Report from Dagstuhl Seminar 12342

Engineering Multiagent Systems

Edited by

Jürgen Dix[1], Koen V. Hindriks[2], Brian Logan[3], and Wayne Wobcke[4]

1 **TU Clausthal, DE,** `dix@tu-clausthal.de`
2 **TU Delft, NL,** `k.v.hindriks@tudelft.nl`
3 **University of Nottingham, GB,** `bsl@cs.nott.ac.uk`
4 **UNSW - Sydney, AU,** `wobcke@cse.unsw.edu.au`

Abstract

This report documents the programme and outcomes of Dagstuhl Seminar 12342 "Engineering Multiagent Systems". The seminar brought together researchers from both academia and industry to identify the potential for and facilitate convergence towards standards for agent technology. As such it was particularly relevant to industrial research. A key objective of the seminar, moreover, has been to establish a roadmap for engineering multiagent systems. Various research areas have been identified as important topics for a research agenda with a focus on the development of multiagent systems. Among others, these include the integration of agent technology and legacy systems, component-based agent design, standards for tooling, establishing benchmarks for agent technology, and the development of frameworks for coordination and organisation of multiagent systems. This report presents a more detailed discussion of these and other research challenges that were identified. The unique atmosphere of Dagstuhl provided the perfect environment for leading researchers from a wide variety of backgrounds to discuss future directions in programming languages, tools and platforms for multiagent systems, and the roadmap produced by the seminar will have a timely and decisive impact on the future of this whole area of research.

Seminar 19.–24. August, 2012 – http://www.dagstuhl.de/12342
1998 ACM Subject Classification I.2.11 Distributed Artificial Intelligence – Multiagent systems, Intelligent agents, D.2 SOFTWARE ENGINEERING
Keywords and phrases Agent-oriented programming, Multiagent systems, Software methodologies for distributed systems, Programming distributed systems, Empirical evaluation
Digital Object Identifier 10.4230/DagRep.2.8.74
Edited in cooperation with Federico Schlesinger

1 Executive Summary

Jürgen Dix
Koen V. Hindriks
Brian Logan
Wayne Wobcke

In 1993, Yoav Shoham's paper on agent-oriented programming was published in the Artificial Intelligence Journal. Shoham's ideas, and the work on agent-oriented programming it inspired, has had a profound impact on the field of multiagent systems, as evidenced by Shoham's paper receiving a 2011 IFAAMAS Influential Paper Award recognising seminal work in the field. Agent-oriented programming offers a natural approach to the development of complex systems in dynamic environments, and technology to support the development of agents and multiagent systems is beginning to play a more important role in today's software development at an industrial level.

Since Shoham's initial work, a range of platforms that support agent orientation have become available, and considerable experience has been gained with these platforms. Some key issues have also emerged from this work, however. First, given the plethora of systems and approaches that have become available in the field for developing multiagent systems, it is no longer clear which of these technologies is most appropriate for developing a particular application or what the distinctive benefits of various approaches are. It is especially important for practitioners to understand the benefits resulting from a particular choice of technology, when and how to apply it, and to develop standards that support the application of agent technology. Secondly, the very different style of agent-oriented programming potentially hampers the uptake of agent development tools and methods. To successfully apply the agent-oriented paradigm and to support the implementation and testing phases of agent-oriented development it is therefore very important to establish best practices and evaluate lessons learned from applying the technology in practice.

The aim of this seminar was to bring together researchers from both academia and industry to identify the potential for and facilitate convergence towards standards for agent technology. The seminar was very relevant for industrial research. The seminar meetings were meant to enable interaction, cross-fertilisation, and mutual feedback among researchers and practitioners from the different, but related areas, and provide the opportunity to discuss diverse views and research findings. The interaction in a Dagstuhl seminar was considered to be ideal for establishing common ground for defining standards, identifying best practices, and developing approaches to applying agent technology to the large scale, realistic scenarios found in industry. The aim of the discussions that were planned was therefore to establish a future research agenda, i.e. a *roadmap*, based on an evaluation of current state-of-the-art of agent-oriented programming languages, tools and techniques that are particularly important for large scale industrial applications.

The seminar took place August 19–24, 2012, with 37 participants from 15 countries. The programme included presentations by the participants and group discussions. Presentations were about 30 minutes long, including questions. We specifically asked participants not to present current research (their next conference paper), but rather asked for what should be considered the next step in their research area.

Participants were encouraged to use their presentations to provide input for discussion about the roadmap. They should show their perspectives and discuss what they think should be on the research agenda, try to explain why, and what it is they think this community should be aiming for. The group discussions took place in the afternoon, after the coffee break until 6pm. We put together four groups of 8-10 members, each headed by one discussion leader (see Section 4 below for more details). The results of each working group were then presented to all participants before dinner. The seminar concluded with a general discussion on Friday morning and a wrap-up.

We identified the following important outcomes of the seminar.

MAS: Understanding of the uptake of multiagent systems technology in industry is seriously hampered by the current situation concerning paper acceptance at scientific conferences and workshops: While new theoretical approaches easily find their way into these events, papers about serious implementations that scale up and put theoretical concepts to work are often considered not innovative enough and are thus not considered appropriate as scientific papers. We need a forum to publish such papers in order to generate research on the transfer of agent technology to industry.

Merger: During the seminar, eight out of 12 steering committee members of three important workshops in the area of agent systems development (ProMAS, DALT, and AOSE) met to discuss the possibility of merging the workshops. Based on the discussions at the seminar, it was generally agreed that greater focus is needed, and a single venue to present work in the field would be desirable. The workshop steering committees therefore decided (during the seminar) to merge ProMAS, DALT and AOSE to form a new workshop *Engineering Multiagent Systems*. 2012 will therefore be the last year in which the workshops will be held separately: They will be replaced by the new EMAS workshop at next year's AAMAS.

Roadmap: The organisers agreed to start a draft on the roadmap, based on the results of the group sessions. We plan to include the group leaders to produce a first draft, discuss it with the participants and afterwards, to finalise it.

