AOSE and Organic Computing – How Can They Benefit from Each Other? Position Paper

Bernhard Bauer and Holger Kasinger

University of Augsburg, 86135 Augsburg, Germany
{bauer, kasinger}@informatik.uni-augsburg.de

Abstract. Organic computing is an upcoming research area with strong relationships to the ideas and concepts of agent-based systems. In this paper, we therefore will have a closer look at agent systems, organic computing systems (as well as autonomic computing systems) and state commonalities and divergences between them. We then propose a common view on these technologies and show, how they can benefit from each other with regard to software engineering.

1 Introduction

Over the past few years technical systems as airplanes, vehicles, telecommunication networks or manufacturing installations became more and more complex. This is not only a result of the continuing evolution in microelectronics but also of the immense embedding of huge hardware and software complexes into these systems. But the producer's painful experiences show that these systems already today are difficult to manage. Thus, with respect to the future evolution, new advanced management principles have to be developed. A feasible principle is an autonomic behavior of the systems which is addressed by two research directions, namely agent technology and organic/autonomic computing.

Agent technology is believed to be able to play a key role in this "revolution", e.g. by automating daily processes, enriching higher level communication or enabling intelligent service provision. An intelligent agent is "a computer system, situated in some environment that is capable of flexible autonomous actions in order to meet its design objectives" [1]. The real strength of agents is based on the community of a multi-agent system and the negotiation mechanisms and coordination facilities. A multi-agent system is "a dynamic federation of software agents that are coupled through shared environments, goals or plans and that cooperate and coordinate their actions" [2]. It is this ability to migrate, communicate, coordinate and cooperate that makes agents and multi-agent systems a worthwhile metaphor in computing and that makes them attractive when it comes to tackling some of the requirements in next-generation systems.

Another worthwhile metaphor is provided by organic computing (OC) systems [3] that can be seen as an extension to autonomic computing (AC) systems [4]. The latter – driven by IBM since 2001 – draw analogies from the human body, in particular from the autonomic nervous system where all reactions occur without explicit override by the human brain – so to say autonomous. By embedding this behavior into technical systems, the administrative complexity of next-generation systems can be left to the systems themselves. IBM refers to this autonomy as "self-management" that includes four so-called "self-x properties", namely self-configuration (configuration and reconfiguration according to policies), self-optimization (permanent improvement of performance and efficiency), self-healing (reactive and proactive detection, diagnostics and reparation of localized SW/HW-problems) and self-protection (defense of the system as a whole). Furthermore, AC systems are self-aware, context-sensitive, non-proprietary, anticipative and adaptive. OC systems instead draw analogies from the biological world and try to use perceptions about the functionality of living systems for the development and management of artificial and technical systems respectively. In addition to the properties of AC systems they are defined as being self-organizing (hence they do not necessarily have to be self-aware).

As OC systems basically have the same objectives and concepts as AC systems, we will mostly treat them as one single technology for the rest of the paper, which is organized as follows: In section 2 we present the concepts of agents as well as autonomic/organic computing and the existing software-engineering approaches for these technologies. Section 3 relates the technologies and presents a common view on them. Based on this view, in section 4 we present a development process, which helps to benefit AOSE and OC from each other before we conclude with open issues and an outlook for further research in section 5.

2 Concepts

In this section we give an overview on agent technology as well as on autonomic/organic computing and consider the associated methodologies.

2.1 Agents

Software agents are software components characterized by autonomy (to act on their own), reactiveness (to process external events), proactiveness (to reach goals), cooperation (to efficiently and effectively solve in common tasks), adaptation (to learn by experience) and mobility (migration to new places). For further details on agent technology see e.g. [5] or [6].

Often, agents are subdivided into three functional sections: The *agent body* wraps a software component (e.g. a database, a calendar or an external service) and controls it through the software API. Connected to external software, the agent acts as an application agent by transforming the application API into agent communication language (ACL) and vice versa. Messages of such ACLs are highly structured and must satisfy standardized communicative (speech) acts which define the type and the content of the messages (like FIPA-ACL [7] or KQML [8]). The order of exchanged messages is fixed in protocols according to the relation of agents or the intention of the communication.

