Combining Multi-Agent-System Methodologies for Organic Computing Systems

Holger Kasinger, Bernhard Bauer
Department of Computer Science, University of Augsburg, 86135 Augsburg, Germany
{ kasinger | bauer }@informatik.uni-augsburg.de

Abstract

As the complexity of computing systems steadily increases, self-managing systems – as Autonomic Computing Systems (ACS) proposed by IBM – are an adequate approach to minimize human effort spent on system administration. While ACS primarily are limited to servers and networks, Organic Computing Systems (OCS) are intended for widespread applications in various domains. In addition to the self-x properties of ACS, OCS are adaptive and context-aware. Thus agent technology is particularly suitable for an implementation of OCS. Nevertheless a key prerequisite for a successful, industrial application remains in a systematical engineering of OCS according to accepted standards. However no single agent methodology is applicable to OCS as self-x properties are not supported directly. Therefore we have combined different proved agent concepts into a system architecture for OCS and developed an adequate, model-driven software engineering methodology based on the Unified Modeling Language (UML) and the Model Driven Architecture (MDA).

1. Introduction

Considering the last 40 years, faster processors, wider memory capacities and higher data transfer rates resulted in more and more multi-functional computing systems. But in the same way as the increasing functionality and interoperability offer great benefits to users, the accruing complexity exhibits enormous challenges to administrators. Today a database expert has to tune hundreds of parameters for optimizing a single company's database. Multiple databases from different vendors working together additionally multiply the administrational effort. Consequently there will be not enough skilled I/T people to keep the world's future computing systems running (see [1]).

Thus the objective of Organic Computing (OC) [2] – as well as Autonomic Computing (AC) [3] – is to confer the administration of future computing systems on the systems themselves and thereby relieve human administrators. While AC addresses this by drawing analogies from the human nervous system, OC draws analogies from complex biological systems, e.g. ant colonies. An OCS is defined as self-organizing system being self-configuring, self-optimizing, self-healing, self-protecting, adaptive and context-aware. For an implementation agent technology is particularly suitable, as agents are autonomous, reactive, proactive and possess social capabilities (see [4]).

However an important key prerequisite to a successful, industrial application of OCS resides in their systematical software engineering. Without an appropriate approach it is almost impossible to cope with the complexity of OCS, especially with their self-x properties. But taking existing agent methodologies into consideration, no approach is applicable, as there is no continuous and consistent development process for self-x properties provided. Thus we have combined appropriate and proved concepts of different multi-agent-system (MAS) methodologies into a system architecture for OCS and designed an adequate development process – with explicit support of self-x properties – to obtain a complete software engineering methodology for OCS.

Section 2 explains the proposed OCS architecture, whereas section 3 presents the related development process. In section 4 we give a brief example of adding a self-optimizing property to a system according to the proposed process. Section 5 concludes with some remarks and gives an outline of future work.

2. The OCS architecture

Figure 1 depicts the metamodel of the proposed OCS architecture. Similar to many existing agent methodologies (for a detailed overview see [5]) a **role**

is the central architectural concept. The set of roles forms the complete **environment** whereas the lifecycle of a single role (and accordingly the enacting agent) is traditionally: It recognizes a certain situation, makes a decision and possibly takes advisable measures.

The recognition of situations (context-awareness) is based on **events**. **Regular events** are already known to roles, e.g. by design or adaptation, whereas **irregular events** (similar to ADELFE's concept of NCS [6]) are unknown and may emerge in failure situations (self-healing) or by intrusion detections (self-protection). Events activate or deactivate **norms** which in turn regulate a role's behavior. A norm is a generalization of an **obligation,** a **permission** and a **prohibition** (similar to the NoA Agent Architecture [7])**,** consisting of a goal, activation and deactivation events.

The decision making is based on **plans** that at the end fire regular events (as state-notification) 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 is chosen according to a goal of an activated norm. Interactions are implemented by one or more **interaction protocol(s)**. The relation between interactions and interaction protocols is similar to interfaces and their implementations. Thus 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 (public / protected / private).

