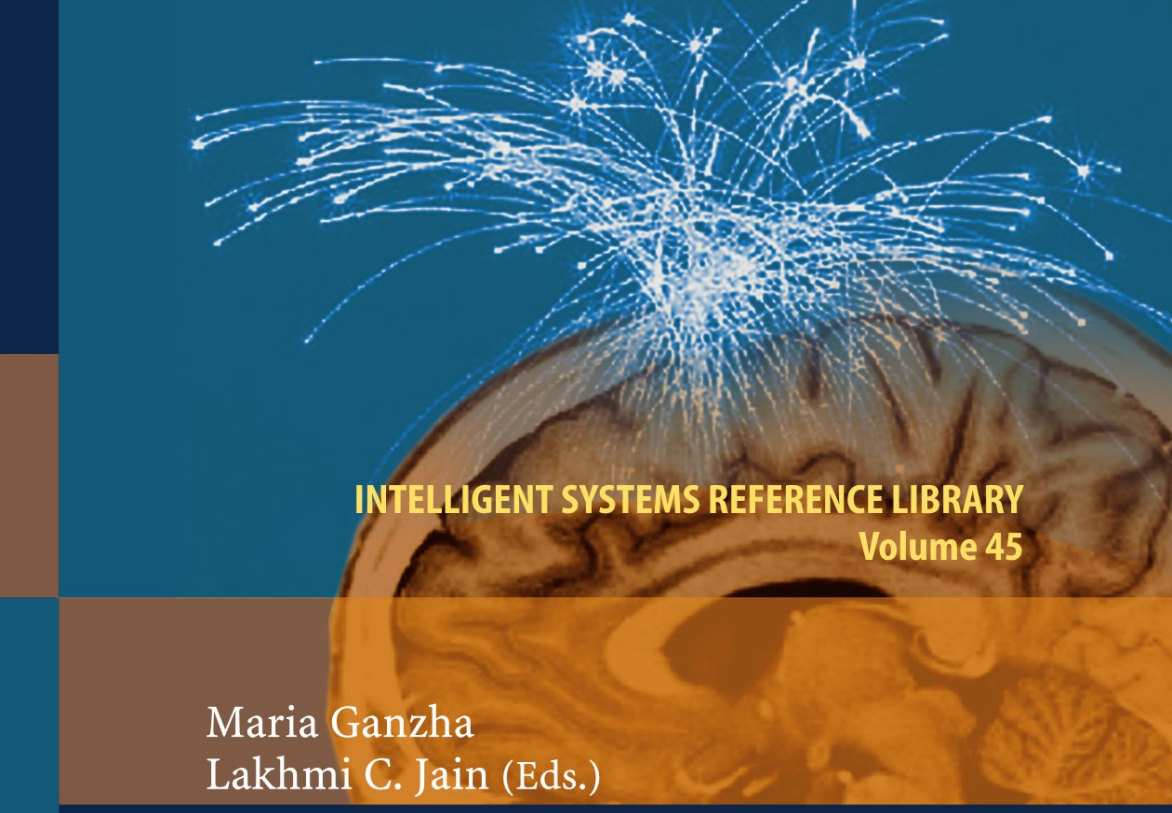

INTELLIGENT SYSTEMS REFERENCE LIBRARY
Volume 45

Maria Ganzha
Lakhmi C. Jain (Eds.)

Multiagent Systems and Applications

Volume 1: Practice and Experience

Springer

Intelligent Systems Reference Library 45

Editors-in-Chief

Prof. Janusz Kacprzyk
Systems Research Institute
Polish Academy of Sciences
ul. Newelska 6
01-447 Warsaw
Poland
E-mail: kacprzyk@ibspan.waw.pl

Prof. Lakhmi C. Jain
School of Electrical and Information Engineering
University of South Australia
Adelaide
South Australia SA 5095
Australia
E-mail: Lakhmi.jain@unisa.edu.au

For further volumes:
http://www.springer.com/series/8578

Maria Ganzha and Lakhmi C. Jain (Eds.)

Multiagent Systems and Applications

Volume 1: Practice and Experience

Springer

Editors
Prof. Dr. Maria Ganzha
University of Gdańsk
Institute of Informatics
Gdańsk
Poland

Prof. Dr. Lakhmi C. Jain
School of Electrical and Information Engineering
University of South Australia
South Australia
Australia

ISSN 1868-4394
ISBN 978-3-642-33322-4
DOI 10.1007/978-3-642-33323-1
Springer Heidelberg New York Dordrecht London

e-ISSN 1868-4408
e-ISBN 978-3-642-33323-1

Library of Congress Control Number: 2012946744

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Foreword

Dear Reader!

Before you start delving into this very interesting book, let me remind you few historical facts. In 1994 P. Maes suggested that software agents should cure the information overload [4]. In her seminal work, she has introduced *personal agents* capable of helping users with management of e-mails and news, and with meeting scheduling. Next, between 1998 and 2005, the EU-funded AgentLink Coordinated Action has been overseeing the development of agent technologies (reaching far beyond the European Research Area). In the meantime, in 2001, N. Jennings has forcefully argued that software agents will facilitate a revolution in software engineering [2]. His reasoning focused on large complex systems of the future. More recently, concepts related to software agent research, start to materialize in movies like "Avatar" or "Surrogates," as well as in virtual worlds (e.g. the "Second Life") and in games like the "World of Warcraft." There, software entities representing humans ("avatars") interact in a mixture of real and virtual worlds.

In this context, one could ask: how close are we to the world filled with personal avatars? (Un)fortunately, when we wake up, we do not talk to our Personal Avatar Lars, who has been working overnight to provide us with everything that we always wanted, but were afraid to ask (as only Lars truly understands our believes, desires and intentions). Obviously, an even more general question arises: what is the current status of agent systems?

A meta-reflection concerning any scientific discipline can be approached from multiple angles. Here, let us use two main perspectives of the philosophy of science. Rationalists, like K. Popper, would probably be quite happy, as for them the key aspect of science is creation of interesting theories. For instance, agent researchers should spend time in their armchairs conceiving all possible agent platforms that could be created. As a matter of fact, this goal has been realized with a great success, as in 2005 already more than 80 different agent platforms existed. While this approach seems to be preferred by agent researchers working in the academia (as illustrated by the second chapter of this book), it has limited practical value. It tells us nothing about creation of real-world personal agents, or having intelligent homes run by a JADEX-based agent system. Therefore, let us turn to the empirical tradition. Here, F. Bacon would say, that to know software agents is to use them. This is

also the perspective assumed by this book. In this way it not only becomes a continuation to the work of R. A. Belecheanu et.al. [1], and to the Agent Computing Roadmap [3], but also follows one of more controversial overview papers concerning agent systems [5]. In their work, H. Nwana and D. Ndumu stated that more **empirical work** is needed in the area of agent system research. It is only through development of (prototypes of) large-scale systems it will be possible to learn the actual pros and cons of agent technologies [3].

In this context, the book at hand constitutes the 2011 snapshot of the state-of-the-art of agent system development. Following the chapter analyzing the content of key journals and conferences related to agent systems, the remaining 9 chapters present (a) three agent platforms, and (b) applications of software agents. The later part can be divided between "agents in simulations" and "other agent applications." The most important point about these chapters is that they are based on the *actual implementations* of agent platforms and systems. This gives their authors an opportunity to reflect on the *lessons learned*. Furthermore, in the introductory chapter, book editors collect most *important issues* to be resolved to take agent technologies to the next level of applicability.

As every snapshot, this one also has its shortcomings. It would be possible to point out to the developments that are missing. Nevertheless, this is a very good book, worthy reading by both specialists in the field and those thinking about starting agent systems research. Finally, as what concerns future of agent systems, I am particularly glad that the need for reviving efforts of FIPA is brought forward. However, as D. Rosengarten said: "life is a matter of taste." While this particular point may be particularly close to my likings, I hope that all readers will find this book very tasty, indeed.

Professor Marcin Paprzycki
Systems Research Institute
Polish Academy of Sciences
Poland

References

[1] Belecheanu, R.A., Munroe, S., Luck, M., Payne, T., Miller, T., McBurney, P., Pechoucek, M.: Commercial Applications of Agents: Lessons, Experiences and Challenges. In: Proceedings of the Fifth International Conference on Autonomous Agents and Multiagent Systems, Hakodate, Japan (2006)

[2] Jennings, N.R.: An agent-based approach for building complex software systems. CACM 44(4), 35–41 (2001)

[3] Luck, M., McBurney, P., Shehory, O., Willmott, S.: Agent Technology: Computing as Interaction (A Roadmap for Agent Based Computing). AgentLink (2005) ISBN 085432 845 9

[4] Maes, P.: Agents that Reduce Work and Information Overload. CACM 37, 31–40 (1994)

[5] Nwana, H.S., Ndumu, D.T.: A *perspective on software agent research*. The Knowledge Engineering Review 14(2), 125–142 (1999)

Preface

According to the Gartner's famous Hype Cycle for Emerging Technologies (http://en.wikipedia.org/wiki/Hype_cycle) each new technology goes through five phases, from the Technology Trigger, through the Peak of Inflated Expectations, Trough of Disillusionment, Slope of Enlightenment, to the Plateau of Productivity. This cycle should also be applicable to the software agents and the agent technologies. In this context, one of the ways of tracking the progress of agent technology would be by the results delivered by the three phases of the EU-funded AgentLink Coordinated Action, which lasted between 1998 and 2005 (www.agentlink.org). Its fundamental deliverable was the Agent Technology Roadmap ([2]). In this publication, on pages 71-74, a number of predictions were made, concerning the advances and the penetration of the agent technologies. However, the most important source of inspiration for this book was the seminal paper published in 2006 ([1]). This paper (also an outcome of the AgentLink action) contains a comprehensive summary of the state-of-the-art of the use of agent technologies in real-world applications. Five years later, in 2011, we have decided to assess the progress, which has been made in the use of software agents in actual applications. To achieve this goal we have invited contributions from the leading experts in the field. We have asked them to describe their implemented applications, to reflect on the process, and to summarize the lessons learned. We have also requested their perspective on the future developments of the agent technology, including the most important challenges that lie ahead.

Synopsis of Book Chapters

This book contains 10 chapters. The first of them (by Balke et al.) provides an interesting context for the whole book. It investigates the status of agent applications in most important agent-related scientific conferences and journals. The main finding is that, in the academic work, the results concerning the actual implementations of agent systems are not given the same recognition as theoretical / foundational work. This is particularly interesting since, as can be found in the remaining nine chapters,

development of both agent tools and applications requires a search for completely novel solutions based on close collaboration between theory and practice. Therefore, cross-feeding between them would be very valuable. In this context, it is worthy to mention work of H. Nwana and D. Ndumu who, in 1999, strongly argued that more practical work is needed to take the agent system development to the next level [3]. Unfortunately, as Chapter 2 shows, this call has not been heard.

Obviously, one of the crucial elements to be able to actually implement agent-based applications is to use agent tools / environments / platforms. Furthermore, observe that agent technology differs from others due to the fact that agent collaboration is usually achieved via messages exchanges, while individual agents are independent software elements that encapsulate autonomic behaviors. This is why it is so important to have in place technologies that support such programming paradigm. On this account, the next three chapters concern the three agent platforms.

Chapter 2, by Pokahr et al., is devoted to a platform allowing development of agent systems based on the, well-known, Belief-Desire-Intention (BDI) approach. BDI agents are characterized by mental-like states with three components: what agent knows (beliefs), what agent can do/fulfill (desires), and what agent strives to achieve (intentions). The beliefs of an agent are its model of the domain it operates in, while desires provide ways of dealing with in incoming events (based on the agents' knowledge). It is worthwhile stressing that, in the BDI model, it is usually assumed that agents' knowledge about the domain (environment) is repeatedly updated on the basis of the information originating from the environment, deciding on which options are available, filtering these options to determine new intentions, and acting on the basis of these intentions.

Jadex is a software platform created to implement the BDI agents. Interestingly, besides cognitive agents, which are particularly suited for complex tasks, Jadex directly supports simple reactive micro agents (so-called active components). These agents are similar to active objects, and are very efficient in terms of low resource consumption. Furthermore, Jadex has been developed with the goal to connect agents tighter to the existing, well-established approaches. Specifically, it is striving to allow inexperienced agent developers, who have experience in Java and XML programming, to develop BDI-based agent systems.

The next chapter (Chapter 3, by Vidakovic et al.) describes an effort to fully utilize Java capabilities to implement software agents. Specifically, the main motivation behind the development of the Extensible Java Enterprise Edition-based Agent Framework (XJAF) was to demonstrate how existing, standardized Java EE technologies, tools, and libraries, such as JNDI, JMS, and EJB, can be used to implement a large subset of functionalities required in the multi-agent systems.

The direct benefits of this approach were, among others, a shorter development time of software systems, the delegation of agent load-balancing to the enterprise server, and a flatter learning curve for new developers of the system. Furthermore, based on common tendencies in software development, recent versions of the XJAF added some new functionalities, e.g. the SOA-based design and the Web Service interfaces.

Particularly interesting is the fact that, before starting to develop their framework, its authors analyzed the most important advantages and disadvantages of other platforms. Based on this research, the XJAF developers try to eliminate the existing "weak points." For instance, in most cases, the MAS solutions do not provide any runtime load-balancing techniques. These few that do, implement their own algorithms from the scratch. In the case of the XJAF, developers can utilize the capabilities of the Java EE. Specifically, the XJAF wraps agents inside of the EJB components and delegates their life-cycle management to the enterprise application server. As a result, with a minimum amount of programming effort, runtime load-balancing of agents and an improvement of the overall performance of the system are achieved.

The last chapter dealing with agent frameworks is devoted to the newest product from the developers of the most popular agent system of today (JADE). This product is a JADE add-on: WADE (Workflows and Agents Development Environment; Chapter 4 by G. Caire). The use of WADE is illustrated through the large agent-based application, which was developed and successfully deployed by the Telecom Italia, in the field of fixed network monitoring and optimization (the Wants-Assurance system). At the beginning of this chapter, readers can find a comprehensive description of WADE, which is a domain independent software platform built on top of the JADE agent platform. This chapter also introduces some important facts about the xDSL application, utilized in the Wants-Assurance system. Its final part is devoted to the details of the development and implementation of the Wants-Assurance application. It is worthy noting that, thanks to its natively distributed agent-based architecture, the Wants-Assurance system continuously (in real-time) monitors about 3.000.000 of xDSL lines.

In this way, Chapter 4 becomes a bridge between the technology and the application chapters, which constitute the remaining parts of the book. In the, above mentioned, overview paper [1] it was clearly illustrated that one of the key areas where agent systems have been extensively used was simulation. Thus, the next three chapters illustrate the current state-of-the-art concerning use of software agents in simulations.

The first chapter of this part of the book was prepared by Braubach and Pokahr and is devoted to the specific characteristics of the JADEX platform, which allow it to support simulation. One of the key issues is: how JADEX helps to establish *simulation transparency* (here, transparency means independence from the simulation specific aspects), which makes it easy to transform a simulation into an application. In the chapter, the usefulness of simulation itself and of simulation techniques applied as part of the application development process, are clearly illustrated. The second addressed issue is that of the *simulation time*. Two possible ways to solve this issue have been discussed and illustrated by the examples. The last fundamental issue addressed in this chapter is the *virtual environmen*t (external environment used to run the simulations). Note that, during the application development, simulation allows intensive testing of the whole application (or some of its specific parts) before putting it into the production environment. Furthermore, the virtual environment allows one to benchmark different implementations in the same environment,

or test implemented system behavior in the changing environments. In the chapter, the specific framework for developing virtual environment extending JADEX, the EnvSupport, is described in details.

The second simulation-related chapter concerns the use of agents in the simulation of cyber attacks. Here, Rafał Leszczyna discusses system, which has been created to study various possible attacks on the IT infrastructures of power plants. Material presented in this chapter matches with and extends the discussion of the virtual environment, introduced in the previous chapter. What is particularly interesting is the fact that the performed simulations allowed to prevent actual attacks, details of which can be found in the chapter.

The next chapter (Chapter 7) is devoted to the interrelated and already completed projects named "Tactical Agent-Fly" and "Tactical AgentScout." Furthermore, it outlines the core objectives of an on-going follow-up project the "AgentFly-In-Air." All these projects deal with cooperation and coordination issues in the multi-robot teams. These teams either carry tactical missions in urban warfare scenarios, or provide information to human troops on the ground. In the chapter, discussion of architectural and technological issues related to the development of the multi-agent platform and the simulation subsystem used in these projects is given. The content of the chapter describes, in chronological order, the co-evolution of the tools and environments and of the subsequent projects undertaken by the authors.

The final part of the book contains three chapters devoted to implementing applications using software agents. In the first chapter (Chapter 8), a novel application field for software agents is a rapidly advancing area of wireless sensor network (WSN). Applications using a combination of both technologies can be found, among others, in e-health, ambient assisted living, or smart environments. Chapter by Fortino and Galzarano contains a discussion of fundamental issues related to merging of software agents and wireless sensors. The analysis presented in this chapter points to some important concerns. For instance, WSN technologies limit agent mobility (it is practically impossible to use a single mobile agent to service multiple sensors – each sensor has to have its own local agent). Furthermore, as of today, development of agent-WSN systems is hindered by the lack of suitable tools and methodologies, capable of dealing with resulting complex systems. In this context, the Platform Based Design (PBD) methodology, proposed in the chapter, can be seen as a step in the direction of solving this problem. In order to discuss the PBD methodology, the TinyMAps platform is introduced and confronted with an earlier MAPS platform. The TinyMaps is Java-based mobile agent system designed for the WSNs with extremely limited hardware resources (e.g. the Sentilla JCreate). A comparative analysis of MAPS and TinyMAPS is provided, showing analogies and differences among the two platforms. It is facilitating the context for the issues involved in designing agent-WSN systems. Finally, a comparison of MAPS with AFME, and other Java-based mobile agent systems running on the SunSPOT devices, and based on different architectures and programming models, is presented.

The next chapter of the book (Chapter 9, by Morge, et.al.) is devoted to agent negotiations and, in this context, argumentation. The focus of the chapter is on how to compose services by negotiating the terms and the conditions between potentially

participating services (e.g. to establish time of use and its duration, according to the requirements and goals both of the clients and of the service providers). The goal of the negotiations is to reduce possible conflicts and to support a successful service composition. The agent model supporting the needed functionalities has been developed within the EU-funded ArguGRID project, as a multi-agent platform. In the chapter one can find a description of this platform as well as more general discussion of the negotiation process and the used argumentation mechanisms.

The final chapter, by Casalicchio and Tucci, is devoted to use of software agents in re-engineering public administration workflows. The focus of the chapter is on the experiences gathered by the authors in the design and implementation of an agent-based modeling and simulation framework. This was used to support the re-engineering of public administration workflows. The project, which started in late 2003, was to analyze and evaluate the performance of the IT infrastructure management and the service provisioning process at the Italian Prime Minister Office for Informatics and Telematics, headed by the second author. On the basis of the solution initially developed to solve the specific problem, a general framework to support public administration processes re-engineering was created. This framework, named Wf-Simulator, has been successfully used in real workflow modeling and simulation. The chapter describes the initial project, the Wf-Simulator framework, and three real-world case studies: service provisioning in public administration, day hospital surgery admission, and blood examination management.

Lessons Learned

Let us now briefly summarize the some of the most important lessons pointed to by the authors of all chapters:

1. Lack of attention to agent-based applications among agent researchers is slowing down advances in "multi-agent system science" as well as their acceptance in real-world applications.
2. FIPA standards developed about 10 years ago for agent system design and implementation have to be revisited, and their new release should take into account advances in agent-based computing that took place in the meantime.
3. There is a need for development techniques that would allow for early testing and validation of application design and implementation.
4. Importance of agents in simulations is increasing, as they are an ideal tool to combine different software artifacts into a common system.
5. Agent technology provides important features such as reasoning, planning and more in general "intelligence" that allow agent-based systems to deal with unforeseen situations. Moreover, it seems that the more complex the situation is, the more successful could be use of agent technology.
6. Unfortunately, the knowledge engineering process can become complex when dealing with multiple and possibly heterogeneous knowledge representations. Utilization of semantics is still a weak point of agent-based applications.

7. Agents are becoming more-and-more "talkative." The initial idea that agents communicate via messaging is systematically extended by turning simple message exchanges into elaborate negotiations with argumentation.
8. An important unsolved issue is the human-agent relations. For instance, while in the academia software agents are to be more-and more autonomous in dealing with real world-problems, today in real-world agent autonomy is not a realistic option; e.g. nobody will allow agents to actually represent humans and make autonomous decisions with potential serious consequences.

References

[1] Belecheanu, R.A., Munroe, S., Luck, M., Payne, T., Miller, T., McBurney, P., Pechoucek, M.: Commercial Applications of Agents: Lessons, Experiences and Challenges. In: Proceedings of the Fifth International Conference on Autonomous Agents and Multiagent Systems, Hakodate, Japan (2006)

[2] Luck, M., McBurney, P., Shehory, O., Willmott, S.: Agent Technology: Computing as Interaction (A Roadmap for Agent Based Computing). AgentLink (2005) ISBN 085432 845 9

[3] Nwana, H.S., Ndumu, D.T.: A perspective on software agent research. The Knowledge Engineering Review 14(2), 125–142 (1999)

Resources

The following resources are recommended to explore the field of intelligent systems further. This list is neither complete, nor exclusive. It can be seen as a starting point to explore the field further.

Journals

- International Journal of Knowledge-Based and Intelligent Engineering Systems, IOS Press, The Netherlands, http://www.iospress.nl/loadtop/load.php?isbn=13272314
- Multiagents and Grid Systems: An International Journal, IOS Press, The Netherlands, http://www.iospress.nl/journal/multiagent-and-grid-systems/
- Autonomous Agents and Multi-Agent Systems Journal (JAAMAS), Springer
- see table 8, Chapter 1

Conferences

- International KES Conference on Agents and Multi-agent Systems –Technologies and Applications (AMSTA KES)
- International Conference on Autonomous Agents and Multiagent Systems (AAMAS)

- International Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS)
- German Conference Series on Multiagent System Technologies (MATES)
- Joint Agent-oriented Workshops in Synergy (JAWS)
- International Workshop on Agent-Oriented Software Engineering

Books

- Bellifemine, F., Caire, G. and Greenwood, D., Developing Multi-Agent Systems with JADE, John Wiley & Sons, 2007
- Bigus, J.P. and Bigus, J., Constructing Intelligent Agents Using Java: Professional Developer's Guide. Wiley, New York, 2nd edition, 2001.
- Cervenka, R.and Trencansky, I., The Agent Modeling Language – AML, Birkhäuser, 2007
- Essaaidi, M., Ganzha, M. and Paprzycki (Eds), M., Software Agents, Agent Systems and Their Applications, IOS Press, 2011
- Jain, L.C., Chen. Z. and Ichalkaranje, N. (Eds), Intelligent Agents and Their Applications, Springer-Verlag, Germany, 2002
- Jain, L.C. and Nguyen, N.T. (Eds), Knowledge Processing and Decision Making in Agent-Based Systems, Springer-Verlag, Germany, 2009.
- Jarvis, J., Ronnquist, R, Jarvis, D. and Jain, L.C., Holonic Execution: A BDI Approach, Springer-Verlag, 2008.
- Jarvis, D., Jarvis, J. Ronnquist, R, and Jain, L.C., Multiagent Systems and Applications, Volume 2: Development using the GORITE BDI Framework, Springer-Verlag, 2013.
- Khosla, R., Ichalkaranje, N. and Jain, L.C. (Eds), Design of Intelligent Multi-Agent Systems, Springer-Verlag, Germany, 2005
- Nguyen, N.T. and Jain, L.C. (Eds), Intelligent Agents in the Evolution of Web and Applications, Springer-Verlag, Germany, 2009.
- Phillips-Wren, G. and Jain, L.C. (Eds), Intelligent Decision Support Systems in Agent-Mediated Environments, IOS Press, The Netherlands, 2005.
- Resconi, G. and Jain, L.C., Intelligent Agents: Theory and Applications, Springer-Verlag, Germany, 2004
- Russell, S.J. and Norvig, P., Artificial Intelligence: A Modern Approach, Prentice Hall, Pearson Education, Inc., Upper Saddle River, NJ, USA, 2nd edition, 2003.
- Srinivasan, D. and Jain, L.C. (Editors), Advances in Multi-agent Systems and Applications, Springer-Verlag, 2010
- Tweedale, J. and Jain, L.C., Embedded Automation in Human-Agent Environment, Springer-Verlag, 2011
- Uhrmacher, A.M. and Weyns, D. (Eds), Multi-Agent Systems: Simulation and Applications, CRC Press, 2009
- Wooldridge, M., An Introduction to Multi Agent Systems, Wiley, 2009

We would like to send out special thanks to Professor Michael Luck for the discussions, while we were working out details of the concept of the book. They turned out to be extremely illuminating. Thank you Michael!

We are also grateful to Professor Marcin Paprzycki for his constant support and ideas, and his very unique foreword.

This book would not have existed without the tremendous contribution by the authors and the reviewers. We remain grateful.

Finally, thanks are also due to the Springer-Verlag for their excellent support during the preparation of the manuscript.

Maria Ganzha
Institute of Informatics
University of Gdańsk
ul. Wita Stwosza 57
80-952 Gdańsk
Poland

and

Systems Research Institute
Polish Academy of Sciences
ul. Newelska 6
01-447 Warszawa
Poland

Lakhmi C. Jain
School of Electrical and Information Engineering
University of South Australia
Mawson Lakes Campus
South Australia SA 5095
Australia

Contents

Editors

Dr. Maria Ganzha is Assistant Professor in the Institute of Informatics, University of Gdańsk, Poland, as well as the Systems Research Institute Polish Academy of Sciences. She completed her Masters' and Doctorate degrees from the Moscow State University in 1987 and 1994, respectively.

Currently her research interests include software engineering, distributed computing, ontology, semantic data processing, and agent-based systems in particular. Results of her research have been published in about 120 peer-reviewed scientific journals, monographs, as well as conference proceedings.

She has been supervising Ph.D. and MsC students in agent-based system, ontology and web services. With her team, she has been developing a prototype of an agent-based system managing resources in the Grid. Dr. Ganzha was the guest-editor of special issues published by journals such as Science of Computer Programming, Journal of Network and Computer Applications (Elsevier) and is on editorial boards of such journals as Computing Now (IEEE), Informatica, The International Journal of Computer Science & Applications, and the International Journal on Information & Communication Technologies.

Professor Lakhmi C. Jain is a Director/Founder of the Knowledge-Based Intelligent Engineering Systems (KES) Centre, located in the University of South Australia. He is a fellow of the Institution of Engineers Australia.

His interests focus on the artificial intelligence paradigms and their applications in complex systems, art-science fusion, virtual systems, e-education, e-healthcare, unmanned air vehicles and intelligent agents.

Chapter 1
Assessing Agent Applications — r&D vs. R&d

Tina Balke, Benjamin Hirsch, and Marco Lützenberger

Abstract. This chapter focusses on the relevancy of implementations of software systems in which agent technology was actually used in agent-oriented conferences and journals. It discusses the discrepancy between the stated aims of agent research with regard to agent applications, and the reality as found in many (general) agent conferences as well as journals. We demonstrate this discrepancy by analysing how implementations of software systems that employed agent technology are represented in publications, both in form of conference as well as journal papers. We analyse some agent related conferences and journals, and look at the distribution of application papers at past conferences/journal issues. In order to lay foundations for a discussion of the reasons for this inconsistency as well as conclusions that can be drawn from it, we analyse relevant stakeholders for agent application papers. The aim of this chapter is to provide a basis for discussing the state and status of agent applications in research oriented conferences and journals.

1 The Case

It is generally agreed that agent technology is a topic mainly confined to the realm of research. From key note speeches at major conferences to the

Tina Balke
Centre for Research in Social Simulation, University of Surrey
e-mail: t.balke@surrey.ac.uk

Benjamin Hirsch
ETISALAT BT Innovation Center, Khalifa University
e-mail: benjamin.hirsch@kustar.ac.ae

Marco Lützenberger
DAI-Labor, Technische Universität Berlin
Faculty of Electrical Engineering and Computer Science
e-mail: marco.luetzenberger@dai-labor.de

M. Ganzha & L. C. Jain (Eds.): Multiagent Systems & Applications, ISRL 45, pp. 1–20.
springerlink.com

AgentLink roadmap [19, Chapter 5], applying agent technology, as well as developing tools and frameworks and applying agent oriented methodologies to real world problems, is one of the key drivers of advancing agent technology. Consequently, looking at the IFAAMAS[1] charter [15] for example, the following mission statements can be found:

Point 1. Promote high quality scientific research and technological practice worldwide in Autonomous Agents and Multiagent Systems, in accordance with standards of excellence and best international scientific practice, giving equal prominence to foundational, theoretical, experimental, and *applied research*.

Point 5. Foster links between the AAMAS community and the international *business community*, and national / international governments / government organisations, where such links will further the goals listed above.

Point 6. Act as an authoritative and responsible international voice for the AAMAS[2] community, informing public opinion and raising *public awareness* of this research and technology.

As these mission statements indicate, the diffusion of agent technology —that could for example be achieved through industry adoption— as well as the increase of business and public awareness for agents, are major aims of the community. In order to achieve these aims, the AgentLink Roadmap [19, p. 50] points at "applications & implementation" as one driving force required, as they attract potentials adopters of agent technology by indicating the benefits of the agent technology to them in a form that they can easily understand. Thus, research and development, or as often abbreviated R&D, should work hand in hand.