The *agent head* is responsible for the agent's intelligence. It is connected to the agent body on one side and to the agent communicator on the other side. The agent head contains knowledge bases storing knowledge of certain types like facts, beliefs, goals or intentions, preferences, motivations and desires concerning the agent itself or associated ones. Further, it contains a world model as an abstraction of relevant states of the real world. It is updated by information from other agents or through real world interfaces, e.g. sensors. The agent head is able to evaluate incoming messages with respect to its goals, plans, tasks, preferences and to the world model.

The *agent communicator* converts logical agent addresses into physical addresses and delivers messages on behalf of the agent head through appropriate channels to the receivers. Furthermore, the communicator listens for incoming messages (e.g. by running an event loop) and forwards them to the agent head. The agent behavior should be benevolent, which means that an agent at least understands the interaction protocols and reacts accordingly.

2.2 Autonomic/Organic Computing

According to [9], AC systems are composed of four levels: On the lowest level *managed resources (MR)*, e.g. HW/SW-components as servers, databases or business applications, are located, together making up the complete IT infrastructure. So-called *touchpoints* on the next level provide a manageability interface – similiar to an API – for each MR by mapping standard sensor and effector interfaces on the sensor and effector mechanisms (e.g. commands, configuration files, events or log files) of a specific MR. The next level is composed of so-called *touchpoint autonomic managers (TAM)* directly collaborating with the MRs and managing them through their touchpoints.

An *autonomic manager (AM)* in general implements an intelligent control loop (closed feedback loop) called *MAPE loop*. The latter is composed of the components *monitor* (collects, aggregates, filters and reports MR's details), *analyze* (correlates and models complex situations), *plan* (constructs actions needed to achieve goals) and *execute* (controls execution of a plan). Additionally, a knowledge component provides the data used by the four components, including policies, historical logs and metrics. Together with one or more MRs, an AM represents an *autonomic element (AE)* (see Fig. 1). A TAM also provides a sensor and an effector to *orchestrating autonomic managers (OAM)* residing on top level. The latter achieve system-wide autonomic behavior, as TAMs are only able to achieve autonomic behavior for their controlled MRs.

As (strong) self-organizing systems (like OC systems) are defined as systems "that change their organization without any explicit – internal or external – central control" [10], there can be no single instance within an OC system that is aware of all system's components or states. From our point of view, system-wide autonomic behavior in OC systems is in contrast to AC systems therefore an emergent behavior of the system's component interactions and not the achievement of a single OAM. This issue has significant impact on software engineering but not on the concepts mentioned above which are also used in OC systems.

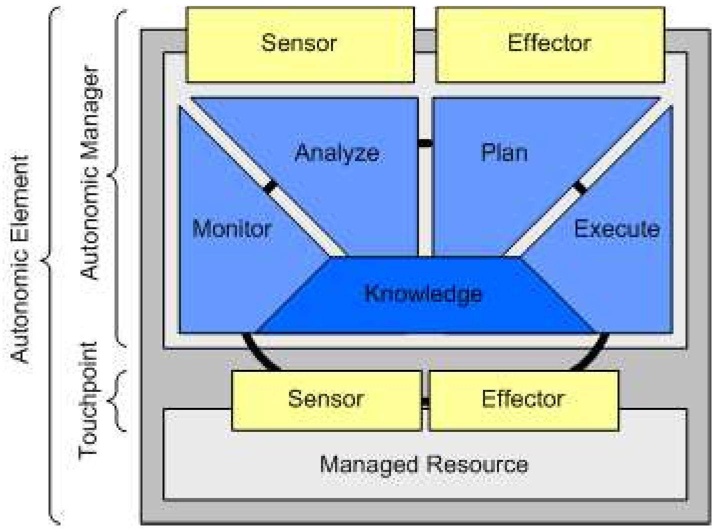

Fig. 1. Logical structure of an autonomic element

2.3 Software Engineering Methodologies

Agent-oriented Software Engineering Methodologies. A considerable number of AOSE methodologies and tools are available today (see our work in [11] or [12] for a more detailed survey), and the agent community is facing the problem of identifying a common vocabulary to support them.