2 Table of Contents

3 Overview of Talks

3.1 Challenges in MAS Verification

Natasha Alechina (University of Nottingham, GB)

Joint work of Alechina, Natasha; Logan, Brian; Nguyen, Nga; Rakib, Abdur; Doan, Trang; Dastani, Mehdi; Meyer, John-Jules

I give a brief overview of the state of the art in verification of multiagent systems where agents are implemented in BDI agent programming languages, and list the challenges. The main challenges are: - Ability to represent MAS at a suitably high level of abstraction- Ability to formulate properties in a suitable language- Scalability improvements

3.2 Agents in Unmanned Aerial Vehicle Applications

Jeremy Baxter (QinetiQ - Malvern, GB)

I discuss my background in multiagent systems and my experience with using multiagent toolkits. I have used agents to control small teams of unmanned air vehicles both in simulation and in test flight. The main focus of the work has been using agents to co-ordinate multiple vehicles and to integrate different types of planning and reasoning. When developing systems only a small part of the effort is a core agent system, the majority is interfaces. Testing is a major element of the development and is not well supported by current agent tools. There is a steep learning curve with new tools and languages which can be hard to justify in a project. I conclude that design patterns and libraries of components might gain better acceptance than complete new languages and development environments.

3.3 Reflections on Multiagent Oriented Programming

Rafael H. Bordini (PUCRS - Porto Alegre, BR)

URL http://dx.doi.org/10.1016/j.scico.2011.10.004

In this talk I discussed some reflections on multiagent oriented programming based on my own experiences, and in particular recent experiences with the JaCaMo platform, in joint work with Jomi Hübner, Olivier Boissier, Alessandro Ricci, and Andrea Santi. I made the point that after years of research we have not yet been able to define precisely what multiagent orientation entails as a programming paradigm. I also argued that achieving such shared view of a paradigm is essential if our work is to reach out to other research communities within computer science.

3.4 Building Multiagent Systems for the Real World: A Company's Perspective

Paolo Busetta (AOS Ltd. - Cambridge, GB)

AOS is one of the few companies on the market whose business is focused on multiagent platforms and applications. AOS' main product, JACK, was originally released in 1997. Since then, AOS has been involved in a large number of diverse research and applicative projects, varying from cognitive simulation in virtual reality to safety-critical embedded systems. In this talk, I will present a few important technical challenges that AOS has faced in its 15 years of existence. These experiences have contributed to shape the new agent platform under development, called C-BDI. I will briefly introduce some of its novelties and how they are meant to address the needs of its expected main domains of application, in particular autonomous operational systems and serious games.

3.5 Experiences with Agent Factory

Rem Collier (University College Dublin, IE)

Joint work of Collier, Rem; O'Hare, Gregory

This talk is broken into two parts. The first part presents an overview of the history of Agent Factory; a cohesive framework for the development and deployment of multiagent systems that has been under development and in constant use since 1996. It briefly reflects on the design choices made for each version of the framework and the improvements made. Where relevant a selection of applications built using the specific version of the framework are described. The second part reflects on some experiences gained from the use of Agent Factory both in terms of the development of demonstrators and in terms of its use as a teaching platform. Specific comments made in part 2 include: the lack of online community resources that promote the field; the challenge of meeting users expectations in terms of tool support; and the lack of significant work on evaluation of agent programming languages / comparison of agent-based solutions with non-agent based solutions.

3.6 Handling High Frequency Perception / Agents and Enterprise Computing

Stephen Cranefield (University of Otago, NZ)

Joint work of Cranefield, Stephen; Ranathunga, Surangika; Purvis, Martin
Main reference Surangika Ranathunga, Stephen Cranefield and Martin Purvis. Identifying Events Taking Place in Second Life Virtual Environments. Applied Artificial Intelligence, (26)1-2:137-181, 2012
URL http://dx.doi.org/10.1080/08839514.2012.629559

In the first part of this talk I briefly discussed some work at the University of Otago on connecting agents with virtual worlds such as Second Life, and used this to motivate a proposal for enhancing agent platforms with support for handling high frequency state changes. In the second part, I considered the role that agents might play in enterprise computing and how they might be integrated into enterprise applications and business processes. I argued that agents can play a useful role as components of larger businesses processes, and that agent development tools should provide an interface between agents and the existing integration technology used in enterprise computing. In particular, I proposed that a simple interface between agents and enterprise computing infrastructure can be provided by defining agent "endpoints" for enterprise message routing and mediation engines such as Apache Camel. These configurable endpoints would translate (selectively) between internal agent entities such as beliefs and ACL messages and the message exchange abstraction used in the enterprise integration patterns (EIPs) of Hohpe and Woolf [1]. Message routing and mediation rules could then be defined outside the agent platform to interconnect the agents with any other protocols and services that have endpoints defined (such as the 130+ that are available for Apache Camel).

References

1 G. Hohpe and B. Woolf. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions.* Addison-Wesley, 2004.

3.7 Engineering Multiagent Systems: Lessons and Challenges

Mehdi Dastani (Utrecht University, NL)

In my presentation I explained the aims of multiagent programming research field as formulated in this community and gave a brief presentation of the activities within this community in the last decade. A distinction is made between academic and industry perspectives. I argued that although both perspectives are valuable and challenging, their activities and aims are different. For each perspective I presented some challenges and future directions for research. I ended the presentation by emphasising the role of transfer of knowledge from the academic perspective to the industry perspective.

3.8 Timeliness Issues in Agent Based Control of Satellites, Among Other Things

Louise Dennis (University of Liverpool, GB)

Joint work of Fisher, Michael; Veres, Sandor; Lincoln, Nicholas; Lisitsa, Alexei; Gao, Yang; Bordini, Rafael; Muller, Berndt

The talk consisted of three parts: 1) The Engineering Autonomous Space Software project investigated the integration of real-time control systems with a rational agent layer for decision making. The focus of the project was on the abstraction of continuous data to discrete data. The implementation ran into a number of issues related to the speed with which data or commands generated by one part of the system could be processed by another part of the system. Since this did not form the core focus of the project these issues were worked around in an ad hoc fashion. This talk provided an overview of the problems, the "quick fixes" and throw out a couple of ideas for how the problems might be dealt with more coherently. 2) An overview of the Agent Infrastructure Layer, a Java-based toolkit for implementing the operational semantics of BDI agent systems and then model checking programs written in the systems. In particular I considered the question of whether the Agent Infrastructure Layer constituted a Virtual Machine for BDI agent languages. 3) An overview of the aims of the newly awarded Reconfigurable Autonomy project.