Yet, although the interplay of R&D sounds like a natural match, when it comes to conferences, R&D seem to have different perceptions and views and the balance between the two seems difficult, especially when understanding the needs of each others domain. Experience shows that papers that either focus on a particular application that uses agent technology, or deal with issues that are close to implementation issues in the context of agents —for example network problems, authorisation, or deployment— are frequently not accepted.

The rest of the chapter is structured as follows. We first define what we mean by application paper, followed by an analysis of various conferences and journals with respect to the number of application papers. Before concluding, we discuss the stakeholders involved with papers.

[1] International Foundation for Autonomous Agents and Multiagent Systems.
[2] International Conference on Autonomous Agents and Multiagent Systems.

2 Application Papers

While many papers clearly fall into the category of "theoretical" work, the question what constitutes an application paper is sometimes not easily answered. In short, we consider a paper to be an application paper if it focuses on aspects and issues regarding the implementation of agent applications. As a special case, we also count papers which describe tools for the implementation of agent applications as application papers. In this chapter, we define application papers to have one of the following characteristics:

- description of an application,
- tools designed to support applications and their actual use,
- implementation aspects of an application,
- real world issues and agent based solutions, and
- application of methodologies to real world problems

An application paper can also discuss an (agent-based) simulation, as long as the paper focuses on engineering aspects or implementation issues which evolved during the implementation or which have been solved. As opposed to that, we do not consider prototypes or toy-implementations as sufficient criterion to count as an application paper. Both, prototypes and toy-implementations are frequently used for evaluation purposes and often feature significant simplifications. This simplification is one of the reasons why proposed concepts are not easily applicable in reality, either because implementational aspects were not comprehensively discussed or because flaws in the approach did not materialise due to the simplification of the problem. At this point we want to stress that we do not question the quality of these papers in any way. It is our opinion that perspectives, concepts and initial implementations are indeed indispensible for industrial quality applications, but do not guarantee for their general applicability. The discussion of implementation and engineering aspects may raise important questions regarding the applicability of concepts and provide valuable feedback for theoretical improvements. By "application papers", we refer to papers which match with the above mentioned criteria. In the next section we now analyze how these papers are represented in the agent community by analysing several agent-oriented conferences as well as journals. Due to a lack of confidential submission information, we can only base this analysis on successfully published papers that we could access. We start with the AAMAS conference as it is highlighted as the main agent conferences by the IFAAMAS whose mission statements we quoted at the beginning of this chapter.

3 Agent-Centered Conferences

3.1 AAMAS

Having a look at the past AAMAS conferences, despite the IFAAMAS mission statements, the number of application & implementation papers is very low.

To emphasise the problem, we examined the main track proceedings [5, 10, 24, 28, 29, 30] of recent AAMAS conferences and analysed papers with a focus on applications. We respectively identified between eight and 13 papers in AAMAS editions from 2006 and 2011. Table 1 illustrates the result of our survey. It shows that during the last five years the number of full papers at AAMAS stayed more or less the same, while the number of application oriented papers dropped from about 10% to little over 6%.

Table 1 Submissions, accepted, and application papers at AAMAS conferences

			Application Papers	
AAMAS	Submissions	Acceptances	# Accepted	Acceptance (%)
2006	550	127	13	10.2
2007	531	121	10	8.3
2008	721	141	10	7.1
2009	651	132	8	6.1
2010	685	163	10	6.1
2011	575	127	8	6.3

Figure 1 shows the trend line over the percentage which clearly is in decline. The reasons for this decline are not clear, but it is our opinion that applications are as relevant today as they were five years ago.

In an early version of this work [14], we presented the above numbers on the 12$^{\text{th}}$ *International Workshop on Agent-Oriented Software Engineering* (*AOSE*). The audience recognised the lack of application papers and agreed with us on the discrepancy between the IFAAMAS mission statement and the reality of the AAMAS conference. Nevertheless, we were advised to additionally examine the so called *Industry Track*, a special session of the AAMAS main track, in which practitioners are given the chance to present their scientific work. Following this suggestion, we extended our survey to include the industry tracks of the previously examined AAMAS conferences. Generally speaking, the works presented within the industry track feature a high level of practical engineering aspects and match our conception of application papers. The overall amount of papers which describe an application ranges approximately from 70% to 85%.

In 2011, no Industry Track was held, but officially, the name Industry Track was changed to *Special Track on Innovative Applications*. However, although

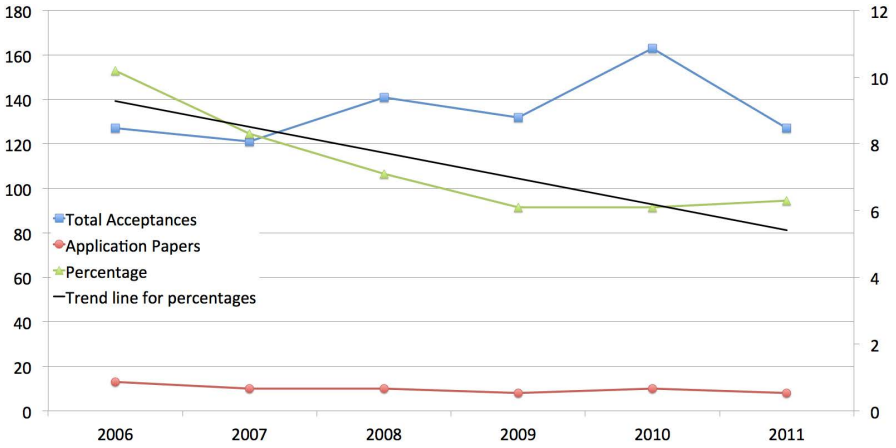

Fig. 1 Amount of accepted full papers and application papers in AAMAS main-track proceedings between 2006 and 2011

Table 2 Overall number of presented papers and number of application papers at AAMAS Industry Tracks

		Application Papers	
AAMAS	Industry Track Papers	# Accepted	Acceptance (%)
2006	14	11	78.6
2007	10	8	80.0
2008	19	16	84.2
2009	12	10	83.3
2010	10	7	70.0
2011	0	0	0.0

there was an explicit call for papers which describe innovative applications, the session was not being held in 2011 [30].

The previous editions however feature an amount of application papers beyond 70%, which indicates that this track counters the shortcoming of engineering aspects and explicitly fosters works which describe practical applications.

Papers which we do not categorise as application papers still had a practical background, so we can state that the AAMAS industry track is exclusively dedicated to the application of agent technology. Nevertheless, due to the shortfall of the Special Track on Innovative Applications in 2011 we can still identify a definite downward trend in the share of application papers. This trend is visualised in Figure 2.

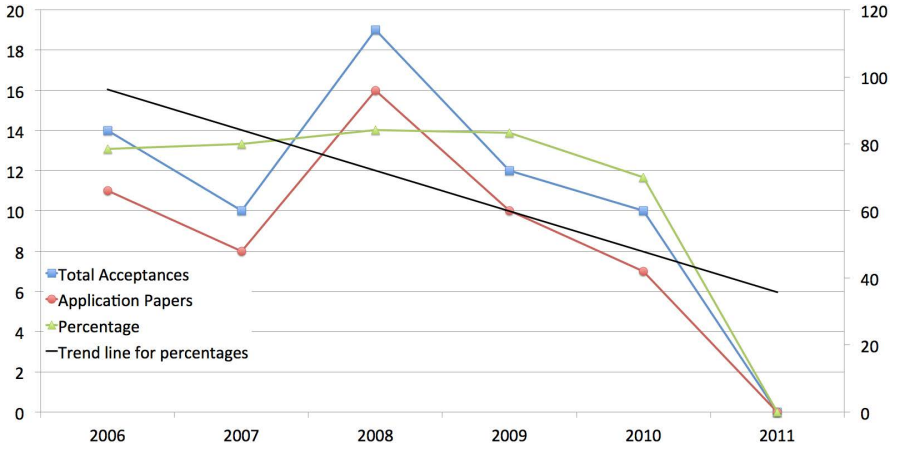

Fig. 2 Amount of accepted full papers and application papers in AAMAS Industry Track proceedings between 2006 and 2011

Based on our analysis, one can argue that applications are adequately represented by means of industry tracks. However, comparing the extend of the AAMAS maintrack to that of the industry track shows that the industry track has only little impact on the overall AAMAS conference. Figure 3 illustrates the industry track's influence on the share of application papers of the entire AAMAS conferences. Despite the outlier in 2008 the number of application papers dropped from about 17% to little over 6%.

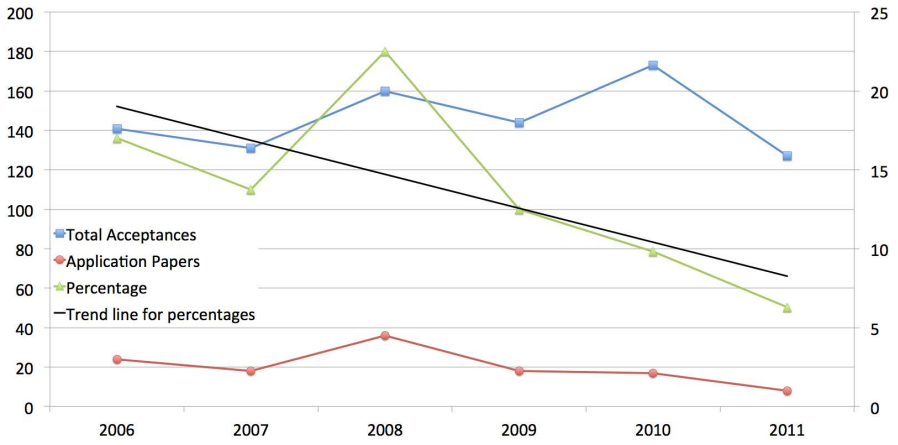

Fig. 3 Amount of accepted full papers and application papers in AAMAS maintrack and industry track proceedings between 2006 and 2011

In addition to the little impact of the industry track on the main conference, we think that the industry tracks are often only visited by like-minded researchers or industry players. Since the industry track is held in parallel to the main track, it is also difficult for the broader community to attend this session. Furthermore, the nominal size of the industry track relative to the overall conference is very small. Table 3 shows the impact of the industry track on the AAMAS conference by presenting respectively the numbers of presented papers.

Table 3 Amount of industry track papers compared to the amount of AAMAS main track papers

AAMAS	Main track	Industry Track	Percentage (%)
2006	127	14	11.0
2007	121	10	8.2
2008	141	19	13.4
2009	132	12	9.1
2010	163	10	6.1
2011	127	0	0.0

3.2 PAAMS

On the other end of the spectrum, the conference on *Practical Application of Agents and Multi-Agent Systems* [6, 7, 8] is explicitly aimed at providing a platform for the dissemination of real-world applications of agents and multi-agent systems. While we have noted before that in our opinion, application oriented papers should be present at main agent conferences, it is also interesting to look at the problem from the other side. Following our argument, we would like to see theoretical work at the main application oriented conference as well. Table 4 shows that the number of application papers is roughly 50% over the last three installments of the conference. It must be noted here that the other papers were usually relevant to agent applications, and very few presented purely theoretical work.

3.3 ICAART

The *International Conference on Agents and Artificial Intelligence* [11, 12, 13] has the stated purpose of bringing together researchers, engineers and practitioners interested in the theory and practise of agents and artificial intelligence. We selected this conference as it is fairly new, yet has already a created considerable interest in the community (2010 they had 364 submissions). It therefore presents a sample of a conference that is not yet set in the type

Table 4 Submission, accepted, and application papers at PAAMS conferences

			Application Papers	
PAAMS	Submissions	Acceptances	# Accepted	Acceptance (%)
2009	92	61	31	50
2010	66	34	17	50
2011	81	39	16	41

Table 5 Submission, accepted, and application papers at ICAART conferences

			Application Papers	
ICAART	Submissions	Acceptances	# Accepted	Acceptance (%)
2009	161	26	3	11.5
2010	364	31	6	19.3
2011	367	32	1	0.3

of topics it generally attracts. Table 5 shows a high fluctuation of application papers between almost none and 20%. Considering that the conference also covers general AI themes such as pattern recognition etc., this is clearly within the stated goals of the conference.

3.4 MATES

The *German Conference on Multi-Agent Systems* [3, 9, 16] solicits theoretical as well as applied research papers. In particular, they focus on enabling technologies for "truly open distributed systems" and request papers reporting on the successful application of agent technologies. Table 6 shows a fairly stable ratio of about 30% of application papers over the last three years. This is in line with the stated aims and goals of the conference.

Table 6 Submission, accepted, and application papers at MATES conferences

			Application Papers	
MATES	Submissions	Acceptances	# Accepted	Acceptance (%)
2009	44	29	11	37.9
2010	34	18	7	38.8
2011	50	18	6	33.3

3.5 Summary of Conferences

In this section we have analysed a number of agent conferences with respect to the number of application papers that are published in the proceedings. While we do realise that there are a large number of conferences and workshops out there, we focused on a cross cutting of conferences that are representing different facets of the field of agents and multi-agent systems. Generally, all of them point to the importance of applications and solicit papers describing real world applications. In the case of PAAMS it is even the main driver of the conference. Other than the main agent conference AAMAS, we found that in general there is a reasonable amount of application papers represented in the analysed conferences.

4 The Agent Technology Journal Landscape

As the agent community and their communication about research results cannot be captured by conferences only, we now have a closer look at the agent technology journal landscape. We do so by first analyse the papers published in the *Autonomous Agents and Multi-Agent Systems Journal* (JAAMAS) in detail, and afterward discuss to what extend r&D is represented in the scopes and aims of the major agent technology journals.

4.1 The Autonomous Agents and Multi-Agent Systems Journal

Looking at the *Autonomous Agents and Multi-Agent Systems Journal* (i.e. the official journal of the IFAAMAS), the aims and scope of the journal specifically asks for "significant original research results in the foundations, theory, development, analysis, and applications of autonomous agents and multi-agent systems", however looking at the list of specific topics of interest: applications are only mentioned once in the third last out of 20 point ("Significant, novel applications of agent technology"). This is also reflected in the papers published in the journal. Table 7 shows the past standard JAAMAS volumes (i.e. special issues are not considered in Table 7, but these will be discussed separately later) and highlights the number (and percentage) of R&D papers in comparison to the total number of research papers published.[3]

In the 23 volumes of the journal that were published since its first publication in 1998, we found 216 research articles. Of these 216 research articles, 15 articles (i.e. approximately 6.9%) feature R&D relevant topics. This is a percentage similar to the percentage of R&D papers found in the AAMAS conference (see Figure 1). Similar to the AAMAS, one worrying aspect of the

[3] We refer to research papers only, as we have neglected editorials, book reviews, etc. in our analysis, as these would distort the results of the analysis.

Table 7 Overall number of published application papers in the Autonomous Agents and Multi-Agent Systems Journal

		Application Papers	
Issue	# Research Papers	#	%
23 (2011)	9	2	22.2
22 (2011)	5	0	0.0
21 (2010)	4	0	0.0
20 (2010)	6	1	16.7
19 (2009)	10	1	10.0
18 (2009)	14	0	0.0
17 (2008)	7	0	0.0
16 (2008)	7	0	0.0
15 (2007)	8	0	0.0
14 (2007)	3	0	0.0
13 (2006)	13	1	7.7
12 (2006)	11	3	27.3
11 (2005)	15	0	0
10 (2005)	9	0	0
9 (2004)	4	0	0
8 (2004)	9	1	11.1
7 (2003)	12	0	0
6 (2003)	10	0	0
5 (2002)	13	1	7.7
4 (2001)	12	1	8.3
3 (2000)	10	1	10
2 (1999)	17	3	17.6
1 (1998)	8	0	0
SUM	216	15	6.9

figures thereby is that —despite a peak in volume 23— in the recent years the percentage of application papers has decreased on average and the 6.9% in Table 7 are partially due to the higher percentages of R&D papers in the earlier years of the journal (i.e. the percentage would be lower if one would analyse the past 5 years only, for example).

In the above analysis, we on purpose neglect the special issues of the JAAMAS journal in order to not distort the figures of the analysis. Instead we treat the special issues separately now. Since 1998, JAAMAS had 21 special issues. The majority of these special issues has been published within the last 3.5 years. Of these special issues one specifically focuses on applications of agents and MAS[4] and a second one featuring "Challenges for Agent-Based

[4] This special issue is the Special Issue on Foundations, Advanced Topics and Industrial Perspectives of Multi-Agent Systems, volume 17(3), December 2008.

Computing"[5] includes several articles that discuss the problem of R&D in-depth. As one interesting aspect, it has to be noted that in both of these special issues – either in the editorial or the papers themselves (see [18, 21, 31] for example) the significance of R&D is pointed out, but little progress can be found so far.

4.2 The General Agent Technology Journal Landscape

Having had a look at the JAAMAS journal in detail, and establishing that especially with respect to the aims and scopes of the journal, r&D seems of secondary importance, we now focus on other agent journals. For this purpose we have analysed the aims and scopes of twelve additional journals focusing on agent technology issues. Besides the focus on agent-related topic, the H-Index of the journals was taken as a decision criterion.[6] These journals were then analysed with respect to their aims and scope with respect to r&D (i.e. not with respect to the actual published papers, but the journals focus defined by the aims and scope). Table 8 shows the results of this analysis.

As Table 8 shows, different to the JAAMAS journal, R&D as well as r&D are present in most of the the journals' scopes and aims. In particular the Applied Artificial Intelligence Journal had a significant focus on r&D and other journals considers papers from this view-point (some of which only for specific application domains). Most important, the majority of the journals considering r&D work, according to their aims and scope demand generalisation and a broad view of the work. This results on some problems for r&D-related research, as it is often rather case-specific and therefore general applicable theory is difficult to gain from the specific results. However, keeping in mind that the general aim of research is not to find only specific solutions, the requirement in the goals is very well understandable.

5 The Stakeholders

Having presented the discrepancy between the IFAAMAS mission statement and the realisation of these statements with regard to the representation of application papers especially in the AAMAS conference and the JAAMAS journal,in this section we focus on the stakeholders of agent application & implementation papers. The stakeholders are the addressees of a paper, i.e. the interests groups that a paper is written for.

[5] This special issue is volume 9 number 3, 2004.

[6] The list of journals we derive the journals and h-indices from can be found on `http://www.scimagojr.com/journalrank.php?category=1702` for the journals to analyse. Thus, from this list of journals focusing n agent-related topics, we selected the 12 journals with the highest h-indices. Journals focusing on other areas of Artificial Intelligence (e.g. robotics) were neglected.

Table 8 r&D in AI Journals (Selection)

Journal	H-Index	r&D-focus
Artificial Intelligence	78	not mentioned as separate topic, but aims and scope highlight that application paper can be submitted if they "describe a principled solution, emphasise its novelty, and present an in-depth evaluation of the AI techniques being exploited"
IEEE Intelligent Systems	61	Emphasises the submission of "R&D activities that can lead to use in real-world applications", more focused on robots then agents
Journal of Artificial Intelligence Research	55	"practical utility" of theories is desired, but not always required; strong focus on R&d
AI Magazine	41	no specific aims & scope given, application-related keywords can be found in list of keywords
Engineering Applications of Artificial Intelligence	39	r&D considered, but only for "real-time automation" applications
International Journal of Intelligent Systems	36	strong focus on R&d, consideration of engineering topics
Autonomous Agents and Multi-Agent Systems	35	see above
Applied Artificial Intelligence	33	strong r&D focus
Artificial Life	32	special focus on "life and life-like phenomena" (e.g. biologically inspired concepts), r&D considered only with this focus
The Knowledge Engineering Review	32	no mentioning of r&D, focus in particular on broad topics and general evaluations, what results in little success chances for specific r&D applications
Artificial Intelligence Review	31	R&D highlighted in aims & scope
Journal of Intelligent Information Systems	30	r&D considered but only with respect to database technologies
Annals of Mathematics and Artificial Intelligence	29	no r&D consideration

Looking at agent application & implementation papers submitted to a conference/journal we can identify 3 groups of potential stakeholder. These are:

– industry, i.e. firms that adopt a scientific idea expressed in a paper and implement/apply it on large scale, or help to perform certain research in the first place,

– academia, consisting of both: the researchers trying to publish their work as well as other researchers that may pick up on an idea expressed in the paper and get to know the work of the author better, and
– reviewers that are supposed to judge the scientific quality of a paper, give feedback to the authors and help the programme committee of a conference to decide which papers to accept and which ones not.

Looking at these stakeholders, how can application & implementation papers become more successful in terms of the IFAAMAS statements? The answer is very straight forward and simple: they need to address the wants and needs of these stakeholders. But what views of the stakeholders should be in a paper? This question we look at in the next sections.

5.1 The Industry View: The Technology Adoption Life-Cycle

To start with, the first interest group's view that we analyse is the industry. As stated earlier, with regard to scientific papers, the industry's main focus is to find new ideas and technologies that can be adopted and incorporated in the business context with the goal of process optimisation and revenue increases.

However, when does industry adopt new technologies and new ideas from papers? One very popular business concept that tries to explain the processes behind industry adoption of technological ideas is the technology adoption life-cycle.

The technology adoption life-cycle is a sociological model that was developed by Joe M. Bohlen, George M. Beal and Everett M. Rogers [1, 2], building on earlier research conducted there by Neal C. Gross and Bryce Ryan [26]. Their original work focused on the adoption (in form of a purchase pattern) of hybrid seed corn by farmers in Iowa and was later generalised to fit the adoption of new ideas and technologies in general [25]. The general life-cycle describes the adoption or acceptance of a new product or innovation, according to the characteristics of defined adopter groups. The model is very believed in by marketers, as it is closely related to the idea that all products and services are subject to a life-cycle, and that can be portrait in the innovation context.

Figure 4 shows the technology adoption life-cycle. The process of adoption over time is typically illustrated as a classical normal distribution or "bell curve". This is because companies respond to new products in different ways. Thus, diffusion of innovations theory, pioneered by Everett Rogers, posits that companies have different levels of readiness for adopting new innovations and that the characteristics of a product affect overall adoption. The model indicates that the first adoption of a new technology begins with a small group of "innovators" that according to Rogers occupy 2.5% of the total group. These are followed by "early adopters" (13.5%). The adoption reaches

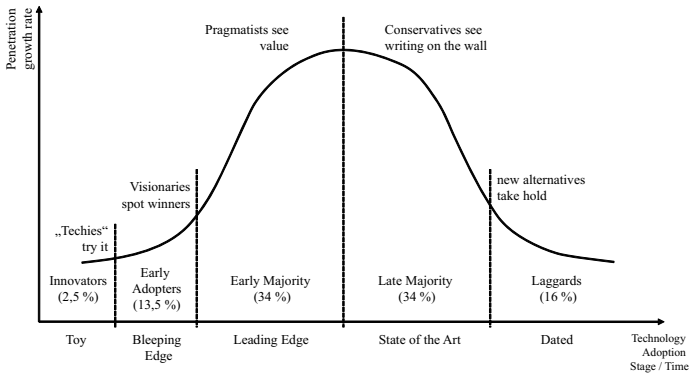

Fig. 4 The Technology Adoption Life Cycle

its growth peak when the "early and late majority" group (34% each) start adopting, and then starts to decline again when only "laggards" (16%) jump on the respective technology bandwagon [25, 17].

Along the lines of these adopter groups in marketing business theory, six technology life-cycle stages have been identified, that are sketched in Figure 4 as well. These stages are:

Toy. At the very beginning the technology is said to be in a toy stage, meaning that it is only known to a very limited number of people, that are highly interested in new technologies, have followed the new innovation from the first day and have the disposable income to indulge their interest. In this early stage the potential of the new technology cannot necessarily be identified and economic risks associated with the adoption of the technology are very high.

Bleeding edge. Technology changes from toy to bleeding edge stages, once the high potential can be identified, but the technology has not been able to demonstrate its value or any kind of consensus about the potential impact of the technology has been agreed on. As a result, the success of the technology is still insecure and adaption is risky (success–wise as well as financially) — early adopters may win big, or may be stuck with a white elephant. That is why normally early adopters tend to be technologically sophisticated, well–informed as well as willing and able to take financial risks.

Leading edge. The leading edge phase is reached once an increasing number of industrial companies learn about the technology and perceived its value with regard to the companies (potentially diverse) needs. However, the technology is still new enough that it may be difficult to find knowledgeable personnel to implement or support it.

State of the art. When everyone agrees that a particular technology is the right solution and knowledgeable personnel for the technology is available, the technology has reached the state of the art stage. At the beginning of this stage the growth rate of the technology penetration is highest and slowly decreases with as the majority of companies have adopted by that time and consequently the number of additional company that could adopt the technology decreases.

Dated. In the dated stage, only laggards are left. The technology is generally perceived as still useful and is still sometimes implemented, but a replacement leading edge technology is readily available.

Obsolete. In the obsolete phase, the technology has been superseded by a new state-of-the-art technology. It is maintained but no longer implemented.

Keeping this theory of the adoption life-cycle in mind, it is useful to consider the position of agent technologies in the curve. Agent technology has not yet entered the two majority stages but can be classified in between the first two stages [23], i.e. in a stage where enthusiasts that share the vision of agents will consider using them, but the majority of potential users has not been reache. Thus, unlike object-oriented technologies for example, only a relatively small number of deployed commercial and industrial applications of agent technology are visible. One reason for this is the relative young age of agent research compared to established technologies (object-oriented concepts had been studied for more than 20 years before being taken up in programming languages). Others argue that (maybe as a consequence) agent technology is not yet visible to many industry players [20].

But what stimulates industry to become aware of a new technology? According to Moore [22], advancing from this stage to stages further ahead is very difficult. He speaks of a "chasm" that technology providers need to bridge. Methods to bridge this chasm Moore identifies are especially in the domain of marketing: thus he points out that adopter need to learn about a technology before they can adopt it. Furthermore he emphasizes the importance of presenting the potential industry adopter the usefulness of the technology by clearly indicating its competitive advantages in comparison to other technologies and show its applicability to several domain[7]. r&D papers are one way of achieving these two objectives, again emphasising the importance of these paper for the agent community. Other authors that have dealt with indusry adoption are Bohlen et al. Thus, in their presentation of the technology adoption life-cycle Bohlen et al. identify two major driving forces for adoption, importance-wise ranking in the order presented here:

1. Usefulness and ease of use needs to be recognised by industry.
2. The perceived risk by companies plays a huge role. So the higher the perceived risk, the lower the likelihood of adoption.

[7] Further requirements for adoption are mentioned in [21].

"Usefullness and ease of use" calls for agent technology example applications for industrial problems but at the same time an awareness for the major specifications of agents that differentiate agents from other technologies. When it comes to perceived risk, the question is how to reduce this risk? The answer given in literature here is the usage of proven methodologies, tools, and complementary products and services. Whereas the latter of the two does not necessarily point to application related publications and work, especially with regard to usefulness they are of an extremely high importance, as in particular example applications in which the respective usefulness of agents has been shown, can motivate industry to invest in agents.

5.2 The Research View

When looking at researchers, i.e. academia, the first thing to note is that academia and industry do not necessarily share congruent goals. Whereas the industry view is revenue oriented, academia's goals are somewhat different. Thus —to put it simple— the main focus of researchers often is to solve problems that are relevant and/or scientifically interesting. Publishing the work is a way to present the achieved results/findings and make them visible to people interested in the field/domain, such as other researchers or industry. Publications are important to start discussions and attract potential future cooperation partners —both from industry and academia— as well as increase scientific recognition and possibly acquire funds for future research. The prerequisite for all the latter, however is the visibility of the research, i.e. the publication in the first place. Due to this importance of publications a high rivalry for them exists. One way of solving this rivalry is by evaluating the quality of the research work with the help of impartial judges. These "impartial judges" are being referred to as reviewers.

5.3 The Reviewers View

According to Smith [27], the task of a reviewer is to evaluate the written work by other researcher that has been submitted for publication in a specific journal or to a conference. On the one hand this involves determining if the work presented is correct and of sufficient quality and on the other hand if the problem studied and the results obtained are new and significant. Based on his knowledge a reviewer is furthermore supposed to make suggestions (if applicable) on how to improve the paper, i.e. give ideas which changes to the paper (and possibly the research behind the paper) might improve the work. Reviewers do this work for free, i.e. without any direct financial advantage, and in their own time. The benefits they have from the task include the contribution they make to the research community, and the fact that they might get ideas for own research or pointers to new references for their work, as well as the possibility to shape the future direction of research in the

domain by either accepting or rejecting a paper and thus by deciding on what is being published and presented and what is not.

In order to perform the review task, reviewers normally are given some guidelines and/or review forms by the journal or conference they are reviewing for. These are supposed to help the reviewer to fulfill his task in a structured manner and make reviews comparable. One particular problem with regard to the r&D work is that due to its practical implementation, it is hard to check for reviewers, especially given the limited time frame of a review period. This, combined with the fact that research is supposed to be general and not only valid for specific cases, results in problems when evaluating r&D work. As pointed out earlier, one of the main industry drivers for adopting a new technology is "usefulness and ease of use". The problem at hand is that this driver does not necessarily go along with the review process. We identify this discrepancy as one reason for the issues raised in Section 1.