The knowledge engineering community inspired most early approaches supporting the SE of agent-based systems: The CommonKADS [13] was developed to support knowledge engineers in modeling expert knowledge and developing design specifications in textual or diagrammatic form. To consider agent-specific aspects CoMoMAS [14] and MAS-CommonKADS [15] were developed.

Gaia [16] is a methodology designed to deal with coarse-grained computational systems, having static organization structures and agents with static abilities and services. ROADMAP [17] extends Gaia by adding elements to deal with the requirements analysis in more detail by using use cases, handling open system environments and specification of interactions. SODA [18] addresses aspects like open systems or self-interested agents, based on the analysis and design of agent societies (exhibiting global (emergent) behavior not deducible from the behavior of the individual agents) and agent environments.

One of the first methodologies for the development of BDI agents based on OO technologies was presented in [13] and [19]. The methodology distinguishes between the external viewpoint – the system is decomposed into agents, modeled as complex objects characterized by their purpose, their responsibilities, the services they perform, the information they require and maintain, and their external interactions – and the internal viewpoint – the elements required by a particular agent architecture must be modeled for each agent, i.e. an agent's beliefs, goals and plans.

MESSAGE [20] is a methodology that extends UML by agent-related concepts (inspired e.g. by Gaia). TROPOS [21] uses UML for the development of

BDI agents. Prometheus [22] it is an iterative methodology covering the complete SE process and aiming at the development of intelligent agents using goals, beliefs, plans and events, resulting in a specification which can for example be implemented with JACK [23]. MaSE [24] has been developed to support the complete software development life cycle. PASSI [25] is an agent-oriented iterative requirement-to-code methodology for the design of multi-agent systems mainly driven from experiments in robotics.

Autonomic / Organic Computing Methodologies. Continuous and consistent SE methodologies for AC/OC systems are more or less not available now, since most of the research activities are in the area of algorithms, middleware, hardware concepts as well as application areas. Nevertheless, the objective in particular of OC has to be on the control of such systems by engineering methods. Traditional SE methods are strictly hierarchic and follow a top-down approach by transforming the entire specification into detailed modules. For emergent and self-organizing systems this strict approach is abandoned. System states have to be reached that are not imagined beforehand. This is a fundamental contradiction between a top-down-control and a creative bottom-up-behavior.

Today it is not clear, how to combine these opposite tendencies. However, there are some approaches based on constraint propagation, the use of assertions and so-called observer/controller architectures. Assertions can be used for monitoring values of special variables. Yet, the limitation of emergent behavior of OC systems will be crucial for their technical application. Thus, constraints play an important role to the limitation of learning in self-organizing systems as constraint violations result in warnings.

3 Relating Agents and Organic Computing

Based on the presented concepts we try to relate agents and OC in this section and propose a common view on these technologies.

Both technologies incorporate managed objects, either software components wrapped in the agent body or managed resources on the OC-side. In addition, both technologies have an institution for intelligent and autonomic behavior, namely the agent head and the autonomic manager respectively. Moreover an agent communicator is in a sense comparable to a touchpoint in OC.

Thus, in order to bring the technologies together, we view an autonomic element from now on as the combination of agents and organic computing with the following properties: Having a BDI mental model about other autonomic elements; using a MAPE loop similar to the control loop of agents, with monitoring and analyzing the environment and messages, consulting the knowledge base, planning and execution; managing the internal behavior automatically, like OC does it, without interaction with the environment; interacts with its environment, not only via direct messages but also via e.g. stigmergy – therefore the environment has to be modeled explicitly, like for swarm intelligence, or ant algorithms. Moreover, an autonomic element community consists of cooperating

