. Other than mere sharing, the VE encapsulates important functionalities to support agent cooperation. Agents can e.g. put marks in the VE to avoid collisions, and the VE takes on the burden of handling the interaction protocol between the various AGVs in the mobile network to synchronise these marks.

Figure 3 represents the AGV application, with the various layers in evidence. The ObjectPlaces middleware implements the distributed environment for the agents. By using it, the designer can define interactions in terms of roles, and can define conditions under which such interactions need to be started between the VE and the various AGVs. The middleware then activates these interactions given the current situation. The Message Transfer System (MTS) is a middleware used to enable communications between AGVs and Transport Agents, while E'nsor ® (Egemin Navigation System On Robot) is a S/W layer running on AGV robots which handles their low-level control.

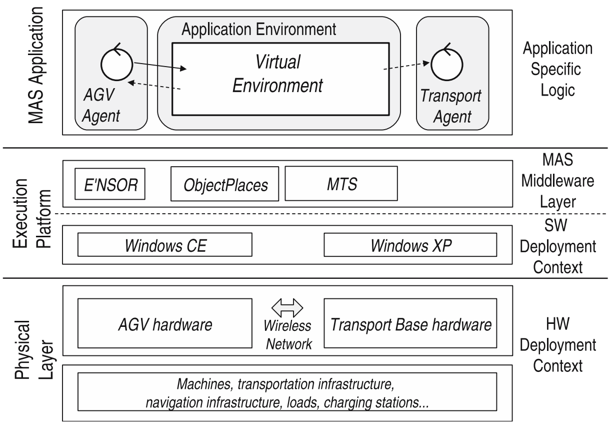

Fig. 3 The AGV application layers

In this application, the two middlewares ObjectPlaces and MTS both contribute in devising an infrastructure providing a shared environment to agents: the global VE can hence be considered here as the global and distributed environment abstraction made available to agents.

3.5 Digital pheromone infrastructure

An interesting application area of environments is the so-called "Command and Control" of multiple robotic entities, exploited e.g. in infrastructures for military air operations. The digital pheremone infrastructure [23] provides services for injecting and perceiving *digital pheromones* in different sites of the physical/logical distributed environment, with the inner ability of aggregating, maintaining and diffusing information according to spatial and temporal criteria—this idea is inspired by stigmergy [17], similarly to the work on TOTA co-fields. In the military context these features can be exploited for several purposes such as surveillance and patrol, target acquisition, and target tracking.

The infrastructure described in Sauter et al. [23] is realised as follows. *Place* agents are stationary entities corresponding to regions of the problem space, forming a graph structure and supporting the environment infrastructure. *Walker* agents have the goal of finding roots in the system, and accordingly communicate with place agents asking for putting and perceiving pheromones of different kinds. *Avatar* agents are created to represent other entities in the environment (either enemies, friendly, or neutral), and can put and perceive pheromones as well. For example, target acquisition can be realised as follows: as long as a walker detects a target agent (that is, another walker) it creates an avatar that keeps pumping a pheromone through the local place agent, with the goal of attracting the target.

In this application, the set of walker agents and avatars exploit environment abstractions resembling repositories of digital pheromones, placed in the nodes of a network. Such an abstraction is realised by a lower-level multiagent system of place agents.