6 Summary

Despite this low representation of r&D in some journals such as the JAAMAS and also the AAMAS conference, we argue that application papers (in form of r&D) are indeed relevant to the agent community at large, and should be presented for the following reasons:

- Engineering problems are often ignored in theoretical papers, but can often be show stoppers for actually *using* the theories. Thus, for industry to consider adopting a new technology, it must not only learn about its advantages, but in addition learn on how it could be *applied* to help and support industrial problems.
- Applications present *real* issues and challenges that in turn (should) point to possible directions for future theoretical research.
- Many national and transnational project calls explicitly ask for industry involvement. Thus, in the general documents of the Community Research and Development Information Service of the European Commission for example, the "Rules for submission of proposals, and the related evaluation, selection and award procedures" emphasises "an appropriate balance between academic and industrial expertise and users" with regard to FP7 projects [4, p. 11].
- In order to learn about industry needs and thinking and get in touch with industry partners, application papers can play a significant role.

It is our belief that application descriptions are an important part of the work being done in the agent community, and that this work should therefore play a larger role in the community at large than it does at the moment. While there are a number of venues specifically for practical issues and implementations of agent-based systems, it is to our mind important to bridge the R and the D, and get the different communities to interact.

We do not assume to have all the answers, but in pointing out the trend of a decreasing number of application oriented papers, especially at AAMAS, we hope to start a discussion on the need for more practical papers. The reasons and possible means to rectify this situation are more complex than what we offer here, but nonetheless we hope that the issue will be taken up by the community.

This book is a first step into this direction by presenting a large number what we refer to as *application papers* in Section 2. The main focus of the papers is on tools designed to support applications and their actual use as well as implementation aspects related to these tools. Some Chapter also present prototypes that are being used in the real-world. In Chapter 2 Alexander Pokahr, Lars Braubach and Kai Jander present the Jadex Project and demonstrate its suitability for the traditional BDI concept. They show extension that facilitates the interactions between agents with services and also provide a common black box view for agents that allows different agent types. Milan Vidaković, Mirjana Ivanović, Dejan Mitrović and Zoran Budimac – in Chapter 3 – present the Extensible Java EE-based Agent Framework they developed. With their framework they demonstrate how existing, standardized Java EE technologies, tools, and libraries, such as JNDI, JMS, and EJB, can be used to implement a large subset of functionalities required from a multi-agent system. Immediate direct benefits of this approach are a shorter development time of the system itself, delegation of agent load-balancing to the enterprise server, flatter learning curve for new developers of the system, etc. The Wade Platform presented by Giovanni Caire in Chapter 4, not only presents a tool for supporting applications but also shows how this tool has been used for dealing with real world problems, namely in the field of Fixed Network monitoring and optimization where it has been employed by Telecom Italia. Chapter 5 by Lars Braubach and Alexander Pokahr again deals with the Jadex project. It focuses on the issues of time and virtual environments in agent simulation and discusses their implementation in the light of establishing simulation transparency. In Chapter 6 Rafał Leszczyna presents MAlSim – a simulator of malicious software based on software agents, which can serve as a testbed for critical infrastructures security. Besides presenting and discussion his design decision, in his chapter the author explains the choice of agent paradigm for the development of the toolkit and also gives a brief description of the application of MAlSim to security evaluation of a power plant. Instead of presenting tools Chapter 7 – written by Peter Novák, Antonín Komenda, Michal Čáp, Jiří Vokřínek and Michal Pěchouček – puts its main focus on prototypes developed for addressing issues of urban warfare. Resulting from the prototype development, the main contribution is an account of architectural and technological issues related to the development of the multi-agent platform and simulation subsystems for the cluster of the project evolving around the prototypes. In Chapter 8 Giancarlo Fortino and Stefano Galzarano focus on the domain of Wireless Sensor Networks (WSN). They promote the use of the mobile agent paradigm for the development

of WSN applications and, specifically, describe issues and solutions for the development of mobile agent systems on resource-constrained wireless sensor platforms. M. Morge, J. McGinnis, S. Bromuri, P. Mancarella, K. Stathis and F. Toni – in Chapter 9 – propose an argumentation-based agent model that supports service and partner selection in service-oriented computing settings and illustrate its functionalities with the help of an distributed e-procurement process example. Finally, in Chapter 10, Emiliano Casalicchio and Salvatore Tucci give account of experience matured by them in the design and implementation of an agent-based modeling and simulation framework to support the re-engineering of Public Administration workflows. They present the Wf-Simulation resulting from this work and show its properties with the help of three real-world case studies.

References

1. Beal, G.M., Rogers, E.M., Bohlen, J.M.: Validity of the concept of stages in the adoption process. Rural Sociology 22(2), 166–168 (1957)
2. Bohlen, J.M., Beal, G.M.: The diffusion process. Special Report 18. Iowa State College (May 1957)
3. Braubach, L., van der Hoek, W., Petta, P., Pokahr, A. (eds.): MATES 2009. LNCS, vol. 5774. Springer, Heidelberg (2009)
4. CORDIS. Rules for submission of proposals, and the related evaluation, selection and award procedures (August 2008)
5. Decker, K.S., Sichman, J.S., Sierra, C., Castelfranci, C. (eds.): Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems, International Foundation for Autonomous Agents and Multiagent Systems, Budapest, Hungary (May 2009)
6. Demazeau, Y., Dignum, F., Corchado, J.M., Bajo, J., Corchuelo, R., Corchado, E., Fernández-Riverola, F., Julián, V.J., Pawlewski, P., Campbell, A. (eds.): Trends in PAAMS. AISC, vol. 71. Springer, Heidelberg (2010)
7. Demazeau, Y., Pavón, J., Corchado, J.M., Bajo, J. (eds.): 7th International Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS 2009). AISC, vol. 55. Springer, Heidelberg (2009)
8. Demazeau, Y., Pechoucek, M., Corchado, J.M., Pérez, J.B. (eds.): Advances on Practical Applications of Agents and Multiagent Systems. AISC, vol. 88. Springer (2011)
9. Dix, J., Witteveen, C. (eds.): MATES 2010. LNCS, vol. 6251. Springer, Heidelberg (2010)
10. Durfee, E., Yokoo, M., Huhns, M., Shehory, O. (eds.): Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems, Honululu, HI, USA. Association for Computing Machinery (May 2007)
11. Filipe, J., Fred, A.L.N. (eds.): Proceedings of the 3rd International Conference on Agents and Artificial Intelligence, Rome, Italy (January 2011)
12. Filipe, J., Fred, A., Sharp, B. (eds.): ICAART 2009. CCIS, vol. 67. Springer, Heidelberg (2010)
13. Filipe, J., Fred, A., Sharp, B. (eds.): ICAART 2010. CCIS, vol. 129. Springer, Heidelberg (2011)

14. Hirsch, B., Balke, T., Lützenberger, M.: Assessing agent applications – r&D vs. R&d. In: Proceedings of the 12th International Workshop on Agent Oriented Software Engineering, Taipei, Taiwan, pp. 93–104 (2011)
15. IFAAMAS. Charter for the international foundation for autonomous agents and multiagent systems
16. Klügl, F., Ossowski, S. (eds.): MATES 2011. LNCS, vol. 6973. Springer, Heidelberg (2011)
17. Levitt, T.: Exploit the product life cycle. Harvard Business Review 43(6), 81–94 (1965)
18. Luck, M., McBurney, P., Preist, C.: A manifesto for agent technology: Towards next generation computing. Autonomous Agents and Multi-Agent Systems 9, 203–252 (2004)
19. Luck, M., McBurney, P., Shehory, O., Willmott, S.: Agent Technology: Computing as Interaction – A Roadmap for Agent Based Computing. In: Agent Link (2005)
20. Marik, V., McFarlane, D.: Industrial adoption of agent-based technologies. In: IEEE Intelligent Systems, vol. 20(1), pp. 27–35. IEEE Educational Activities Department (2005)
21. McKean, J., Shorter, H., Luck, M., McBurney, P., Willmott, S.: Technology diffusion: analysing the diffusion of agent technologies. Autonomous Agents and Multi-Agent Systems 17, 372–396 (2008)
22. Moore, G.A.: Crossing the Chasm: Marketing and Selling High-Tech Products to Mainstream Customers. Harper Business Essentials, New York (1991)
23. Munroe, S., Miller, T., Belecheanu, R.A., Pěchouček, M., McBurney, P., Luck, M.: Crossing the agent technology chasm: Lessons, experiences and challenges in commercial applications of agents. Knowledge Engineering Review 21(4), 345–392 (2006)
24. Padgham, L., Parkes, D.C., Müller, J., Parsons, S. (eds.): Proceedings of the 7th International Conference on Autonomous Agents and Multiagent Systems, International Foundation for Autonomous Agents and Multiagent Systems, Estoril, Portugal (May 2008)
25. Rogers, E.M.: Diffusion of Innovations. Free Press (1962)
26. Ryan, B., Gross, N.C.: The diffusion of hybrid seed corn in two iowa communities. Rural Sociology 8, 15–24 (1943)
27. Smith, A.J.: The task of the referee. Computer 23(4), 65–71 (1990)
28. Stone, P., Weiß, G.: Proceedings of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems, Hakodate, Japan. Association for Computing Machinery (May 2006)
29. van der Hoek, W., Kaminka, G.A., Lespérance, Y., Luck, M., Sen, S. (eds.): Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems, Toronto, Canada. International Foundation for Autonomous Agents and Multiagent Systems (2010)
30. Yolum, P., Tumer, K., Stone, P., Sonenberg, L. (eds.): Proceedings of the 10thInternational Conference on Autonomous Agents and Multiagent Systems, International Foundation for Autonomous Agents and Multiagent Systems, Taipei, Taiwan (May 2011)
31. Zambonelli, F., Omicini, A.: Challenges and research directions in agent-oriented software engineering. In: Autonomous Agents and Multi-Agent Systems, vol. 9, pp. 253–283 (2004)

Chapter 2
The Jadex Project: Programming Model

Alexander Pokahr, Lars Braubach, and Kai Jander

Abstract. This chapter describes the priciples of the Jadex programming model. The programming model can be considered on two levels. The intra-agent level deals with programming concepts for single agents and the inter-agent level deals with interactions between agents. Regarding the first, the Jadex belief-desire-intention (BDI) model will be presented, which has been developed for agents based on XML and Java encompassing the full BDI reasoning cycle with goal deliberation and means-end reasoning. The success of the BDI model in general also led to the development goal based workflow descriptions, which are converted to traditional BDI agents and can thus be executed in the same infrastructure. Regarding the latter, the Jadex active components approach will be introduced. This programming model facilitates the interactions between agents with services and also provides a common back box view for agents that allows different agent types, being it BDI or simple reactive architectures, being used in the same application.

1 Introduction

This chapter is one of two chapters describing practical applications built with the Jadex agent framework. The applications are structured according to the main features of Jadex that were required for building these applications. In this chapter, the focus is on features regarding the programming model of Jadex. Therefore, this chapter is subdivided into three thematic sections that cover different programming model aspects and applications. Each section starts with a short background about why a certain topic was considered important for programming in Jadex, followed by a more general motivation about the relevance of the concept itself. A related work section is presented

Distributed Systems and Information Systems
Computer Science Department, University of Hamburg
e-mail: {pokahr,braubach,jander}@informatik.uni-hamburg.de

M. Ganzha & L. C. Jain (Eds.): Multiagent Systems & Applications, ISRL 45, pp. 21–53.
springerlink.com

for each concept, trying to give an overview of the field with pointers to other relevant works in the area. Afterwards the approach as implemented in Jadex is covered in detail and further illustrated by example applications that have been built. Each section closes with a short summary.

In particular, the following topics are described in this chapter. Section 2 discusses the behavior model of agents which, in Jadex, was initially realized according to the belief-desire-intention (BDI) model that was extended for Jadex in several substantial ways. With workflows, Section 3 addresses an interesting application area for agents regarding the support of e.g. complex and dynamic business processes. The last topic in Section 4 is called active components and introduces a unification of agent concepts with concepts from service- and component-based software engineering. Finally, Section 5 summarizes the chapter and identifies important challenges with respect to the programming model that remain to be tackled for promoting industrial take-up of agent technology.

2 Agent Programming: BDI Architecture

The ever increasing computational power causes an ever increasing complexity of software systems. The tasks performed by computer systems become more and more advanced including e.g. automating complex processes or providing intelligent support for humans during their execution of activities. Engineering science strives to develop new concepts, methods and tools for dealing with the increasing complexity of systems. All systems are ultimately built by humans for humans. Therefore, ideas from disciplines like philosophy or psychology have been applied to engineering for better supporting the process of comprehension of typical human system engineers and human system users. One well known example is the so called *Intentional Stance* coined by Daniel Dennett [23]. When applied to software systems, it allows considering system components as intentional entities that have certain responsibilities with respect to local and overall system goals and that act rationally and independently of each other towards achieving these goals. This approach fits well to the way how humans conceive their own thinking processes (a.k.a. folk psychology) and thus simplifies reasoning and discussing about system designs.

Intentional approaches haven proven useful early on, for example with respect to goal-driven requirements engineering [21]. When considering more and more complex systems, where typically autonomous and/or adaptive behavior is required from the system's components, it becomes apparent that intentional notions such as goals and rational action are useful also for improving system design and implementation. An intentional approach simplifies tracing requirements to design and implementation artifacts, as each are based on the same mental model of responsibilities, system goals and rational action. As an additional advantage, systems start to "behave like humans

would do", i.e. they behave understandably according to the mental models of system designers and system users. This further simplifies, e.g. debugging of the system and leads to an intuitive usage.

2.1 Related Work

The term *agent architecture* is used to describe the concepts and constructs for specifying behavior. In this respect between internal and social agent architectures is distinguished. The first refers to architectures that deal with concepts for programming a single agent while the latter are concerned with how group behavior and teamwork can be described and programmed. With regard to different application contexts, simple or complex agent architectures may be better suited. Figure 1 shows an overview of well-known agent architectures. The figure highlights how the architectures are influenced by theories from different disciplines, such as philosophy and psychology. E.g. the agent architectures AOP [47], 3-APL [22], IRMA [5] and PRS [44] incorporate the Intentional Stance and are therefore related to philosophical theories like the belief-desire-intention (BDI) model. Theories from the field of psychology focus on lower-level cognitive processes such as learning and have led to architectures like SOAR [30] that largely differ from those that originate from philosophical theories. For social architectures that focus on coordination in multi-agent systems, organization theory and sociology have been sources of inspiration, e.g. the Joint Intentions theory [19] as incorporated in the Joint Responsibility model [27]. Finally, the Subsumption architecture [13] is a biologically inspired architecture for building simple reactive insect-like agents.

The BDI model [4] is a good trade-off between complexity and expressiveness as it is based on a simple set of intuitive concepts with a natural meaning (e.g. beliefs representing the knowledge of an agent about the world). The first implemented system based on a BDI-like model was the procedural reasoning system PRS [24]. The mapping to BDI was later made explicit and formalized in [44]. A number of successor systems have transported original PRS ideas to newer runtime infrastructures, e.g. the Java-based JAM [25] and the commercial JACK [17]. In addition, with AgentSpeak(L) a BDI-style programming language has been proposed in [43], which is supported by interpreters such as Jason [3].

2.2 Approach

The Jadex BDI architecture has been conceived and realized with conceptual as well as technical goals in mind. Conceptual aim was developing an agent behavior model that intuitively resembles human decision making. This model should act as a blueprint (pattern) for commonly found problems in agent systems. The Jadex BDI agent architecture thus provides ready to use

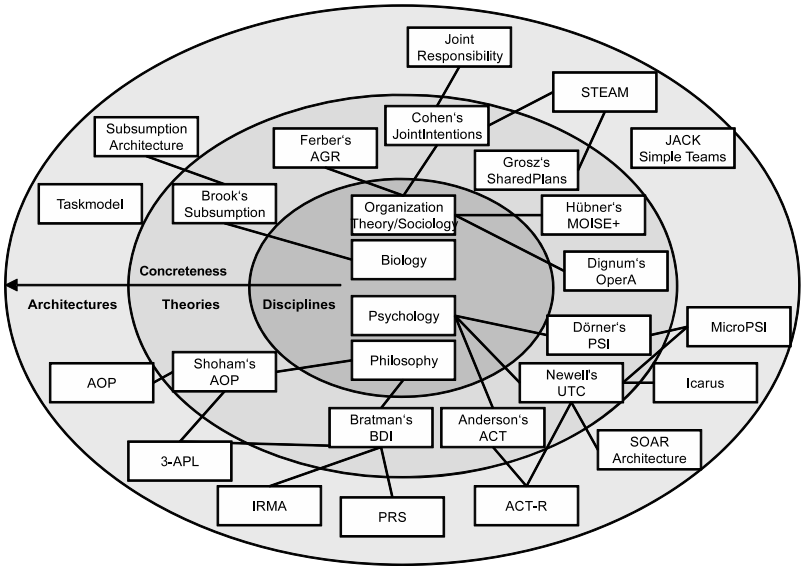

Fig. 1 Agent architectures (from [12])

functionality and reduces the need for manually coding aspects of the agent behavior.

On a technical level the idea is making agents more close to mainstream programming. Therefore, the realization makes use of established technologies like Java and XML. This facilitates the integration with existing technologies, 3rd-party libraries and legacy systems and further allows developing agent applications using existing development environments.

2.2.1 Goal Representation and Processing

The Jadex BDI architecture comprises several aspects of agent behavior and development support. In the following, the basic goal-based behavior model will be described. Put simply, it allows defining agent behavior in terms of goals to be achieved and plans to be executed towards achieving the intended goals. The behavior model is based on the means-end reasoning process found in earlier PRS systems. These realize a reactive planning approach as follows: Given a goal or event, the agent will choose a plan from a library of procedural plans and execute the plan in a step-by-step fashion. Each plan specification incorporates one or more triggering events, i.e. goals for which the plan may be applicable. If the plan succeeds (i.e. completes without error), the goal is considered achieved. Otherwise the agent may choose another plan from the plan library and start over. The PRS reasoning cycle is well-suited for realizing adaptive behavior as plans are selected based on their applicability to the current situation. An agent can react to changing environments by

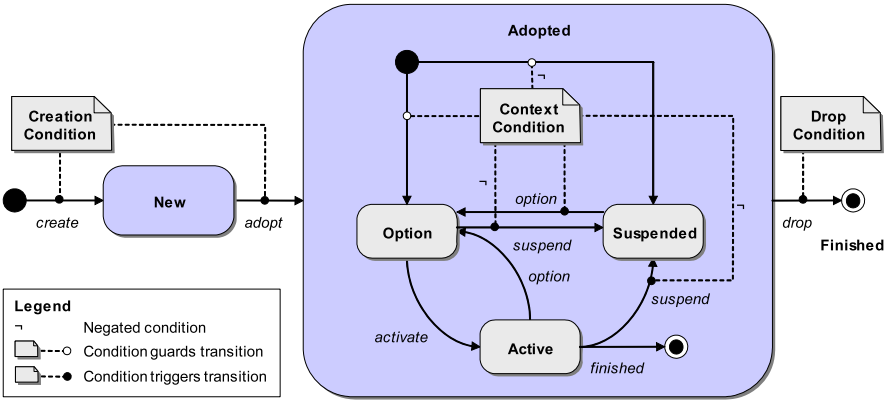

Fig. 2 Goal lifecycle (from [11])

simply retrying with a different plan. Furthermore, the PRS approach facilitates an extensible system design, as new plans can be added to the plan library without the need of touching other parts of the agent code.

Goal Lifecycle

In PRS, goals are only considered as ephemeral events. Jadex extends the PRS model by introducing a lifecycle for goals that allows treating goals as first class programming concepts [11]. The goal lifecycle is depicted in Figure 2 in an extended state-chart notation. The rounded rectangles represent the possible lifecycle states of a goal and the arrows indicate the possible transitions between the states. A goal can be created (state *New*) as a programming construct to configure its contents before making it accessible to an agent. Once the goal is *adopted*, the agent is aware of the goal such that it may influence the agents behavior. To simplify dealing with many goals at a time, three substates of the adopted state are introduced. Only *active* goals are currently pursued following the PRS reasoning approach described above. Goals may be *suspended*, when they cannot be pursued, e.g. due to external conditions. Furthermore, goals can be *options*, when their processing is delayed, e.g. in favor of other more important goals. To stop the agent from working on a goal, a goal may be dropped, putting the goal in the *finished* state.

The transitions between goal states can be performed manually by the agent programmer (e.g. writing code in a plan to create or suspend some goals). Additionally, the goal specification can be equipped with declarative conditions to indicate situations, when state transitions should happen automatically. These are shown in the figure as note boxes. The *creation condition* leads to the creation of new goals, which are initialized with contents according to the condition (e.g. the creation condition might state to create a new

goal for each new item observed by the agent) and directly adopted by the agent. The *context condition* controls in which of the substates of the adopted state a goal is in. When the context is valid, the goal becomes an option and may be activated. Otherwise, the goal is automatically suspended. In some situations it is useful to stop processing of a goal, even when it is not achieved (e.g. when a goal has become obsolete). Such situations can be declaratively specified using the *drop condition*.

Goal Kinds

The goal lifecycle as introduced above facilities the management of goals as a first class programming construct. Yet, it does not further clarify the semantics of the goal itself, i.e. how an agent should behave according to its currently active goals. Therefore, the active state is further refined in different goal kinds. In the literature, many kinds of goals can be found [11] and a common classification considers goals as a specification of a world state and an intention towards this world state (e.g. achieve, maintain, avoid, ...). Jadex supports four goal kinds, which cover a wide variety of usage patterns.

The *perform* goal is the simplest goal type and comes close to the original PRS semantics. The goal tries to execute all applicable plans, succeeding if at least one plan could be found. The *achieve* goal specifies a desired world state as a so called target condition. The goal succeeds, when the target condition is fulfilled, regardless if plans have been executed or not. Thus, the success of a perform goal only depends on the availability of plans while the success of an achieve goal is only related to the world state. Therefore, the former is often called a procedural goal, while the latter represents a declarative goal. Another common kind of declarative goal is the *maintain* goal. Unlike the achieve goal, which describes a state to be achieved only once, a maintain goal intends to keep a state after it has been achieved. Therefore, every time the state is violated plans are executed for re-achieving the state. A maintain goal is never considered succeeded and is thus only dropped, when explicitly requested by the agent programmer or the optional drop condition. The final goal kind is the *query* goal. It is similar to an achieve goal with the difference that the target condition does not represent a potentially external world state, but instead demands some information from the agent's beliefs. If the information is readily available, no plans need to be executed. Otherwise, the executed plans are expected to lead to the adoption of the required information as beliefs.

2.2.2 Goal Deliberation

The goal representation described in the previous section allows for dealing with multiple goals at once. Following the goal lifecycle one can influence the order in which goals are processed by moving goals between the option and active state. The mechanism of selecting goals to actively pursue is called

goal deliberation strategy. While such a strategy can also be implemented manually, Jadex provides a default deliberation strategy that allows an intuitive specification and covers many recurrent application cases [41]. The so called "easy deliberation" strategy is based on two concepts: a *cardinality* to restrict the number of active goals of a given type and *inhibition arcs* to define a partial order of importance between goals.

Both concepts allow a developer to take a local perspective when writing goal specifications. The cardinality is concerned only with a single type of goal. The inhibition arc expresses a local conflict or precedence between two types of goals. It specifies that the first goal "inhibits" the second, meaning that if both are options the first may become active. Inhibition arcs can be specified on the type level or on the instance level. A type level inhibition arc means that as long as one goal of the first type is active no goal of the second type may be pursued. An instance level inhibition arc contains an expression restricting to which specific goal instances the arc applies. This allows also drawing arcs between two goals of the same type and establishing an order for goal processing based on goal properties.

2.2.3 Capabilities

An important concept in software engineering is modularization as it allows reducing system complexity by decomposition in software modules, which can be to some extent treated (e.g. designed, implemented, tested, ...) in isolation. The BDI architecture as such does not support modularization with regard to a single agent. Although plans can be developed independently of each other they typically require access to global data structures like the agent's beliefs. The capability concept, initially proposed by Busetta et al. in [16], allows grouping BDI elements (e.g. beliefs, goals and plans) pertaining to a specific functionality into a separate module. The agent implementation can then be composed of existing modules. The concept has been adopted and extended for Jadex [10].

The extensions concern important software engineering aspects like parameterization, which allows external configuration of existing capabilities for making them applicable to different usage contexts, and dynamic composition, i.e. the addition and removal of capabilities during the life time of an agent. Another important extension is a generic import/export mechanism that allows establishing relationships between elements from different capabilities without violating module independence. Therefore one may specify plans that are triggered in response to goals from other capabilities and also establish inhibition arcs for goal deliberation across capabilities.

2.2.4 Goal-Oriented Interaction Protocols

The concepts that have been described until now have only considered the (intelligent) behavior of a single agent. In multi-agent systems the interaction

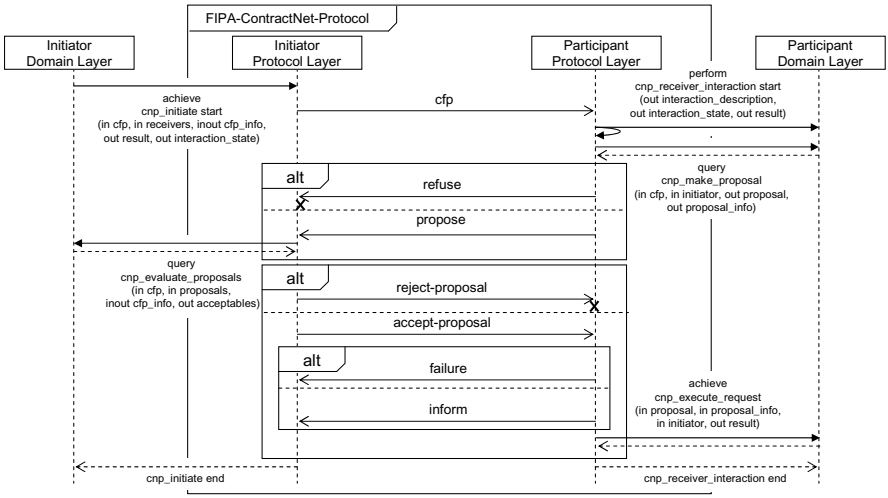

Fig. 3 Goal-oriented contract-net protocol (from [6])

between agents, typically based on asynchronous message exchange, also plays an important role. Therefore the question arises how the internal behavior can be linked to the external communication. As a manual approach one can send messages directly in plans. The disadvantage is that the complete code for a potentially complex negotiation needs to be placed in a single plan leading to poorly maintainable code. The concept of goal-oriented interaction protocols, proposed in [6], allows capturing agent intentions pertaining to interactions. The concept allows making use of deliberation and goal/plan decompositions for interactions as well.

The general approach defines a process for analyzing an interaction protocol, which describes the allowed sequences of messages, and attaching goals to each role in the interaction. Based on such an interaction specification, the developer can simply define separate plans for the activities and decisions required during an interaction. Besides the general approach, several ready-to-use goal oriented interaction specifications are included in Jadex that implement standardized interaction patterns like Dutch or English auction and contract-net negotiations.

Figure 2.2.3 shows the result of the protocol analysis for the contract-net protocol. The left hand side represents the *initiator* role of the negotiation while the right hand side illustrates the behavior of each of the potentially many *participants*. The relationship between the *domain layer* (i.e. business logic) and *protocol layer* (i.e. exchanged messages) is captured in a number of goals, which may be posted or handled at each role. The domain layer of the initiator role starts the interaction by creating the *achieve cnp_initiate* goal. During the negotiation, the *query cnp_evaluate_proposals* goal is created by the initiator's protocol layer and needs to be handled in the domain

layer. When the negotiation ends, the result is made available as success or failure of the *cnp_initiate* goal, such that the initiator domain layer can proceed appropriately. At the participant side all goals are created automatically in the protocol layer. The participant's domain layer handles the *query cnp_make_proposal* goal to generate an offer to be sent to the initiator. In case a participant's offer is accepted, the *achieve cnp_execute_request* goal causes the execution of the requested task in the domain layer.

2.3 Application: MedPAge

The described features of the Jadex BDI architecture will be illustrated with an example application called *MedPAge*, which is a real world multi-agent application that additionally makes use of capabilities for modularization and reusability as well as goals, goal-oriented interaction protocols for complex negotiations. The aim of the MedPAge ("*Medical Path Agents*") project [38, 37, 52] was improving patient scheduling in hospitals. Approach of the project was representing the different goals of the involved stakeholders by intelligent agents. E.g. patient agents would try to minimize the waiting times for their patients, whereas resource agents would try to maximize the utilization of hospital resources such as radiology units. As these goals are usually in conflict, the agents perform autonomous negotiations for producing schedules that balance the individual goals.

The project was part of a larger initiative investigating the applicability of agent technology to real world business applications. The DFG-funded[1] priority research programme SPP 1083 was conducted from 2000-2006 and involved projects from the areas of hospital logistics as well as manufacturing logistics.[2]

The hospital setting considered for the MedPAge project was derived from a real German hospital with hundreds of patients as well as several functional units with different resources. The resulting agent-based application thus exhibits much more complexity compared to the rather toy-like cleaner world application. Therefore, besides using goal representation and goal deliberation for defining the behavior of the individual agents, also capabilities and goal-oriented interaction protocols have been employed in the implementation.

