

Multi-Agent System: A Guiding Metaphor for the Organization of Software Development Projects

Lawrence Cabac

University of Hamburg, Dept. of Informatics, Vogt-Kölln-Str. 30, D-22527 Hamburg
<http://www.informatik.uni-hamburg.de/TGI>

Abstract In this work we propose the introduction of multi-agent concepts for the organization of software development projects of (especially multi-agent) application design and implementation. This is expressed by the guiding metaphor (German: Leitbild) of a *multi-agent system of developers*.

Team orientation and concurrent development are two aspects that are crucial in every large development project. Consequently, the organizational structure of the programming team has to take account for both. If the developed application is distributed, concurrent and team-oriented – e.g. a multi-agent application – one approach is to aim for a comparable (homomorphic) structure of a developed system and development team. We achieve this by reintroducing the multi-agent system metaphor into the organizational structure of the development team.

Agent attributes such as autonomy, communication, cooperation, self-organization and the capacity for teamwork are transferred by the guiding metaphor back to team members. Concurrency and distribution of resources and processes is naturally supported by the guiding metaphor. This guiding metaphor can be applied to any project organization. However, it is best suited for the organization of multi-agent application development, due to the similarity in structure.

Keywords: agents, guiding metaphors, *multi-agent system of developers*, Leitbild, metaphor, project management, software development approach, team organization

1 Introduction

Multi-agent systems are applications based on encapsulated, autonomous software entities that can flexibly achieve their objectives by interacting with one another in terms of high-level interaction protocols and languages. Agents balance their reactive behavior in response to influences from the environment with their proactive behavior towards the achievement of design objectives.

The agent metaphor is highly abstract and it is necessary to develop software engineering techniques and methodologies that particularly fit the agent-oriented paradigm. Traditional software development techniques such as for example object-oriented analysis and design are inadequate to capture the flexibility and autonomy of an agent's problem-solving capabilities, the richness of

agent interactions and the (social) organizational structure of a multi-agent system as a whole. Many agent-oriented software development methodologies have been brought forward over the last years, many of them already in mature state.

Agent-oriented development methodologies such as GAIA [16,18], MASE [2] or PROMETHEUS [9] are well-established. Similarities can be found in methods and abstractions such as use cases, system structure (organization) diagrams, role models, interaction diagrams and interaction protocols. However, it is not a trivial task to decide on a suitable implementation platform as pointed out by Sudeikat et al. [12].

Similar claims hold for the management of development processes, the organization and guidance of a team as well as for project management. As well as for methodologies and techniques of software development, there also exists a necessity to develop approaches for the management of projects that particularly fit the agent-oriented paradigm. As already proposed by Petrie et al. [10] the organization of projects can be oriented towards the agent concept. The proposal here is to increase even more the symmetry between the project management and the software being build.

We present a guiding metaphor that is capable to dynamically adapt to the needs of the team and development processes. Criteria for a powerful and acceptable metaphor is its simplicity, flexibility and the range of the commonly known concepts. It should take account of the main concepts and design objectives of the developed system; e.g. for a multi-agent application these are concepts such as distribution, concurrency and dynamical structures.

Section 2 introduces the term *guiding metaphor* and explains the guiding metaphor *multi-agent system of developers*¹ for the development of multi-agent-based projects in detail. Section 3 describes the utilization and our experiences with this guiding metaphor.

2 Leitbild: MAS

Before we start with our approach we will elaborate on the notion of the *guiding metaphor*. Then we will describe the guiding metaphor of a *multi-agent system of developers* in regard to three aspects. First, we describe the guiding metaphor in more detail in its role as a Leitbild [19] regarding orientation, notions, strategies and terminology in the environment of multi-agent application development. Second, we go into detail of the guiding metaphor's manifestation in the organizational structure of a (multi-agent application) development project especially in regard to concurrent and distributed development. Third, we focus on communication, coordination, project organization and team management.

¹ We include all participants of a development process, such as programmers, users, supporting staff, etc. We could thus also call the metaphor *multi-agent system of participants* but in the context of system development we regard all participants as developers of the system.

2.1 Guiding Metaphor

A guiding metaphor (German: Leitbild [19]) is a strong and well-established concept that can guide the participants of a development team in a general sense. While the term originated in business management, it is also well established in software engineering. A guiding metaphor should have four functions. It should offer orientation and have a strong integrative force. Decision processes should be supported by the guiding metaphor and it should also be a means of coordination. Züllighoven et al. define a guiding metaphor as follows.