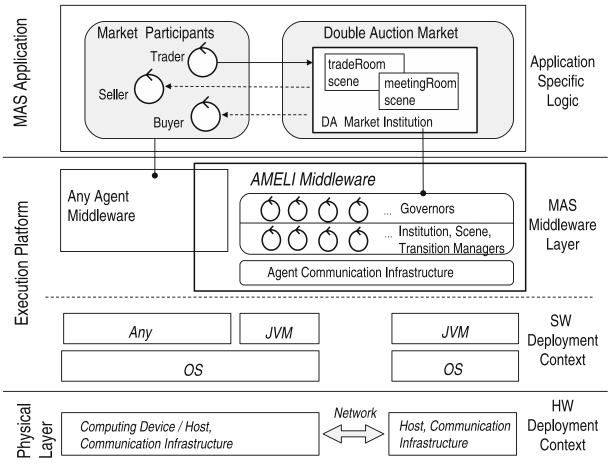

Fig. 4 A market application on top of AMELI middleware

3.6 AMELI middleware for electronic institutions

One of the most challenging issues concerning the engineering of agent-based systems is balancing agent autonomy and social order [4], in particular in the context of *open* multiagent systems. This issue can be tackled by introducing some kind of *regulatory structure* establishing permission and denials for agents, namely, a *normative* system [12]. As in the case of coordination, the mediation role of environment abstractions makes them a possible and effective place where to encapsulate and enact norms and—more generally—organizational rules.

An example of infrastructure supporting organization with electronic institutions is AMELI [6], a computational environment for possibly heterogeneous agents. Each institution represents a normative system, and includes *scenes* that agents can enter and exploit in order to exchange messages.

An application example is given by *markets*, where traders (buyers and sellers) meet to trade their goods under the supervision of the institution, and where auctions are used as trading protocols. The institution as computational environment encapsulates and enforces the norms that characterise the trade.

Figure 4 shows a market as a MAS application, where the middleware layer is formed by the AMELI infrastructure. Note that this infrastructure is itself built using a multiagent system running over a lower communication infrastructure. Hence, agents participating in the institution do not communicate each other directly, but actually interact with the AMELI agents of the infrastructure, with actions like `enterInstitution`, `moveToScene`, and `saySceneMessage`.

The environment abstraction provided by AMELI is hence an institution of scenes, which is a "context" where agents can meet and exchange messages in a regulated way, and where prohibited interactions are disallowed by the infrastructure. Designers exploiting AMELI are able to organise agents cooperation, regulating their access to the institution and enforcing the necessary rules and norms.

4 Conclusions and open research issues

In this paper, we showed that supporting environment abstractions through an infrastructure is a useful interpretation for many existing works on environments for multiagent systems. On the other hand, this very idea can also be endorsed as a starting point for interesting research directions. These are all centered around the attempt of including the environment notion into the current research and practice of multiagent system development.

We note that the vast majority of current AOSE methodologies—e.g. MASE [31], TROPOS [3], and PASSI [5]—minimises the role of environment, and simply assumes that agents interact via a communication language. Some exceptions are Gaia [32] and SODA [14]: though, their interpretation of what the environment comprises is minimal, for design support is limited to the representation of resources and simple access control to the resources. Identifying environment abstractions as building blocks of a multiagent system can foster the introduction of a methodology that tackles in a more systematic way the issue of environment engineering. Fundamental steps would be (i) identifying the proper environment infrastructure(s) to use, (ii) designing the environment abstractions to be included in the specific multiagent system, and (iii) taking into account environment abstractions in agent development.

From an architectural viewpoint, the environment abstractions that a particular infrastructure supports are considered as architectural elements. The choice of infrastructure—and thus of environment abstractions—is an important early design decision that has a significant influence on the architecture of the application, and it is difficult to reverse later. Software architectures and architecture description languages [1, 2, 7] provide means to deal with the specification of the environment abstraction's provisions, requirements, assumptions, functional and quality attributes. Furthermore, software architectures can provide a contribution to the development of environment middleware, as in the reference architecture introduced in Weyns et al. [29]

Finally, another interesting direction of exploration is identifying a general model for environment abstractions, that could help streamlining research on theory and practice of environment in multiagent systems. An example model is coordination artifacts [21, 26]: environment abstractions for coordination whose structure and behaviour can be dynamically inspected by an agent and exploited in a rational way. Interesting perspectives of this work include the study of the interplay between environment and agent intelligence/rationality/cognition, as well as the development of general-purpose infrastructures.