Architecture and Design

The main goal of the MedPAge system consists in generating an efficient treatment scheduling plan. Thus the main goal of performing treatments can be refined towards two subgoals for each side. With respect to the hospital side, the main objective is to achieve a high resource utilization while the

[1] Deutsche Forschungsgemeinschaft (German Research Council): http://www.dfg.de

[2] More details can still be found on the programme web site: http://www.realagents.org/

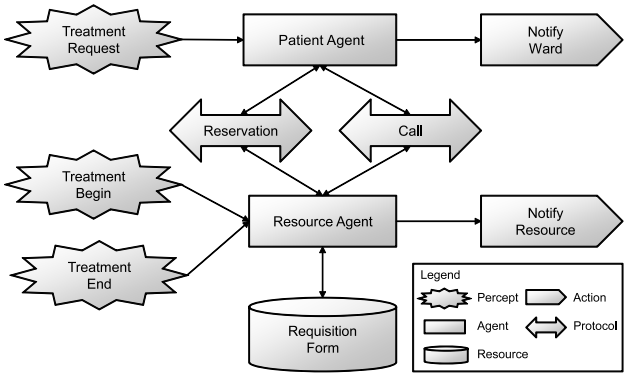

Fig. 4 MedPAge system overview diagram

patient side is interested in seeing patient needs being satisfied, e.g. having short waiting times or giving priority to patients with severe diseases. Of course, the pursuit of these system goals has to respect the fundamental medical conditions in place.

The MedPAge system has been developed following the Prometheus methodology [36]. The core of the architecture is the system overview diagram, which is depicted in Fig. 4. This design contains two agent types that represent patients and hospital resources respectively. This allows a natural modeling and assignment of goals to the different coordination objects (wards and patients) and also adequately reflects the decentralized structure of hospitals. The patient agent is responsible for announcing these requested treatments at a corresponding functional unit (e.g. at the x-ray unit). Furthermore, it ensures that patients visit treatment rooms and are afterwards brought back to their ward. A resource agent accepts appointment requests from patient agents and is in charge to create treatment schedule. The resource agent is notified whenever a new treatment can begin. In this case it calls the patient from the ward and also informs the resource about the planned treatment and patient. After treatment end the patient is sent back to its ward.

In order to implement the MedPAge system the high-level system design has been further concretized to the patient and resource agent design shown in Fig. 5. These diagrams visualize the goals, plans, events, knowledge bases as well as the incoming percepts and outgoing environmental actions of the agents. The agent functionalities have been modeled as goals and plans, which can express the proactive as well as reactive agent behavior. The patient agent reacts on treatment percepts by creating a new make reservation goal for a specific appointment. The goal is handled by the reserve appointment plan, which uses a registry to find resource agents representing the functional unit it needs for the planned treatment. The set of resource agents is subsequently used to find a suitable appointment for the patient by performing negotiations that aim at respecting patient (e.g. health state) as well as resource (e.g.

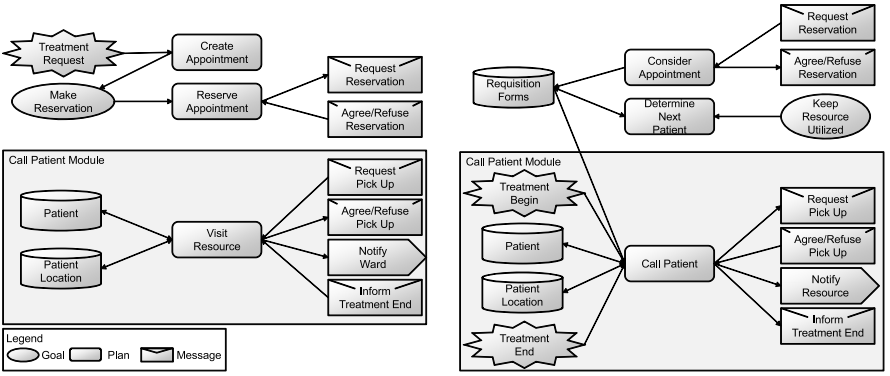

Fig. 5 a) patient agent b) resource agent

other appointments and utilization) needs. At resource side the new requisition form has to be taken into account and is thus added to the agent's knowledge base. The knowledge base is monitored by a keep resource utilized goal, which is used to assure a beneficial appointment ordering from the resource's point of view. Similar to the appointment reservation the patient pick up mechanism has been modeled.

The functional unit signals the readiness for a new treatment to the resource agent, which activates the call patient plan that contacts the patient agent with a pick up request. The receiving patient agent starts the visit resource plan and decides if the visit is possible (e.g. the patient could not be at the ward). The resource agent is informed about the decision. Furthermore, if the decision is positive, the ward is notified to send to patient to the functional unit and the internal beliefs of the patient location is updated. The treatment end is again announced to the resource agent. It reacts by using the call patient plan to update its beliefs and forward the information to the patient agent.

The design diagrams from Fig. 5 have been used to implement the application with Jadex BDI agents. The high correspondence between the Prometheus design concepts and the Jadex BDI concepts led to a straight forward implementation process that directly mimics the design.

Capabilities

Capabilities allow decomposition and reusability of agent functionality. In MedPAge, different scheduling mechanisms have been tested under realistic conditions. To keep implementation efforts low it was critical to modularize the agent designs and factor out common functionality. The primary components of the application were the patient and resource agents, which were accompanied by some support agents [39] of limited complexity. Common functionality of the patient as well as resources agents that was independent of the

scheduling algorithm concerns the *call patient* module, introduced above. Regardless of how the agents negotiate the time slots for treatments and examinations, the actual calling of a patient from the ward to the corresponding resource has to be performed as a separate step allowing manual intervention of hospital personnel in case of, e.g., emergencies. Additionally, the implementation of the call patient module might differ with respect to the existing IT systems already available in the hospital.

For each tested scheduling algorithm, two capabilities have been implemented: one for the patient side and one for the resource side. Using the import/export interfaces of the capability concept, these modules can be seamlessly integrated into the agents and coupled with the remaining functionality, such as the call patient module. Each implemented scheduling approach defines a different pattern of message exchange according to an interaction protocol. The capabilities for the patient and resource agent complementarily implement either the initiator or participant role of this protocol. Details of the protocol implementations are given in the next section.

Goal-oriented Interaction Protocols

In MedPAge, scheduling mechanisms of varying complexity were implemented. The MedPaCo ("Medical Path Coordination") algorithm incorporates stochastic knowledge about the probability of future treatments based on predefined clinical pathways as well as statistical data on previous patients with the same diagnosis. Based on this knowledge, a patient agent can estimate the value of a time slot offered by some required hospital resource. E.g. a slot would be assigned a higher value, when waiting for the next slot would significantly increase the overall staying time of the patient at the hospital. The resource agents collect estimations from multiple patients and adapt their local schedule accordingly.

The MedPaCo protocol is shown in Figure 6. The protocol is split into four phases. The first two phases involve communication the need for a time slot from the patient to the resource (*subscription phase*) and announcing the start of an auction for an upcoming time slot (*announcement phase*). The last two phases correspond to the contract-net protocol as already introduced in Section 2.2.4. In the *bidding phase*, the resource agent collects the bids from the patient and selects the winning patient in the *awarding phase*. At any time multiple negotiations between overlapping sets of patient and resource agents may take place. Therefore a patient might simultaneously win two negotiations at different resources. As a result, the awarding phase needs to be cyclic, because a winning patient might have accepted another time slot for the same treatment already (*cancel(treatment)*) or for a different treatment (*refuse(not-available)*).

Based on the goal-oriented interaction protocols approach, the business logic of the negotiation can be cleanly separated from the protocol specification. Important domain interaction points of the protocol are the evaluation

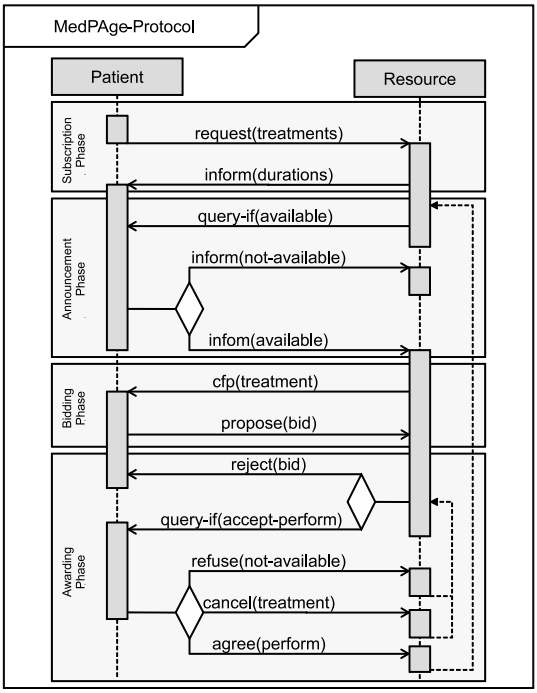

Fig. 6 MedPaCo3 negotiation protocol

of the time slot by the patient agent after receiving the *cfp(treatment)* message and the evaluation of the patient proposals by the resource agent to reject or accept bids.

2.4 Summary

The Jadex BDI architecture simplifies agent programming as it allows for intuitively decomposing agent behavior into responsibilities and abilities, which can be treated separately. Responsibilities of an agent can be obtained from a requirements analysis or an abstract system design and are described explicitly as goals (e.g. world states to be achieved or maintained). The abilities are defined as plans, i.e. procedural recipes how some goals might be pursued. The built-in goal deliberation strategy further allows intuitively controlling the order of goal processing by taking a local perspective of conflicts and precedence relations between goals. Capabilities are a modularization concept that respects all aspects of the BDI architecture and deliberation and can be used for decomposing an agent design into parts that can be independently developed. The goal-oriented interaction protocols approach connects the internal BDI concepts to message-based interaction multi-agent systems and thus allows a seamless integration of both. Ready-to-use

predefined interaction protocols, such as the contract-net, further simplify the development of common interaction patterns.

One design focus of the Jadex BDI architecture was providing a means of agent programming that can be easily learned by programmers with a traditional (e.g. object-oriented) background. On the other hand, the programming model should fit well with a high-level intuitive understanding of an intelligent agent. Experiences with the Jadex framework in numerous software projects as well as teaching courses have shown that the BDI model can be easily understood and represents a natural way of thinking. Following the provided Jadex programming tutorials, students with only Java-knowledge are usually capable of developing their own agents in a short time frame.

In the MedPAge project using agent technology helped with several difficult problems. First, it perfectly mimics the decentralized nature of hospitals with wards and different functional units. The approach respects the existing autonomy of these entities and uses the agent metaphor to represent them explicitly. This allowed modeling the scheduling problem as decentralized coordination approach, in which self-interested patient and resource agents negotiate with each other to reach their goals. Using Jadex facilitated the implementation of the MedPAge system in several ways. Most noteworthy, it allowed a high level system design using Prometheus with a direct mapping to a Jadex implementation, it enabled reuse of functionalities using agent modules and it helped hiding negotiation complexities using interaction goals.

3 BDI in Workflows: GPMN

While a number of challenges in business process management, especially in the area of production workflows, have been addressed in various ways [31], there remains a set of business processes with particular challenges. For example, processes like car model development cover a considerable time span, often multiple years, yet the processes themselves are dynamic. Specific practices may change while the process is in progress and unforeseen events outside the process may have an impact selecting the next set of actions in the process. Furthermore, collaborative processes like product development tend to be unstructured in terms of control flow. The control flow of such a process depends on the actions and discussions of the process participants and is difficult to predict in advance.

Faced with these challenges, it can be seen that a new approach is necessary to address a changing process environment and dynamic business processes if those processes are to be modeled as executable workflows. Since most aspects of the processes are subject to change, the question becomes which parts of the processes are actually stable and can be modeled in an executable workflow. It became clear that the only stable aspects these processes were strategic aspects like business goals. For example, during car development, the business goal of developing a new car model remains the same, even if

the actual means of achieving the goal, the order in which they are achieved or the process environment like new parts or schedules may change over the years.

Thus, a goal-driven workflow modeling language would allow for the required flexibility and agility of the processes. Goals would have to be evaluated during execution and appropriate actions should be selected to further the currently active goals. Since the BDI agent model already offers a goal-centric approach, it is a good candidate for the execution of such workflows. The integration of workflow concepts in Jadex began with the DFG project "Go4Flex" [9] in cooperation with Daimler AG based on previous research conducted at Daimler Group Research regarding goal-oriented workflow concepts [15].

3.1 Related Work

A diverse collection of workflow languages are available both in literature and practice. Often, each language has a particular focus on either business domain-oriented modeling of business processes or the automated execution of processes as workflows. Examples for business domain-oriented approaches include languages such as Yet Another Workflow Language (YAWL [50]), Event-driven Process Chains (EPCs [45]) and BPMN. Execution-centered approaches include ECA (Event Condition Action [29]), Petri nets and the Business Process Execution Language (BPEL [34]).

This distinction is primarily one of degree and not of fundamental limitations. For example, it is certainly possible, provided the semantics are sufficiently defined, to directly execute BPMN using an interpreter and it is also possible, albeit inconvenient, to directly implement a business process in BPEL. In addition, conversion of, for example, BPMN models to BPEL workflow has become a common practice [35].

The languages can be evaluated based on how they address the five perspectives of the holistic business process view proposed by List and Korherr [32] based on earlier work of Curtis et al. [20]. The *functional view* focuses on the actions of a process, i.e. the execution of tasks. This view introduces modeling concepts such as atomic tasks and subprocesses. The *behavior view* centers around the control flow by defining the sequence of the elements of the functional view. This view is often represented using sequence edges and branching elements like XOR- or AND-splits and joins. The necessary data for tasks and data produced by tasks are represented in the *informational view*. This can include both simple information as well as complex business data structures, products and services. Organizational structures such as roles, actors and organizational units are represented in the *organizational view*. This includes the representation of work distribution and responsibilities. Finally, process meta issues and important process characteristics like strategic and

operational business goals and their performance metrics in the form of key performance indicators are included in the *context perspective*.

The first four perspectives are relatively well-established and represented in workflow and business process modeling language to a varying degree. The most comprehensive approach in this regard is the ARIS house of business engineering [45]. The context perspective is a more recent addition and, as a result, tends to be less represented and connected to the other four perspectives. Most modeling languages like YAWL, BPEL and BPMN are strongly focused on the behavior and functional perspectives, featuring a limited support for the organizational and informational perspectives, often relying on external models and means to provide more comprehensive support. The context perspective generally receives little support or is completely ignored.

This situation is based both on practical consideration as well as difficulties integrating the various perspectives in a comprehensive model. The ARIS approach, which tends to be the most comprehensive, solves the problem of multiple perspectives by introducing a myriad of models to represent them. The disadvantage of this approach is the lack of integration and the risk of diverging models during both the initial development of a workflow model and later workflow reengineering.

The business goals of a process could potentially be used to integrate both the context perspective and the behavior perspective. They not only represent the reasons and motivation for the process but they would also influence the execution of a workflow model in a workflow engine depending on their specification. Before our approach, attempts have been made to integrate the context perspective using the user requirements notation (URN) in conjunction with use case maps (UCM) and the goal-oriented requirements language (GRL) [42]. However, unlike the approach presented here, this does not use goals as both functional and non-functional features and therefore does not integrate the context and behavior perspective.

Our approach is based on earlier work on the goal-context method developed at Daimler AG [15], which has also been spun off as a commercial tool [18]. However, this commercial tool uses are more straightforward processing of the goals and does not include the BDI reasoning process central to our approach.

3.2 Approach

Most business process modeling languages are centered on the ordering and execution of tasks. For example, BPMN uses sequence edges and gates to direct the control flow towards the appropriate task elements. In contrast, the approach presented here attempts to focus on the business reasons for the process instead of the individual actions that are required to satisfy the process. This shifts the perspective away from the question of how to solve

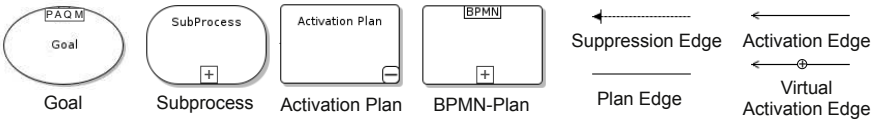

Fig. 7 GPMN elements (from [26])

a problem and emphasizes why action is needed and what target state is desired.

This is accomplished by introducing business goals as process modeling element. In order to model a new workflow, the workflow engineers first determine the central business goal that the process aims to accomplish. For example, in case of a car development process, this central goal can be to develop a new car model. This first goal tends to be very abstract and cannot be easily reflected with concrete tasks and actions. Therefore, the next step involves decomposing the goal into multiple *subgoals*, which, when accomplished, implicitly achieve the original goal. These subgoals then can be further broken down into more subgoals until the goals are sufficiently concrete and simple enough to accomplish them using a relatively basic and straightforward set of actions, which are then expressed as a simple BPMN workflow fragment.

This section will elaborate on the goal modeling language used to specify such goal-oriented processes and describe the technical infrastructure used to support such processes in a productive environment.

3.2.1 Goal-Oriented Process Modeling Notation

Since current workflow languages like BPMN are task-centric, a new language or at least language elements are needed to represent functional business goals in a process. While it is technically simpler to represent goal hierarchies in a purely textual fashion like BPEL represents traditional workflow models, the goal hierarchy is supposed to represent an abstraction from the technical details and center around business functionality, which also help non-technical and more business-centric people to understand the workflow models. As a result, a graphical representation for the language is desirable.

This language called the Goal-oriented Process Modeling Notation (GPMN) currently consists of four node elements and four edge elements shown in Fig. 7. These elements have been developed and tested within a number of both synthetic test workflows as well as real world workflows found at Daimler AG. The most obvious element is the goal element, which can represent an overall ("top level") business goal of a process or a subgoal of another goal. The language currently offers four kinds of goals that have been derived from the underlying BDI reasoning when found useful in a process. The most common kind of goal is the *achieve goal* which aims to reach a certain process state. On the other hand, if the process simply requires to perform a certain action without

regard for the process state, the *perform goal* is used. The third kind of goal, the *query goal,* is used to acquire information relevant to the process. Finally, the most complex goal kind is the *maintain goal* which constantly monitors a condition and if that condition is violated, aims to re-establish a state in which the condition becomes true again.

In order to express available actions to accomplish a goal, one or more *plans* can be attached to the goals using a *plan edge*. Each plan represents an option for achieving the goal and multiple plans may be tried before a goal is achieved. Currently there are two types of plans available. The most direct way of associating tasks with a goal is to attach a *BPMN plan*. This type of plan represents a workflow fragment implemented in BPMN, specifying exactly which tasks are required to attempt to achieve the goal. However, in order to decompose goals into subgoals, the second type called *activation plan*, is needed. This type of plan is used to activate further subgoals which together achieve the plan's goal. The subgoals are defined by connecting the activation plan with the subgoals using the *activation edge*. Since the activation plan is very simple and is often unnecessary to understand the process, it is possible to hide it. The plan edge, the activation plan and the activation edges are then replaced by multiple *virtual activation edges* directly connecting the main goal and its subgoals.

Sometimes goals are in conflict with each other or can possibly interfere with each other if both are active at the same time. One way of resolving this conflict is to consider one goal to be more important and temporarily suppressing the other goal while it is active. This situation can be modeled using *suppression edges*. A goal with a suppression edge pointing to a second goal will suppress that second goal until it becomes inactive either through success or failure.

Finally, a *subprocess* element enables the workflow engineer to modularize the workflow. This is useful when the workflow is very large and the resulting model would consist of an overwhelmingly complex goal hierarchy. The subprocess element lets the workflow engineer split off part of that hierarchy and integrate it in a separate process model.

3.2.2 Process Context

The order of execution in the workflow is influenced by conditions based on the *process context*. The context not only contains the complete state of the workflow during execution but also reflects the environment of the workflow. This can include information such as customer information, delivery estimates, machine states and information about unusual events which have impact on the workflow.

Both goals and plans have a number of conditions whose state is influenced by the context. For example, a drop condition will, if it becomes true, cause the goal to be dropped and no longer considered while a creation condition will pick up a new goal once the condition becomes true. A number of

conditions are specific to the goal kind. Achieve conditions specify the context state when an achieve goal should be considered successful. Maintain conditions on the other hand define the context state that a maintain goal aims to maintain. The context conditions of plans are used by the workflow to decide whether a particular plan is applicable under the current circumstances. For example, an achieve goal which tries to acquire transportation for an employee between two locations may have two plans, one for booking plane flights and one for train rides. However, if one of the locations lack an airport, the plan for booking flights is inadequate for achieving the goal and thus is excluded based on the context.

The process context emphasizes the context perspective and deemphasizes the behavior view by making task selection and order implicit and context-dependent instead of explicit using sequences and branches in traditional workflow languages. This allows the workflow engineer to trivially include escalation and exception handling in the workflow by adding an appropriate set of maintain goals instead of including a large number of branches and event triggers within the workflow.

3.2.3 Technical Implementation

A number of tools have been implemented to support GPMN workflows. Modeling and reengineering GPMN workflows is done using two editors. The GPMN editor is used to model the goal hierarchy and define the process context. The second editor is used to model the workflow fragments used for the BPMN plans. Both editors generate XML files which contain the model of the workflow.

The next step after generating the workflow models using the editors involves their execution using a workflow engine. A workflow engine creates an instance of the workflow based on the workflow model, coordinates the execution of workflow steps and manages the workflow state and context. Since GPMN workflows are inspired by BDI semantics, using a BDI agent platform like Jadex as the basis for a workflow engine was considered to be a good starting point. The models provided by the editors are first loaded and then transformed into a BDI agent model by adding additional parts needed for the agent such as the predefined activation plans.

In order to enable the BDI agent to execute the BPMN workflow fragments used for the BPMN plans, a BPMN interpreter has been developed. This editor uses a loaded BPMN model in conjunction with an internal BPMN state to interpret the BPMN elements in the model. As BPMN tends to contain some ambiguities and inconsistencies in its semantics, only a subset of BPMN elements is currently supported. The BPMN interpreter itself can also be used as an interpreter for standalone BPMN processes, enabling Jadex to execute BPMN workflows as well.

In addition, a workflow management system (WfMS) has been developed around Jadex as the workflow engine roughly based on the reference model of

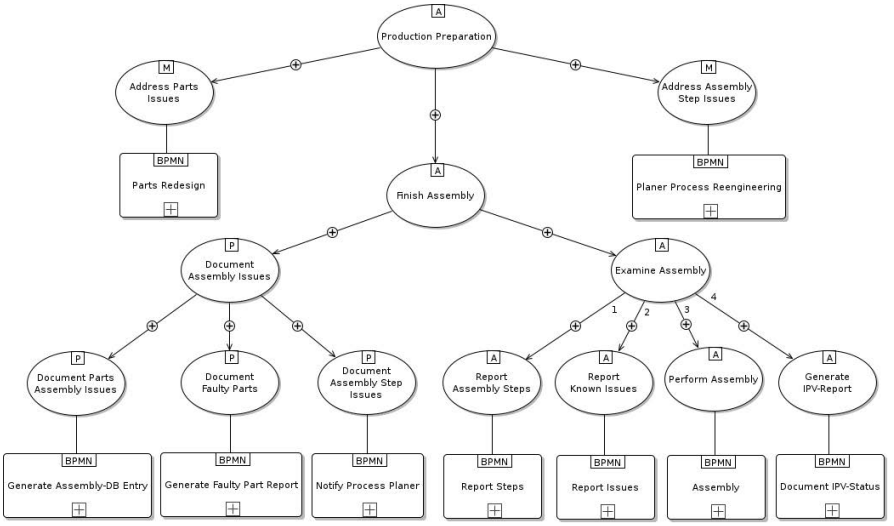

Fig. 8 Partial production preparation process (from [26])

the Workflow Management Coalition (WfMC) [51]. This system provides addition components like user management, security and administrative features like monitoring and model deployment. This workflow management system can be accessed by client software for which an example implementation is also available.

3.3 Application

Goal-oriented workflow modeling has been used in a number of applications at Daimler AG. The example presented here is a partial model of a process used for preparing the production of a new car model.[3] During this process, the production of the car as well as the parts of the car are tested in a production-like environment in order to identify issues both with the car parts as well as the production process. This allows the designers of the parts and workflow engineers to address issues in their respective areas before the new car model is put into factory production.

The process shown in Figure 8 starts with the main "Production Preparation" goal which has to be achieved in order to reach the business goal of the process. From there, it decomposes into multiple subgoals which address three different areas. The first area is the test assembly of the vehicle itself which is a comparably regular and sequential part of the process following a predefined order. This part of the process is consolidated under the "Examine Assembly" goal.

[3] The original workflow has been made abstract due to business secrecy reasons.

While the assembly is in progress, issues that are identified by examining the assembly have to be documented to be addressed at a later point. This is the second area of the process which is summarized by the "Document Assembly Issues" goal. It involves the documentation of part changes that may easy the assembly, parts that are faulty to the point of not allowing proper assembly and defects in the steps of the assembly process, such as missing assembly steps or improper order of assembly steps.

Both the "Examine Assembly" and the "Document Assembly Issues" goals are part of the test assembly and as such are subgoals of the overarching "Finish Assembly" goal which controls the overall performance of the test assembly. Outside the test assembly, the issues that have been found need to be addressed in the appropriate parts or production process workshops. This is accomplished by the third area of the process which consists of the two maintain goals "Address Parts Issues" and "Address Assembly Step Issues". The maintain condition of these goals aim to keep the list of outstanding issues empty. If new issue are found during assembly, the maintain condition is violated and the associated plan is executed which then schedules a workshop to address this new issue.

Since these maintain goals stay active during the whole product preparation process, they are direct subgoals of the main "Production Preparation" goal along with the actual test assembly subgoal "Finish Assembly". This means the main goal and thus the process is not considered to be successful until the test assembly is over and all issues found during assembly have been resolved.

Most activities in the described process involve human tasks. The aim of the agent-based workflow management system is supporting human experts ("knowledge workers") in their activities by improving their coordination. The goal-oriented process description allows the agent to determine dynamically, which activities are enabled or required in response to certain events. The agent thus knows to re-enable corresponding activities automatically (e.g. scheduling a "Parts Redesign" when hen a faulty part issue was found). Using techniques such as work item lists, the knowledge workers can quickly asses the state of the process and which activities are required by them. The process state is automatically managed by the agent and updated to reflect the current situation. E.g. if a faulty part issue was found, but a change of the overall car design no longer requires the part, then the issue is automatically removed, because resolving the part issue is no longer a subgoal of the process.

3.4 Summary

The GPMN workflows presented demonstrate how BDI reasoning and agent-centric approaches can be used to address challenges in the area of business process management and workflow modeling. The language has been found

particularly useful for processes that are either subject to a highly dynamic process context, are particularly long-running or have a low degree of structuring like collaborative and development processes. The goal-based approach lets the workflow engineer focus more on the process objectives than on the order of tasks and puts the context perspective into focus instead of modeling the workflow around the behavior perspective. The result of this additional abstraction allows the workflows to be more accessible by non-technical participants who are more focused on the business side of process management since the concept of business goals are already well known and map well onto GPMN processes.

Overall, GPMN processes offer some unique opportunities to business process management. In addition, they already have a background of being tested against real world challenges at Daimler AG that have so far been hard to address using traditional means and known workflow modeling languages.

4 Agents, Components and Services: Active Components

In practice only few agent-based systems have been developed and deployed in an operative setting. In contrast, other programming models such as object, component and service orientation have gained wide industrial acceptance. One could argue that agent orientation is still a very new conceptual approach and its market penetration will is still to come and will steadily increase in future. An argument that debilitates this view is the fact that service orientation is newer than agent orientation and industry interest has been much higher already since the beginnings of the adoption. The reasons for not using agent technology in practice are manifold but several obstacles can be clearly identified.

One such obstacle of particular importance is the set of programming abstractions for agent systems, which is very different from the other programming paradigms. A developer has to deal with ontologies, asynchronous message based communication, speech acts, internal and possibly social agent architectures. So the learning effort required for developers is high and existing knowledge e.g. from object orientation only partially helps to cope with these new concepts. In order to alleviate the low conceptual integration of agents the active component metaphor has been conceived. The objective consists in combining the advantages of agents with those of services and components by bringing together their main characteristics in a new conceptual entity. The resulting active components still have all characteristics of agents but extend and enhance the software technical construction means by fostering explicitly reusability, modularity and service based interactions. This does not only lead to a steeper learning curve as active components are more similar to already known approaches, it also helps using active components, hence agents, in the context of today's predominant service oriented projects.

4.1 Related Work

There are many approaches aiming at a combination of different software technical strands, whereby these can be distinguished by the dominating paradigm that was used as starting point for the fusion. Furthermore, the approaches can be classified according to the integration layer targeted, i.e. is a conceptual or a rather technical solution sought.

Considering agents as primary conceptual background most approaches remain oriented towards a technical integration of agents with services. Prototypical examples are the WSIG [2] and WADE projects, which are extensions of the widely used JADE agent platform [2]. WSIG is the web services integration gateway and facilitates the interaction of web services and JADE agents. On the other hand, WADE extends agents with workflows, so that agent behavior can be modeled graphically as processes.