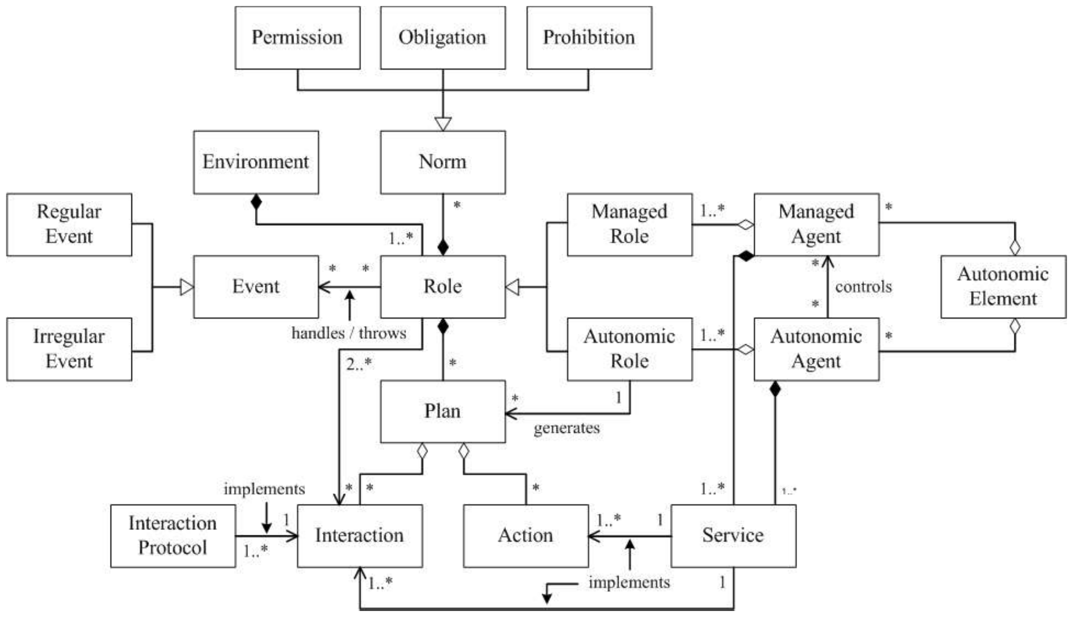

Fig. 2. The meta model for organic computing systems

autonomic elements explicitly communicating based on speech acts and interaction protocols or implicitly via the environment. Additionally these cooperating elements have to satisfy global system rules such that no unintentional behavior of the system takes place.

Having this in mind we propose a meta model for both a MASs with OC properties and OC systems as MASs (see Fig. 2). Therefore, we have combined different proved concepts of existing agent architectures and their SE methodologies as well as AC/OC concepts.

Similar to many existing agent methodologies a *role* is the central architectural concept. The complete set of roles builds up the *environment*. The life cycle of a role is traditionally: A role or rather the enacting agent recognizes a situation, makes a decision based upon it and executes appropriate activities. The recognition of situations is based on *events*. *Regular events* are familiar to a role, e.g. by design or by adaption, whereas *irregular events* are new to a role, e.g. by failure appearance. *Norms* regulate the behavior of a role and are a generalization of either a *permission*, an *obligation* or a *prohibition* and consist of a goal and activation as well as deactivation events. The decision making is based on *plans* that fire certain events at the end (as notification of being in a certain state) which may correspond to a norm's goal or event respectively. A plan consists of actions (internal activities of a role) and interactions (external activities between different roles) and are chosen accordingly to a goal of an activated norm. *Interactions* are implemented by specific *interaction protocols*. The relation between interactions and interaction protocols is the same as between interfaces and their implementations. Thus, according to diverse requirements, an interaction may be implemented by different kinds of protocols for direct (e.g. by auctions) or indirect (e.g. by stigmergy) communication. Interactions and actions are both implemented by *services* with different visibilities.

Roles are logically divided into *managed roles (MR)* and *autonomic roles (AR)* (similar to the AC concepts). MRs are responsible for the business logic of a system and reside on versatile resources. They are controlled by one or more ARs that are responsible for the self-management of a system. ARs do not necessarily have to be located at the same resource as its MRs. In contrast to MRs the ARs are able to generate new plans based on the received data of their MRs. The latter do not have to generate new plans as they communicate the occurrence of irregular events to their monitoring ARs and mostly are not in possession of further required information. Both roles are taken over dynamically by *managed agents* and *autonomic agents* respectively. *Autonomic elements* contain one or more autonomic agents and managed agents at the same time.