References

1. Allen, R., & Garlan, D. (1997). A formal basis for architectural connection. *ACM Transactions on Software Engineering and Methodology*.
2. Bass, L., Clements, P., & Kazman, R. (2003). *Software architectures in practice (2nd edn.)*. Addison-Wesley.
3. Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., & Mylopoulos, J. (2001). A knowledge level software engineering methodology for agent oriented programming. In *Proceedings of the 5th international conference on autonomous agents*, (pp. 648–655) Montreal (CA): ACM Press.
4. Castelfranchi, C. (2000). Engineering social order. In *Engineering societies in the agents world*, Vol. 1972 of *LNAI* (pp. 1–18) Springer-Verlag.
5. Chella, A., Cossentino, M., Sabatucci, L., & Seidita, V. (2006). PASSI: An agile process for designing agents. *Journal of Computer Systems Science and Engineering*.

6. Esteva, M., Rosell, B., Rodríguez-Aguilar, J. A., & Arcos, J. L. (2004). AMELI: An agent-based middleware for electronic institutions. In N. R. Jennings, C. Sierra, L. Sonenberg, & M. Tambe (Eds.), *3rd international joint conference on autonomous agents and multiagent systems (AAMAS 2004)* (Vol. 1). New York, USA, pp. 236–243, ACM.
7. Garlan, D., Monroe, R. T., & Wile, D. (2000). Acme: Architectural description of component-based systems. In G. T. Leavens & M. Sitaraman (Eds.), *Foundations of component-based systems*. Cambridge University Press, pp. 47–68.
8. Gasser, L. (2001). MAS infrastructure: Definitions, needs and prospects. In T. Wagner & O. F. Rana (Eds.), *Infrastructure for agents, multi-agent systems, and scalable multi-agent systems, international workshop on infrastructure for multi-agent systems, Barcelona, Spain, June 3–7, 2000, Revised Papers*, Vol. 1887 of *Lecture Notes in Computer Science* (pp. 1–11). Springer.
9. JADE-board (2000). Java agent DEvelopment framework. http://sharon.cselt.it/ projects/jade/.
10. Jennings, N. R. (2001). An agent-based approach for building complex software systems. *Communications, ACM 44*(4), 35–41.
11. Mamei, M., & Zambonelli, F. (2005). Programming stigmergic coordination with the TOTA middleware. In F. Dignum, V. Dignum, S. Koenig, S. Kraus, M. P. Singh, & M. Wooldridge (Eds.), *4th international joint conference on autonomous agents and multiagent systems (AAMAS 2005), July 25–29*, 2005 (pp. 415–422) Utrecht, The Netherlands: ACM.
12. Noriega, P., & Sierra, C. (2002). Electronic institutions: Future trends and challenges. In M. Klusch, S. Ossowski, & O. Shehory (Eds.), *Cooperative information agents VI*, Vol. 2446 of *LNCS*. Springer Verlag.
13. Norling, E., & Ritter, F. E. (2001). Embodying the JACK agent architecture. In *Australian joint conference on artificial intelligence*, Vol. 2256 of *Lecture Notes in Computer Science* (pp. 368–377). Springer.
14. Omicini, A. (2001). SODA: Societies and infrastructures in the analysis and design of agent-based systems. In P. Ciancarini & M. J. Wooldridge (Eds.), *Agent-oriented software engineering*, Vol. 1957 of *LNCS* (pp. 185–193), Springer-Verlag. 1st International Workshop (AOSE 2000), Limerick, Ireland, 10 June 2000. Revised Papers.
15. Omicini, A., Ossowski, S., & Ricci, A. (2004). Coordination infrastructures in the engineering of multiagent systems. In F. Bergenti, M.-P. Gleizes, & F. Zambonelli (Eds.), *Methodologies and software engineering for agent systems: The agent-oriented software engineering handbook*, Vol. 11 of *multiagent systems, artificial societies, and simulated organizations*. Kluwer Academic Publishers, Chapt. 14, pp. 273–296.
16. Omicini, A., & Zambonelli, F. (1999). Coordination for internet application development. *Autonomous agents and multi-agent systems, 2*(3), 251–269.
17. Parunak, V. D. (1997). Go to the ant: Engineering principles from natural agent systems. *Annals of Operations Research, 75*, 69–101.
18. Picco, G. P., Murphy, A. L., & Roman, G.-C. (2000). Developing mobile computing applications with LIME. In ICSE. pp. 766–769.
19. Platon, E., Mamei, M., et al. (2006). “Mechanisms” paper in this issue.
20. Ricci, A., Omicini, A., & Denti, E. (2002). Virtual enterprises and workflow management as agent coordination issues. *International Journal of Cooperative Information Systems, 11*(3/4), 355–379. Special issue: Cooperative information agents—Best Papers of CIA 2001.
21. Ricci, A., & Viroli, M. (2005). Coordination artifacts: A unifying abstraction for engineering environment-mediated coordination in MAS. *Informatica, 29*, 433–443.
22. Rossi, D., Cabri, G., & Denti, E. (2001). Tuple-based technologies for coordination. In A. Omicini, F. Zambonelli, M. Klusch & R. Tolksdorf (Eds.), *Coordination of internet agents: models, technologies and applications*. Springer, Chapt. 4, pp. 83–109.
23. Sauter, J. A., Matthews, R. S., Parunak, H. V. D., & Brueckner, S. (2005). Performance of digital pheromones for swarming vehicle control. In *4th international joint conference on autonomous agents and multiagent systems (AAMAS 2005), July 25–29*, 2005 (pp. 903–910) Utrecht, The Netherlands: ACM.
24. Schelfthout, K., Weyns, D., & Holvoet, T. (2005). Middleware for protocol-based coordination in dynamic networks. In *Proceedings of 3rd international workshop on middleware for pervasive and ad-hoc computing (MPAC05)*.
25. Sycara, K., Paolucci, M., van Velsen, M., & Giampapa, J. (2003). The RETSINA MAS infrastructure. *Special Joint Issue of Autonomous Agents and MAS, 7*(1–2).
26. Viroli, M., & Ricci, A. (2004). Instructions-based semantics of agent mediated interaction. In N. R. Jennings, C. Sierra, L. Sonenberg, & M. Tambe (Eds.), *3rd international joint conference on autonomous agents and multiagent systems (AAMAS 2004)*, Vol. 1. (pp. 102–110). New York, USA: ACM.