In the area of component models several approaches exist that target distribution and concurrency and for this reason partially adopt agent or actor model ideas. An example is the Fractal framework [14], which has been advanced in the ProActive [1] project towards active objects. Similarly, in the JCoBox project [46] a component model with active object ideas has been devised. This model introduces coboxes as active entities that own passive objects and use tasks inside of coboxes for behavior execution. The model isolates objects of coboxes from other coboxes and thus adopts the typical separated actor memory model. In addition, with AmbientTalk [49] a new programming language for ambient intelligence has been proposed. The ambient communication and memory model is similar to JCoBox but its focus is on providing solutions for mobile ad-hoc networks. Furthermore, the component model of AmbientTalk is rather restricted and does not provide composition means so far. Both approaches, JCoBox and AmbientTalk, share some important conceptual ideas with active components with the main differences that they do not introduce internal component architectures for behavior definition and follow a language based instead of a framework based specification path.

It can be seen that conceptual integration of agent, component and services has been tackled partially by other existing approaches. Most close to active components are two promising strands of research. Firstly, SCA[33] successfully integrates services with components and leverages the way SOA based application can be built. Secondly, some component models like JCoBox and AmbientTalk bring together concurrency and distribution with traditional component concepts. Hence, they foster the usage of component models in dynamic application scenarios. Active component combines these efforts and further leverages the behavior specification means by introducing the internal architecture concept from agents.

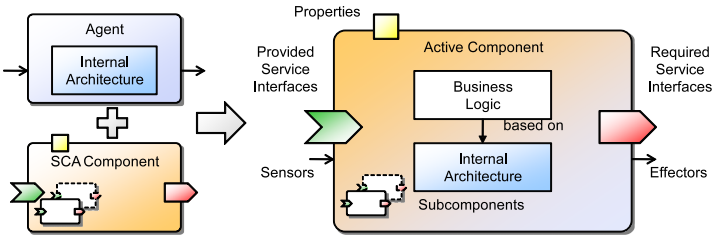

Fig. 9 Active component structure

4.2 Approach

Recently, major IT industry vendors such as IBM, Oracle and TIBCO have proposed a new software engineering approach called service component architecture (SCA) [33], which is meant to be a unification of component and service oriented architecture (SOA) concepts. The general idea of SCA consists in introducing a hierarchical component model for distributed systems. The SCA approach fosters dealing with *complexity* and *reuse*. Complexity is addressed by separating the programming model from concrete communication protocols so that these protocols are largely part of the application configuration and not of the functional program part itself. In this way SCA shields developers from protocol details and allows building applications that communicate using a different set of protocols. Reuse is facilitated by SCA by relying on components and services as basic building blocks of software. Per definition components are considered as rather self-contained entities that exactly define what they need and offer via required and provided services. Hence, components make clear in which contexts they can be used and which functionality can be expected from them. Active components aim at combining the SCA model with agent characteristics in order to conceive a programming model that is capable to deal with scenarios that exhibit a *highly dynamic* and *concurrently acting* set of service providers. In the following subsections the structure, behavior and composition of active components are explained in detail.

4.2.1 Structure

Fig. 9 presents an overview of the synthesis of SCA and agents to active components. On the left hand side schematic views of an agent and an SCA component are depicted. In can be seen that an agent is characterized by its capability of interacting via asynchronous message passing and internally uses an internal agent architecture for encapsulating its behavior control. In contrast, an SCA component interacts with other components by relying on interconnected required and provided services. By including subcomponents higher-level functionalities can be composed of available lower-level component building blocks. Furthermore, an SCA component has clearly defined

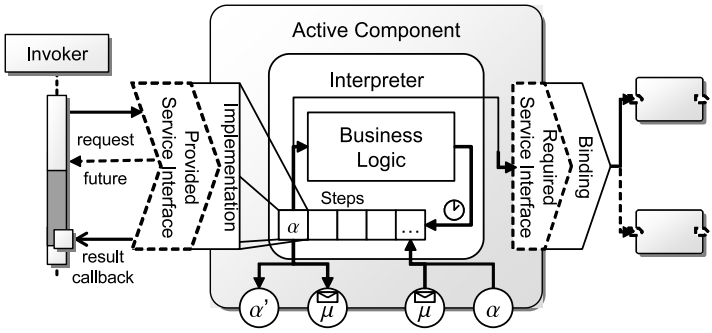

Fig. 10 Active component behavior

configuration points called properties, which can be used to equip it with specific startup argument values.

The merger of both approaches is shown at the right hand side of Fig. 9. Using a black-box perspective, an active component looks very similar to a traditional SCA component except for the small extension that an active component allows for message based interactions in the same way as agents do. The most significant enhancement of the SCA component concept results from the inclusion of the internal architecture concept as component part. This allows active components to realize autonomous behavior that goes beyond its passive service functionalities. In contrast, internal architectures enable the development of reactive and also proactive component behavior that can e.g. be used for expressing workflow logic.

4.2.2 Behavior

In Figure 10 the behavior model of active components is shown. It is considerably different if compared with agents and SCA components because it has to combine a service oriented with an agent oriented perspective on how behavior can be realized. Especially, active components need to respect the most important property of agents, their autonomy, in order to be usable for constructing scenarios of components with possibly cooperative or defective intentions. With respect to active components, autonomy needs to be reflected in the way service calls are processed. No active component should be forced to execute a service call if it cannot or does not want to do it. Hence, service calls have always to be decoupled from the caller to allow the called component to reason about the service request. The only contract that is ensured by service invocations is that the caller is eventually notified about the call result, being it a value or an exception.

Technically, the decoupling is realized using futures [48], which represent place holders for the result of asynchronous processing. For each arriving service call a component immediately returns a future return value and also

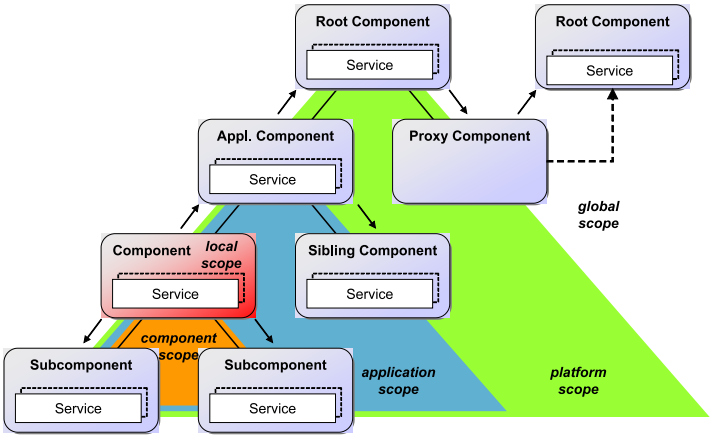

Fig. 11 Predefined dynamic binding scopes

schedules an action representing the call in an action queue. Each component is equipped with an interpreter operating on the queue and processing the contained actions one by one. The action representing the call may optionally lead to reasoning about the call and eventually to its execution or refusal. After service processing has finished, the interpreter fills the future with the real result or exception triggering the resumption of the caller's processing.

In addition to incoming service calls a component also has to deal with outgoing service calls. These calls are targeted on another active component and a required service binding defines how this component is found. The mechanisms for specifying and managing such bindings are part of the active component composition as described next.

4.2.3 Composition

The composition model of active components allows for *static* as well as *dynamic* component interconnections. Static composition means that developers use a deployment specification in order to directly wire specific component instances with each other. The advantage of this classical composition model, that is adopted by SCA and other component models, consists in the possibility of creating self-contained components that use their own subcomponents to bring about internally needed functionality on their own. Therefore, such components can be made applicable in many usage contexts by minimizing the number of required service interfaces on the component top-level. On the other hand, static wiring is not an acceptable solution for many dynamic real world application scenarios, in which service providers may appear and vanish at runtime [28].

For this reason, the active components approach supports besides a static wiring also dynamic binding based on search specifications (cf. [40]). Dynamic

binding specification use search scopes for locating appropriate services in components depending on the proximity with respect to the searching component. Some predefined scopes, that proved to be useful in practice, are depicted in Fig. 11. They range from local scope, which only considers services of the searching component itself, over component and application scope, which extend the area towards sub- and all application components respectively towards platform and global scopes that include the whole platform and even all accessible remote sites. Further it is planned to support also the application dependent definition of user search scopes allowing developers to reflect their specific domain needs.

4.3 Application: JadexCloud

To illustrate the active component development approach, the JadexCloud infrastructure will be presented. JadexCloud represents a middleware for private enterprise cloud scenarios and especially highlights the advantages of the active component programming model for utility computing. JadexCloud [8] is itself a middleware, currently in development, for running applications based on the active component concept within private enterprise clouds. The general idea consists in supporting cloud application not only in homogeneous high-end data centers, but also in existing heterogeneous company computing networks, which typically consist of a mix of differently powerful and utilized machines. In such a setting cloud applications have to be designed in a very modular fashion, so that dynamic relocations of certain application parts can be performed at runtime, whenever the infrastructure or application needs change. JadexCloud makes use of active components in two ways. First, the infrastructure is built itself based on active component concepts and secondly, it supports the execution of cloud applications developed with active components.

Key concept of the proposed JadexCloud architecture is a layer model that helps separating responsibilities and managing complexity. It is composed of the following three layers: *daemon layer*, *platform layer* and *application layer*.

The daemon layer is the foundation for creating a cloud of interconnected nodes. This is done by small daemon platforms that need to run on each host that should participate in the cloud. The daemon platform includes an awareness service, which is capable of automatically detecting other platforms. The awareness service relies of different discovery mechanisms that can be used to discover new nodes. Currently, several mechanisms for detecting nodes in a local network exist relying IP broadcast and multicast schemes. Furthermore, to build up networks with hosts from different networks a relay discovery mechanism has been developed, which acts as a bridge between platforms. It is planned to further extend the relay mechanism in the direction of a redundant supernode structure known from several peer-to-peer networks. In the network a single node can always construct an actual

view of available network resources. Furthermore, the daemon layer allows for basic management functionalities for application handling. Concretely, application components can be started and terminated. In order to enforce a strict separation between applications those components are started on newly started application platforms that are controlled by the daemon. Application management further requires that software bundles of applications can be accessed in specific versions, for normal startup as well as for rolling out updates of existing applications. The daemon layer handles this by utilizing software repositories that can be hierarchically organized, i.e. distinguishing local, companywide and global repositories.

On top of the daemon layer the platform layer offers a global administration view for deployment and management of applications within the cloud. The entry point for the platform layer is the so called JCC (Jadex Control Center), which offers a canon of remotable tools for setting up an application and monitoring its behavior. All nodes build up the cloud from their local perspective so that an arbitrary node with JCC can be used for application management. Based on local configuration options and user privileges, the JCC provides access to a subset of the existing nodes called the *cloud view*. The administrator can choose, which nodes to include in the deployment of an application, by assigning application components to the platforms running on the different nodes. To start the separate components, each platform will obtain the required component implementations from the repository.

The application layer, sitting on top of the platform layer, deals with how a distributed application can be built based on the active components paradigm as well as providing tools for debugging and testing applications during development. Besides the already presented general concepts of active components providing cloud ready applications need to especially consider the specification of non-functional aspects like resource needs of component instances. These aspects will be part of a deployment description for an application, which can be evaluated by the platform layer for creating a deployment plan, i.e. an ideal initial component-resource mapping, and also for component relocations at runtime. One non functional key property that has to be ensured at runtime is fault tolerance of software components. In case fault tolerance is needed for specific components they will be replicated and checkpointing will be employed to ensure that components can be restarted after a crash has occurred. Furthermore, it is planned to wrap already existing cloud services, for example for storage of data in the cloud, and make them in this way accessible for active components in a natural way.

4.4 Summary

This section has briefly introduced the active component concept, which unifies central component with agent ideas. The integration has been done by extending the SCA component model with internal architectures. As a result

components may own not only service driven passive but also autonomous self governed behavior. Looking at active components from the outside they appear no different to traditional SCA components so that the advantages of managing complexity and reuse by hierarchical composition and abstraction from technology dependent communication means remain established. Details of active component structure, behavior and composition have been introduced and further explained. A common objection that is put forward against active components concerns the potential loss of autonomy that is caused by using service, i.e. method based interactions. This argument is not valid as a service provider is still free to reason about performing service requests before they are actually executed. Services just introduce sound software technical foundations for typed interactions. An in depth discussion about method calls and agents is out of scope for this chapter but can be found in [7].

In JadexCloud, agent and active component technology is helpful with respect to several aspects. It defines a new programming model for cloud applications that naturally supports a louse coupling of the components and thus allows for dynamic reconfigurations of the applications according to the system demands. Furthermore, the complexity of the JadexCloud architecture became better manageable by explicit service interfaces introduced in active components. In this way the interfaces between the layers and between service requesters and providers could be cleanly software technically described.

5 Conclusion and Outlook

Agent technology offers intuitive concepts for describing distributed systems but implementing them is hard, time consuming and very different to other established technologies. In the following, some of the lessons learnt regarding the programming model for agent systems are summarized:

- The concepts for programming agents depend on the internal agent architecture used. In literature many different agent architectures have been proposed, often inspired from other disciplines like biology, psychology or philosophy. At the modeling and implementation layer this leads to a huge heterogeneity of approaches and requires considerable learning efforts. From a developer perspective it is advantageous to use an agent architecture that fits the concrete project requirements, especially the complexity of the agent functionalities is an indicator the choice of the right programming model. Due to this wanted flexibility agent platforms should not prescribe a specific programming model but either allow developers to choose a model among different options or provide a simple model that can be used to build custom extensions on top of it. One architecture of specific importance is the BDI model as it is a hybrid approach that combines reactive with proactive features, i.e. it is able to timely react

of environmental events and is also capable cognitive behavior based on the way human rational action is explained. In Jadex both aspects have been taken into account. On the one hand, different agent architectures are supported by the kernel concept and on the other hand a BDI kernel exists that fits many common use cases.

- Besides the agent itself, the inter agent layer plays an important role for realizing multi-agent system. It has been found that building interaction based purely on speech act based asynchronous messages is error prone and difficult as no compile time checks can be done and profound knowledge regarding the FIPA message format, ontologies and interaction protocols is required. Furthermore, agent systems are typically peer-to-peer and lack a mechanism for hierarchical decomposition, which is essential for handling complexity in large systems. Conceptually, holons fill this gap but existing frameworks do not pick up these ideas. In Jadex these challenges have been addressed by the active components metaphor, which allows service-based asynchronous interactions and allows components to have subcomponents.

The main motivation of Jadex has also been to facilitate the practical usability of agent technology. In this respect all developments described in this chapter have strived to deliver conceptual solutions that are bound to generically usable software. Respecting the problems and challenges from above, Jadex has been developed with the rationale in mind to connect agents tighter to established approaches. Concretely, with a BDI approach on basis of established programming languages like Java and XML programming of goal directed intentional agents was made easily accessible also for inexperienced agent developers. Furthermore, it has been shown how BDI agent concepts can be adopted for goal driven workflow descriptions. As goal are considered more stable than activities these kinds of workflows help to cope with frequently changing business processes. Finally, with active components a unification of agent, services and components has been introduced. Active components strongly contribute to the problem of lacking industry adoption as they are based on the standardized and industry driven SCA model. As part of ongoing work, active components are field tested in commercial applications, concretely tackling the area of business intelligence, as well as scientific mass calculations.

References

1. Baude, F., Caromel, D., Morel, M.: From Distributed Objects to Hierarchical Grid Components. In: Meersman, R., Schmidt, D.C. (eds.) CoopIS 2003, DOA 2003, and ODBASE 2003. LNCS, vol. 2888, pp. 1226–1242. Springer, Heidelberg (2003)
2. Bellifemine, F., Caire, G., Greenwood, D.: Developing Multi-Agent systems with JADE. John Wiley & Sons (2007)

3. Bordini, R., Hübner, J.F., Vieira, R.: Jason and the Golden Fleece of Agent-Oriented Programming. In: Bordini, R., Dastani, M., Dix, J., El Fallah Seghrouchni, A. (eds.) Multi-Agent Programming: Languages, Platforms and Applications, pp. 3–37. Springer (2005)
4. Bratman, M.: Intention, Plans, and Practical Reason. Harvard University Press (1987)
5. Bratman, M., Israel, D., Pollack, M.: Plans and Resource-Bounded Practical Reasoning. Computational Intelligence 4(4), 349–355 (1988)
6. Braubach, L., Pokahr, A.: Goal-Oriented Interaction Protocols. In: Petta, P., Müller, J.P., Klusch, M., Georgeff, M. (eds.) MATES 2007. LNCS (LNAI), vol. 4687, pp. 85–97. Springer, Heidelberg (2007)
7. Braubach, L., Pokahr, A.: Method calls not considered harmful for agent interactions. International Transactions on Systems Science and Applications (ITSSA) 1/2(7), 51–69 (2011)
8. Braubach, L., Pokahr, A., Jander, K.: JadexCloud - An Infrastructure for Enterprise Cloud Applications. In: Klügl, F., Ossowski, S. (eds.) MATES 2011. LNCS, vol. 6973, pp. 3–15. Springer, Heidelberg (2011)
9. Braubach, L., Pokahr, A., Jander, K., Lamersdorf, W., Burmeister, B.: Go4Flex: Goal-Oriented Process Modelling. In: Essaaidi, M., Malgeri, M., Badica, C. (eds.) Intelligent Distributed Computing IV. SCI, vol. 315, pp. 77–87. Springer, Heidelberg (2010)
10. Braubach, L., Pokahr, A., Lamersdorf, W.: Extending the Capability Concept for Flexible BDI Agent Modularization. In: Bordini, R.H., Dastani, M.M., Dix, J., El Fallah Seghrouchni, A. (eds.) PROMAS 2005. LNCS (LNAI), vol. 3862, pp. 139–155. Springer, Heidelberg (2006)
11. Braubach, L., Pokahr, A., Moldt, D., Lamersdorf, W.: Goal Representation for BDI Agent Systems. In: Bordini, R.H., Dastani, M.M., Dix, J., El Fallah Seghrouchni, A. (eds.) PROMAS 2004. LNCS (LNAI), vol. 3346, pp. 44–65. Springer, Heidelberg (2005)
12. Braubach, L., Pokahr, A., Lamersdorf, W.: A universal criteria catalog for evaluation of heterogeneous agent development artifacts. In: Sixth International Workshop From Agent Theory to Agent Implementation, AT2AI-6 (2008)
13. Brooks, R.: A Robust Layered Control System For A Mobile Robot. IEEE Journal of Robotics and Automation 2(1), 24–30 (1986)
14. Bruneton, E., Coupaye, T., Leclercq, M., Quéma, V., Stefani, J.-B.: The fractal component model and its support in java: Experiences with auto-adaptive and reconfigurable systems. Softw. Pract. Exper. 36(11-12), 1257–1284 (2006)
15. Burmeister, B., Arnold, M., Copaciu, F., Rimassa, G.: Bdi-agents for agile goal-oriented business processes. In: AAMAS 2008, pp. 37–44. IFAAMAS (2008)
16. Busetta, P., Howden, N., Rönnquist, R., Hodgson, A.: Structuring BDI Agents in Functional Clusters. In: Jennings, N.R. (ed.) ATAL 1999. LNCS, vol. 1757, pp. 277–289. Springer, Heidelberg (2000)
17. Busetta, P., Rönnquist, R., Hodgson, A., Lucas, A.: Jack Intelligent Agents - Components for Intelligent Agents in Java. AgentLink News (2), 2–5 (1999)
18. Calisti, M., Greenwood, D.P.A.: Goal-Oriented Autonomic Process Modeling and Execution for Next Generation Networks. In: van der Meer, S., Burgess, M., Denazis, S. (eds.) MACE 2008. LNCS, vol. 5276, pp. 38–49. Springer, Heidelberg (2008)
19. Cohen, P.R., Levesque, H.J.: Teamwork. Technical Report Technote 504, SRI International, Menlo Park, CA (March 1991)

20. Curtis, B., Kellner, M., Over, J.: Process modeling. Com. ACM 35(9), 75–90 (1992)
21. Dardenne, A., van Lamsweerde, A., Fickas, S.: Goal-directed requirements acquisition. Science of Computer Programming 20(1–2), 3–50 (1993)
22. Dastani, M., van Riemsdijk, B., Meyer, J.J.: Programming Multi-Agent Systems in 3APL. In: Bordini, R., Dastani, M., Dix, J., El Fallah Seghrouchni, A. (eds.) Multi-Agent Programming: Languages, Platforms and Applications, pp. 39–67. Springer (2005)
23. Dennett, D.: Intentional systems. Journal of Philosophy (68), 87–106 (1971)
24. Georgeff, M., Lansky, A.: A system for reasoning in dynamic domains: Fault diagnosis on the space shuttle. Technical Report Technical Note 375, Artificial Intelligence Center, SRI International, Menlo Park, California (1986)
25. Huber, M.: JAM: A BDI-Theoretic Mobile Agent Architecture. In: Etzioni, O., Müller, J., Bradshaw, J. (eds.) Proceedings of the 3rd Annual Conference on Autonomous Agents (AGENTS 1999), pp. 236–243. ACM Press (1999)
26. Jander, K., Braubach, L., Pokahr, A., Lamersdorf, W.: Goal-oriented processes with gpmn. International Journal on Artificial Intelligence Tools, IJAIT (2011)
27. Jennings, N., Mamdani, E.: Using Joint Responsibility to Coordinate Collaborative Problem Solving in Dynamic Environments. In: AAAI, pp. 269–275 (1992)
28. Ježek, P., Bureš, T., Hnětynka, P.: Supporting Real-Life Applications in Hierarchical Component Systems. In: Lee, R., Ishii, N. (eds.) Software Engineering Research, Management and Applications 2009. SCI, vol. 253, pp. 107–118. Springer, Heidelberg (2009)
29. Knolmayer, G., Endl, R., Pfahrer, M.: Modeling processes and workflows by business rules. In: Business Process Management, Models, Techniques, and Empirical Studies, pp. 16–29. Springer, London (2000)
30. Lehman, J.F., Laird, J., Rosenbloom, P.: A gentle introduction to Soar, an architecture for human cognition. In: Sternberg, S., Scarborough, D. (eds.) Invitation to Cognitive Science, vol. 4, pp. 212–249. MIT Press (1996)
31. Leymann, F., Roller, D.: Production Workflow: Concepts and Techniques. Prentice Hall PTR (2000)
32. List, B., Korherr, B.: An evaluation of conceptual business process modelling languages. In: SAC 2006, pp. 1532–1539. ACM (2006)
33. Marino, J., Rowley, M.: Understanding SCA (Service Component Architecture), 1st edn. Addison-Wesley Professional (2009)
34. OASIS. Web Services Business Process Execution Language (WSPBEL) Specification, version 2.0 edition (2007)
35. Ouyang, C., Dumas, M., ter Hofstede, A., van der Aalst, W.: From bpmn process models to bpel web services. In: Proc. of ICWS 2006, pp. 285–292. IEEE (2006)
36. Padgham, L., Winikoff, M.: Prometheus: a methodology for developing intelligent agents. In: Gini, M., Ishida, T., Castelfranchi, C., Lewis Johnson, W. (eds.) Proceedings of the 1st International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2002), pp. 37–38. ACM Press (July 2002)
37. Paulussen, T., Zöller, A., Rothlauf, F., Heinzl, A., Braubach, L., Pokahr, A., Lamersdorf, W.: Agent-based patient scheduling in hospitals. In: Lockemann, P., Spaniol, O., Kirn, S., Herzog, O. (eds.) Multiagent Engineering - Theory and Applications in Enterprises, pp. 255–275 (June 2006)

38. Paulussen, T.O., Jennings, N.R., Decker, K.S., Heinzl, A.: Distributed Patient Scheduling in Hospitals. In: Gottlob, G., Walsh, T. (eds.) Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI 2003), Morgan Kaufmann (2003)
39. Paulussen, T.O., Zöller, A., Heinzl, A., Pokahr, A., Braubach, L., Lamersdorf, W.: Dynamic Patient Scheduling in Hospitals. In: Bichler, M., Holtmann, C., Kirn, S., Müller, J., Weinhardt, C. (eds.) Coordination and Agent Technology in Value Network. GITO, Berlin (2004)
40. Pokahr, A., Braubach, L.: Active Components: A Software Paradigm for Distributed Systems. In: Proceedings of the 2011 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT 2011), IEEE Computer Society (2011)
41. Pokahr, A., Braubach, L., Lamersdorf, W.: A Goal Deliberation Strategy for BDI Agent Systems. In: Eymann, T., Klügl, F., Lamersdorf, W., Klusch, M., Huhns, M.N. (eds.) MATES 2005. LNCS (LNAI), vol. 3550, pp. 82–93. Springer, Heidelberg (2005)
42. Pourshahid, A., Amyot, D., Peyton, L., Ghanavati, S., Chen, P., Weiss, M., Forster, A.: Business process management with the user requirements notation. Electronic Commerce Research 9(4), 269–316 (2009)
43. Rao, A.: AgentSpeak(L): BDI Agents Speak Out in a Logical Computable Language. In: Perram, J., Van de Velde, W. (eds.) MAAMAW 1996. LNCS, vol. 1038, pp. 42–55. Springer, Heidelberg (1996)
44. Rao, A., Georgeff, M.: BDI Agents: from theory to practice. In: Lesser, V. (ed.) Proceedings of the 1st International Conference on Multi-Agent Systems (ICMAS 1995), pp. 312–319. MIT Press (1995)
45. Scheer, A.-W., Nüttgens, M.: Aris architecture and reference models for business process management. In: Business Process Management, Models, Techniques, and Empirical Studies. Springer (2000)
46. Schäfer, J., Poetzsch-Heffter, A.: JCoBox: Generalizing Active Objects to Concurrent Components. In: D'Hondt, T. (ed.) ECOOP 2010. LNCS, vol. 6183, pp. 275–299. Springer, Heidelberg (2010)
47. Shoham, Y.: Agent-oriented programming. Artificial Intelligence 60(1), 51–92 (1993)
48. Sutter, H., Larus, J.: Software and the concurrency revolution. ACM Queue 3(7), 54–62 (2005)
49. Van Cutsem, T., Mostinckx, S., Boix, E.G., Dedecker, J., De Meuter, W.: Ambienttalk: Object-oriented event-driven programming in mobile ad hoc networks. In: International Conference of the Chilean Computer Science Society, vol. 0, pp. 3–12 (2007)
50. van der Aalst, W.M.P., ter Hofstede, A.H.M.: Yawl: yet another workflow language. Information Systems 30(4), 245–275 (2005)
51. Workflow Management Coalition (WfMC). Workflow Reference Model (January 1995)
52. Zöller, A., Braubach, L., Pokahr, A., Paulussen, T., Rothlauf, F., Lamersdorf, W., Heinzl, A.: Evaluation of a multi-agent system for hospital patient scheduling. International Transactions on Systems Science and Applications (ITSSA) 1, 375–380 (2006)

Chapter 3
Extensible Java EE-Based Agent Framework – Past, Present, Future

Milan Vidaković, Mirjana Ivanović, Dejan Mitrović*, and Zoran Budimac

Abstract. *EXtensible Java EE-based Agent Framework* (*XJAF*) is a modular, *FIPA*-compliant multi-agent system developed by the authors of this chapter. The main motivation behind the development of *XJAF* was to demonstrate how existing, standardized Java EE technologies, tools, and libraries, such as *JNDI*, *JMS*, and *EJB*, can be used to implement a large subset of functionalities required from a multi-agent system. Immediate direct benefits of this approach are shorter development time of the system itself, delegation of agent load-balancing to the enterprise server, flatter learning curve for new developers of the system, etc. The first implementation of *XJAF* has been published several years ago and has since been used in several real-life applications. In the meantime, some disadvantages and weaknesses of the system were noticed, and the work is underway to provide a new implementation with an improved quality. The most recent focus of improvements has been on the addition of fault-tolerant techniques, and the increase of interoperability through a *SOA*-based design and web service interfaces.

1 Introduction

Agent technology represents one of the most consistent approaches to distributed software development. *Software agents* can be defined as executable software entities with varying degrees of intelligence, that act autonomously while pursuing their goals.

Milan Vidaković
Faculty of Technical Sciences, University of Novi Sad, Serbia

Mirjana Ivanović · Dejan Mitrović · Zoran Budimac
Department of Mathematics and Informatics, Faculty of Sciences,
University of Novi Sad, Serbia
e-mail: dejan@dmi.uns.ac.rs

* Corresponding author.

M. Ganzha & L. C. Jain (Eds.): Multiagent Systems & Applications, ISRL 45, pp. 55–88.
springerlink.com

A *multi-agent system* (*MAS*) is a software system with an infrastructural support for its agents. Its core functionalities include [7] the agent life-cycle management, a messaging infrastructure, and a service subsystem that effectively supports agents, giving them the possibility of accessing resource, executing complex algorithms, etc. In addition, *MAS*s in many cases offer security features (e.g. agent code and data integrity, and encryption of messages), a connection mechanism that allows cooperation of agents in physically distributed environments, and a support for agent persistence and mobility.

By analyzing *MAS* solutions that existed back in 2003, however, it was concluded that in the majority of cases many of the functional requirements were implemented from scratch. Developers used their own implementations of the message exchange infrastructure, directories of agents and services, and security features, instead of re-using technical solutions to these problems already present in Java EE.