A guiding metaphor in software development defines a frame of orientation for involved groups in development processes as well as during utilization. It supports the design, utilization and the evaluation of software and is based on values and goals. A guiding metaphor can be used constructively or analytically [19, from German, p. 73].

An important feature is that the guiding metaphor is so general and common that every potentially involved person has at least a good idea of the organizational concepts, structures, notions and rules. A good guiding metaphor comes with a whole set of other metaphors that do not have to be named explicitly.² In the context of developing software we can distinguish three different forms of guiding metaphor. It can be used to characterize the software systems, the development process and also the team organization (respectively project management).³ Examples of guiding metaphors are the *tools & material* approach [19] or the *expert work place* [6] for software systems. Guiding metaphors for team organizations are *the factory*, *the office*, *the workshop* or the *(free) jazz band* [15].

One interesting approach as to how to define a new guiding metaphor for team organization has been done by Mack [7]. He proposes the guiding metaphor of an *expedition* for the development process and derives some aspects that are useful in everyday (work) life of a software developer. Here we will not go into detail of this guiding metaphor but we would like to elaborate on the notions that are instantly linked to this example to show the potentials of a guiding metaphor.

For a (development) expedition one will need a team (developers, supporting users and other staff) and resources (computers, software, rooms, paper, etc.). There should be a good notion on how much everyone can carry (individual capabilities of team members) on the way. The organizers need to work out a plan in advance that is detailed enough to take as many aspects as possible into account and flexible enough to allow the team members to react to sudden changes and dangers. In an expedition it seems clear that all members have to support each other and that conflicts that are left unsolved can lead to difficulties that can endanger the expedition (software project). A good communication between members of the team is essential in all stages of the expedition. We

² In this way the guiding metaphor can be compared to an *extended metaphor* or even a *parable* as used in literature.

³ In this work we focus on the function of the guiding metaphor for the team's organizational structures / project management.

know that an expedition is a socially challenging project that can be adventurous as well as hard work. In addition, the outcome of an expedition is open in the beginning.

The example shows that a strong guiding metaphor offers many notions (common in the team) and a multitude of metaphors. These help team members to find orientation in the project and by this the guiding metaphor succeeds in guiding a team.

In the following sections we describe a guiding metaphor that is well applicable to the development of multi-agent application and is also well known in the multi-agent community. It is the multi-agent system.

2.2 Multi-Agent System of Developers

Our approach of organizing projects for multi-agent application development is described by the guiding metaphor of *multi-agent system of developers*. Developer teams, their members and their actions are characterized by the attributes usually related to agents [17], multi-agent systems [3] and cooperative workflows [14].⁴ In the team members are acting in a self-organized, autonomous, independent and cooperative way. They all have individual goals that culminate in a common vision of the system that is to be developed.

Like agents in a multi-agent system, developers are situated in an environment, in which they communicate with other developers and other participants of the development process. Moreover, the environment offers services or restricts the possibilities of actions for the developers.

Multi-Agent System of Developers

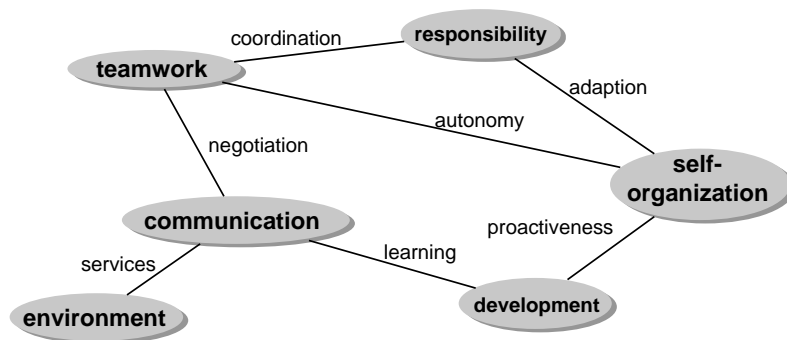


Figure 1. Agent concepts used in the context of team organization (selection).

⁴ In the following many agent concepts are used to describe behavior or attributes of members of the development team. These are used for the metaphorical power.