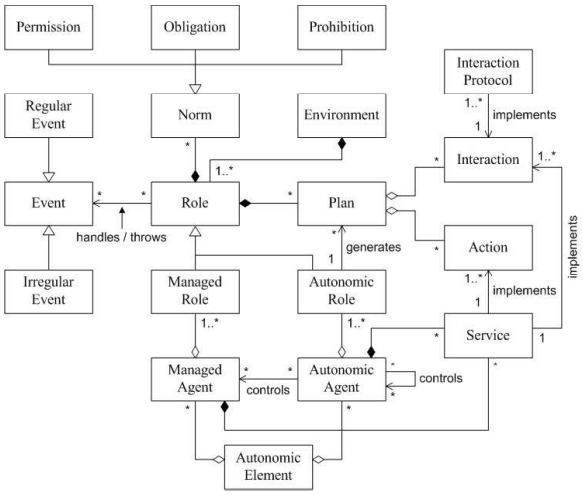

Figure 1. OCS architecture meta model

Roles are divided into **Managed Roles** (MR) and **Autonomic Roles** (AR) (similar to the concepts by the AC reference architecture [3]). The system's business logic is spread over the MRs which may reside on versatile resources (applications, sensors, handhelds, etc.) and are supervised by one or more ARs. The latter are responsible for the self-management of a system and do not necessarily have to be located on the same resource as their MRs. Only ARs are able to generate new plans based on the received data of their MRs (self-configuration, self-optimization, adaptivity). Both roles are dynamically taken over by **Managed** or **Autonomic Agents** respectively. An **Autonomic Elements** in turn contains one ore more Autonomic Agent(s) and Managed Agent(s) at the same time.

3. The OCS development process

The proposed development process is based on the UML and the MDA [8]. In the context of MDA the system's functionality is separated from the technology specific implementation. This is addressed by different levels of abstraction (Computational Independent Model (CIM), Platform Independent Model (PIM) and Platform Specific Model (PSM)), constructive models over all phases of the process and transformations between the models on different levels. The models are not only used for an abstract description of the later system but for the (semi-)automatic generation of later models and finally components of the system.

The development process consists of 19 specified activities and encompasses an analysis and design phase whereas an implementation phase is not considered yet, but can be added easily in future. Each activity results in a specific model (see Figure 2). Notice, the process does not prescribe a process model.

3.1 The analysis phase

The analysis phase encompasses five activities: (1) Definition of the business context, (2) Definition of supported business processes, (3) Characterization of the environment, (4) Assembly of potential use cases and (5) Assembly of common vocabulary.

As result of (1) the business context of the future system is modeled in the **Business Context Model** by an UML activity diagram. This model only considers relationships between business processes. As result of (2) only the supported business processes are modeled by the **Business Process Model** by an UML activity diagram again. Each activity of a business process has to be assigned to an actor (e.g. worker, machine or database) of the company (represented as partitions). As result of (3) affected environment objects of arbitrary types are modeled by the **Environment Model** by an UML class diagram. Thus the system environment is specified in detail by e.g. attributes. As

result of (4) the usage of the system is abstractly declared by the **Use Case Model** by an UML use case diagram. This model is supported by an UML sequence diagram to explain the system's message flow clearly. As result of (5) the common vocabulary is categorized by the **Ontology Model** by an UML class model.

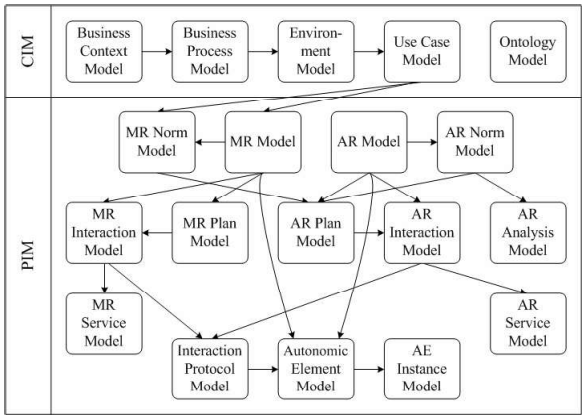

Figure 2. OCS development process models

3.2 The design phase

The design phase encompasses 14 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 Autonomic Elements and (19) Deployment of Autonomic Elements.