EXtensible Java EE-based Agent Framework (*XJAF*) [47, 48, 49] is a *FIPA*-compliant [16] *MAS* developed by the authors of this chapter. Given the authors' own practical experience in developing Java EE-based applications, as well the strong interest in the agent technology (e.g. [40]), the initial motivation for the development of *XJAF* was to evaluate whether *MAS*s can benefit from re-using existing, standardized Java EE tools and libraries. At the same time, there was a need for an efficient *virtual central catalog* implementation for the library information system *BISIS* [42]. Additionally, the *DIGLIB* system [10] for *Networked Digital Library of Theses and Dissertations* [36] required a framework to harvest metadata from heterogeneous content providers. The two main design goals for *XJAF* have thus been defined:

- To provide an efficient and standards-compliant *MAS* implementation. Java EE has been chosen as the main implementation platform, given its success in the development of scalable, secure, and reliable software solutions for large enterprises.
- To develop a framework with aforementioned problems in minds, but with a high level of reusability. This has been achieved by defining a system of abstract *services* that can be specialized to serve the needs of a particular application.

The choice of Java EE as the *XJAF* implementation platform has proven to be beneficial. Direct advantages of this approach were shorter development time of the system itself, and harnessing of advanced programming features such as runtime load-balancing. More concretely, *XJAF* relies on the following Java EE technologies for its functioning:

- *Java Naming and Directory Interface* (*JNDI*) [28]: used for implementing directories of agents and services.
- *Java Message Service* (*JMS*) [21]: provides the communication (inter-agent messaging) infrastructure.
- *Enterprise JavaBeans* (*EJB*) [11]: used as placeholders for agents and services
- *Java Serialization*: supports agent persistence and mobility.
- *Java PKI API* [25]: used for implementing some of the security features.

- *Java Architecture for XML Binding* (*JAXB*) [26]: simplifies the use of *XML*-based task descriptions.

JMS, for example, has been chosen instead of *IIOP* or *RMI* commonly used by other *MAS*s because it is allows for asynchronous communication, both internal (e.g. among agents residing in the same XJAF) and distributed. *IIOP* and *RMI* are designed primarily for distributed, synchronous communication. Therefore, instead of, for example, developing a messaging infrastructure from scratch, existing *JMS* technology was used. Additionally, because large parts of the system are based on standardized, widely-known technologies, *XJAF* has a flatter learning curve for new developers. Besides these advantages related to the development process itself, an additional important runtime benefit was gained. In *XJAF*, agents and services are initially created as *Plain Old Java Objects* (*POJO*s), but are immediately wrapped inside *EJB* components. *XJAF* itself is managed by a modern enterprise application server, which offers *EJB pooling* and runtime load-balancing. So, with the minimum amount of effort, *XJAF* is able to effectively balance the number of running agents with the available system resources. More details on these benefits are given in Section 3.

The second aforementioned design goal of *XJAF* has also been achieved. By employing the concepts of agent technology, such as mobility, message exchange, and services, the problem of searching for a specific library record in a large number of distributed library catalogues was solved in less than 100K lines of code – significantly lower that any other, non-agent based solution written from scratch (see Section 4 for more details).

At the high-level of abstraction, *XJAF* was designed as a set of loosely-coupled module called *managers*. Each module handles a distinct part of the overall agent management process, such as message passing, task execution, data and code security, etc. There are several advantages of the manager-based approach. New managers can be added dynamically and are recognized solely by their interfaces, which allows for custom implementations. Also, the list of manager is not hard-coded, but rather loaded from a configuration file, which means that new functionalities can be added without the need for changing the system itself.

The rest of the chapter is organized as follows. Section 2 provides a comparison of existing *MAS* architectures with the approaches taken in the development of *XJAF*. Section 3 provides detailed information on the architecture of *XJAF*, both the old and the new implementation. A simple example of using *XJAF*, as well as some existing, more complex practical usages of the system are presented in Section 4. Recent changes and improvements made to the system are presented in Section 5. Finally, Section 6 outlines general conclusions and future work on the presented topic.

2 An Overview of Existing *MAS* Architectures

Various authors have made different design and implementation choices during the development of their *MAS*s. A representative subset of these solutions, with

interesting and efficient approaches to solving specific problems include *ABLE*, *Aglets*, *DimaX*, *FUSION@*, *JADE*, and *Voyager*. These systems are analyzed in greater details in the following subsections.

2.1 ABLE

Agent Building and Learning Environment (*ABLE*) [1, 5, 31] is a Java-based, *FIPA*-compliant *MAS* and a framework for agent development based on machine learning and reasoning. It uses the *JavaBeans* technology for building both agents (*AbleAgents*) and general-purpose, reusable software components (*AbleBeans*). Because *ABLE* agents are, in essence, *JavaBeans* components, they can be constructed from *AbleBeans* as well as from other agents. The environment includes a number of standard beans for data manipulation, rule processing, and learning.

ABLE supports rule-based reasoning. Rules can be written as simple *if-then*s, or by using more complex, *Prolog*-like predicate logic. For this purpose, *ABLE* offers a specialized *Able Rule Language* (*ARL*) similar to Java, with advanced features such as rule priorities, rule metadata, templates, and *OMG OCL* [37] collections. The system includes a number of inference engines, based on backward and forward rule chaining, fuzzy logic, neural networks, scripting, and more.

Agents in *ABLE* communicate using *FIPA ACL* messages. In case of physically distributed environments, Java *RMI* is employed. Persistence of beans and agents is achieved via the *AbleSerializable* interface, which is backed-up by Java serialization.

2.2 Aglets

Aglets [2, 29, 43] is a Java-based *MAS* with the support for agent mobility, synchronous and asynchronous message exchange, and security mechanisms. It is a multi-tier environment, consisting of the following tiers:

- *AgletsRuntime*: a host for agents, *proxies*, and *contexts*. A proxy is an agent placeholder. It serves as a security wrapper, protecting the agent from outside malicious attacks, and vice versa. A context is a link between an agent and its environment.
- *Agent Transport and Communication Interface* (*ATCI*): supports agent mobility and communication over a network, at the higher level of abstraction. It relies on the *ATP* layer for functioning.
- *Agent Transport Protocol* (*ATP*): a low-level infrastructure for sending agents and messages across a *TCP/IP* network. Supported operations include *dispatch*, for sending an agent to a remote server, *retract*, which pulls the agent from the remote server, *fetch*, for the exchange of information, and *message* for sending messages over the network.

Aglets supports event-driven programming, through a set of pre-defined events and appropriate listeners. Example events include *CloneEvent*, *MobilityEvent*, and *PersistencyEvent*.

The security model of *Aglets* is used to control the access to the file system, network resources, and environment threads. Agents can be categorized as *trusted* or *untrusted*, and the access rights for each category can be defined separately.

Although well designed and supporting advanced agent development features, the *Aglets MAS* does not seem to be developed or maintained anymore.

2.3 DimaX

DimaX [12, 20] is a *MAS* built around the concept of fault-tolerance. It is constructed by layering an older Java-based *MAS* named *DIMA* [19] on top of the *DarX* [30] replication framework.

Main services of *DimaX* include agent directory, fault-detection, monitoring, and adaptive agent replication. Fault-detection is based on the *heartbeat* technique which involves agents *pinging* each other to indicate their availability. If an agent does not send any pings within a dynamically determined timeframe, its replica is used to restore it. A similar heartbeat-based approach is used to detect failure of a distributed *DimaX* server.

The monitoring service tries to predict future failures of the system or individual agents. It utilizes specialized agents called *host monitors* and *agent monitors*. Each *DimaX* server contains a single host monitor which collects information at the system level, e.g. about the CPU. This allows agents to leave the troubling environment in a timely manner. Agent monitors, on the other hand, keep track of agents in the system, measuring the amount of sent and received messages, the number of executed tasks, etc. This data is used to create *interdependency graphs* in order to identify agents that are more important than others, and adapt the number of replicas accordingly.

2.4 FUSION@

FUSION@ [44, 45] is a modular *SOA*-based *MAS* with a layered organization. At the lower level, services are used to implement functionalities of the framework. They can be invoked locally or remotely, and can be organized as local or web services. Task distribution is controlled by deliberate, *BDI*-style agents at the upper layer. They employ load-balancing and quality assurance analysis in order to assign the given task to the most appropriate service.

External clients interact with *FUSION@* using *SOAP*, while inner inter-agent communication uses *FIPA ACL*.

There is a set of pre-defined types of specialized agents responsible for performing certain assignments within the system. Examples include:

- *CommAgent*: handles all communication between external users and underlying services.
- *Security Agent*: analyzes the structure and syntax of all incoming and outgoing messages.

- *Directory Agent*: manages the list of services available in the system.
- *Supervisor Agent*: monitors the status of all agents, by occasionally pinging them.

FUSION@ allows developers to changes the behavior of all pre-defined agents, as well as to add new agents in accordance to the needs of their project.

2.5 JADE

Java Agent DEvelopment Framework (*JADE*) [4, 24] is probably the most widely-used *MAS* implementation. The system is very well documented, and it's supported by a large community of developers. *JADE* is *FIPA*-compliant and relies on advanced Java features for its functioning, which include *RMI*, *CORBA IDL*, Serialization, and *Reflection API*.

At runtime, *JADE* framework can be organized into multiple containers, which can then be distributed over a local area network. A container serves as a fully-featured agent environment. All containers are linked to a single *Main* container.

Internally, agents communicate using *FIPA ACL* messages. Supported communication patterns also include message exchange with other *JADE* environments, as well as with *MAS*s from other vendors. In the first case, messages are serialized and transported using *RMI*. In the latter case, *IIOP* can be used, but the system also supports the addition of new protocols.

Each *JADE* agent has its own thread of control. Multi-threaded agents can also be developed, in which case *JADE* provides synchronization of the concurrent access to the agent's message queue. An agent task is defined as a direct sub-class of the *Behavior* class, or one of its more specialized sub-classes. For example, *CyclicBehavior* defines tasks that are executed in a continuous loop.

JADE has a built-in support for agent mobility and includes various security features, such as authentication, authorization, and encryption of exchanged messages. However, one of the most important feature of the framework is extensibility through third-party add-ons. Add-ons can be used to modify, fine-tune, or add completely new functionalities to the system. An example add-on is *Web Service Integration Gateway* (*WSIG*) [6] which automatically publishes agent's services as web services.

2.6 Voyager

Voyager [50] is primarily an application server, but it can also be effectively used for agent development [17]. The *Voyager* package includes following components (among others):

- *Voyager Application Server*: an *EJB* server, with the support for *JSP*.
- *Voyager ORB*: enables communication between the *Voyager* system and external entities using *SOAP*, *CORBA*, *RMI*, and *DCOM* objects. In addition, it includes a distributed naming service, a sub-system for object persistency, etc.

- *Voyager ORB Professional*: a layer on top of *Voyager ORB*, it includes a *GUI* management console, and *JNDI*, load-balancing, and *CORBA* naming services.
- *Voyager Security*: authentication and authorization system, which utilizes *SSL*, *Firewall Tunneling* using the *SOCKS* protocol, as well as *HTTP*.
- *Voyager Transactions*: supports *Object Transaction Service* and *JTA*.

Voyager agents communicate by exchanging messages, using synchronous (i.e. blocking), one-way, one-way multi-cast and *future* communication patterns. The system relies on the *JMS* technology for message transport. Agent mobility is also supported, through Java Serialization.

2.7 *Comparisons with* XJAF

The main difference between *XJAF* and *MAS*s presented above is in the use of Java EE as the implementation platform. Endorsed by a large portion of corporate enterprizes, Java EE includes technologies, libraries and tools that ease the development of efficient, reliable, and scalable software. This makes it an excellent platform for *MAS* development. For example, by wrapping agents inside *EJB* components, *XJAF* delegates the agent life-cycle management to the underlying enterprize application server. This approach increases the system's performance, because modern application servers employ *EJB pooling* and runtime load-balancing techniques. Out of the *MAS*s described earlier, only *Voyager* attempts to fully utilize the Java EE platform. However, *Voyager* is an application server written from scratch, and represents a commercial product. *XJAF*, on the other hand, executes inside an existing, open-source solution (i.e. GlassFish [18]). *BlueJADE* [9] is a project aimed at incorporating Java EE with *JADE*, but it only transforms the entire framework into a manageable service.

In a recent improvement of the *XJAF* architecture, a special care has been dedicated to the creation and maintenance of fault-tolerant networks. In a distributed environment, it allows the remaining interconnected *XJAF* instances to cooperate correctly regardless of the number of broken nodes in the network. In addition, a robust agent tracking technique has been introduced, allowing the system to locate a mobile agent even if some nodes in its path break. None of the presented systems have taken the fault-tolerant network management to this extent. On the other hand, support for agent replication is at the core of *DimaX*, and it is also present in *JADE*. *FUSION@* employs monitoring of agents and services in order to detect failures. Implementation of these techniques within *XJAF* is planned in the future.

Integration of agents and web services has been supported by *IEEE FIPA Agents and Web Services Interoperability Working Group* (*AWSI-WG*) [23]. Out of the presented systems, only *FUSION@* uses web services as building blocks, while *JADE* relies on an add-on to employ web services for interoperability purposes. The *SOA*-based design has recently been applied to *XJAF*.

Table 1 The summary of features offered by different *MAS* implementations

MAS	Java EE	*FIPA*	Mobility	Fault-tolerance	WS integration
ABLE	No	Yes	Yes	No	No
Aglets	No	No	Yes	No	No
DimaX	No	Yes	No	Yes, agent replication	No
FUSION@	No	Yes	No	Yes, monitoring of agents and services	Yes
JADE	No	Yes	Yes	Yes, agent replication	Yes, through an add-on
Voyager	Yes	Yes	Yes	No	No, but some standards are supported (e.g. *SOAP*)
XJAF	Yes	Yes	Yes	Yes, fault-tolerant networks	Yes

Table 1 summarizes the similarities and differences between *XJAF* and each of the *MAS*s described above. Categories used for comparison are:

- The usage of Java EE as the implementation platform
- *FIPA*-compliance
- Support for agent mobility
- Fault-tolerance techniques
- Web services integration

3 The *XJAF* Architecture

XJAF is designed as a pluggable software architecture. It is defined as a set of loosely-coupled, dynamically loaded modules called *managers*. Each manager is responsible for handling a distinct part of the overall agent-management process. The design based on pluggable managers has several benefits. Managers are recognized and used solely by their interfaces, which allows for custom implementations. Also, the set of managers is not hard-coded, but rather loaded from a configuration file at startup. This means that new managers (and thus, new functionalities) can be added as needed, requiring no changes to the underlying implementation.

The set of managers available in the default implementation of *XJAF* is as follows:

- *AgentManager*: in charge of allocating and releasing agents (i.e. controls the agent's life-cycle).
- *TaskManager*: manages tasks that can be performed by registered agents.

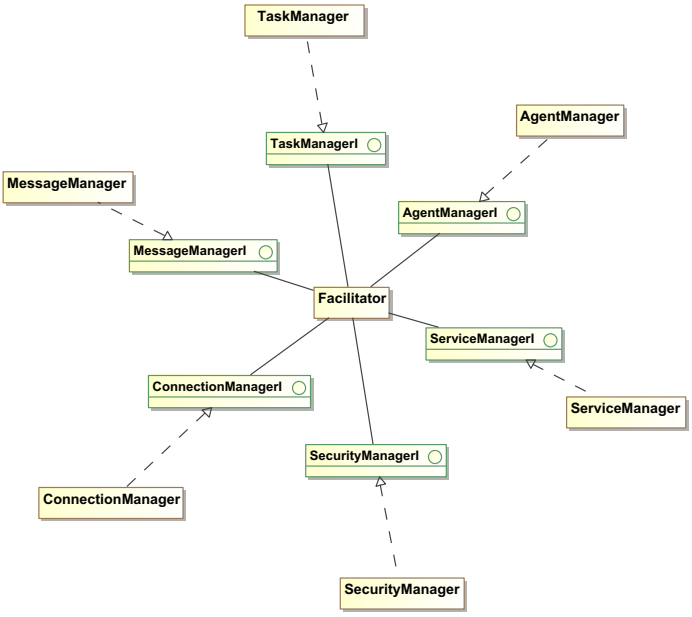

Fig. 1 Managers of *XJAF* and the *Facilitator* component

- *MessageManager*: provides the means for inter-agent communication.
- *ConnectionManager*: used for connecting physically distributed instances of *XJAF*.
- *SecurityManager*: provides security mechanisms.
- *ServiceManager*: manages services that can be used by other entities of the system.

All managers of *XJAF* are controlled by *Facilitator*, the central component of the system, as shown in Fig. 1. *Facilitator* provides a working environment for the agents, through an exposed *API*. Implementations of most of its methods are usually trivial, and consist of transferring the call to an appropriate manager, and, in case of blocking methods, waiting for the response and returning the result.

3.1 Agent Management

AgentManager includes functionalities that correspond to those specified in the *FIPA*'s *Agent Directory Service* [14]. Therefore, its main tasks are to keep a directory of agents, and to activate and deactivate agents on demand.

As noted before, *XJAF* relies on existing Java EE technologies for advanced programming features. *AgentManager* uses *JNDI* for implementing agent directory services of the *FIPA* specification. More important goals, when it comes to agent management, include minimizing the overhead of allocating new agents, and

employing some form of agent load-balancing at runtime. In order to achieve these goals, *AgentManager* will additionally wrap each agent inside an *EJB* component, and then pass it on to the enterprise application server. Modern application servers utilize the *EJB pooling* technique, keeping a number of *EJB* instances in memory at all time. Once a request for a new agent arrives, an instance is recycled from the pool, rather than being re-allocated from scratch. The number of instances kept in the pool is increased or decreased in response to the number of simultaneous requests. These features enable *XJAF* to balance the number of running agent with available system resources, with the minimal amount of programming effort.

XJAF has a built-in support for agent mobility. When an agent decides to migrate, it makes an appropriate call to its *Facilitator*. The call converts the agent to a stream of raw bytes using *Java Serialization* features. Raw bytes are transferred to the target *XJAF*, where the agent is de-serialized, and then its *onArrival()* method is called. This means that *XJAF* supports the so-called *weak* agent mobility.

In order to support agent mobility, *AgentManager* actually keeps two directories of agents: one for agents available locally, and another for agents that have moved to another *XJAF*. Each record in the *local directory* is a pair *(ID, ref)*, where *ID* identifies the agent, and *ref* holds a reference to it. Once an agent is created, its newly assigned identifier and reference are stored in the local directory. This directory record is removed once the agent is destroyed. Other parts of the system (including other agents) interact with an existing agent using its ID. So, for example, if an agent *A* wishes to send a message to agent *B*, it will do so by setting the *B*'s ID as the message recipient. This value will eventually be used to get a hold of the actual reference to the agent *B* from the local directory and place the message in its message queue.

For mobile agents, *AgentManager* utilizes another, *remote directory*. If an agent leaves its current host and moves to another *XJAF*, the agent's record is removed from the local directory, and a new record is inserted into the remote directory. Each record in the remote directory is a pair *(ID, addr)*, where *ID* is a newly-assigned identifier of the agent, while *addr* is the address of *XJAF* the agent has moved to. If the agent continues to move, e.g. to a third *XJAF*, the second instance will again keep the *(ID, addr)* record in its own remote directory.

When a message is sent to an agent, *AgentManager* will:

1. Look for the agent's ID in the local directory. If it is found, the message is put in the agent's message queue using the available reference.
2. Look for the agent's ID in the remote directory. If it is found, the message is forwarded to the remote *XJAF*, using the available address. The remote instance then follows the same procedure.

The agent migration process is depicted in more details in Fig. 2.

3.2 Managing Tasks

Capabilities of *XJAF* agents are described in form of *tasks*. Each agent publishes a set of tasks it can perform into a centralized repository handled by *TaskManager*.

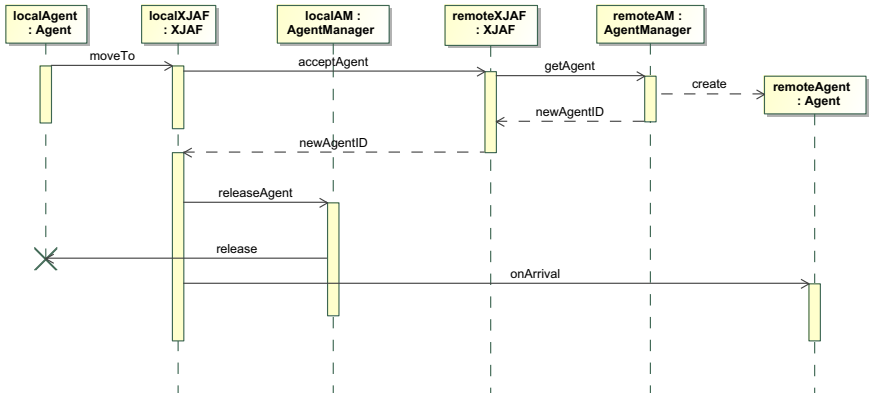

Fig. 2 Agent migration process in *XJAF*

A client (e.g. another agent, external user, etc.) can ask for a task execution by providing the task ID to *TaskManager*. The manager then finds the best agent for the task, sends it an appropriate message, and later returns the result to the client.

For interoperability reasons, agent tasks are represented using the standardized, platform-independent, and widely-used *W3C XML Schema* language [51].

Agents exchange task requests and execution results using the messaging system. For external, non-agent clients, *XJAF* uses the concept of *event listeners*. A listener is a *POJO* implementing a predefined *AgentListener* interface. For example, the task execution result is returned to the external client as a parameter of an *actionPerformed* event. The parameter is actually an instance of the *AgentResult* class, with a set of getter methods for extracting the actual result value, error code in case of a failure, etc.

3.3 Agent Communication

Three types of communication can be recognized in *XJAF*:

1. *Agent* $\leftrightarrow$ *XJAF*. Each agent holds a reference to the *Facilitator* instance, and can invoke its methods directly. The system, on the other hand, can interact with an agent by sending it a message.
2. *Agent* $\leftrightarrow$ *Agent*, where both agents are located in the same *XJAF*. This *local* inter-agent communication is performed using messages, with the assistance of *MessageManager*.
3. *Agent* $\leftrightarrow$ *Agent*, where agents are distributed over a network. This *remote* inter-agent communication also relies on the message exchange using *MessageManager*. Additionally, in this scenario *ConnectionManager* (described in Subsection 3.4) is employed for transferring the message to the remote agent.

Originally, *XJAF* used *Knowledge Query and Manipulation Language* (*KQML*) [13] as the message format. When the first version of the system was developed, in 2003

[47], *KQML* was the *de facto* standard for the agent communication language. Over the time, however, *FIPA ACL* [15] has emerged as a new standard, and was adopted by many *MAS*s, as shown previously in Section 2. In order to increase the interoperability of *XJAF* agents, the work is underway to update the existing messaging infrastructure of *XJAF* and replace *KQML* with *FIPA ACL*.

Following the idea of employing existing Java EE technologies in *MAS* development, *XJAF* relies on *Java Message Service* (*JMS*) [22] for the message exchange. *JMS* is a Java *API* used with loosely-coupled components in distributed applications, for reliable and asynchronous communication. When a *KQML* message is sent (e.g. from one agent to another), it is embedded into a *JMS* message. The *JMS* message is then published to all subscribed *XJAF* instances. Only the instance hosting the target agent, however, will receive the message. Upon receiving the message, the hosting *XJAF* extracts its *KQML* content and passes it to the target agent, by invoking its *onKQMLMessage* method. Fig. 3 outlines components in a *JMS*-based message exchange.

If case of security requirements, exchanged messages can be encrypted using the functionality offered by *SecurityManager*, as described in Subsection 3.5.

For a developer, however, manual construction and usage of messages can be labor-intensive tasks. To simplify the agent development process, *XJAF* offers the

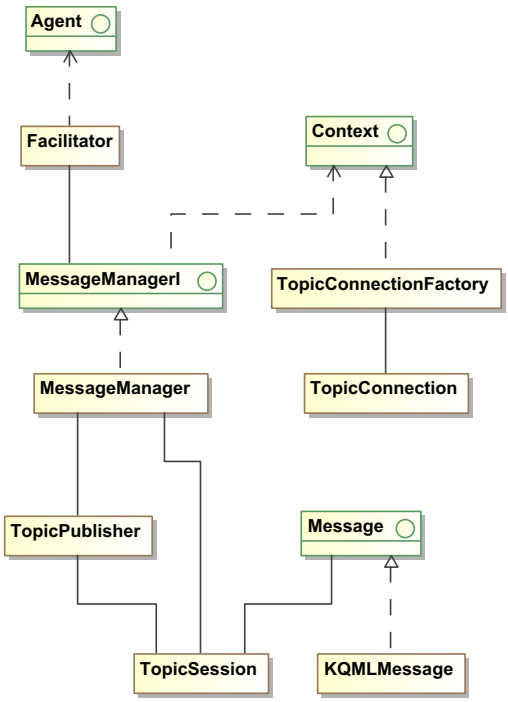

Fig. 3 Components of a *JMS*-based message exchange

so-called *programmatic* mode. The idea is to use methods that correspond to *KQML* performatives in order to reduce the process of constructing a message, sending it, and waiting for the reply to a single method invocation. For example, if an agent *A* wants another agent to perform some task, it needs to:

- Perform marshalling of the task description object.
- Construct a message manually and send it to the *Facilitator* component.
- Wait until the *Facilitator* component responds with the *tell* performative that corresponds to the original request.
- Extract the recommended agent's ID from the newly received message.

In the programmatic mode, the agent *A* can simply invoke the *Facilitator*'s *recommendOne* method and receive the recommended agent's ID as a return value.

It is important to note that no additional configuration of the system is needed to switch between the programmatic mode and raw messaging. If the agent *A* passes the received ID and task object in a call to the *Facilitator*'s *execute* method, it will eventually invoke the *execute* method of the target agent. This is an example of a programmatic mode approach. Alternatively, a raw *achieve* message could be sent by using the *Facilitator*'s *sendKQMLMessage* method, ending up as an input parameter of the target agent's *onKQMLMessage* method.

3.4 *Connecting Distributed* XJAF *Instances*

XJAF has a built-in support for agent mobility – while pursuing their goals, agents can move freely between interconnected, distributed instances of the environment. In the original implementation, *ConnectionManager* was the manager in charge of establishing a network of running *XJAF* instances. The network formed a tree-like organizational structure, with the *primary XJAF* representing the root of the tree. Fig. 4 shows an example of such a structure, with *Instance1* being the primary node.

New instances could easily be added to an existing network. If an instance was supposed to be a member of a network, its configuration file would contain the

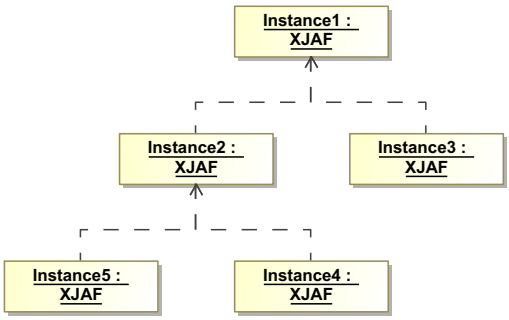

Fig. 4 Tree-like structure of distributed *XJAF* instances

address of the parent *XJAF*. During the startup, the new instance would inspect this file and then register itself with the specified parent accordingly.

Load-balancing is the main motivation behind the establishment of *XJAF* networks. That is, instead of running a large number of agents on a single machine, it would be more efficient to use several (possibly low-end) computers, and then distribute agents among them. For that matter, *ConnectionManager* offers a set of methods for locating an agent in the network and/or forwarding a message to it. Additionally, once an *XJAF* instance becomes a member of a network, it can rely on *ConnectionManager*'s methods for communicating with other members.

Recently, the tree-like structure was replaced by a fully-connected graph, in an effort to increase the fault-tolerance of *XJAF* networks, as described in Section 5.

3.5 Security Features of XJAF

In the area of *MAS* development, security features include the protection of agents and the system itself from malicious attacks, code and data integrity, and the integrity and confidentiality of exchanged messages. However, the application of security mechanisms often imposes significant overhead to the code execution and is not mandatory for all agent-based software solutions. Therefore, security features of *XJAF* are available in a distinct manager named *SecurityManager*, to be applied as needed.

The set of methods defined by *SecurityManager* is outlined in Table 2. All methods operate on raw bytes, and can be used to protect any part of the system, such as a serialized agent or a message.

The choice of techniques and algorithms to be used for implementing *SecurityManager*'s method is left to the system's developer. The default implementation uses *Public Key Infrastructure* (*PKI*). The proposed model for *SecurityManager* is shown in Fig. 5.

Tasks of the individual components of the *SecurityManager* model are as follows. *AccessSecurityHandler* manages agent's access permissions to system resources. Initially, these permissions are set through an external file, but can later be updated using the same interface. Permission checking is performed automatically by *JVM*, through the *XJAFSecurityManager* sub-class of *java.lang.SecurityManager*. *CryptoManager* performs date encryption/decryption, as well as signing and verification. Encryption and decryption are delegated to *CryptoAlgorithm*, while the job of digital

Table 2 Methods offered by *SecurityManager*

Method	Description
encrypt(source)	Encrypts the given array of bytes.
decrypt(source)	Decrypts the given array of bytes.
sign(source)	Performs digital signing of the given byte array.
verify(source, signature)	Validates the signed array of bytes.

signature generation and validation is transferred to *SignAlgorithm*. *PKI* administration is performed by *KeyHandler*, which includes issuing, revoking, and changing certificates, as well as generating key-pairs and secret keys. Key generation is governed by a class implementing the *KeyGenerator* interface. Finally, *AccessHandler* is used in a network environment, preventing unauthorized connections from distributed instances of *XJAF*.

