Lecture Notes in Computer Science 3950

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

Jörg P. Müller Franco Zambonelli (Eds.)

Agent-Oriented Software Engineering VI

6th International Workshop, AOSE 2005
Utrecht, The Netherlands, July 25, 2005
Revised and Invited Papers

Springer

Volume Editors

Jörg P. Müller
Technische Universität Clausthal
Institut für Informatik
Julius-Albert-Str. 4, 38678 Clausthal-Zellerfeld, Germany
E-mail: joerg.p.mueller.ext@siemens.com

Franco Zambonelli
University of Modena and Reggio Emilia
Facoltà di Ingegneria, Sede di Reggio Emilia
Dipartimento di Scienze e Metodi dell'Ingegneria
Via Allegri 13, 42100 Reggio Emilia, Italy
E-mail: franco.zambonelli@unimore.it

Library of Congress Control Number: 2006924852

CR Subject Classification (1998): D.2, I.2.11, F.3, D.1, C.2.4, D.3

LNCS Sublibrary: SL 2 – Programming and Software Engineering

ISSN 0302-9743
ISBN-10 3-540-34097-1 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-34097-3 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11752660 06/3142 5 4 3 2 1 0

Preface

New technology developments, such as Ambient Intelligence, the Internet of Things, the Grid, and Autonomic/Organic Computing, impose new requirements on the engineering of software systems. Nowadays, software is to be based on open architectures that continuously change and evolve to accommodate new components and meet new requirements. Software must also operate on different platforms, without recompilation, and with minimal assumptions about its operating environment and its users. Furthermore, software must be robust and autonomous, capable of serving a user with a minimum of overhead and interference.

Agent and multiagent concepts provide a number of interesting properties to respond to these challenges. They offer higher level abstractions and mechanisms which address issues such as knowledge representation and reasoning, communication, coordination, cooperation among heterogeneous and autonomous parties, perception, commitments, goals, beliefs, and intentions all of which need conceptual modeling. The implementation of these concepts can lead to advanced functionalities, e.g., in inference-based query answering, transaction control, adaptive workflows, brokering and integration of disparate information sources, and automated communication processes. At the same time, successful research is being performed to provide links between the modeling of agent systems and state-of-the-art software modeling techniques and tools, such as the Model-Driven Architecture™, or the Unified Modeling Language.

Like its very successful predecessors of the years 2000 to 2004 (*Lecture Notes in Computer Science*, Volume 1957, 2222, 2585, 2935, and 3382), the AOSE 2005 workshop sought to examine the credentials of agent-based approaches as a software engineering paradigm, and to gain an insight into what agent-oriented software engineering will look like, and what its benefits will be.

AOSE 2005 was hosted by the 4th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2005) held in Utrecht, The Netherlands, in July 2005. The workshop received 35 paper submissions. After a round of reviews where each paper received at least three reviews from independent reviewers, 13 papers were selected for regular presentation at the workshop, plus seven additional papers for short presentation. The workshop program included an invited talk, technical sessions in which the accepted papers were presented and discussed, and a closing plenary session. It congregated more then 50 attendees among researchers, students and practitioners, who contributed to the discussion of research problems related to the main topics in AOSE. After the workshop, the authors of the accepted papers were asked to review their papers based on both the reviewers' comments and the outcomes of the workshop discussion. Then, after a second round of reviews, 18 papers made it into this book, in which we are confident the readers will find a

comprehensive high-quality overview of the state of the art in agent-oriented software engineering.

This book is organized into five sections, each dealing with various very important aspects of multiagent systems development: Modeling Tools, Analysis and Validation Tools, Multiagent Systems Design, Implementation Tools, and Experiences and Comparative Evaluations.

Section 1: Modeling Tools

The first section focuses on the issue of modeling multiagent systems, and includes three papers.

The first paper, titled *Operational Modeling of Agent Autonomy*, by Weiß et al., focuses on the concept of autonomy, which is a central one in AOSE. The authors correctly discuss that, while there has been considerable progress in the theoretical aspects related to the autonomy concept, little has been done so far into transforming autonomy into a practical software property. To this end, the authors proposes ASL (Autonomy Specification Language) as a first step in this direction. ASL helps modeling autonomy in terms of the degrees of freedom left to the agents for the execution of their activities, and allows for the precise identification of the activities to be carried on by a set of agents.