Figure 1 shows a selection of typical multi-agent concepts and their inter-relationships that are utilized in the development project context as metaphors. Lippert et al. [6] identify a selection of key metaphors as shown in the figure as a metaphor design space.

The agent metaphor leads to dynamic and flexible structures in the team's organization. All members can form (sub-)teams with other members during the development process. This is not only encouraged but also a main aspect of the self-responsible and autonomous actions of team members. The structure of a team is not static. Sub-teams are able to decide their own dissolution and to proactively decide on new alliances. From this point of view concurrent and distributed work is a natural phenomenon.

According to the *multi-agent system of developers* metaphor control, project management and organizational matters in the development process are managed through mechanisms typically owned by social agents [13]. Thus social norms, conventions and motivation become important forces in the team's behavioral patterns.

At first glance it seems odd to re-transfer the concept (metaphor) of a multi-agent system, which has been used to define and organize software systems in the manner of (sociological) organizations, back to an organizational structure of people. However, the metaphor of a multi-agent system has grown so strong in recent years that many developers are well acquainted with the notions and key elements of agent concepts. Therefore, the multi-agent system is a reasonable, well-established and powerful guiding metaphor. But even for participants of the team that do not share the concepts of multi-agent systems as paradigm – e.g. users with no technical background – still all the concepts are well known, since they are rooted in social organizations.

In the following two sections we elaborate on two main aspects of agent-oriented development. These are the communication of agents and the concurrency and distribution. Through the guiding metaphor both aspects take a leading role in our vision on the project organization.

2.3 Matrix Organization

In a multi-agent application development project the organizational structure has to be defined, such that responsibilities for certain aspects can be assumed by team members or sub-teams. The general perspectives in the area of a multi-agent system and – therefore also here – for the development process are *structure*, *behavior* and *terminology*. These perspectives are orthogonal with connecting points at some intersections (compare Figure 2).

The structure of a multi-agent system is given by the agents, their roles, knowledge bases and decision components [4,11]. The behavior of a multi-agent system is given by the interactions of the agents, their communicative acts and the internal actions related to the interactions [1]. The terminology of a multi-agent system is given as a domain-specific ontology that enables agents to refer to the same objects, actions and facts. The agents' common ontology is crucial for their successful interactions.

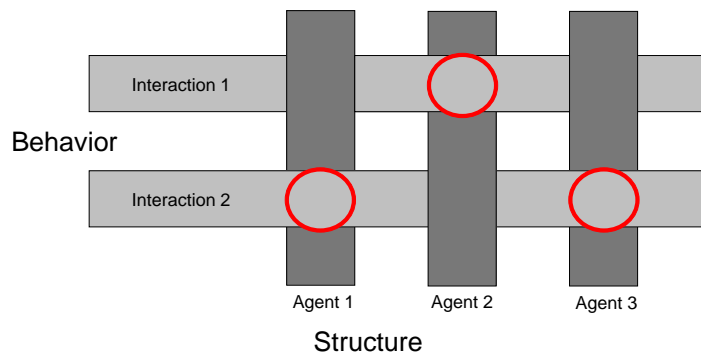


Figure 2. Two dimensional matrix showing perspectives (*behavior, structure*).

A schematic two dimensional matrix is depicted in Figure 2 showing the independence and interconnections of agents and interactions. Neither is there any direct relationship between any pair of agents, nor between any pair of interactions. Thus these architectural elements are independent and drawn in parallel to each other. Agents and interactions are orthogonal because each agent is involved in some interactions and the same holds the other way around. When an agent and an interaction are coupled, a circle marks the interconnection point.

The general case for any two structural and/or behavioral elements is independence. In the diagram interconnections are explicitly marked. The ontology, which is omitted in the diagram, is the third dimension of perspectives. This perspective is orthogonal to the other two perspectives, but it tends to have many interconnection points because each interaction and each agent needs parts of the ontology definition to fulfill its purpose.

Since the three perspectives are orthogonal and independent within each perspective, it is easily possible to divide the tasks of design and implementation into independent perspectives and independent parts. This means that different interactions can be developed by independent sub-teams and different agents can be designed by other independent sub-teams. Between agent teams and interaction teams, coordination is needed for the crucial parts only (circles).

Following this method, the different parts of the system can be developed independently and concurrently as long as there is enough coordination / synchronization between intersecting groups.