As result of (6) MRs are identified by the **Managed Role Model**. As UML defines no notation element for roles we have defined a new element similar to a class in an UML composition structure diagram. As result of (7) norms of MRs are specified by the **MR Norm Model**. Again there is no existing UML notation which gets us to the definition of a notation similar to a class in an UML class model. However there are no attributes or methods but goals, activation and deactivation events. As result of (8) plans of MRs are modeled by the **MR Plan Model** by an UML activity diagram. Plans contain input and output parameters, actions and interactions, and events as well. Additionally they are separated into partitions corresponding to the MRs. As result of (9) the interactions between MRs are derived and modeled by the **MR Interaction Model** by an UML sequence diagram. In this model only the exchanged objects (information carriers) are embodied, not a concrete protocol. As result of (10) signatures of provided services of a MR are specified by the **MR Service Model**. They are modeled similar to a class in a UML class diagram again. Instead of attributes and methods it contains visibility, input and output parameters.

As result of (11) ARs are identified by the **Autonomic Role Model**. The notation of these roles is the same as in the Managed Role Model. This activity is to be executed in parallel with activity (12). As result of (12) norms for ARs are specified according to desired self-x properties by the **AR Norm Mod**el. Notice, a norm of an AR symbolizes a part of a certain self-x property of a system. As result of (13) monitoring and analysis of events and data by an AR is modeled by the **AR Analysis Model** by an UML activity diagram. This is an important premise for the appropriate choosing of a plan to satisfy an AR's norm. As result of (14), (15) and (16) plans, interactions and services of an AR are modeled by the **AR Plan Model**, **AR Interaction Model** and **AR Service Model**, similar to the respective MR models.

As result of (17) the interaction protocols for the interactions between arbitrary roles are specified by the **Interaction Protocol Model** by an UML sequence diagram. Depending on requirements the kind of interaction can be direct or indirect. As result of (18) MRs and ARs are combined to Autonomic Elements by the **Autonomic Element Model**. The notation of an Autonomic Element is similar to a class in an UML composition structure diagram. As result of (19) the deployment of the Autonomic Elements on resources is defined by the **Autonomic Element Instance Model** which is similar to an UML deployment diagram.

In addition we have defined a set of transformations between various models (depicted as arrows) which will not be described in this paper. Thereby some of the process activities were completely automated.

4. Case study: Manufacturing control

In order to evaluate the development process we have redesigned an existing MAS and added different self-x properties according to the proposed process. We only will illustrate activities (12), (13) and (14) by a short example in what way an application can be extended with self-optimization.

4.1 A MAS production planning system

Today's manufacturing industry is facing a major shift from a supplier's to a customer's market. Thus

the requirements on the manufacturing process itself increase permanently: Instant demand satisfaction, higher product variety and cost reducing are just some of them. Thus Valckenaers et al. [9] developed a multi-agent coordination and control system based on stigmergy (coordination mechanism based on indirect communication, e.g. used by food foraging ants).

The system consists of three different types of agents: **Resource agents**, each assigned to a machine or switch in the manufacturing plant, **order agents**, each routing a product instance through the plant while reserving processing time on appropriate machines, and **product agents**, each containing the construction plan for a specific product. In addition resource and order agents use intelligent ant agents propagating and collecting information throughout the plant.