3.9 What We Talk About When We Talk About Agents

Virginia Dignum (TU Delft, NL)

Joint work of Dignum, Virginia; Frank Dignum
Main reference Virginia Dignum, Frank Dignum: Designing agent systems: state of the practice; International Journal of Agent-oriented Software Engineering - IJAOSE , vol. 4, no. 3, pp. 224-243, 2010
URL http://dx.doi.org/10.1504/IJAOSE.2010.036983

In this presentation, I discussed different views on agent technology and its applications. Guidelines to decide on agent approaches and its consequences for (agent-oriented) software engineering lifecycle. I furthermore introduced a few extra issues to be included in the roadmap: Scaling / multi-level models; Evolution / re-design; and the role of people in the loop.

3.10 Agent Technology integration with Maven for an Ambient Assisted Living Case Study

Jorge J. Gomez-Sanz (Univ. Comp. de Madrid, ES)

URL http://sociaal.sf.net, http://ingenias.sf.net

The talk introduces some recent advances in the INGENIAS Development Kit to deal with challenges found in an Ambient Assisted Living (AAL) oriented project and how the Maven project management tool contributed to this goal. Ambient Assisted Living systems tries to make the life of people easier by assisting them in different ways. Most of them have to do with a smart use of sensors and actuators situated all of them in the environment of the user. Literature in AAL tells a natural candidate for become the main building block in this kind of proposal is agent technology. The name of project mentioned in this talk is SociAAL, because it tries to focus on social aspects that influence this kind of systems. The project is inherently challenging because of the mixture of different technologies for which standard agent oriented development environments are not the best choice. Current tools do not strongly support integration and require installing different plugins that do not ensure two technologies can work together. A candidate solution is the Maven framework. Maven is a tool created by the Apache Foundation. Quoting them "Apache Maven is a software project management and comprehension tool". In SociAAL project, Maven integration has meant a possibility of putting together in a straight forward way the development of OSGi objects, XML documents, XML transformations, INGENIAS agents, and other artefacts.

The introduction and justification of this framework has served to explore stages of the development which are not covered usually by AOSE methodologies. As a result, the talk introduces how well is AOSE dealing with a complete software development using as driver the software lifecycle according to the standard IEEE Glossary of Software Engineering. The conclusion of the talk is that many work is needed in AOSE to understand the role of agent technology in a long term development. In this endeavour, frameworks like Maven can help, since they are widely used mainstream software engineering tools and can be trusted to identify meaningful aspects of a development. By integrating with Maven, AOSE will have to tell what "compiling","generating sources", "documenting", "testing", or "packaging" the multiagent system means. This will introduce better our technology to people used to work with software.

3.11 Perspectives and Roadmap for Engineering Multiagent Systems

Christian Guttmann (IBM R&D Labs, Melbourne, AU)

On our agent roadmap, we have not advanced as far as we ought to. Academic and industrial agent projects still lack consistent and unified design and engineering patterns, and advantages of agent engineering over other engineering approaches are not entirely clear. Hence, it is difficult to evaluate the benefit and potential of agents as an approach and methodology for ambitious projects, and hence it is difficult to make a well informed choice of using agents. I will support this statement by revisiting how far we have come on existing agent roadmaps, and also by reporting on my recent experience on defining and leading R&D projects that extend and use agent technologies in the area of health and medicine. A few ideas are offered to extend the agent roadmaps. Our community may benefit from engaging more in the technology transfer process (showing the value of agent engineering), and engaging more with other research communities and stakeholders, where the key is to identify and define challenges together, rather than in isolated labs and research groups.

3.12 Multiagent Oriented Programming with JaCaMo

Jomi Hübner (Federal University of Santa Catarina - Brazil, BR)

This talk brings together agent-oriented programming, organisation-oriented programming and environment-oriented programming, all of which are programming paradigms that emerged out of research in the area of multiagent systems. In putting together a programming model and concrete platform called JaCaMo which integrates important results and technologies in all those research directions, we show in this paper that with the combined paradigm, that we prefer to call "multiagent oriented programming", the full potential of multiagent systems as a programming paradigm. JaCaMo builds upon three existing platforms: Jason for programming autonomous agents, Moise for programming agent organisations, and CArtAgO for programming shared environments.

3.13 Lessons and Perspectives on Agent Languages

Yves Lespérance (York University - Toronto, CA)

In the talk I briefly review the main features of the Golog family of Situation Calculus-based agent programming languages and application where they have been used and put out some ideas for future research on Engineering Multiagent Systems. One topic raised for future work is modeling and reasoning about the mental states of other agents (Theory of Mind) in agent programming languages, and I briefly discuss some initial work in that area.

3.14 Programming Agents

Brian Logan (University of Nottingham, GB)

Joint work of Logan, Brian; Alechina, Natasha; Bordini, Rafael; Dastani, Mehdi; Gordon, Elizabeth; Hindriks, Koen; Madden, Neil; Meyer, John-Jules; Sloman, Aaron; Vikhorev, Konstantin

In this talk, I consider agent programming from the perspective of Artificial Intelligence. I briefly outline some lessons learned from our work on developing approaches to tractable deliberation for intention scheduling in the agent programming languages ARTS, AgentSpeak(RT) and N-2APL, and highlight some unsolved problems in deliberation about deadlines and plan durations. I also sketch some future directions for agent programming.

3.15 On Engineering Emotional Agent Systems

John-Jules Ch. Meyer (Utrecht University, NL)

In this talk I go through the following: Why emotional agent systems? The main idea. Methodology. How far we have got. Intuition of 4 basic types of emotion. Deliberation with emotions. Future work.

3.16 Observations from Current and Past Projects: 1. Shaping the Intelligent Home of the Future, 2. Settlers of Catan

Berndt Müller (University of Glamorgan, GB)

We discuss our experience from projects using agent-based techniques and point out some research topics that will have to be addressed if multiagent based systems become ubiquitous. These include legal and ethical issues, security, and verification. The latter needs to be more rigorous (generating reasoning engines from semantic specifications) and ideally needs to take notions of resource and location into account. Of further importance to the acceptance of MAS as a programming paradigm, is the availability of a component-based approach and the availability of agent libraries. This is illustrated by an example of agent-based development of a turn-based game using high-level Petri nets based on the nets-within-nets paradigm.