4 Software Engineering for OC and AO Systems

As a result of the common view presented in the previous section, we propose a development process in this section which can be used for both AOSE and OC. The process is based on the Model Driven Architecture (MDA), a framework for software development driven by the Object Management Group (OMG). It comprises a *Computation Independent Model (CIM)* (model of a system that abstracts from any computation), a *Platform Independent Model (PIM)* (model of a system that abstracts from any specific platform) and a *Platform Specific Model (PSM)* (model of a system that is tailored to one or more specific implementation platforms). For a more detailed description see [26].

The process consists of 19 activities and encompasses an analysis phase (activities 1-5) and a design phase (activities 6-19). Each activity results in a specific model either in the CIM (analysis phase) or the PIM (design phase) (see Fig. 3). An implementation phase is not considered yet, but can be added smoothly in the future. Notice, the process does not prescribe a process model.

The analysis phase consists of the activities (1) 'Definition of the business context', (2) 'Definition of business processes being supported', (3) 'Characteri-

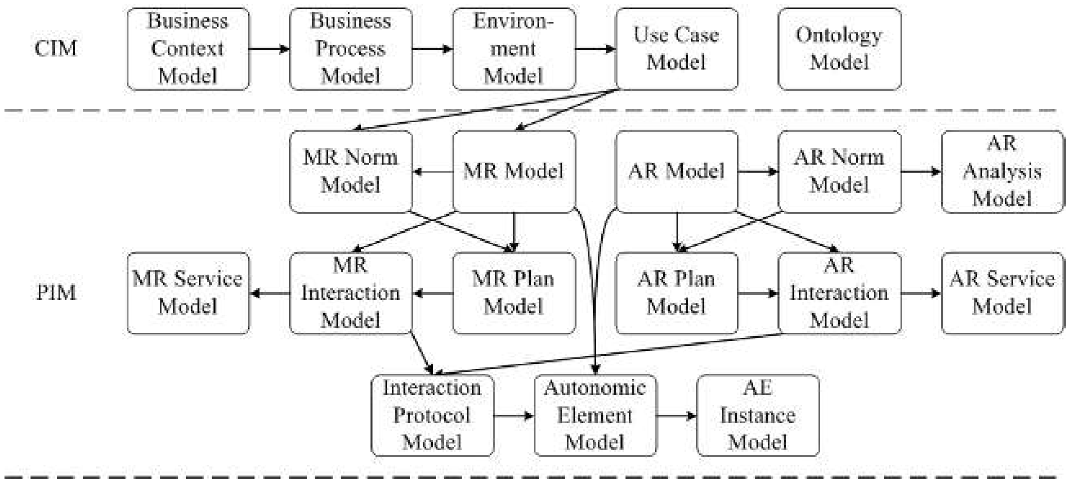

Fig. 3. MDA-based development process models for agent and OC systems

zation of the environment', (4) 'Assembly of potential use cases' and (5) 'Assembly of common vocabulary'. The resulting models are: *Business Context Model:* As a result of (1) the business context of the future system is modeled by an UML activity diagram. This model only considers higher level correlations and abstracts from concrete business processes; *Business Process Model:* As a result of (2) the business processes supported by the later system are modeled by an UML activity diagram; *Environment Model:* As a result of (3) important environment objects of all types are modeled by an UML class diagram; *Use Case Model:* As a result of (4) the system application is declared abstractly in an UML use case diagram. The model is supported by an UML sequence diagram to explain the message flow of the system clearly; *Ontology Model:* As a result of (5) all important knowledge blocks and common vocabulary are categorized in an UML class model.