The coordination mechanism works as follows: Resource agents assigned to machines permanently send ant agents opposite to the production line through the plant. These ants deposit the processing capabilities of their sending resource agent/machine as so called pheromones (information carriers for indirect communication, used by ants) on every switch they cross. Order agents also create ant agents in a certain interval and send them down the production line. Based on the deposited pheromones on a switch the ants decide to which machine they will travel next. When they arrive at this machine, they request an offer for a specific process step (e.g. duration, earliest start) and as soon as they receive the offer they continue traveling. If the end of the plant is reached the ants return to their order agents and report their chosen route. An order agent decides which of its ants has found the best way and routes its product instance accordingly.

4.2 Adding Self-Optimization

The existing system already copes with machine breaks and short-termed changes. Nevertheless the system can be improved by certain self-x properties, e.g. self-optimization.

Consider a single resource agent simultaneously responding to a multitude of offer requests of present ant agents, the response time can exceed in a way that the performance of the complete production system may slow down. In order to prevent this situation we added an **AR - resource** to the **MR - resource** which measures the response time and if needed informs the order agents to reduce their ant agent generation interval to minimize the amount of concurrent offer requests. For the measurement the norm **offer request response time optimization** (see Figure 3) has been specified for the **AR - resource** as result of (12). This <<obligation>> forces the AR to achieve the goal **offer request response time in bounds**, is activated by the event **offer request response time out of bounds** and deactivated by the event homonymous to the goal.

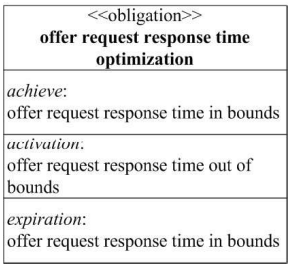

Figure 3. Norm for self-optimization

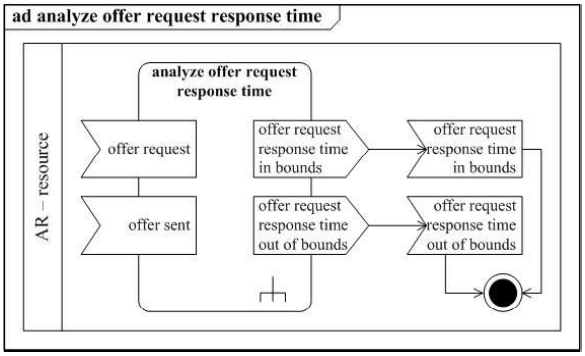

Figure 4. Monitoring and analysis of events

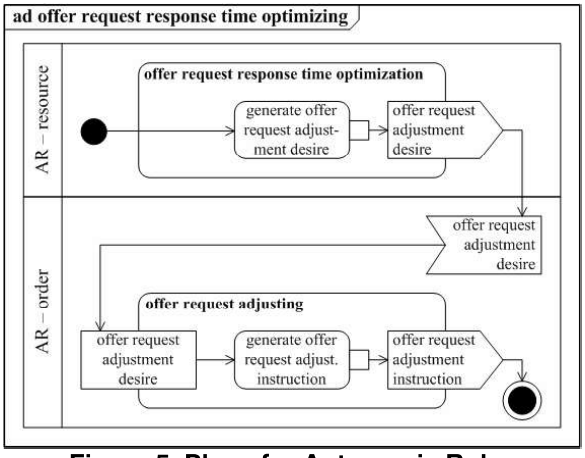

Figure 5. Plans for Autonomic Roles

To provide the **AR - resource** with the ability to analyze if this norm is activated or not, the monitoring and analysis (see Figure 4) of specified events is modeled as result of (13). The **AR – resource** listens to **offer request events** and **offer sent events** fired by the **MR - resource**. Within the analysis action the

response time is determined according to given rules (not modeled in this figure) and – depending on the result – a corresponding event is fired. By catching this event the **AR - resource** can determine whether or not the above norm is activated or deactivated.

If the analysis marks the norm as activated, the **AR - resource** has to choose a plan for informing the order agents to reduce their generation interval. Such a plan is modeled within an action (see Figure 5) as result of (14). The **AR - resource** generates an **offer request adjustment desire** and sends it to every order agent or rather to every **AR - order** (similar to a broadcast), possibly propagated by ant agents again. Such an AR in turn generates an **offer request adjustment instruction** for its controlled **MR - order** which on its part slows down the generation interval (not modeled in this figure).