3.17 Application Impact of Multiagent Systems and Technologies

Jörg P. Müller (TU Clausthal, DE)

There appears to be a common perception among Multiagent Systems (MAS) researchers that their research field has still some room left in creating impact outside our own research community. However, there are no recent studies that allow us to back up or rebut this hypothesis. In this talk I am providing some thoughts and observations related to the application impact of MAS. I review some models of ICT impact known from the literature and discuss their applicability to MAS research. Further, I discuss previous work related to research on application impact in our research community. I come to the conclusion that it is beneficial to include research activities related to the study of application impact into a research roadmap on multiagent systems and technologies. I propose some desiderata for such research, and inform about an ongoing survey activity.

3.18 Exploring Agents as a Mainstream Programming Paradigm: The simpAL Project

Alessandro Ricci (University of Bologna, IT)

Main reference Alessandro Ricci, Andrea Santi. "Designing a general-purpose programming language based on agent-oriented abstractions: the simpAL project". Proceeding ofSPLASH '11 Workshops Proceedings of the compilation of the co-located workshops on DSM'11, TMC'11, AGERE!'11, AOOPES'11, NEAT'11, VMIL'11.
URL http://dx.doi.org/10.1145/2095050.2095078

Agent-Oriented Programming has been explored so far mainly in the context of Distributed AI and Multiagent Systems, and it is almost totally unknown in the context of programming languages and software engineering. In spite of that, we argue that agent-oriented concepts and abstractions could be effective to tackle main problems that affect modern programming, beyond object-oriented programming and actor-based programming. Accordingly, our medium-term objective is to shape a new programming paradigm based on agent-oriented abstractions, as a natural evolution of the object and actor ones. This calls for devising programming languages – as well as related models and technologies – that, besides being based on agent-oriented abstractions, would provide features and mechanisms that are important when programming and software development is of concerns. In this presentation we discuss our progress in that direction, represented by the simpAL programming language and platform.

3.19 MAS for Engineering Complex Systems

Amal El Fallah Seghrouchni (UPMC - Paris, FR)

This talk will present my main experiences of MAS engineering within industrial context. It aims to show where and how MAS may bring an added-value for complex systems design.

The first part of my talk outlines some examples of simulations of complex systems and also some prospective projects we have developed in the aerospace domain. Two main projects are described: 1) SCALA is a project for mission interception based on reactive multiagent planning (collaboration with Dassault-Aviation) and 2) the coordination of fleet of UAVs where several aspects of MAS are involved such as planning, elicitation of preferences and multiagent decision (collaboration with Thales Airborne Systems).

The second part of my talk goes on to relate some lessons learnt from the development of two languages for MAS programming, namely CLAIM and its extension S-CLAIM (Smart Claim) to deploy MAS an smart devices. S-CLAIM is a declarative agent-oriented language for Ambient Intelligence (AmI) - S-CLAIM - that allows programming reactive or cognitive mobile agents in a simple, easy-to-use manner while meeting AmI requirements. Based on a hierarchical representation of the agents, the language offers a natural solution to achieve context-sensitivity. S-CLAIM is an evolution of the CLAIM language, its predecessor. It is light-weight and, being transparently underpinned by the JADE framework, allows deployment on mobile devices and easy interoperation with other components by means of web services. The usefulness of the proposed language for AmI is illustrated through a scenario and a demo featuring an AmI application in a Smart Room (see also the attached paper presented at ANT'2012). My talk concludes with a positive note concerning the transfer in the field of MAS from academia to industry.

3.20 Agents in Space for Real: Lessons Learned from Applying Agent Technology in NASAs Mission Control

Maarten Sierhuis (Ejenta Inc., US)

This talk provides lessons learned from developing and implementing the first multiagent workflow system that automates the work of the OCA flight controller in NASA's Mission Control Centre for the International Space Station. OCAMS was first simulated and then developed and deployed using the Brahms agent language and NASA's Brahms environment. Ejenta, Inc. is a startup in San Francisco, CA and has as its mission to develop intelligent personal agent technology. Ejenta provides a commercial version of the Brahms agent simulation and development environment and its associated NASA applications and technology, including the OCAMS multiagent procedure execution workflow environment and the Individual Mobile Agent System. For more information, please contact the author.

3.21 Empirical Software Engineering for Agent Programming

Birna van Riemsdijk (TU Delft, NL)

Joint work of van Riemsdijk, M. Birna; Hindriks, Koen; Jonker, Catholijn M.
Main reference M. B. van Riemsdijk, K. V. Hindriks, and C. M. Jonker. An empirical study of cognitive agent programs. Multiagent and Grid Systems (MAGS), 8(2):187-222, 2012.

In this talk I argue for increasing use of empirical software engineering in the development of agent programming languages and techniques. Empirical software engineering is a branch of computer science in which empirical methods are used to study how people use the technologies and to what extent certain techniques are better than others. We need to investigate how software quality characteristics as identified in mainstream software engineering apply in the context of engineering multiagent systems, and define dedicated attributes and metrics to measure to what extent these are present in the software product. In this way we can improve the techniques based on data. Also we need to develop or come to agreement concerning what 'counts' as good empirical research for engineering multiagent systems.

3.22 Engineering Multiagent Systems - Reflections

Jørgen Villadsen (Technical University of Denmark - Lyngby, DK)

Joint work of Villadsen, Jørgen; Jakobsen, Troels Christian
Main reference Troels Christian Jakobsen: "I wouldn't have thought of it myself" - Emergence and unexpected intelligence in theater performances designed as self-organising critical systems. In proceedings: Algolog Multiagent Programming Seminar 2011 (AMAPS2011) - Technical University of Denmark - Lyngby
URL http://www.imm.dtu.dk/algolog/index.php?n=Home.AMAPS

In the first part I look at a theater performance by artistic director Troels Christian Jakobsen as a multiagent system. It is designed as a self-organising critical system using a framework where within its borders but without a script there is real interaction between the elements of the performance. In the second part I discuss the ideas behind my recent monograph on propositional attitudes and inconsistency tolerance. Natural language sentences are parsed using a categorial grammar and correctness of arguments are decided using a paraconsistent logic. In the third part I present a curriculum for the MSc in Computer Science and Engineering program at the Technical University of Denmark with a focus on multiagent systems. As the director of studies I have observed that the students are working hard and with much creativity in advanced courses and projects involving intelligent agents, in particular in the agent contest 2009-2012.