The starting point of the paper authored by Cheong and Winikoff and titled *Hermes: Designing Goal-Oriented Interactions* is that interactions between agents are traditionally specified by using notations such as Petri Nets, AUML, etc., which – being message-centric – hardly fit autonomous goal-oriented agents. Also, since interaction protocols typically prescribe how interactions must be carried out by agents, they may limit the flexibility of interactions and the overall robustness of a multiagent system. Based on these assumptions, the authors present a new goal-oriented approach to practically model interactions, Hermes, which includes a methodology for designing goal-oriented interactions, failure-handling mechanisms, and a process for mapping design artifacts into an executable implementation.

The third paper, *Modeling Social Aspects of Multiagent Systems: the AML Approach*, by Cervenka et al., outlines the need for any modeling tool to provide suitable ways for representing the social aspects of multiagent systems, including the social structure of a multiagent system, the social behavior driving the overall dynamics of the system, and the social attitudes of the individuals in the system. In this context, the authors propose a new Agent Modeling Language (AML) to model social aspects. The paper specifically focuses on analyzing those aspects of AML related to social structure modeling and to role modeling, and evaluates the effectiveness of AML with the help of application examples.

Section 2: Analysis and Validation Tools

This section focuses on the important issue of analyzing and validating multiagent systems.

The starting point of the paper *Requirements Elicitation for Agent-Based Applications*, authored by Fuentes Fernandez et al., is that the analysis of

requirements and, specifically, requirements elicitation is a key stage for the development of complex multiagent systems. Given the need for proper tools to support requirements elicitation, the authors propose a new tool based on activity theory and social sciences, the Requirements Elicitation Guide. The guide empowers teams devoted to developing multiagent systems with the necessary knowledge and experience required to succesfully perform requirements elicitation.

In their paper titled *Formalization and Analysis of the Temporal Dynamics of Conditioning*, Bosse et al. outline that it is very important for AOSE techniques and for AOSE analysis to be able to properly incorporate learning mechanisms into agent systems. By focussing on the specific learning mechanism of classical conditioning, the authors point out that traditional modeling mechanisms – based on dynamical systems theories – mismatch with the traditional way of modeling software systems (and multiagent systems), typically based on logical languages. Accordingly, the authors explore a new logical approach to model classical conditioning, which may be more suitable for integration into current AOSE techniques.

In the last paper of this section, titled *Incorporating Committment Protocols into Tropos*, Mallya and Singh attempt synthetizing two trends in the engineering of agent systems. On the one hand, modern methodologies focus on the key phases of agent development, but tend to miss properly modeling flexible interactions. On the other hand, committment protocols are deeply studied to model flexibly behaviors and interactions, but are not properly integrated into an engineering framework. The proposal of the paper is thus that the analysis phase of a multiagent system should incorporate committment protocols as a primary concern, at the same level of goals and agents, and the authors show how this can be done with regard to the Tropos methodology.

Section 3: Multiagent Systems Design

This section consists of four papers that investigate different aspects of the design of multiagent systems.

The first paper, titled *Zooming Multi-Agent Systems*, by Molesini et al., proposes a new technique for a multi-layered description and analysis of multiagent systems called *zooming*, and describes how the SODA methodology for agent-oriented software engineering can be extended to include a simple zooming mechanism. A case study concerning the management of a university course Internet website is provided to demonstrate the applicability and potential benefits of the new technique.

The second paper, authored by Hill et al., deals with *Improving AOSE with an Enriched Modeling Framework*. The authors observe that existing agent-oriented methodologies neglect (or, e.g., in the case of Tropos, only provide rudimentary support for) the case of early requirements gathering and analysis. Their contribution targets conceptual knowledge modeling to be used in early requirements engineering. The paper proposes a MAS design framework that provides *conceptual graphs* as a modeling notation. Based on these, a transaction-based

architecture is described which enables model verification during the requirements gathering phase. To allow their approach to leverage the capability of other AOSE methodologies and agent development environments, the description of a mapping of a conceptual graph model to AUML is included in the paper.

The third paper in this section, titled *Dealing with Adaptive Multi-Agent Organizations in the Gaia Methodology*, by Cernuzzi and Zambonelli, investigates factors, parameters and requirements for designing multiagent systems with view to adaptability. In particular, the paper analyzes the GAIA agent methodology with regard to its suitability in supporting and facilitating changes in the organization of a multiagent system. By means of a conference management example, the authors show how the above-mentioned factors and concepts can be taken into account when modeling a MAS using Gaia. The authors compare other methodologies with Gaia regarding their support for adaptation. They argue that while older methodologies (such as Roadmap, Prometheus, or MaSE) require MAS organizations to be derived in a more or less implicit way from the identification of roles and their interactions, the more recently proposed approaches (such as MASSIVE or Tropos) explicitly address the requirement of *design for change* to some degree.