Many of the described functionalities rely of the technologies already included in the Java platform, such as *Java PKI API*.

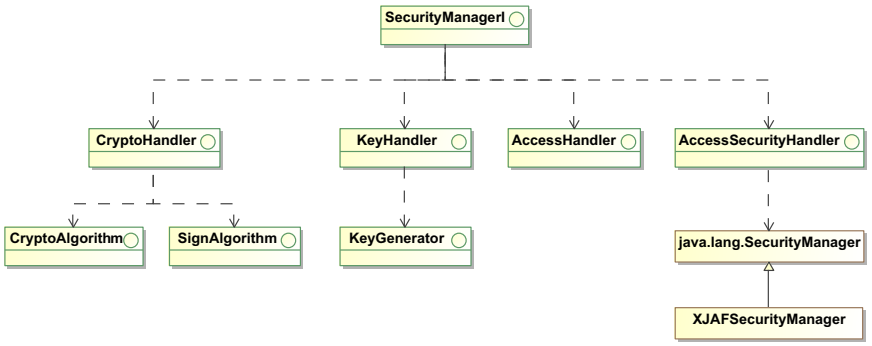

Fig. 5 The *SecurityManager* model

3.6 Service Manager

In *XJAF*, a *service* is a reusable software component that implements some specific functionality. It can be used by agents and other entities of the system. The basic idea is to extract some common tasks (e.g. input validation) into services and avoid repeating the implementation for each agent. The manager in charge of handling *XJAF* services is *ServiceManager*.

A service is a *POJO* implementing the *Service* interface. This interface contains only one method, *remove()*, which is called automatically once the client is finished working with this particular instance of the service. However, clients of a service usually do not deal with an instance of the service class. Rather, they specify a task that needs to be executed, as well as the identifier of the service that should execute it.

Similarly to *AgentManager* described earlier, *ServiceManager* uses *JNDI* for service directory implementation. In addition, after it initializes a service, *ServiceManager* will wrap it inside an *EJB* component, to be handled by the enterprise application server. Because of this feature, a single service instance can be recycled from a memory pool to serve many different agents. This reduces the number of memory allocations/deallocations to a minimum. The pool of instances is automatically increased or decreased by the enterprise server to match the actual demand. All these

features help to balance the overall resource requirements of *XJAF* with the amount of resource available in the system.

4 Practical Applications of *XJAF*

XJAF has been applied to two particular software systems in the field of distributed digital library catalogues: the *virtual central catalogue*, and *metadata harvesting* [47, 48]. Fundamental features of the agent technology employed in these applications include mobility and agents inter-relationship. In the virtual central catalogue, agents migrate from one node to another in search of the content specified by the query issued from the central node. In case of metadata harvesting, one central agent delegates tasks to agents in subordinate nodes and collects results.

In both cases, the use of agent technology provided a simple and effective mechanism for a dynamic setup of the distributed catalogues network. The main contribution of this approach is a solution that gives a simple and flexible mechanism for automatic maintenance of dynamically changing networks, as well as an environment suitable for the implementation of additional services in distributed libraries.

4.1 Example: Factorial *Agent*

In order to demonstrate the usage of *XJAF*, a simple framework for calculating the factorial of a specified number is developed. Although the presented example is artificial and simple, it is adequate for demonstrating the fundamentals of writing and employing *XJAF* agents, and for gaining a deeper understanding of the system's functioning. The framework consists of:

- The factorial task description
- *FactorialAgent*, an agent that performs the factorial calculation

FactorialAgent follows the next algorithm: if it can calculate the factorial directly (i.e. $n \in \{0,1\}$), it will do so and return the result; otherwise (i.e. $n \geq 2$), it will ask for another agent to calculate $(n-1)!$ and then return the value of multiplying n with that sub-result.

The factorial task description defines types of expected input and calculated return values. When working with tasks, clients, agents and managers of *XJAF* use *Java Architecture for XML Binding* (*JAXB*) [39] for generating the appropriate wrapper classes, performing object marshalling and document unmarshalling, etc. In this particular example, a *FactorialTask* class is generated from the defined factorial task *XML Schema* document.

Core implementation of the *FactorialAgent* (i.e. its *execute* method) is shown in Listing 1.

Listing 1 Core implementation of *FactorialAgent*

```
public AgentResult execute(AgentTask task, Facilitator f, Object aid){
  if (task instanceof FactorialTask) {
    // extract input value from the task
    FactorialTask factTask = (FactorialTask)task;
    int value = factTask.getInputValue();
    // n!, n >= 2
    if (value >= 2) {
      try {
        // recommend another agent for this task
        Object nextId = f.recommendOne(task);
        // ask it to calculate (n - 1)!
        factTask.setInputValue(value - 1);
        AgentResult subResult =
          f.execute(nextId, task);
        // return the final result
        value *= subResult.getContent();
        return new AgentResult(value);
      }
      catch (AgentNotFoundException ex) { return null; }
      catch (RemoteException ex) { return null; } }
    return new AgentResult(1); } // 1!, 0!
  return null; }
```

External clients interact with agents in *XJAF* through the *FacilitatorProxy* component. This component is an abstraction over *Facilitator*, hiding all the "low-level" details, such as *JNDI* lookup and *JMS* message composition. To interact with an agent, the client simply needs to create an instance of *FacilitatorProxy* and pass it the task description and concrete parameter values. The process of task execution though *FacilitatorProxy* is shown in Fig. 6.

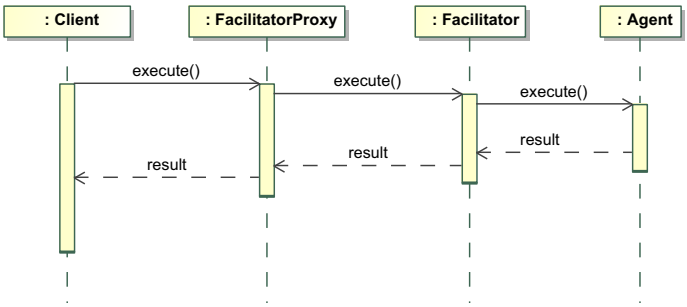

Fig. 6 The process of task execution in *XJAF*

Implementation of the factorial client is shown in Listing 2. The client's constructor receives the input value for *n* and generates an appropriate *XML*-formatted string, incorporating the value. It then creates an instance of *FacilitatorProxy* and requests

the task execution. The last parameter of the *FacilitatorProxy*'s *execute* method is an instance of the class implementing the *AgentListener* interface. As described earlier, this interface is used for asynchronous communication between *XJAF* and external clients. Once the task execution is done (either successfully of unsuccessfully), the listener's *actionPerformed* method will be called.

Listing 2 Implementation of the factorial client

```
public class FactorialClient implements AgentListener {
  private FacilitatorProxy fp;
  private int n;

  public FactorialClient(int n) {
    this.n = n;
    // construct the task
    String task =
      "<?xml version=\"1.0\" encoding=\"UTF-8\"?>" +
      "<factorialTask " +
          "xmlns=\"tasks/example/FactorialTask\">" +
        "<inputValue>" + n + "</inputValue>" +
      "</factorialTask>";
    // ask for task execution
    fp = new FacilitatorProxy();
    fp.execute(task, this); }

  public void actionPerformed(AgentEvent e) {
    // extract the result
    AgentResult result = e.getResult();
    String content = result.getContent();
    if (result.successful())
      System.out.printf("%d! = %s", n, content);
    else
      System.out.println("Error: " + content);
    fp.close(); } // free resources

  public void actionStarted(AgentEvent e) { }
  public void actionPerforming(AgentEvent e) { } }
```

4.2 Distributed Library Catalogues

Library information systems based on local record databases do not provide anything more than local record database searches. These systems could instead be incorporated in a library network that would provide the *shared cataloguing* feature.

Shared cataloguing enables data exchange on a record level between many libraries. Its purpose is to help librarians in reusing a record from a distant library database, instead of creating one manually. That is, shared cataloguing creates a controlled environment which enables librarians to complete their own local record

databases using library records from other sources, which results in an increased productivity. In order to offer this functionality, the environment must provide the librarian with the possibility of searching distant record databases, downloading selected distant records into the local database, and, optionally, modifying it in accordance to the specific needs of the local library.

Shared cataloging is implemented as a system that incorporates all local record databases into one joint database. This can be achieved in one of the following manners: *joint record server*, and *virtual central catalogue*. Joint record server represents a central record database that incorporates all library records from local databases. It requires regular updates, usually on a daily basis, which means that a powerful (i.e. an expensive) computer system is needed. The other downside is that this concept is characterized by a single point of failure – if the central computer or its network connection fail, so does the whole system.

Virtual central catalogue, on the other hand, maintains a network of local record databases. All queries sent to the virtual central catalogue are forwarded to all members of the network. Their sub-results are joined into a single result presented to the user. This concept can be efficiently implemented using agent technology features, namely distributed execution of the agent code, agent mobility, and inter-agent communication.

4.2.1 Using *XJAF* in *BISIS*

An implementation of the virtual central catalogue based on *XJAF* was provided [47] for the library information system *BISIS* [42]. The *BISIS* software system supports library operations and is designed and implemented using the object-oriented approach, *UML*, and the Java programming language. The system supports the Unicode standard [8], which makes it an appropriate tool for the multi-language environments.

Fig. 7 displays the virtual central catalogue consisting of n local *BISIS* library servers. Each local library server is connected to a local *XJAF* instance (*XJAF*1, *XJAF*2, …, *XJAFn*), which provides the working environment for library agents.

The virtual central catalogue is connected to the primary *XJAF* that hosts library agents. Once a query is made, the central catalogue forwards it to all available library agents, which then migrate to local *XJAF* instances and perform local database searches. An agent can access a local *BISIS* server using a library service implemented for this purpose. The benefit of using a specialized service is that it hides the library server details from the agent. By providing an abstraction layer, the service enables the creation of a heterogeneous library server network, without the need for changing the agent implementation.

After performing the local search, each agent sends its result back to the central catalogue. The catalogue then merges all these sub-results into a single result presented to the user. The entire process is outlined in Fig. 8.

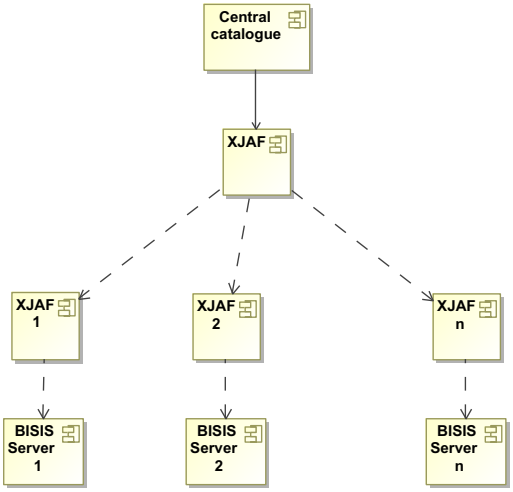

Fig. 7 Component diagram of a virtual central catalogue with *n* local library servers

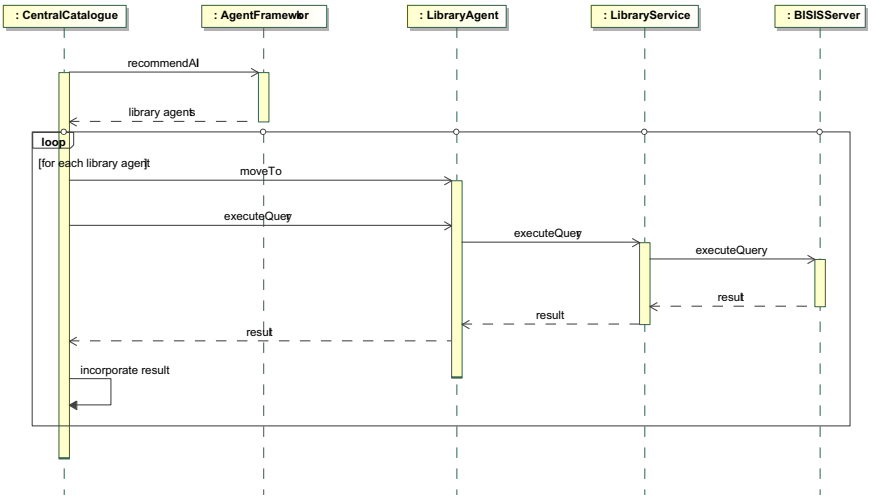

Fig. 8 Agent-based search of distributed library catalogues

4.3 Metadata Harvesting

Open Archive Initiative (*OAI*) [46] is an initiative for the foundation of electronic archives, with the goal of achieving enhanced accessibility to their contents. In the context of *OAI*, the term *archive* resembles a repository of scholarly and scientific papers. The framework for data interchange is defined by *Protocol for Metadata Harvesting* (*PMH*). *OAI-PMH* metadata harvesting is used as a standard proto-

col within *Networked Digital Library of Theses and Dissertations* (*NDLTD*) [36]. *NDLTD* aims at building a digital library of *Electronic Theses and Dissertations* (*ETD*) authored by students of member institutions.

A member of *OAI* can have both or either of the following roles:

- Data Provider: a system which supports *OAI-PMH* as the means of exposing metadata
- Service Provider: a system which uses metadata harvested via *OAI-PMH* as a basis for building value-added services

DIGLIB [10] is a *NDLTD* implementation for entering and querying electronic versions of theses and dissertations. *XJAF* agents have been integrated with the system [48] in order to harvest metadata from *OAI-PMH* data providers. By relying on the functionality of *ConnectionManager*, *DIGLIB* is able to automatically maintain the network of providers, i.e. without the need for manual addition or removal. Agents harvest metadata using the network of *XJAF* instances and corresponding data providers as shown in Fig. 9.

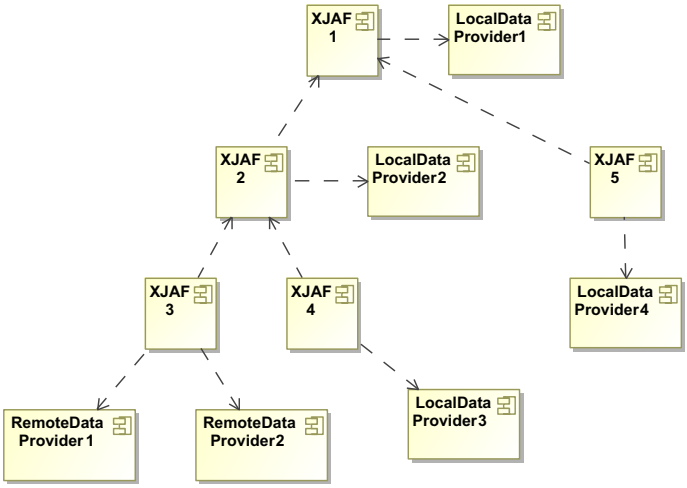

Fig. 9 An example of an *XJAF*-based *OAI-PMH* network

There are two distinct classes of data providers: *local* and *remote*. A local data provider is directly linked to an *XJAF* instance, and can be accessed by its agents as a local resource. Remote data providers, on the other hand, cannot be directly linked to any particular *XJAF* instance, for various technical, security, etc. reasons. Remote data providers are manually organized into lists which are fed into harvesting agents.

As with *BISIS*, metadata harvesting agents use a specialized service, named *OAIPMHService*, in order to communicate with data providers via *OAI-PMH*. In

case of remote data providers, an additional specialized web service was developed to act as a communication layer between *OAIPMHService* and the provider. Fig. 10 shows the sequence diagram of metadata harvesting using *XJAF* agents and *OAIPMHService*.

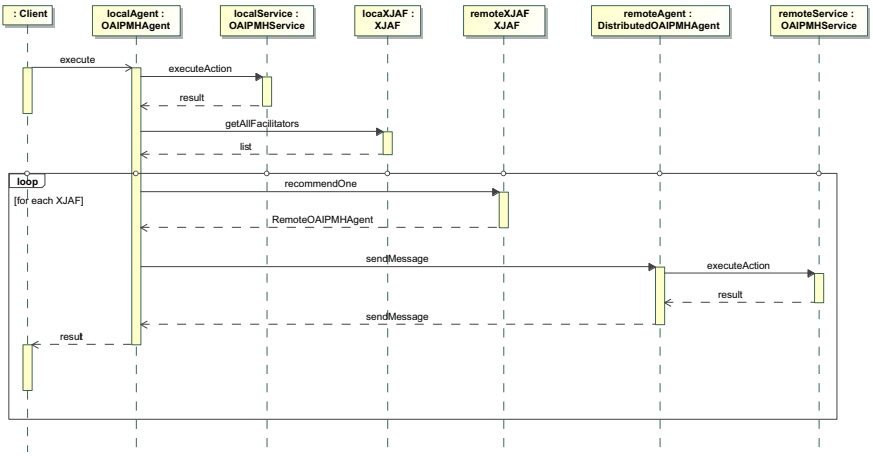

Fig. 10 Sequence diagram of the metadata harvesting

Two types of agents can be distinguished in the system: *OAIPMHAgent* and *DistributedOAIPMHAgent*. *OAIPMHAgent* receives metadata harvesting tasks and searches local data providers. It can also request a distributed search from *DistributedOAIPMHAgent*s which are distributed across the network and can communicate with both local and remote data providers. In the remote case, a *DistributedOAIPMHAgent* is given a list of providers to search.

OAIPMHAgent uses the messaging infrastructure for employing *DistributedOAIPMHAgent*s. Based on the task formulation contained in the message, the receiving *DistributedOAIPMHAgent* performs metadata harvesting, gathers the results, and then sends them back to the invoking agent in a message. *OAIPMHAgent* gathers all the results and presents them to the client.

4.4 The Benefits of Using XJAF

Agent-based implementation of the virtual central catalogue solves the distributed search problem in an efficient way because agent technology concepts offer the way for finding and recruiting agents, true parallelism, code migration, and message exchange. The virtual central catalogue implementation based on *XJAF* and *BISIS* offers the creation of a virtual network of libraries on a higher level of abstraction. That is, the network topology is defined by *XJAF* instances that are connected to library servers. Furthermore, this system is not related to any particular type of a

Table 3 Comparisons of source code sizes for various functionalities required by *BISIS* and *DIGLIB*, with and without using *XJAF*

Functionality	Without *XJAF*	With *XJAF*	Percentage
Communication and mobility	3908	90	2.30%
Regular library record search	28636	479	1.67%
Intelligent library record search	28603	446	1.56%
Quality of library records	28380	223	0.78%
Metadata harvesting	4328	441	9.50%

library server, and therefore provides the means for establishing and using heterogeneous networks of library servers.

When it comes to metadata harvesting, there are two main advantages of using *XJAF* over the conventional solutions [38]:

1. Automated, dynamic formation of data providers networks
2. Because *XJAF* itself already provides the network interconnection feature, developers of agents and services can concentrate on data processing

Consequently, for both systems, the source code is considerably smaller. Rough estimates of source code sizes required to implement various functionalities for *BISIS* and *DIGLIB* are outlined in Table 3. Description of each column is as follows:

1. Description of the functionality
2. Rough estimate of the source code size needed to implement the corresponding functionality without using *XJAF*
3. Size of an *XJAF* agent that was implemented to achieve the same functionality
4. Ratio, column 3 over column 2, expressed in percentages

Values of the second column have been estimated by measuring sizes of *XJAF* components that were needed to support the corresponding agent in the third column (plus the size of the agent itself).

The overall conclusion is that the usage of *XJAF* shortens and simplifies the software development time significantly. More concretely, the size of an agent that performs intelligent library search is only 1.56% the size of a non agent-based, classical software solution offering the same functionality. Because re-usability was one of the main design goals for *XJAF*, the framework can easily be applied in other domains as well.

Undoubtedly, the use of any framework (and not just an agent-based) can result in shorter software development times. However, the use of agents brings additional benefits. For example, *BISIS* is a closed system which cannot be accessed by external entities, residing outside of the system itself. Mobile *XJAF* agents, on the other hand, are able to migrate into and operate from within *BISIS*, assuming the roles of (human) librarians, and so gaining the access to protected resources.

5 Recent Improvements of the Architecture

Besides improving *BISIS* and *DIGLIB*, practical applications of *XJAF* had helped pointing out the weaknesses of the system itself. The tree-like structure of interconnected instances, although simple to build, was inflexible and prone to errors. For example, the failure of node *XJAF*2 in Fig. 9 would divide the system into two sets of mutually unreachable nodes. Furthermore, the original mobile agent tracking technique was not robust enough as the failure of any *XJAF* instance in the agent's migration path would render the agent lost. These issues have been solved by the introduction of fault-tolerant techniques [32, 33].

Another problem was the lack of interoperability. Being implemented in Java (EE), the system was available only to Java-based external clients – a severely limiting factor. In order to increase the interoperability of *XJAF*, the service-oriented philosophy was used [34]. The new implementation still utilizes many existing Java EE tools and libraries, but the system itself has been redesigned as a set of web services. This means that any external client with the support for web services is able to interact with agents of *XJAF*. In addition, for each new agent the system now automatically generates a web service interface, allowing the direct client $\leftrightarrow$ agent interaction. *Java API for XML-Based Web Services* (*JAX-WS*) [27] is used as the fundamental Java EE *API* for introducing the *SOA*-based design to *XJAF*.

Therefore, despite of its successful usage, and along with further developments in the domain of agent technology, the original design and implementation of *XJAF* have recently been critically evaluated and improved. These improvements are discussed next.

5.1 *Fault-Tolerant Networks of* XJAF *Instances*

In order to increase the fault-tolerance of *XJAF* networks, the tree-like structure of interconnected instances has been replaced by a fully-connected graph [32, 33]. Fully-connected graphs are convenient because, for a mobile agent the problem of finding a path to the target *XJAF* becomes trivial. This approach introduces no significant overhead, since each *XJAF* instance keeps a simple list of all other instances. Finally, the increase in time needed to add a new instance to an existing network is managed by exploiting the mobility feature of agents.

The new approach to creating and maintaining *XJAF* networks is includes a new type of a specialized, mobile agent named *ConnectionAgent*. The initial job of *ConnectionAgent* is to register its newly created host *XJAF* with an existing network. Afterwards, the agent performs tasks related to detecting a broken neighbor and informing other *ConnectionAgent*s of the problem.

An *XJAF* instance that needs to become a member of a network is pre-configured with a list of network addresses it should try to connect to. During startup, *ConnectionManager* will consult this list, trying to establish a connection. Once a connection is made, *ConnectionManager* initiates and dispatches *ConnectionAgent*, a light-weight mobile agent that visits all existing nodes in the network and spreads the information about newly created *XJAF*.

ConnectionAgent carries with itself a set of all *XJAF* instances it has detected so far. An element of this set is a triplet $\{address, state, timestamp\}$, where *address* is the instance's network address, *state* is its running state, as perceived by the agent (e.g. *running*, *unresponsive*, etc.), while *timestamp* marks the time at which the perceived state was reached. In general, *ConnectionAgent* relies on timestamps to distinguish between mutually exclusive messages. If, for example, it receives two messages about a *XJAF* instance, one telling it that the instance is up and running, and the other that the instance is unresponsive, it can detect the correct state by choosing the message with the later timestamp. Any inconsistencies between clocks are handled by always associating information about an *XJAF* instance with a timestamp generated by the instance itself, as described later.

Once *ConnectionAgent* visits an *XJAF* in the network, it performs a *synchronize* operation with it. During the synchronization, the hosting *XJAF* will learn about the *ConnectionAgent*'s home. At the same time, *ConnectionAgent* will learn about all instances its current host is or was connected to. Any changes made to the agent's set are sent back home.

ConnectionAgent occasionally *pings* its home *XJAF*, because it might occur that the instance breaks during the registration procedure. If this happens, *ConnectionAgent* will revert the registration: it will go to all previously visited instances in the network and inform them that its home is down. Similarly, if the home does not receive any pings from it *ConnectionAgent* for a certain amount of time, it will assume the agent is lost and will dispatch a new copy.

The mobility feature of *ConnectionAgent* has been used to reduce the overall time needed to add a new node to a network, especially if it contains a large number of nodes. After the synchronization procedure, the agent divides its set of yet unvisited instances into two distinct subsets. It then *spawns* a clone of itself and assigns one of these subsets to it, keeping the other subset for itself. Each agent then migrates to the next unvisited host from its own subset, repeating the synchronization procedure, and re-spawning new *ConnectionAgent*s at each subsequently visited *XJAF*. The registration procedure is finished for an agent once it has no more unvisited instances. By performing the set division and agent re-spawning at each visited host, the overall dependence of the registration time on the number of existing hosts becomes insignificant [33].

Once the registration procedure is finished, *ConnectionAgent*s return to their home *XJAF*. One of the instance is selected to establish a *heartbeat* connection with its neighbors in order to perform failure detection. When *ConnectionAgent* detects that one of its neighbors is down, it informs the remaining instances of the problem. To prevent the situation in which many *ConnectionAgent*s detect the same problem and start informing each other about it, the following approach is used:

- Each *ConnectionAgent* sorts its list of all *XJAF* instances in the network according to their respective timestamps. Let instance *A* be *before* instance *B* if it has earlier timestamp than *B*. Similarly, let *B* be *after A* if it has later timestamp than *A*.
- The agent establishes a heartbeat connection with agents directly before and directly after itself.

- If an agent detects a failure of the instance after itself, it sends a notification to the agent before itself, and establishes a new heartbeat connection with the agent after the failed one.
- Similarly, if an agent detects a failure of the instance before itself, it sends a notification to the agent after itself, and establishes a new heartbeat connection with the agent before the failed one.
- Whenever an agent learns about a failure, it updates its own set and forwards this information to the agent at the opposite side from the notification source.
- Whenever two agents establish a heartbeat connection, they synchronize their respective lists of neighbors. Any changes are propagated similarly as above.

Timestamps are again used to distinguish between mutually exclusive messages. To avoid any differences in clocks, it is crucial for a timestamp of an *XJAF* instance to be always generated by the instance itself. An agent a_i that receives a ping from a heartbeat neighbor returns the timestamp of its home *XJAF* as a result. The sender of the ping stores this value as the a_i's timestamp. In case of a failure of a_i's host, the saved value (plus an ε) is used in notification messages forwarded to other agents.

By following this simple set of rules, *ConnectionAgent*s can always establish a consistent view of the network, i.e. even in very unstable environments.

5.2 Agent Tracking Improvements

The agent tracking process used in the original implementation of *XJAF* (as described in subsection 3.1) was based on the *forwarding pointers* technique [35]. Similarly to the tree-like structure of distributed *XJAF* instances, the forwarding pointers technique was easy to construct, but was also characterized by a single point of failure: a breakdown of any node in the agent's path would have rendered the agent lost for all systems hierarchically above the breaking point. Therefore, an algorithm has been devised [32, 33] to increase the fault-tolerance of the agent tracking approach. The solution is again proposed in form of a new type of a specialized, light-weight agent, named *RemnantAgent*.

For each mobile agent M_a in a network, there is a single instance of *RemnantAgent* in each *XJAF* node in the M_a's path. *RemnantAgent* keeps two sets of information about its mobile agent:

1. An unordered set of all *XJAF* nodes in the M_a's path (S_a).
2. An ordered set of all nodes the M_a has visited after leaving the *RemnantAgent*'s host (S_f).

RemnantAgent is a purely reactive agent. It responds to two distinct events:

1. M_a has moved to another *XJAF* (E_f).
2. M_a has returned to this *XJAF* (E_r).

Event E_f is signaled to all *RemnantAgent*s in the agent's path, except for the target one. In response, corresponding *RemnantAgent*s update their respective sets S_a and S_f, with the new host being appended to the end of S_f.

Event E_r is signaled along with E_f, but to *RemnantAgent* in the target *XJAF* only. As a result, the target *RemnantAgent* will update its set S_a, while clearing the set S_f.

Fig. 11 depicts a scenario in which there are four *XJAF* instances, *A*, *B*, *C*, and *D* comprising a network [33]. Within each symbol representing an *XJAF*, current values of S_a and S_f are shown. There is a single mobile agent, initially moving from *A* to *B* (Fig. 11(a)), then from *B* to *C* (Fig. 11(b)), and finally from *C* to *D* (Fig. 11(c)).

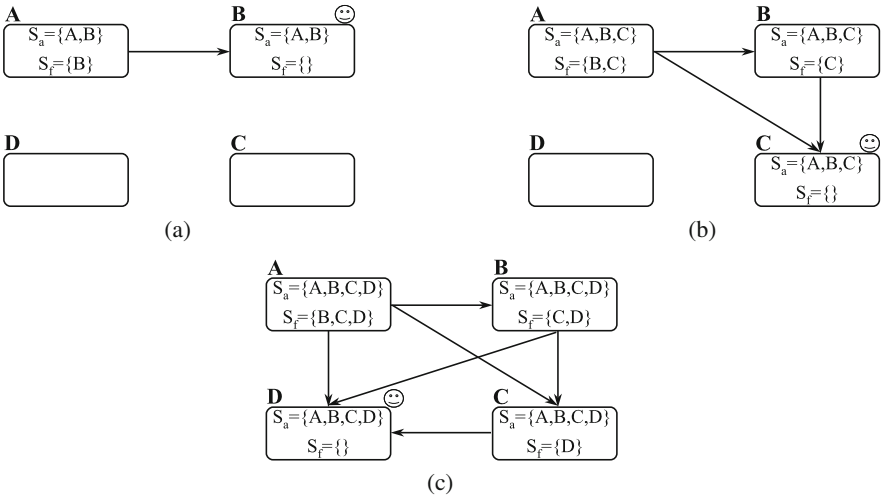

Fig. 11 Fault-tolerant mobile agent tracking algorithm [33]

Suppose that, in the situation depicted by Fig. 11(c), *A* needs to deliver a message to the agent. By inspecting its set S_f, it learns that the agent is in *D* and delivers the message directly there. However, let the link between *A* and *D* be broken. *A* now tries to deliver the message through the second-to-last element of S_f, which is *C*. By inspecting its own set S_f, *C* concludes that the agent is in *D*, and forwards the message there. In total, there are four alternative paths for delivering the message:

1. $A \rightarrow D$
2. $A \rightarrow C \rightarrow D$
3. $A \rightarrow B \rightarrow D$
4. $A \rightarrow B \rightarrow C \rightarrow D$

For very unstable environments, a configurable parameter can be used to trigger instantiation of additional *RemnantAgent*s with each migration step. These *RemnantAgent*s are then distributed across different *XJAF*s, providing even more alternative routes to the agent.