3.23 Challenges and Directions for Engineering Multiagent Systems

Michael Winikoff (University of Otago, NZ)

Main reference "Future directions for agent-based software engineering", Int. J. Agent-Oriented Software Engineering, Vol. 3, No. 4, pp. 402-410.
URL http://dx.doi.org/10.1504/IJAOSE.2009.025319
URL http://arxiv.org/pdf/1209.1428.pdf

In this talk I review where we stand regarding the engineering of multiagent systems. There is both good news and bad news. The good news is that over the past decade we've made considerable progress on techniques for engineering multiagent systems: we have good, usable methodologies, and mature tools. Furthermore, we've seen a wide range of demonstrated applications, and have even begun to quantify the advantages of agent technology. However, industry involvement in AAMAS appears to be declining (as measured by industry sponsorship of the conference), and industry affiliated attendants at AAMAS 2012 were few (1-2%). Furthermore, looking at the applications of agents being reported at recent AAMAS, usage of Agent Oriented Software Engineering (AOSE) and of Agent Oriented Programming Languages (AOPL) is quite limited, which is also supported by the results of a 2008 survey by Frank and Virginia Dignum ("Designing agent systems: state of the practice", IJAOSE 2010, 4(3):224-243). Based on these observations, I make five recommendations: 1. Re-engage with industry 2. Stop designing AOPLs and AOSE methodologies ... and instead ... 3. Move to the "macro" level: develop techniques for designing and implementing interaction, integrate micro (single cognitive agent) and macro (MAS) design and implementation 4. Develop techniques for the Assurance of MAS 5. Re-engage with the US.

3.24 Decoupling in Industry

Cees Witteveen (TU Delft, NL)

I discuss an application of agent technology in maintenance. The industrial partner is part of the Dutch Railway company responsible for maintenance. They are interested in flexible schedules and distribution of a global operational scheduling problem over several teams. These teams should be able to schedule their activities independently. We applie some ideas derived from temporal decoupling, but also from classical OR to solve their problems. The main lessons learned are: (1) use the language and concepts of your partner, (2) make very concrete promises and fulfil them in a verifiable way; (3) do not hesitate to consider agents only as a useful metaphor.

3.25 Engineering Multiagent Systems: Where is the Pain (and the Opportunity)?

Wayne Wobcke (UNSW - Sydney, AU)

I first describe two multiagent systems, one a "smart personal assistant" providing spoken dialogue interaction with a collection of personal agents in e-mail and calendar management (built using JACK), and the second an agent-based model for risk assessment of routine clinical processes, estimating the risk associated with patient misidentification and infection control (built using Brahms). I then reflect on the main difficulties in engineering these systems and the opportunities presented for further research. In summary, the main problems are not with particular programming languages or platforms, but of two types: (i) integration, where the agent aspect of the system is a small part of a much larger system, and (ii) validation of an agent model against reality. I conclude with a proposal of developing tools for semi-automatically constructing agent models using a mixture of knowledge acquisition and machine learning/data mining techniques, validating against traces of existing system behaviour, with particular application to the medical domain. This approach has recently become feasible due to the availability of "big" data sets.

4 Working Groups

There were three group-discussion sessions. The organisers separated participants into four groups (of size 8–10), which varied for the different sessions. The outcomes of the discussions in each group were presented to the other groups at the end of each session and a general discussion followed.

For the first session, we did not specify a particular topic. As this session took place on the very first day of the seminar, we just collected ideas and identified important topics.

For the second session, the four groups were assigned different topics, focussing on particular research areas: *Integration and Validation*, *Coordination and Organisation*, *Tools, Languages and Technologies* and *Component-Based Agent Design*.

The third session was again directed towards the structure of the roadmap. The motto for this session was:

> *What do you want to do in the next 10 years (research, applications)? Pick the top 10 topics out of a list or add new ones. Put them into clusters or state how they relate to each other.*

4.1 Integration and Validation

One objective of this Dagstuhl seminar was to bring together academic researchers and industry practitioners working on a variety of applications, to compare experiences, identify common problems in deploying agent technology, and propose ways to alleviate these problems in the future. A premise of the seminar was that there is currently a large variety of deployed agent applications, but that this was not widely recognised by the agents research community. The breadth of existing work was confirmed with presentations on: (i) avionics and defence (unmanned aerial vehicles), (ii) smart cities (ambient intelligence, crowd simulation), (iii) transportation (traffic modelling and simulation), (iv) logistics (warehouse management, maintenance scheduling), (v) workflow management, (vi) decision support (process monitoring, disaster management), (vii) healthcare (agent-based modelling, care coordination, patient monitoring), (viii) personal assistants (dialogue management), (ix) real-time control systems (mission planning), (x) power engineering (grid management), (xi) information integration (sensor networks), and (xii) virtual environments (games, training).

Most participants in this working group agreed on two major distinctive benefits of agent technology: **autonomous decision making** and **explicit problem decomposition and coordination mechanisms**. There was also clear consensus that the main problems facing deployment of agent/multiagent systems fall under three related areas: **integration**, **validation** and **software engineering**.

4.1.1 Integration

A basic problem is that an agent/multiagent system is generally only a small part of a much larger system containing any number of other complex hardware/software components. While traditionally a system can be organised "hierarchically" as a collection of agents (providing interfaces to the user and wrappers for other components), this is not generally the case for many applications. More commonly, agents need to interact with other non-agent components without the use of agent communication protocols, and need to be provided with information about the rest of the system's behaviour in order to perform their function. This means that the agent part of the system cannot easily be isolated from the remainder of the system, both in software development and for the purpose of reasoning about agent and system behaviour.

Integration can take place at multiple layers of abstraction, e.g. the implementation level, conceptual level, business level, etc. Furthermore, the design of the non-agent part of the system is typically outside the agent developer's control, so the agent programming language/platform needs to be highly flexible in allowing integration with interchangeable components (that may be at different abstraction layers), facilitate customisation of the agent/multiagent system to different application scenarios as needed, and support system maintenance of the whole system as it evolves. It is also desirable (for simplicity and efficiency) to be able to select only a subset of features of an agent platform needed for a given application rather than being required to use all features of a large and complicated agent platform.