The design phase consists of the activities (6) 'Identification of MRs', (7) 'Specification of norms for MRs', (8) 'Development of plans for MRs', (9) 'Derivation of interactions between MRs', (10) 'Specification of services of MRs', (11) 'Identification of ARs', (12) 'Specification of norms for ARs', (13) 'Development of an analysis for ARs', (14) 'Development of plans for ARs', (15) 'Derivation of interactions between ARs', (16) 'Specification of services of ARs', (17) 'Development of interaction protocols', (18) 'Identification of AE' and (19) 'Deployment of AE'. The resulting models of this phase are: *Managed Role Model:* As a result of (6) the MRs are identified and modeled similar to a class in an UML composition structure diagram; *MR Norm Model:* As a result of (7) the norms (containing goals, activation and deactivation events) of MRs are specified and modeled similar to a class in an UML class model; *MR Plan Model:* As a result of (8) the plans (containing input and output parameters, actions and interactions, and events) of MRs are modeled in an UML activity diagram; *MR Interaction Model:* As a result of (9) the interactions between MRs are derived and the exchanged objects (information carriers) are modeled in an UML sequence diagram; *MR Service Model:* As a result of (10) the signature of provided services (containing visibility, input and output parameters) of a MR are specified and modeled similar to a class in a UML class diagram again.

The results of activities (11), (14), (15) and (16), the *Autonomic Role Model*, the *AR Plan Model*, the *AR Interaction Model* and the *AR Service Model* are similar to the corresponding MR models. Further resulting models are: *AR Norm Model:* As a result of (12) and parallel to (11) the norms for ARs are specified according to desired self-x properties. Notice, a norm of an AR realizes a part of a certain self-x property of a system; *AR Analysis Model:* As a result of (13) the monitoring and analysis of events and data by an AR is modeled in an UML activity diagram as a premise for the right choosing of a plan; *Interaction Protocol Model:* As a result of (17) the interaction protocols for the (direct/indirect) interactions between all types of roles are specified in an UML sequence diagram; *Autonomic Element Model:* As a result of (18) MRs and ARs are combined into AEs that are modeled similar to a class in an UML composition structure diagram again; *Autonomic Element Instance Model:* As a result of (19) the deployment of the AEs onto resources is defined similar to an UML deployment diagram. Note, activities (11)-(16) are logically separated and represent the way of self-x property development.

5 Conclusion, Open Issues and Outlook

As described in this paper, agent systems and OC systems have many conceptual commonalities which result in benefits for both AOSE and OC: On the one side open agent systems can be developed that exhibit OC properties, on the other side OC can make use of the experiences in AOSE and adopt existing concepts.

The open issues in this context for us are: Where are the borders between an autonomic element, an agent or multi-agent system? How to deal with the emergent behavior of the system such that no unintentional behavior of the system occurs? How to define emergency strategies if the system is out of control, with regard to the emergent behavior? Should we have an hierarchical composition, like grouping autonomic elements to autonomic communities, view these communities as autonomic elements and grouping them to autonomic communities, etc.? How to model self-x properties in the local as well as in the global sense and how does the local behaviors result in a global behavior? How to integrate interaction (communication protocols) in such OC systems? What is the appropriate middleware/platform for OC systems (web services, grid computing middleware, agent platforms, . . .)?

In this context our vision is to combine different but related technologies, like grid computing, semantic web, (semantic) web services and web service composition, P2P, business processes and OC with its self-x properties, since these technologies deal with similar aspects (service provisioning, service access, service and data distribution, service and resource work loading, processes in distributed environments) and use similar standards.

References

1. Jennings, N.R., Sycara, K., Wooldridge, M.J.: A Roadmap of Agent Research and Development. Autonomous Agents and Multi-Agent Systems, 1(1) (1998) 7–38
2. Huhns, M.N.: Multiagent Systems. Tutorial at the European Agent Systems Summer School (EASSS 99) (1999)
3. Organic Computing website: http://www.organic-computing.org
4. Horn, P.: Autonomic Computing: IBM's Perspective on the State of Information Technology. http://www.research.ibm.com/autonomic/manifesto/ autonomic_computing.pdf (2001)
5. Müller, J. P.: The design of intelligent agents. A layered approach. Lecture Notes of Artificial Intelligence, Volume 1177. Springer-Verlag (1996)
6. Huhns, M.N., Singh, M.P.: Agents and Multiagent Systems: Themes, Approaches, and Challenges. Readings in Agents, Morgan-Kaufmann (1998), 1–24
7. FIPA: http://www.fipa.org
8. Finin, T., Fritzson, R., McKay, D., McEntire, R..: KQML as an Agent Communication Language. Proceedings of the Third International Conference on Information and Knowledge Management (CIKM'94). ACM Press (1994) 456–463