The last paper in this section addresses the problem of providing transformations from verifiable formal goal-based specifications of agents to implementation models, so-called behavior automata. Simon and Flouret describe an approach that is based on an agent design model called goal decomposition tree (GDT), allowing designers to specify agent behaviors in a temporal logic formula (a subset of TLA). A proof system is given to enable verification of agent behaviors specified in GDT. The focus of the paper is the description of an implementation model that is based on automata, which can be automatically generated from the verified GDT agent model. In this implementation model, the behavior automaton of an agent is constructed by combining elementary automata using so-called *automata composition patterns*. These patterns are associated with the goal decomposition operators as specified in the GDT language.

Section 4: Implementation Tools

The papers in this section describe up-to-date research efforts on the development of tools for developing agent and multiagent systems and applications.

In their paper titled *An Approach to Dynamically Generated User-Specified MAS*, Jayaputera et al. present an approach for designing multiagent systems relying on the concept of what they call *missions*. A mission is the description of a goal plus associated (partial) plan on how to achieve the goal. Given a mission, a set of agents (mobile or stationary) are created to work on the misison. The authors claim that introducing this abstraction allows them to focus on the application rather than on individual agents. Using the mission concept, the authors describe and empirically evaluate the eHermes platform that creates the (appropriate number of) agents required for a mission at run-time 'on demand.' The dynamic features of their approach provide an increased robustness of the

system if the environmental conditions change during execution, and the ability to maintain state and data of the mission so that it can be suspended and resumed at a later stage and at a different location.

The second paper in this section — *Supporting the Development of Multi-Agent Interactions via Roles* — by Cabri et al. starts from the assumption that the modeling of interactions between agents by roles can simplify the development of multiagent systems. The authors introduce the BRAIN framework for developing agent systems; the key elements of BRAIN are threefold: (i) a semi-formal model of a role (defined as a set of capabilities and its expected behavior); (ii) the XRole language, an XML-based notation for roles; and (iii) interaction infrastructures building on the XRole notation and the role model. The focus in this paper is on how roles in BRAIN can be employed in different phases of applications development, covering analysis, design, and implementation, thus establishing a central repository of information usable throughout the whole application development lifecycle. The concepts are illustrated by using an application example.

The third paper, *Automating Model Transformations in Agent-Oriented Modeling*, by Perini and Susi, advocates the usage of OMG's Model-Driven Architecture (MDA$^{\text{TM}}$) to provide a model-based approach for the analysis and design of multiagent systems. In particular, the paper discusses the role of model transformations in agent-oriented software engineering. Using the Tropos methodology, the paper discusses how MDA concepts can be applied to the different phases of agent systems development. In this context, the authors describe a model-to-model transformation from a Tropos plan decomposition model to a UML 2 activity diagram. Also, the paper describes a set of tools that the authors created for supporting the use of the Tropos methodology according to the MDA paradigm by re-engineering the TAOM Tropos modeler in the Eclipse platform. The results of this are two Eclipse plug-ins, the *TAOM4e model* implementing the Tropos meta-model, and the *TAOM4e* platform implementing the modeler functions required for building and managing Tropos models.

A similar problem is dealt with by García-Ojeda et al., the authors of the fourth paper in this section. In *Paving the Way for Implementing Multiagent Systems: Integrating Gaia with Agent-UML*, they describe a MAS development process that incrementally refines a design made using Gaia by applying agent-oriented extensions of UML (in particular Agent-UML). The authors claim that by combining Gaia with Agent-UML, a MAS design can be made more concrete. Technically, this is achieved by mapping the core models of Gaia (Interaction Model, Roles Model, Organizational Structure Model, Service Model, and Agent Model) into the three layers of the agent interaction protocols as defined in Agent-UML. Thus, it is possible to extend the representation of protocols and interaction models, agents, and organizational structures in Gaia with the corresponding concepts in Agent-UML. The applicability and potential benefits of the authors' work are illustrated by using a sample scenario involving the design of a conference management system.