5. Conclusion

The combination of different proved agent concepts led to an appropriate system architecture for OCS which may be implemented by agent technology. The corresponding development process incorporates the ability to develop both a (multi-agent-)system based on this architecture and additionally certain self-x properties for the system's self-management. In contrast to other role-based agent methodologies, e.g. Gaia [10], ROADMAP [11] or TROPOS [12], this separation clearly yields to two advantages: On the one hand the development process is applicable for a continuous and consistent engineering of ordinary MAS within the framework of the MDA, on the other hand self-x properties can be developed independently from the business logic of a system. Moreover the use of MDA as framework establishes a basis for various implementations, as the PIM only depends on platform independent concepts. By construction of different model transformations between the PIM and PSM (implementation phase) further technologies can be applied.

Nevertheless much future work has to be done: In order to prove the proposed system architecture we have to implement an OCS according to the development process. This will lead us to detailed experiences and possibly to an improvement of the process as a whole. Thereby implementation, project and quality management phases will be added to obtain a complete methodology. Furthermore a tool support has to be developed as this is the only way quality OCS can be engineered efficiently in time.

6. References

[1] IBM Research, *Autonomic Computing Manifesto*, http://www.research.ibm.com/autonomic/manifesto/autonomic_computing.pdf

[2] Gesellschaft für Informatik: *VDE/ITG/GI-Positionspapier zum Organic Computing*, 2003 (german), http://www.gi-ev.de/download/VDE-ITG-GIPositionspapier%20Organic%20Computing.pdf.

[3] IBM Research, An architectural blueprint for autonomic computing, 2004, http://www-306.ibm.com/autonomic/pdfs/ACBP2 2004-10-04.pdf

[4] M. Wooldridge and P. Ciancarini, "Agent-Oriented Software Engineering: The State of the Art", *Agent-Oriented Software Engineering*, P. Ciancarini and M. Wooldridge (eds), LNAI 1957, 2000, pp. 1–28.

[5] B. Bauer and J.P. Müller, "Methodologies and Modelling Languages", *Agent-Based Software Development*, M. Luck, R. Ashri and M. d'Inverno (eds), Artech House, 2004, pp. 77–131.

[6] C. Bernon, M.-P. Gleizes, G. Picard and P. Glize, "The ADELFE methodology for an intranet system design", presented at *AOIS2002*, Toronto, 2002.

[7] M.J. Kollingbaum and T.J. Norman, "Supervised Interaction - Creating a Web of Trust for Contracting Agents in Electronic Environments", Proceedings of the *AAMAS 2002*, ACM Press, New York, 2002, pp. 272–279.

[8] Object Management Group, *Model Driven Architecture: A Technical Perspective*, ormsc/01-07-01, 2001.

[9] P. Valckenaers, H. Van Brussel, M. Kollingbaum and O. Bochmann, "Multiagent coordination and control using stigmergy applied to manufacturing control", *Multi-Agent Systems and Applications*, LNAI 2086, 2001, pp. 317–334.

[10] M. Wooldridge, N. Jennings and D. Kinny, "The Gaia Methodology for Agent-Oriented Analysis and Design", *Journal of Autonomous Agents and Multi-Agent Systems*, 3(3), 2000, pp. 285–312.

[11] T. Juan, A. Pierce, and L. Sterling, Roadmap: "Extending the Gaia methodology for complex open systems", Proceedings of the *1st ACM Joint Conference on Autonomous Agents and Multi-Agent Systems*, Bologna, Italy, 2002, pp. 3–10.

[12] J. Castro, M. Kolp, and J. Mylopoulos, "Towards requirements-driven information systems engineering: the TROPOS project", *Information Systems*, 27, 2002, pp. 365-389.