27. Weyns, D., Omicini, A., & Odell, J. (2006) "Definition" paper in this issue.
28. Weyns, D., Schelfthout, K., Holvoet, T., & Lefever, T. (2005a). Decentralized control of E'GV transportation systems. In *4th international joint conference on autonomous agents and multiagent systems (AAMAS 2005), July 25–29*, 2005 (pp. 67–74). Utrecht, The Netherlands: ACM.
29. Weyns, D., Schelfthout, K., Holvoet, T., Lefever, T., & Wielemans, J. (2005b). Architecture-centric development of an AGV transportation system. In *Proceedings of the 4th international/central and eastern european conference on multi-agent systems*, Vol. 3690 of *Lecture Notes in Computer Science*.
30. Weyns, D., Vizarri, G., & Holvoet, T. (2005c). Environments for situated multi-agent systems: Beyond infrastructure. In D. Weyns, H. V. D. Parunak, & F. Michel (Eds.), *2nd international workshop "environments for multi-agent systems" (E4MAS 2005)*. AAMAS (2005). Utrecht, The Netherlands. Available at E4MAS 2005 website.
31. Wood, M., DeLoach, S. A., & Sparkman, C. (2001). Multiagent system engineering. *International Journal of Software Engineering and Knowledge Engineering, 11*(3), 231–258.
32. Zambonelli, F., Jennings, N. R., & Wooldridge, M. (2003). Developing multiagent systems: The Gaia methodology. *ACM Transactions on Software Engineering Methodology, 12*(3), 317–370.
33. Zambonelli, F., & Parunak, H. V. D. (2003). Signs of a revolution in computer science and software engineering. In *3rd international workshop on engineering societies in the agents world*, Vol. 2577 of *LNCS* (pp. 13–28). Springer Verlag.