In general it is not a good idea to assign tasks of orthogonal dimensions to the same sub-team because then the responsibilities of the different dimensions might become blurred. However, developers are well advised to look for similarities between independent elements of the same dimension, like for example a set of similar interactions. In such a situation, code reuse becomes possible if a sub-team is responsible for multiple parallel elements.

The (agent-based) software system imposes its matrix structure onto the team organization. In the metaphor of *multi-agent system of developers* this is naturally supported.

2.4 Communication, Coordination and Synchronization

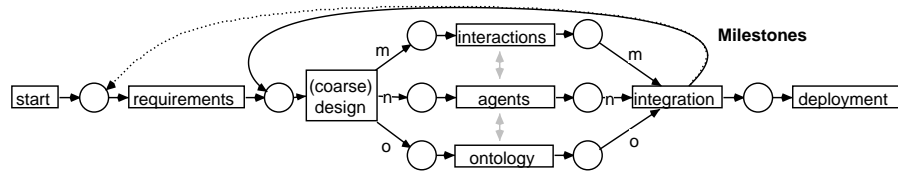


Figure 3. Schematic and coarse Petri net model of the PAOSE development process.

We can identify four main phases when applying the guiding metaphor of a *multi-agent system of developers* to the time schedule: (1) the requirements analysis, (2) the (coarse) design of ontology/roles/interactions, (3) the concurrent and highly interactive implementation of ontology/agents/interactions and (4) an intense and concurrent integration and testing phase. The time schedule is iterative in all phases, however, in normal settings iterations in phases two to four would suffice.

Figure 3 shows a schematic Petri net model of the development process. The design phase results in several independent tasks for interaction, agent and ontology implementation.

The synchronizations between concurrent processes during implementation in the form of communication between the groups have to be supported during development, both through synchronous and asynchronous communication. This is achieved by physical meetings (synchronous), through (web-based) tool support (synchronous and asynchronous) and implicit communication in documentation of activities and code (asynchronous). At the end of the implementation phase a thorough integration phase is necessary to obtain a milestone / running system. Each phase in itself is a process with its own structure.

While the processes of independent activities are concurrent, some synchronizations are necessary during implementation between orthogonal groups (gray arrows). Also phase shifts should be coordinated. This is implied in Figure 3 and explicitly shown at integration, which should be entered synchronously by the whole team.

All team members are attributed with the sociality of communicating agents. The team structure is self-organized and controlled through participating developers by observation, negotiation, rules and norms.

Awareness of participants is an important factor in avoiding problems resulting from miscoordination. Unfortunately, the support for user awareness in our tool-set is not sufficient yet. Thus, we have to compensate with extensive communication about changes in design and implementation. Nevertheless, some simple elements in our communication platform exist, which enable us to track recent changes. Improvements are being discussed.

3 MAS of Developers in the Project Context

The concept of the guiding metaphor has to be backed up with the utilization in the context of a multi-agent application development project. Here the guiding metaphor can unveil its usefulness.

3.1 Employing the Guiding Metaphor

Following the guiding metaphor of *multi-agent system of developers* project organizers or initiators will be able to anticipate the needs of the team members during the development. Good equipment, enough resources and an adequate team composition are essential to any project. Here also the means of communication, coordination, learning, reorganization and the possibility to take responsibility are important parts in the development process. These processes have to be supported by adequate means, for example regular meetings for direct communication and teamwork sessions and/or a (web-based) communication system for asynchronous (and synchronous) communication. These communication means have to be integrated into the environment (platform) of the developers (agents).

The organizers have a powerful means to guide the actions, the way of thinking and the general behavior of participants in the context of the project. Here the main responsibility is that the metaphor is well conveyed to all participants. If all participants have a good notion of agent concepts, everyone will be able to live the metaphor (and the team will profit from that). This means that all participants are aware of the fact that participation (coordination, negotiation) in the development process and in the decision processes of team members as well as the possibility for the team to exercise the sociological prosperity is of importance. The ease of the adaption to the guiding metaphor – borrowed from sociological theories about organizations – can lead to higher motivation, integration and identification with the group and the common goals, which in turn leads to quicker orientation in the project and higher productivity.

In addition to the metaphor's inherent organizational powers, the developers can benefit from a structural organization of the development process that resembles the structure of the developed system.

3.2 Homomorphic Structure

The advantages to work with a homomorphic – similar – structure in software organization and project organization are manifold. In general, they are the same advantages as those of multi-agent systems over conventional paradigms.