4.1.2 Validation

The major potential benefits of agent technology arising from autonomous decision making present, paradoxically, a significant barrier to the adoption of the technology, since users typically require performance guarantees (preferably quantifiable and verifiable) that agent actions, though understood to be not completely predictable, are within *acceptable* bounds. Thus *validation* or *assurance* of agent/multiagent systems is particularly important. The level of system assurance required varies with the application area, but is most stringent in defence, where formal certification is required. Other sorts of applications need to be "trusted" by users, regulators and the community.

The type of properties that may need validation include safety, security, scalability, quality, maintainability, performance and interoperability. Validation is extremely difficult due to the complexity of agent behaviour and interactions, particularly as validation needs to be not only of the agent part of the system but of the system as a whole (and as noted above, agent/multiagent systems are tightly integrated into larger systems of agent and non-agent components). Validation of the agent part of a system is difficult because this part of the system cannot (in general) be isolated from the rest of the system, and because some desired properties may need to be derived from those of the agent development platform, which in turn may need to be validated/certified. In the specific area of agent-based modelling, validation is often ignored and datasets are often insufficient to provide adequate rigorous validation of models. Verification may sometimes be possible, but this is the exception rather than the rule with today's complex applications, considering the limitations of current approaches for specification and verification.

4.1.3 Software Engineering

It was commented by one participant that agent-based software development is 10% multiagent systems engineering and 90% standard software engineering. Whatever the breakdown, a consequence of the need for integration of agents into larger systems is that standard software engineering is heavily involved in the deployment of agent/multiagent systems. To improve the ease of adoption of agent/multiagent systems, what is needed are *not* more special-purpose "agent-oriented software engineering" methodologies, which often emphasise the distinctive nature of agent-based systems or which are closely tied to particular agent languages or platforms. Similarly, special-purpose agent programming languages present a barrier to deployment if they do not provide support for integration with existing software, creating undesirable "lock-in" to particular platforms and/or duplication of effort when non-agent components need to be translated into a particular agent model/language to enable interaction with agents.

Instead, what is required is better incorporation of agent-oriented software development *within* mainstream software engineering practices, and conversely, the use of more standard software engineering methodologies and tools within agent-oriented software development. This involves: (i) providing support for agent development within the whole software development lifecycle, from requirements engineering and architectural design through to testing and maintenance, (ii) integration with mainstream software development environments and especially tools, (iii) adoption of widely used software engineering approaches such as design patterns and pattern languages, (iv) compliance with software engineering standards, (v) "reaching out" to the software engineering research community through publication in software engineering venues, and (vi) comparison of agent-based software development platforms with standard programming language environments. The overall objective is to make it easier to deploy and maintain agent/multiagent systems within mainstream applications.

The term *component* is used above loosely to refer to some part of a larger hardware/software system. The topic of a specific "component-based" software engineering paradigm for agents was a subtheme of this working group discussion, but meant a number of things: (i) treating agents as interchangeable components in a larger system ("plug-and-play" agents or agent components, perhaps taken from a component library or repository), (ii) a declarative platform-independent representation for agents to enable reuse of agents from one system to another or to make it easier to construct agent models using third party tools, and (iii) building single agents out of simpler interchangeable components (e.g. belief database, reasoning engine, etc.). Despite the unresolved ambiguity, the idea of component-based agent software engineering was felt worthy of much further research.

4.2 Coordination and Organisation

The notions of *interaction* and *organisation* are important in Multiagent Systems (MAS), but they are important in other systems as well. However, the coordination and organisation have never been studied in a homogeneous fashion. We need answers to why there are such big differences. For example it should be explained why elements which are a concern in other systems are not a concern in MAS. As an example, for a newcomer with some knowledge in distributed systems, it is surprising that classical problems in concurrent systems, like deadlock or starvation, are hardly mentioned today. While we cannot define the problems with deadlock and starvation away, we claim that it is the agent paradigm, with its levels of abstraction and decentralised solutions, that helps to define these notions appropriately. We feel agent technology helps to better understand the problem, provides tools to deal with them and, in the end, verification of these and other properties is possible. One community dedicated to these topics is the COIN (COIN: Coordination, Organisation, Institutions and Norms). A big part of COIN is more concerned with abstractions and much less with implementations. There are important connections to the multiagent oriented paradigm that are not yet fully explored. Different technologies to enable coordination are not able to represent concepts needed in MAOP (Multi-Agent Oriented Programming) beyond messages. We believe that a semantical underpinning (as opposed to classical tuple spaces without any messages) makes message interchange easily possible and also helps to implement it.In a solution for *interaction* and *organisation* applied to a multiagent system, we would look for an *organisation model* subsuming both aspects. In an organisational model, we need to determine (1) the elements required to define a coordination, and (2) what goals should be pursued. Agents then acquire or are given these goals and commit to them in ways compliant with the organisation model.

Ideally, we are looking for an organisation specification language that could be translated into explicit organisations at the *execution level.* We can look for optimisations at two levels: (1) looking for first-class-citizen representation of concepts belonging to the organisation model; or (2) focusing on protocols/algorithms that implement the coordination/organisation which have certain properties (e.g., being deadlock-free) we might even verify. Also, we assume such MAS can change its coordination behavior (the proper (local or global) strategy) each time. In any case, the organisational approach makes explicit the strategy of the system in those cases. These observations are also important for our planned roadmap. Coordination in industry is often solved with dirty hacks. There is no general methodology. Techniques and concepts are needed.We feel we really need an agreement about the kind of high level concepts that define the coordination. In academia we often program just single agents instead of defining the MAS from the very beginning. Once the language is chosen, the coordination problem needs to be worked out and different coordination solutions can be compared. We need some kind of *coordination engineering.*

To sum up, we considered the following tasks to be particularly important:(1) to develop coordination mechanisms for large distributed open systems,(2) to take *runtime organisation* seriously, (3) to develop platforms that incorporate coordination/organisation support,(4) to make an organisation live as a distributed system (this is not just the design of the organisation), (5) to develop both top-down as well as bottom-up methods (from agents to organisations and back).