9. IBM: An architectural blueprint for autonomic computing. http://www-03.ibm.com/autonomic/pdfs/ACBP2_2004-10-04.pdf (2004)
10. Di Marzo Serugendo, G., Gleizes, M.-P., Karageorgos, A.: Self-Organisation in Multi-Agent Systems. AgentLink News (16) (2004) 23–24
11. Bauer, B., Müller, J.P.: Methodologies and Modeling Languages. In: Luck M., Ashri R. D'Inverno M. (eds.): Agent-Based Software Development. Artech House Publishers, Boston, London (2004)
12. Iglesias, C.A., Garijo, M., Centeno-González, J.: A Survey of Agent-Oriented Methodologies. In Proceedings of Fifth International Workshop on Agent Theories, Architectures, and Languages (ATAL 98) (1998) 317–330
13. Kinny, D., Georgeff, M., Rao, A.: A Methodology and Modeling Technique for Systems of BDI Agents. 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW 96), LNAI 1038, Springer (1996) 56–71
14. Glaser, N.: Contribution to Knowledge Modelling in a Multi-Agent Framework (the Co-MoMAS Approach). PhD thesis, L'Universtité Henri Poincaré, Nancy I, France (1996)
15. Iglesias, C.A., Garijo, M., Centeno-González, J., Velasco, J.R.: A methodological proposal for multiagent systems development extending CommonKADS. In Proceedings of 10th Knowledge Acquisition for Knowledge-Based Systems Workshop (KAW 96), Banoe, Canada (1996)
16. Wooldridge, M., Jennings, N.R., Kinny, D.: The Gaia Methodology for Agent-Oriented Analysis and Design. Journal of Autonomous Agents and Multi-Agent Systems, 3 (3) (2000) 285–312
17. Juan, Th., Pearce, A., Sterling, L.: ROADMAP: Extending the Gaia Methodology for Complex Open Systems. In Proc. of the First Int. Joint Conf. on Autonomous Agents and Multiagent Aystems (AAMAS 02), ACM Press (2002) 3–10
18. Omicini, A.: SODA: Societies and Infrastructures in the Analysis and Design Of Agent-based Systems. In Proceedings of Agent Oriented Software Engineering (AOSE 00), LNCS 1957, Springer (2000) 185–193
19. Kinny, D., Georgeff, M: Modelling and Design of Multi-Agent Systems. Intelligent Agents III: Proceedings of Third International Workshop on Agent Theories, Architectures, and Languages (ATAL 96), LNAI 1193, Springer (1996)
20. Caire, G., Coulier, W., Garijo, F., Gomez, J., Pavon, J., Massonet, P., Leal, F., Chainho, P., Kearney, P., Stark, J., Evans, R.: Agent Oriented Analysis using MESSAGE/UML. In Proceedings of the Second International Workshop on Agent-Oriented Software Engineering II (AOSE 01), Springer (2002) 119–135
21. Bresciani, P., Giorgini, P., Giunchiglia, F., Mylopoulos, J., Perini, A.: Tropos: An Agent-Oriented Software Development Methodology. Journal of Autonomous Agent and Multi-Agent Systems, 8 (3) (2004) 203–236
22. Padgham, L., Winikoff, M.: Developing Intelligent Agent Systems: A Practical Guide. John Wiley & Sons (2004)
23. Busetta, P., Rönnquist, R., Hodgson, A., Lucas, A.: JACK Intelligent Agents - Components for Intelligent Agents in Java. AgentLink News (2) (1999) 2–5.
24. DeLoach, S.A., Wood, M.F., Sparkman, C.H.: Multiagent Systems Engineering. The International Journal of Software Engineering and Knowledge Engineering, 11 (3) (2001) 231–258
25. Cossentino, M., Potts, C.: A CASE tool supported methodology for the design of multi-agent systems.. In Proceedings of the 2002 International Conference on Software Engineering Research and Practice (SERP'02), Las Vegas, USA (2002)
26. Model Driven Architecture website: http://www.omg.org/mda