In the last paper in this section, titled *Applying Multi-agent Concepts to Dynamic Plug-in Architectures*, Duvigneau et al. apply concepts of agent orientation to the plug-in-based architecture which are currently being developed in software engineering research. The work described in this paper aims at plug-in frameworks like Eclipse or NetBeans. The main goal of this research is to pull the design of plug-in-based applications up to a conceptual modeling level. The authors use their MULAN architecture, to build a conceptual model of a plug-in-based system. The core idea is to achieve extensibility by the idea of nested platform agents and to support this concept by message-based horizontal and vertical communication between the agents representing the components in the system. Based on the conceptual model, a plug-in system using the Renew platform is described, which enables dynamic configuration of the plug-in system.

Section 5: Experiences and Comparative Evaluations

In this last section, three papers are included that report interesting experiences and evaluations of specific AOSE-related issues.

The paper *Using the Analytic Hierarchy Process for Evaluating Multiagent Systems Architecture Candidates*, by Davidsson et al., starts from the consideration that, although a number of different multiagent systems architectures are being proposed and implemented, little has been done so far to systematically evaluate them. In particular, the authors argue that – when developing a multiagent system – it is important to evaluate possible architecture candidates with respect to their suitability to the specific application scenario. In this context, the authors focus on the problem of load balancing in intelligent networks, and they evaluate four different architectures that can be used to handle this task. These architectures are then studied via simulations, and metrics measurements are recorded and analyzed using the analytic hierarchy process, which is proposed as a useful analysis tool for deciding which architecture candidate is the most appropriate in different situations.

The paper *Estimating Costs for Agent-Oriented Software*, by Gomez-Sanz et al., focuses on software economics and on the need to carefully evaluate the costs involved in developing agent-oriented software systems. The authors correctly claim that there is a lack of shared experience in evaluating the costs associated to the development of multiagent systems, and provide some results related to this. Specifically, the authors exploit data collected from real agent-based projects, and give hints for the application of existing software cost estimation models – e.g., the well-assessed COCOMO model – and for what would be appropriate metrics for agent-based software development. These results can assist agent developers to elaborate tentative estimations of how much effort they should dedicate to their projects and determine their costs.

The last paper of this section and of the book, *Aspects in Agent-Oriented Software Engineering: Lessons Learned*, by Garcia et al., focuses on the issue of modularity in multiagent systems. The paper shows that several concerns in the development of multiagent systems cannot be represented in a modular way, since they crosscut several system modules and do not easily fit into the

traditional abstractions of agent-oriented software engineering. Thus, the authors argue that is is important to systematically verify whether emerging development paradigms support improved modularization of the crosscutting concerns relative to multiagent systems. The paper then reports some lessons learned based on their experiences in using aspect-oriented methods and techniques to address these problems. In the light of these lessons, the authors also discuss related work in the area and are able to discuss a number of promising future research directions.

We believe that this thoroughly prepared volume is of particular value to all readers interested in key topics and the most recent developments in the very exciting field of agent-oriented software engineering.

February 2006

Jörg P. Müller
Franco Zambonelli

Organization

Organizing Committee

Jörg P. Müller (Co-chair)
Clausthal University of Technology, Germany
Email: mueller@in.tu-clausthal.de

Franco Zambonelli (Co-chair)
Universitá di Modena e Reggio Emilia, Italy
Email: franco.zambonelli@unimo.it

Steering Committee

Paolo Ciancarini, University of Bologna, Italy
Gerhard Weiß, Technische Universität München, Germany
Michael Wooldridge, University of Liverpool, England

Program Committee

Federico Bergenti (Italy)
Carole Bernon (France)
Giacomo Cabri (Italy)
Paolo Ciancarini (Italy)
Massimo Cossentino (Italy)
Scott DeLoach (USA)
Bruce Edmonds (UK)
Alessandro Fabricio Garcia (Brazil)
Paolo Giorgini (Italy)
Marc-Philippe Huget (France)
Michael Huhns (USA)
Carlos Iglesias (Spain)
Matthias Jarke (Germany)
Catholijn Jonker (Netherlands)
Thomas Juan (Australia)
David Kinny (Australia)
Manuel Kolp (Canada)
Juergen Lind (Germany)
Sehl Mellouli (Canada)
Andrea Omicini (Italy)
Van Parunak (USA)
Anna Perini (Italy)
Michael Rovatsos (Germany)
Brian Henderson Sellers (Australia)
Onn Shehory (Israel)
Gerhard Weiß (Germany)
Michael Winikoff (Australia)
Mike Wooldridge (UK)

Table of Contents

Modeling Tools

Analysis and Validation Tools

Multiagent Systems Design

Implementation Tools

Experiences and Comparative Evaluations