4.3 Tools, Languages and Technologies

Tooling and programming languages for multiagent systems are very important for the uptake of the multiagent programming paradigm. In particular, the need for more sophisticated approaches and tools for testing and debugging was clear to the participants of the seminar. Multiagent systems pose many new challenges in this area. First of all, the behavior of agents is often dynamic and may change over time. Multiagent systems are also typically distributed systems which introduces additional challenges. Multiagent systems, moreover, are used to control complex, dynamic and non-deterministic environments. Such environments do not support, for example, easy replay of one and the same test case. A key challenge therefore is to establish an approach for testing such complex systems. In order to manage this complexity tests are needed at different levels of a multiagent system similar to unit and integration tests in more traditional object-oriented approaches. There is a need to identify the equivalents of these tests for agent technology. Different techniques may need to be introduced such as mock agents for protocol testing. For example, it was felt that a language for expressing test cases may advance the state of the art significantly. Such a language would need to provide support for defining the state of the agents' environment and for the state of the agents themselves.

A need for test beds that are made widely available for collecting data on and for comparing various platforms was also identified. The variety of platforms available for engineering multiagent systems for developers raises the issue of which to choose. Various benchmarks should be developed to identify the benefits and weaknesses of platforms. As a community, we should agree on a list of common benchmarks that relate to specific aspects of a multiagent system. Relevant aspects that are specific for multiagent systems include, for example, components of agents such as percept processing, belief revision, and intention reconsideration, as well as more general aspects such as reactivity and scalability. An important issue is how to ensure that similar things are measured in different platforms. One solution would be to use standardised interfaces for e.g. connecting to environments. It was suggested to create and use a web portal to publish and discuss benchmarks to make progress in this area.

A related but different topic concerns the usability of different agent platforms. The learning curve associated with one platform may differ greatly from that of another, while usability from the perspective of the capabilities offered by the latter platform may be rated higher by expert users than that of the former platform. Methodologies are needed to be able to perform systematic studies into usability aspects. Here both qualitative as well as quantitative techniques will need to be used, in particular at this stage of the research where only few studies are available that look at usability issues. As in the case for benchmarks, a range of different tasks will need to be designed that can be used as test cases in usability studies. Moreover, the skill and experience level of users will need to be taken into account. In order to be able to study differences between platforms we also need to develop techniques for comparing different solutions programmed in different agent programming languages (e.g. a simple comparison of lines of codes will not do). Usability studies may also be used to enhance teaching and improve courses on how to engineer multiagent systems. This could even lead to an increase of the number of universities that adopt agent technology and programming languages in courses related to engineering intelligent and multiagent systems.

One of the main contributions of the multiagent programming paradigm is to introduce a new set of abstractions for programming software systems. Apart from the notion of an agent, the focus of the multiagent paradigm on concurrent and event-driven programming may provide the proper level of abstraction for programming distributed systems by abstracting away low-level concerns related to, for example, threads. If research in our community would focus more on these aspects this could possibly lead to (re)connecting multiagent programming to mainstream computing science research on related issues. Also, at the conceptual level, multiagent systems raise important new issues such as how to incorporate norms and program the organisational structure of a multiagent system. Finally, various more technical challenges need to be faced relating to the scalability of multiagent systems. The management of huge number of agents in, for example, large-scale (cognitive) agent-based simulations remains an important challenge that needs to be addressed.

4.4 Component-Based Agent Design

The working group explored the hypothesis that the monolithic nature of many current agent programming languages and platforms is both a barrier to the adoption of agent technology in "mainstream" software development and industry and an impediment to research, and results in a dilution of effort in the development and maintenance of agent platforms, with useful new features or capabilities being reimplemented for different platforms rather than being improved.[1]

Feedback from industrial participants and academics working on large scale deployed applications stressed both the relatively small size of the "agent component" in many systems employing agent technology, and the need for the agent components to integrate with existing software engineering methodologies and tools (see Section 4.1.3). In this context, both the overarching agent-centric nature of many agent development methodologies and the monolithic nature of agent platforms are an issue. In some cases, ideas prototyped in an agent programming language/platform have been re-implemented using "traditional" software development methodologies and languages when the system is deployed, either to facilitate integration and maintenance of the agent components by traditional software developers or because the overheads of a complex agent platform could not be justified when only a subset of its capabilities is required. A more modular approach would address these concerns, by allowing developers to directly incorporate only those features that are required for a particular application ("allowing agent language complexity to be application specific"). In addition, the overhead of learning new agent technologies is also reduced for mainstream developers and in undergraduate teaching (seen as a barrier for many). Only those components and APIs relevant to the current project must be mastered, facilitating a piecemeal, demand-driven integration of agent technology, starting with simple applications of agents, e.g., simple decision making, and progressively expanding outwards to more complex capabilities, e.g., negotiation, as developers gain experience with, and confidence in, agent technology.

A more component-based approach would facilitate research, particularly at the single-agent level, where considerable work remains to be done. Currently, extending a feature of an agent language or platform, such as extending deliberation to incorporate reasoning under uncertainty, or adding a new feature, such as learning, involves mastering the details of the agent platform, and often requires a project of PhD length. This hinders innovation and makes it difficult to create ad-hoc prototypes, e.g., to demonstrate the benefits of agent technology to other communities. A more modular approach with standardised interfaces would address these concerns by allowing researchers to target a single component or small

[1] Component-based agent software engineering was also discussed in the Integration and Validation group.

number of components, rather than the agent platform as a whole. The loose coupling inherent in a component-based approach may also facilitate the development of novel agent architectures incorporating, e.g., multiple asynchronous deliberation cycles, or concurrent reactive and deliberative cycles. This may in turn help "bridge the gap" between architectures for software agents and those found in autonomous robots such as UAVs and spacecraft. Much can be learned from the experiences of the robotics community, which is coalescing around component-based platforms such as ROS that provide libraries and tools to help researchers and software developers create robot applications.[2] A more modular approach also raises novel short-term research challenges at the component integration level. Key issues include interfaces between components (should these be based on standard languages for representing beliefs, goals and plans, or on syntax neutral approaches based on queries), how coordination between components can be coordinated, and how such coordination rules can best be expressed. In the medium to longer term, componentisation of agent designs should foster the development of a reference model for agent technology. Such a reference model would be a powerful tool both for structuring agents research, and in clarifying to the mainstream software development community that "agents" represents a suite of technologies that can be adopted as needed in applications.