The multi-agent system organization of the development team allows for and supports distributed as well as concurrent development. In this context it is important that developers act self-responsible and consider self-reorganization if necessary. Structures in the team should emerge from the processes during development. Thus independence and flexibility as well as means for communication and mobility are supported in this approach as first-order concepts. One main advantage of the similar structure for the developed software – and a successful project – is that the same principles, concepts and organization help the developers also to design a truly agent-oriented software system. Distribution, autonomy and concurrency in the organizational structure will automatically foster the same attributes in the designed system.

Some disadvantages also exist and – not surprising – these are the same disadvantages as those of multi-agent systems again. To succeed with the project by employing the *multi-agent system of developers* metaphor a strong emphasis on communication and adaptive processes has to be made. This leads to a large communication overhead. Due to the flexible and dynamical organization, the inherent concurrency and distribution, the complexity of the project organization is very high. This leads to more management overhead (compared to a non-distributed and non-concurrent development).

3.3 Experiences

Especially in our teaching projects the guiding metaphor of *multi-agent system of developers* works extremely fine. This results to some extent from the fact that our students have a well-founded background knowledge of basic and advanced agent concepts. Usually these concepts are conveyed through conceptualized object Petri net models, which have a strong graphical representation for concurrency, locality and hierarchical nesting.

The main aims of multi-agent system development (concurrent, independent development) are reached with the support of the guiding metaphor. However, it is still useful to gather the source code in a central repository even if parts of the system are run exclusively in disjunct places. This eases the deployment of system and framework.

In addition, common elements have to be made available to all members. Many documents like overview diagrams (multi-agent system structure) or ontology definitions respectively models are also still designed in a central (non-distributed) fashion. Here, still more flexibility can be added to the development process. However, it is not essential to work concurrently (of independently) on these elements, since the ontology for instance is meant to be common to all agents as well as common to all developers. Moreover, these *central* specification elements (especially ontology) can be used by the project leaders to actively

control the direction of the development. The software MAS ontology becomes a common language for the developer MAS as well.

Many improvements in support of the development team, communication means and increase of flexibility are possible and the extend of the guiding metaphor has not reached its limits, yet. We would like to include direct and indirect communication, inline documentation and workflow capabilities into our development environments (RENEW [5], MULAN [4,?,11], Eclipse) to better support the interactive means of the developers in their environment. Web-based documentation and groupware features can also be more heavily exploited.

4 Conclusion

In this work we present a guiding metaphor for the organization of (multi-agent) application development projects. The guiding metaphor itself is taken from agent technologies. It is the multi-agent system metaphor applied to the team of developers (and other participants). By this self-reflective view on the organization of development teams a coherent structure in all parts of the system and all processes is defined.

Guiding metaphors are well suited to give a common orientation in a development team. The multi-agent system is, through its origination from socio-organizational structures, its generality, its ease of accessibility and its recognition of distribution, well suited to serve as the guiding metaphor for project organization. We believe that it is an especially powerful metaphor when it comes to multi-agent application development. And in the spirit of this guiding metaphor we believe that the organizational structure and the teams notion of the guiding metaphor is subject to change, adaption, self-organization and emergence. Thus the power of the metaphor will improve during the development process.

The principle behind the usage of guiding metaphors can add to the socio-organizational processes in the development team. Thus, the project managers have a powerful concept tool⁵ that enables guidance on an abstract level.

With the organization of the development team as multi-agent system we have achieved agent-oriented software engineering (AOSE) in two ways. In the original meaning of the term *AOSE* the software system is the objective. In our approach also the development team is oriented (guided) by the multi-agent system metaphor.

Acknowledgements I thank my colleagues Till Döriges, Michael Duvigneau, Michael Köhler, Daniel Moldt, Christine Reese, Heiko Rölke and Matthias Wester-Ebbinghaus for their participation in our AOSE projects as well as for the fruitful discussions regarding the *multi-agent system of developers* metaphor.

⁵ A tool or concept to guide and organize (or even transmit) one's thoughts. The artificial German term *Denkzeug* [8], a mix of *denken* (to think) and *Werkzeug* (tool), fits better.