Lastly, it was argued that component-based approaches would also help agent technologies achieve critical mass, both within the agent programming community and, more generally, in the mainstream software development community. The relatively small agent programming community is currently structured around several competing agent programming languages and platforms. While this has been very successful in driving innovation, useful innovations must be re-implemented for each platform (at considerable cost) rather than effort being concentrated on expanding and improving innovative features and their documentation. Focussing on common components would lead to more rapid advances in "whole platform" capabilities (since features no longer have to be re-implemented), promote standardisation regarding key concepts and technologies, and should result in higher quality implementations more likely to be adopted by mainstream developers. Again, there is much that can be learned from recent developments in the robotics community and in other related communities, such as computer vision and the re-use of high-quality BDD libraries in model checking.

A key challenge in adopting a component-based approach is to identify and develop components and their APIs. Fortunately, there is a pool of existing agent platform implementations that can serve as a basis for components, and the agent development community has already made some initial steps in the direction of common interfaces, such as the environment interface standard,[3] and modularisation (within a single platform) is now common. However much remains to be done. Another key challenge is in the development of middleware to support the interaction of components, and how this interaction can best be specified. This area is less explored, but even here there is preliminary work on which the community can build.

[2] www.ros.org
[3] http://sourceforge.net/projects/apleis

5 Open Problems

- While much research will continue to be devoted to foundational work, there needs to be an increased appreciation within the community of the challenges involved in engineering large-scale multiagent systems. Agents conferences and workshops should encourage submission and acceptance of papers that address these concerns, and should work towards setting and maintaining standards to ensure that work of this kind is of high quality.
- From the industry perspective, although it is very clear what steps can be taken to facilitate the deployment of agent technology, it is uncertain (a) which organisations are best capable of doing this work, (b) how this work will be funded, and (c) whether the needs of end users are sufficient to provide the impetus for the work to be done commercially.
- There remain significant technological barriers to the deployment of multiagent systems which requires research into new techniques, lessons learned from applications, and more generally software engineering type of papers that use existing agent technologies (and not common languages such as Java to build multiagent systems).
- Another challenge that remains is to identify the application areas and types of applications where agent technology provides a critical advantage (such as "autonomous decision making" or explicit multiagent coordination mechanisms), and if possible, to quantify the benefits of using the technology.

6 Panel Discussions

A plenary session was organised on Friday, the last day of the seminar, in which summary reports of the four groups were presented and discussed. The purpose of this session was to identify key challenges and ideas for future research based on discussions during the seminar.

One of the ideas that repeatedly came up during discussions relates to the development of a *modular* or *component-based* agent architecture. The idea is that engineering multiagent systems may be facilitated by a set of components that can be relatively easily exchanged and reused between agent-based applications. Developers within industry may be interested in using some instead of all components of existing agent architectures. Moreover, developing such components may also give rise to some degree of standardisation. It may also give rise to a reference model for agent technology. The main challenge here remains to identify and develop these components.

A related topic concerns the need to continue *research at the single agent level.* The notion of BDI+ was coined to refer to the need to integrate, for example, emotions in a more systematic way into agent architectures. Another example in this area concerns learning. Integrating learning into the agent architecture raises new and interesting challenges that are different from the typical issues studied in the machine learning community. Another challenge is to design new components that extend the capabilities of agents in order to support, for example, reasoning under uncertainty. Finally, more research is needed on the capability of agents to explain their behavior which not only may provide a selling point for the technology but also may be used in debugging tools to identify the reasons for observed behavior.

Another topic that has been quite extensively discussed during the seminar and obtained quite a lot of support as a topic for the research agenda concerns *metrics* and the development of *benchmarks* for agent technology. Some agent programming languages and frameworks may, for example, facilitate the design of scalable systems. But how do we identify these languages and the features that support the engineering of scalable multiagent systems? Are there specific metrics that apply to multiagent systems. How can we measure, for example, concepts that are often mentioned in the literature such as believability and flexibility of agents?

Tooling has been identified as a main topic for future research as it is very important for the uptake of any technology. The application of agent technology in commercial and business applications requires integration of this technology into the full software life cycle. However, we should not reinvent but rather reuse techniques and tools wherever possible. More research is needed to more clearly identify where tooling developed within the more broader software engineering community can be used to provide this support and where agent-specific tools are needed. Generally, a need was felt to focus on debugging support initially as providing assurance for a multiagent system seems to be of key importance. Moreover, it may be useful to connect to and integrate the work from the Agent-Oriented Programming and Software Engineering communities better.

Participants

- Natasha Alechina
University of Nottingham, GB
- Jeremy Baxter
QinetiQ - Malvern, GB
- Michal Bida
Charles University - Prague, CZ
- Olivier Boissier
Ecole des Mines - St. Etienne, FR
- Rafael H. Bordini
PUCRS - Porto Alegre, BR
- Lars Braubach
Universität Hamburg, DE
- Paolo Busetta
AOS Ltd. - Cambridge, GB
- Rem Collier
University College Dublin, IE
- Stephen Cranefield
University of Otago, NZ
- Mehdi Dastani
Utrecht University, NL
- Louise Dennis
University of Liverpool, GB
- Virginia Dignum
TU Delft, NL
- Jürgen Dix
TU Clausthal, DE
- Jorge J. Gomez-Sanz
Univ. Comp. de Madrid, ES
- Christian Guttmann
IBM R&D Labs; AU; EBTIC Abu Dhabi, AE; Monash University, AU
- Axel Heßler
TU Berlin, DE
- Koen V. Hindriks
TU Delft, NL
- Tom Holvoet
KU Leuven, BE
- Jomi Hübner
Federal University of Santa Catarina - Brazil, BR
- Yves Lespérance
York University - Toronto, CA
- Brian Logan
University of Nottingham, GB
- John-Jules Ch. Meyer
Utrecht University, NL
- Berndt Müller
University of Glamorgan, GB
- Jörg P. Müller
TU Clausthal, DE
- Alexander Pokahr
Universität Hamburg, DE
- Alessandro Ricci
University of Bologna, IT
- Andrea Santi
University of Bologna, IT
- Federico Schlesinger
TU Clausthal, DE
- Amal El Fallah Seghrouchni
UPMC - Paris, FR
- Maarten Sierhuis
Ejenta Inc., US
- Marija Slavkovik
University of Liverpool, GB
- Bas J. G. Testerink
Utrecht University, NL
- Birna van Riemsdijk
TU Delft, NL
- Jørgen Villadsen
Technical University of Denmark - Lyngby, DK
- Michael Winikoff
University of Otago, NZ
- Cees Witteveen
TU Delft, NL
- Wayne Wobcke
UNSW - Sydney, AU