References

1. Lawrence Cabac, Daniel Moldt, and Heiko Rölke. A proposal for structuring Petri net-based agent interaction protocols. In Wil van der Aalst and E. Best, editors, *24th International Conference on Application and Theory of Petri Nets, Eindhoven, Netherlands, June 2003*, volume 2679 of *Lecture Notes in Computer Science*, pages 102–120. Springer-Verlag, June 2003.
2. Scott DeLoach. Engineering organization-based multiagent systems. In *Software Engineering for Large-Scale Multi-Agent Systems (SELMAS)*, volume 3914 of *Lecture Notes in Computer Science*, pages 109–125. Springer Verlag, 2005.
3. Jacques Ferber. *Multi-agent Systems: An Introduction to Distributed Artificial Intelligence*. Addison-Wesley, Harlow [u.a.], 1999.
4. Michael Köhler, Daniel Moldt, and Heiko Rölke. Modelling the structure and behaviour of Petri net agents. In J.M. Colom and M. Koutny, editors, *Proceedings of the 22nd Conference on Application and Theory of Petri Nets 2001*, volume 2075 of *Lecture Notes in Computer Science*, pages 224–241. Springer-Verlag, 2001.
5. Olaf Kummer, Frank Wienberg, and Michael Duvigneau. Renew – The Reference Net Workshop. <http://www.renew.de>, March 2007. Release 2.1.
6. Martin Lippert, Axel Schmolitzky, and Heinz Züllighoven. Metaphor design spaces. In *Extreme Programming and Agile Processes in Software Engineering*, Lecture Notes in Computer Science, pages 33 – 40, 2003.
7. Julian Mack. *Softwareentwicklung als Expedition: Entwicklung eines Leitbildes und einer Vorgehensweise für die professionelle Softwareentwicklung*. PhD thesis, Universität Hamburg, Fachbereich Informatik, 2001.
8. Daniel Moldt. Petrinetze als DENKZEUG. pages 51–70, Vogt-Kölln Str. 30, 22527 Hamburg, August 2005. Universität Hamburg, Fachbereich Informatik.
9. Lin Padgham and Michael Winikoff. Prometheus: A pragmatic methodology for engineering intelligent agents. In *Proceedings of the OOPSLA 2002 Workshop on Agent-Oriented Methodologies*, pages 97–108, 2002.
10. Charles J. Petrie, Sigrid Goldmann, and Andreas Raquet. Agent-based project management. In *Artificial Intelligence Today*, pages 339–363. 1999.
11. Heiko Rölke. *Modellierung von Agenten und Multiagentensystemen – Grundlagen und Anwendungen*, volume 2 of *Agent Technology – Theory and Applications*. Logos Verlag, Berlin, 2004.
12. Jan Sudeikat, Lars Braubach, Alexander Pokahr, and Winfried Lamersdorf. Evaluation of agent-oriented software methodologies - examination of the gap between modeling and platform. In *Agent-Oriented Software Engineering V, Fifth International Workshop AOSE 2004*, pages 126–141, 2004.
13. Rolf v. Lüde, Daniela Spresny, and Rüdiger Valk. Sozionik: Modellierung soziologischer Theorie. In Rolf v. Lüde, Daniel Moldt, and Rüdiger Valk, editors, *Sozionik: Modellierung soziologischer Theorie*, volume 2 of *Reihe: Wirtschaft – Arbeit – Technik*, chapter Rationalität und organisierte Anarchie oder: James Bond im Garbage Can, pages 9–45. Lit-Verlag, Münster - Hamburg - London, 2003.
14. WfMC. Workflow reference model. <http://www.wfmc.org/standards/model.htm>, 2005.
15. Kim Wikström and Alf Rehn. Playing the live jazz of project management. online, 2002. <http://www.reformingprojectmanagement.com/docs/playing-the-live-jazz-of-project-management.pdf>.
16. Michael Wooldridge, Nicholas Jennings, and David Kinny. The Gaia methodology for agent-oriented analysis and design. *The International Journal of Autonomous Agents and Multi-Agent Systems*, 3, 2000.

17. Michael Wooldridge and Nicholas R. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10(2):115–152, 1995.
18. Franco Zambonelli, Nicholas Jennings, and Michael Wooldridge. Developing multi-agent systems: The Gaia methodology. *ACM Transactions on Software Engineering and Methodology*, 12(3):317–370, 2003.
19. Heinz Züllighoven. *Object-Oriented Construction Handbook*. dpunkt Verlag/Copublication with Morgan-Kaufmann, October 2004. ISBN 3-89864-254-2.