5.3 SOM: SOA-*Based* MAS

A major disadvantage of the original *XJAF* architecture was that it had been locked into a particular development platform (i.e. Java). The consequence of this approach, however, was that only Java-based external clients could use the system and interact with its agents. In order to increase the interoperability of *XJAF*, the system has been redesigned. Although it still uses Java EE as the implementation platform, managers of the system have been re-implemented as *web services*. The new system is named SOA-*based* MAS (*SOM*) [34].

Because it represents a conceptual specification of web services, their functionality and interaction, *SOM* can be implemented using many modern programming languages. The default implementation is provided in Java EE, following the same idea behind *XJAF* – reuse of existing technologies for *MAS* development as much as possible. Recently, successful examples of using other development platforms (e.g. Python [41]) have emerged. This, however, poses an interoperability problem – an agent written for a Java EE-based implementation of *SOM* cannot migrate to a Python-based implementation, undermining the intended goal of *SOM*.

In order to overcome this issue, a new agent-oriented programming language, named *Agent LAnguage for* SOM (*ALAS*) has been proposed [34]. Besides providing developers with programming constructs that hide the underlying complexity of agent development, one of the main features of the new language is *hot compilation*. This feature is used in software solutions based on mobile agents that need to migrate across a network consisting of heterogeneous *SOM* implementations. When an agent moves from one host in the network to another, it carries with itself its source code written in *ALAS*, as well as its internal state. When the agent reaches the target host, its *ALAS*-based source code is transformed on-the-fly into the source code of the target implementation platform. This output is forwarded to a *native* compiler, which produces the executable code, allowing the agent to continue its task execution.

An example of *FactorialAgent* written in *ALAS* is shown in Listing 3. The agent is defined inside a *namespace* and contains a single *service* named *calculate*. In *ALAS*, a service is a functionality that the agent offers to other, internal or external, users of the system. The given agent implementation follows the same algorithm for calculating the factorial described in Subsection 4.1.

Listing 3 An example of *FactorialAgent* written in *ALAS*

```
namespace "http://www.example.org/alas/factAgent";

ws agent FactorialAgent {
  service int calculate(int n) {
    if (n >= 2) {
      String taskId = "http://www.example.org/alas/" +
        "factAgent/CalculateTask";
      int subResult = facilitator.execute(taskId,
        "<n>" + (n - 1) + "</n>");
      return subResult * n; }
    return 1; } }
```

Behavior of the *ALAS* compiler can be controlled by a set of so-called *modifiers*. In the example shown in Listing 3, the *ws* agent modifier is used, which instructs the compiler to generate an appropriate web service interface to the agent. This feature increases the interoperability of *SOM* even further – external clients can interact with agents directly, through the exposed web service interface, i.e. in a familiar fashion, using the standardized *SOAP* communication protocol. All calls to the web service methods are automatically transformed into messages and then routed to the agent, even if it has migrated to another *SOM* instance.

6 Conclusion and Future Work

XJAF is a *FIPA*-compliant *MAS*, designed as a pluggable software architecture consisting of dynamically-loaded, loosely-coupled managers. The basic set of managers includes *AgentManager*, for controlling the agent life-cycle, *TaskManager*, for managing tasks that can be performed by agents, *MessageManager*, which provides the messaging infrastructure, *ConnectionManager*, used in networking environments for connecting distributed *XJAF* instances, *SecurityManager*, which offers security features, and *ServiceManager*, for managing *XJAF* services, reusable software components.

As concluded in [3], some of the most important technical requirements imposed on industrial applications of the agent technology are robustness, reliability, scalability, and interoperability. According to our personal opinion, not enough effort has been put into satisfying these requirements. For example, existing *MAS* solutions, in most cases, do not provide any runtime load-balancing techniques; those few that do, implement their own algorithms from scratch. Additionally, there is an overall lack of interoperability between various, even *FIPA*-compliant *MAS* implementations, as well as between *MAS* implementations and legacy, non agent-based code.

The main advantage of *XJAF* over many other existing *MAS*s is the full employment of Java EE technologies, tools, and libraries. Java EE has been adopted by a large portion of corporate enterprises, because it simplifies the development of robust, reliable, and scalable software. As such, it represents an excellent tool for *MAS* development, covering almost all of the aforementioned technical requirements. Although certain *MAS* architectures employ some of the features offered by Java EE, none have taken the extensive approach used during the development of *XJAF*. For example, *XJAF* wraps agents inside *EJB* components and delegates their life-cycle management to the enterprise application server, gaining in such a way, and with a minimum amount of programming effort, runtime load-balancing of agents, improving the overall performance of the system.

The interoperability of *XJAF* has recently been increased significantly by following the *SOA* philosophy. In the new system, named *SOM*, managers have been redesigned as web services, while still relying on Java EE for implementation. The *SOA*-based design enables any external client to use *SOM* and interact with its agents. In addition, web service interfaces are now automatically generated for each

new agent, providing the means for the direct communication with the underlying agent, even if it moves to another host in a network.

However, the employment of web services is not enough for achieving the full interoperability. The new agent-oriented programming language named *ALAS* and its supporting tools provide an opportunity for agent mobility in, platform-wise, heterogeneous networks. The execution code of an agent written in *ALAS* can be regenerated at runtime, during the migration process, to match the implementation platform of the new host.

Additionally, the original *XJAF* has recently been improved with the introduction of fault-tolerant techniques. New algorithms for creating and maintaining a network of distributed *XJAF* instances enable remaining instances to cooperate undisturbed in unstable environments, e.g. with network nodes failing unexpectedly. In addition, the introduced robust agent tracking technique can locate a mobile agent even if a number of intermediary *XJAF* instances in its path break.

The future work on *SOM* will be in improving its interoperability and fault-tolerance even further. As the first step, the *KQML* messaging system will be replaced by *FIPA ACL*, used by the majority of modern *MAS* architectures. Efficient agent replication techniques will help in restoring the execution state of agents in case of host failures.

The focus of further development will also be on *ALAS*, in enriching the language with new programming constructs, improving its performance, and adding support for the development of *BDI*-style agents.

Acknowledgements. The work is partially supported by Ministry of Education and Science of the Republic of Serbia, through project no. OI174023: "Intelligent techniques and their integration into wide-spectrum decision support".

References

1. ABLE homepage, http://www.alphaworks.ibm.com/tech/able (retrieved on July 22, 2011)
2. Aglets homepage, http://aglets.sourceforge.net (retrieved on July 22, 2011)
3. Belecheanu, R.A., Munroe, S., Luck, M., Payne, T., Miller, T., McBurney, P., Pěchouček, M.: Commercial applications of agents: lessons, experiences and challenges. In: Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS 2006, pp. 1549–1555. ACM, New York (2006), doi:http://doi.acm.org/10.1145/1160633.1160932, http://doi.acm.org/10.1145/1160633.1160932
4. Bellifemine, F.L., Caire, G., Greenwood, D.: Developing multi-agent systems with JADE. John Wiley and Sons (2007)
5. Bigus, J.P., Schlosnagle, D.A., Pilgrim, J.R., Mills III, W.N., Diao, Y.: ABLE: a toolkit for building multiagent autonomic systems. IBM Systems Journal 41(3), 350–371 (2002)
6. Board, J.: JADE web services integration gateway (WSIG) guide (2008), http://jade.tilab.com/doc/tutorials/WSIG_Guide.pdf (retrieved on July 22, 2011)

7. Bădică, C., Budimac, Z., Burkhard, H., Ivanović, M.: Software agents: languages, tools, platforms. Computer Science and Information Systems, ComSIS 8(2), 255–296 (2011)
8. Consortium, T.U.: The Unicode Standard, http://www.unicode.org (retrieved on July 22, 2011)
9. Cowan, D., Griss, M., Burg, B.: BlueJADE – a service for managing software agents. Tech. rep. Hewlett-Packard Company (2002)
10. DIGLIB homepage, http://www.diglib.uns.ac.rs/frontOffice/index.jsp (retrieved on July 22, 2011)
11. EJB homepage, http://www.oracle.com/technetwork/java/javaee/ejb/index.html (retrieved on March 5, 2012)
12. Faci, N., Guessoum, Z., Marin, O.: DimaX: a fault-tolerant multi-agent platform. In: Proceedings of the 2006 International Workshop on Software Engineering for Large-Scale Multi-Agent Systems, pp. 13–20 (2006)
13. Finin, T., Fritzson, R., McKay, D., McEntire, R.: KQML as an agent communication language. In: Proceedings of the third international conference on Information and knowledge management, CIKM 1994, pp. 456–463. ACM, New York (1994), http://doi.acm.org/10.1145/191246.191322, doi:http://doi.acm.org/10.1145/191246.191322
14. FIPA abstract architecture specification(2002), http://www.fipa.org/specs/fipa00001/SC00001L.pdf (retrieved on July 22, 2011)
15. FIPA ACL message structure specification (2002), http://www.fipa.org/specs/fipa00061/SC00061G.pdf (retrieved on July 22, 2011)
16. FIPA homepage, http://www.fipa.org/ (retrieved on July 22, 2011)
17. Ganguly, P., Ray, P., Low, G.: Software agent based approach towards tele-electrocardiography. In: Proceedings of the 13th IEEE Symposium on Computer-Based Medical Systems (CBMS 2000), pp. 275–280 (2000)
18. GlassFish open source application server homepage, http://glassfish.java.net/ (retrieved on July 22, 2011)
19. Guessoum, Z., Briot, J.: From active object to autonomous agents. IEEE Concurrency 7(3), 68–78 (1999)
20. Guessoum, Z., Faci, N., Briot, J.P.: Adaptive replication of large-scale multi-agent systems: towards a fault-tolerant multi-agent platform. In: Proceedings of the Fourth International Workshop on Software Engineering For Large-Scale Multi-Agent Systems, pp. 1–6 (2005)
21. Haase, K.: Java message system tutorial, http://docs.oracle.com/javaee/1.3/jms/tutorial/1_3_1-fcs/doc/jms_tutorialTOC.html
22. Hapner, M., Burridge, R., Sharma, R., Fialli, J., Stout, K.: Java Message Service (JMS) specification (2002), http://www.oracle.com/technetwork/java/jms/index.html (retrieved on July 22, 2011)
23. IEEE FIPA Agents and Web Services Interoperability Working Group (AWSI-WG) homepage, http://www.fipa.org/subgroups/AWSI-WG.html (retrieved on July 22, 2011)
24. JADE homepage, http://jade.tilab.com/ (retrieved on July 22, 2011)

25. Java PKI programmer's guide, http://docs.oracle.com/javase/6/docs/technotes/guides/security/certpath/CertPathProgGuide.html (retrieved on March 5, 2012)
26. JAXB homepage, http://jaxb.java.net/ (retrieved on March 5, 2012)
27. JAX-WS homepage, http://jax-ws.java.net/ (retrieved on March 5, 2012)
28. JNDI homepage, http://www.oracle.com/technetwork/java/jndi/index.html (retrieved on July 22, 2011)
29. Karjoth, G., Lange, D.B., Oshima, M.: A security model for aglets. IEEE Internet Computing 1(4) (1997)
30. Marin, O., Sens, P., Briot, J., Guessoum, Z.: Towards adaptive fault-tolerance for distributed multi-agents systems. In: Proceedings of the Fourth European Research Seminar on Advances in Distributed Systems, pp. 195–201 (2001)
31. Meyer, M.: The features and facets of the Agent Building and Learning Environment, ABLE (2004), http://www.ibm.com/developerworks/autonomic/library/ac-able1/ (retrieved on July 22, 2011)
32. Mitrović, D., Budimac., Z., Ivanović, M., Vidaković, M.: Improving fault-tolerance of distributed multi-agent systems with mobile network-management agents. In: Proceedings of the International Multiconference on Computer Science and Information Technology, vol. 5, pp. 217–222 (2010)
33. Mitrović, D., Budimac, Z., Ivanović, M., Vidaković, M.: Agent-based approaches to managing fault-tolerant networks of distributed multi-agent systems. Multiagent and Grid Systems 7(6), 203–218 (2011)
34. Mitrović, D., Ivanović, M., Vidaković, M.: Introducing ALAS: a novel agent-oriented programming language. In: Simos, T.E. (ed.) Proceedings of Symposium on Computer Languages, Implementations, and Tools (SCLIT 2011) held within International Conference on Numerical Analysis and Applied Mathematics ICNAAM 2011. AIP Conference Proceedings, vol. 1389, pp. 861–864 (2011)
35. Moreau, L.: Distributed directory service and message router for mobile agents. Science of Computer Programming 39(2–3), 249–272 (2001)
36. NDLTD homepage, http://www.ndltd.org (retrieved on July 22, 2011)
37. OMG OCL specification page, http://www.omg.org/technology/documents/modeling_spec_catalog.htm#OCL (retrieved on July 22, 2011)
38. Open Archives Initiative Tools, http://www.openarchives.org/pmh/tools/tools.php (retrieved on July 22, 2011)
39. Ort, E., Mehta, B.: Java Architecture for XML Binding, JAXB (2003), http://www.oracle.com/technetwork/articles/javase/index-140168.html (retrieved on July 22, 2011)
40. Pešović, D., Vidaković, M., Ivanović, M., Budimac, Z., Vidaković, J.: Usage of agents in document management. Computer Science and Information Systems, ComSIS 8(1), 193–210 (2011)
41. SPADE homepage, http://code.google.com/p/spade2/ (retrieved on July 22, 2011)
42. Surla, D., Konjović, Z.: Distributed library information system BISIS. Group for Information Technologies, Novi Sad (2004) ISBN: 96-7444-006-1
43. Tai, H., Kosaka, K.: The Aglets project. Communications of the ACM 42(3), 100–101 (1999)

44. Tapia, D.I., Bajo, J., Corchado, J.M.: Distributing Functionalities in a SOA-Based Multi-agent Architecture. In: Demazeau, Y., Pavón, J., Corchado, J.M., Bajo, J. (eds.) 7th International Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS 2009). AISC, vol. 55, pp. 20–29. Springer, Heidelberg (2009)
45. Tapia, D.I., Rodríguez, S., Bajo, J., Corchado, J.M.: FUSION@, a SOA-based multi-agent architecture. In: Corchado, J.M., Rodríguez, S., Llinas, J., Molina, J. (eds.) International Symposium on Distributed Computing and Artificial Intelligence (DCAI 2008). AISC, vol. 50, pp. 99–107. Springer, Heidelberg (2009)
46. The Open Archives Initiative Protocol for Metadata Harvesting, http://www.openarchives.org/OAI/openarchivesprotocol.html (retrieved on July 22, 2011)
47. Vidaković, M.: Extensible java based agent framework. Ph.D. thesis, Faculty of Technical Sciences, University of Novi Sad, Serbia (2003)
48. Vidaković, M., Milosavljević, B., Konjović, Z., Sladić, G.: EXtensible Java EE-based agent framework and its application on distributed library catalogues. Computer Science and Information Systems, ComSIS 6(2), 1–16 (2009)
49. Vidaković, M., Sladić, G., Konjović, Z.: Security management in J2EE based intelligent agent framework. In: Proceedings of the 7th IASTED International Conference on Software Engineering and Applications (SEA 2003), pp. 128–133 (2003)
50. Voyager homepage, http://www.recursionsw.com/Products/voyager.html (retrieved on July 22, 2011)
51. (W3C), W.W.W.C.: XML Schema, http://www.w3.org/XML/Schema (retrieved on July 22, 2011)

Authors' Bios

PhD Milan Vidaković. Since 2009 holds the associate professor position at Faculty of Technical Sciences, Novi Sad, Serbia. He received his PhD degree (2003) in Computer Science from the University of Novi Sad, Faculty of Technical Sciences. Since 1998 he has been with the Faculty of Technical Sciences in Novi Sad. Mr. Vidaković participated in several science projects. He published more than 60 scientific and professional papers. His main research interests include web and internet programming, distributed computing, software agents, and language internationalization and localization.

PhD Mirjana Ivanović. Since 2002 holds the position of full professor at Faculty of Sciences, University of Novi Sad, Serbia. She is the head of Chair of Computer Science and a member of University Council for informatics. Author or co-author is, of 13 textbooks and of more than 230 research papers on multi-agent systems, e-learning and web-based learning, software engineering education, intelligent techniques (CBR, data and web mining), most of which are published in international journals and international conferences. She is/was a member of Program Committees of more than 80 international Conferences and is Editor-in-Chief of Computer Science and Information Systems Journal.

MSc Dejan Mitrović. Since 2008 holds the position of teaching and research assistant at Faculty of Sciences, University of Novi Sad, Serbia. He graduated in 2006 (informatics), and received master's degree (computer science) in 2008, enrolling

the PhD studies afterwards. His is an author or co-author of 6 research papers on software agents and multi-agent systems. He is a member of several national and bilateral research projects.

PhD Zoran Budimac. Since 2004 holds the position of full professor at Faculty of Sciences, University of Novi Sad, Serbia. Currently, he is the head of Computing laboratory. His fields of research interests involve: Educational Technologies, Agents and WFMS, Case-Based Reasoning, Programming Languages. He was the principal investigator of more than 20 projects. He is an author of 13 textbooks and more than 220 research papers, most of which are published in international journals and international conferences. He is/was a member of Program Committees of more than 60 international Conferences, and a member of Editorial Board of Computer Science and Information Systems Journal.

Chapter 4
Agent-Based XDSL Monitoring and Optimization

Giovanni Caire

Abstract. This chapter focuses on a large agent-based system developed and successfully deployed by Telecom Italia in the field of Fixed Network monitoring and optimization. Thanks to its natively distributed agent-based architecture, this system, called Wants-Assurance, continuously monitors about 3.000.000 of xDSL lines in real time. Wants-Assurance is developed on top of WADE (Workflows and Agents Development Environment) a domain independent software platform that allows creating distributed applications leveraging the agent paradigm in conjunction with the workflow metaphor. The chapter is organized as follows: section 1 presents WADE describing its architecture and main features. Section 2 gives an overview of the xDSL network domain highlighting the main phenomena that affect its quality. Section 3 focuses on the Wants-Assurance system, describes its internal architecture and shows how it exploits WADE features to face the challenges set by the application domain.

1 WADE

WADE (Workflows and Agents Development Environment) [1] is a domain independent software platform built on top of ***JADE*** [2], a popular open source middleware conceived to facilitate the development of distributed applications based on the ***agent-oriented paradigm***.

As depicted in Figure 1, JADE provides a distributed runtime environment, the agent and behaviour (a task performed by an agent) abstractions, peer to peer communication between agents and basic agent lifecycle management and discovery mechanisms. WADE adds to JADE the support for the execution of tasks defined according to the ***workflow*** metaphor and a number of mechanisms and

Giovanni Caire
Telecom Italia S.p.A.
Via Reiss Romoli 274 – 10148 Turin
Italy
e-mail: giovanni.caire@telecomitalia.it

M. Ganzha & L. C. Jain (Eds.): Multiagent Systems & Applications, ISRL 45, pp. 89–106.
springerlink.com

components that help ***managing the complexity of the distribution*** both in terms of administration and fault management.

Thanks to the combination of distributed agents and workflows, WADE is particularly suited to develop applications that require a high degree of scalability and/or imply the execution of possibly long and articulated tasks.

In principle WADE supports "notepad-programming" in the sense that there is no hidden mechanism that developers can't control. However, especially considering that one of the main advantages of the workflow approach is the possibility of representing processes in a friendly graphical form, WADE comes with a development environment called ***WOLF*** [3] that facilitates the creation of WADE-based application. WOLF is an Eclipse [4] plug-in and, as a consequence, allows WADE developers to exploit the full power of the Eclipse IDE plus additional WADE-specific features.

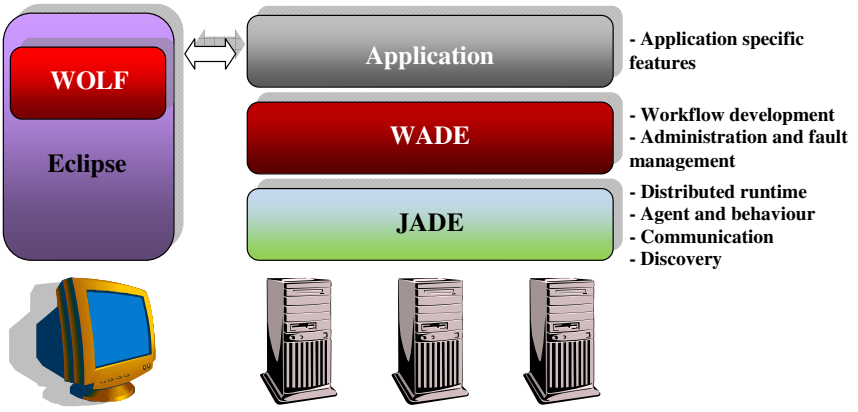

Fig. 1 The WADE platform

1.1 Distribution

WADE inherits from JADE a distributed runtime composed of several "***Containers***" that can run on different hosts and each one can contain a number of ***agents*** [5]. Agents are actually the components that make up a JADE based application as well as a WADE based application. Each type of agent corresponds to a class that inherits from the `jade.core.Agent` class of the JADE library and many instances of a type of agent can be active in the system.

Containers are the abstraction by means of which the agents that compose an application can be distributed across several hosts. Of course the application can be designed as if all its components (agents) were running locally. Agent distribution across containers and hosts can be defined at deployment time according to scalability requirements.

Even if this is not strictly mandatory, most of the time a container corresponds to a JVM. The set of active containers is called a ***Platform***. As depicted in Figure 2 a special container exists in the platform called "***Main Container***". The Main Container must be the first one to start and all other containers (typically called peripheral containers) register to it at bootstrap time. Furthermore the Main Container holds two special agents.

The ***AMS*** (Agent Management System) that represents the ***authority in the platform***, i.e. it is the only agent that can activate platform management actions such as creating/killing other agents, killing containers and shutting down the platform. Normal agents wishing to perform such actions must request them to the AMS.

The ***DF*** (Directory Facilitator) that implements the ***yellow pages service*** by means of which agents can advertise their services and find other agents offering services they need.

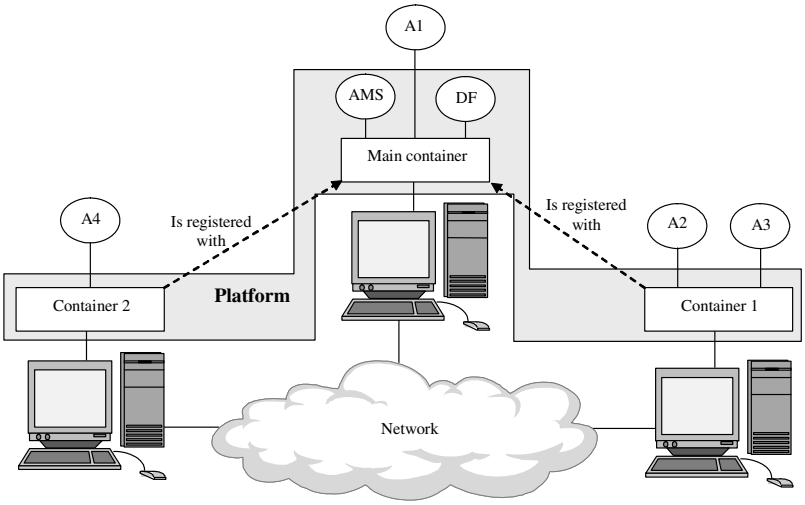

Fig. 2 JADE distributed architecture

1.1.1 The Communication Model

All agents in the system interact by means of an asynchronous message passing mechanism. More in details each agent has a sort of mailbox (the agent message queue) where the JADE runtime posts messages sent by other agents. Whenever a message is posted in the message queue the receiving agent is notified. If and when the agent actually picks up the message from the message queue to process it is completely up to the programmer.

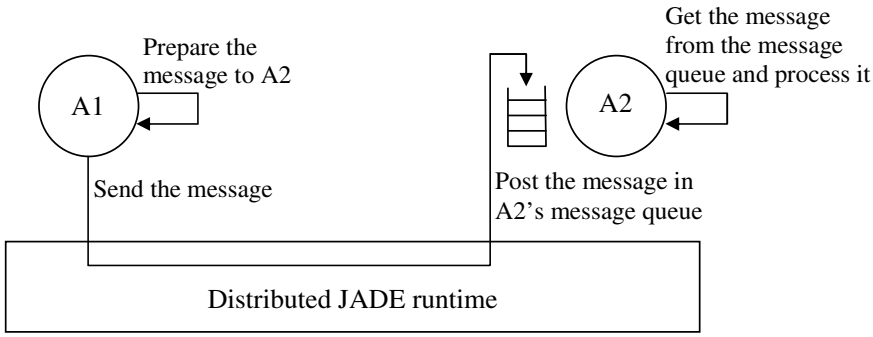

Fig. 3 The JADE asynchronous message passing paradigm

Messages exchanged by JADE agents have a format specified by the ACL language defined by the FIPA [6] international standard for agent interoperability. This format comprises a number of fields and in particular:

- The ***sender*** of the message
- The list of ***receivers***
- The communicative intention (also called "***performative***") indicating what the sender intends to achieve by sending the message. The performative can be REQUEST, if the sender wants the receiver to perform an action, INFORM, if the sender wants the receiver to be aware a fact, QUERY_IF, if the sender wants to know whether or not a given condition holds, CFP (call for proposal), PROPOSE, ACCEPT_PROPOSAL, REJECT_PROPOSAL, if the sender and receiver are engaged in a negotiation, and more.
- The ***content*** i.e. the actual information included in the message (i.e. the action to be performed in a REQUEST message, the fact that the sender wants to disclose in an INFORM message …).
- The content ***language*** i.e. the syntax used to express the content (both the sender and the receiver must be able to encode/parse expressions compliant to this syntax for the communication to be effective).
- The ***ontology*** i.e. the vocabulary of the symbols used in the content and their meaning (both the sender and the receiver must ascribe the same meaning to symbols for the communication to be effective).
- Some fields used to control several concurrent conversations and to specify timeouts for receiving a reply such as ***conversation-id***, ***reply-with***, ***in-reply-to***, ***reply-by***.

1.1.2 Administration Features

Distribution is animportant characteristic, especially for applications that are expected to support heavy loads, since a distributed application can be deployed on highly scalable hardware architectures such as blades. It is clear however that

administering a distributed application is more complex than administering a monolithic application unless proper tools are made available. Furthermore the probability that a host crashes increases proportionally with the number of hosts the application is distributed on and proper recovery mechanisms are necessary to ensure the application can survive. Clustering systems are often used for those purposes, but they are typically quite expensive and difficult to configure.

WADE faces these problems by providing a number of mechanisms that help the administrator in

- Installing the application
- Configuring it (tuning parameters).
- Activating/deactivating the application spreading components (i.e. containers and agents) across available hosts according to specific needs.
- Monitoring runtime events and critical conditions such as disk and memory consumption.
- Deploying new/modified system logics at runtime without system down.
- Automatically recovering from host, container and agent faults.

For instance it is possible to specify which agents to start, where (i.e. on which containers and on which hosts) to deploy them and which parameters to pass to them by means of a simple xml file as exemplified in the snippet below. At this point it is possible to activate the application according to the defined configuration file from the WADE Management Console.

```
<platform name="Develop"
description="WA Development configuration file">
  <hosts>
    <host name="localhost">
      <containers>
        <container name="Admin-Container">
          <agents>
            <agent name="nma" type="Network Manager Agent"/>
            <agent name="nh" type="Notification Handler Agent">
              <parameters>
                <parameter key="subnet" value="AWS1"/>
                <parameter key="port" value="21025"/>
              </parameters>
            </agent>
          </agents>
        </container>

        <container name="RP-Container>
          <agents>
            <agent name="rp01" type="Resource Proxy Agent"/>
            <agent name="rp02" type="Resource Proxy Agent"/>
            <agent name="rp03" type="Resource Proxy Agent"/>
            <agent name="rp04" type="Resource Proxy Agent"/>
            <agent name="rp05" type="Resource Proxy Agent"/>
          </agents>
        </container>
      </containers>
    </host>
  </hosts>
</platform>
```

The presented snippet is a simplified version of the configuration file used for the development installation of the Wants Assurance application that will be presented in section3. In particular this configuration includes a single host (the local host) with two containers called Admin-Container and RP-Container respectively. The former contains two agents: *nma* of type Network Manager Agent and *nh* of type Notification Handler Agent. The latter contains five agents of type Resource Proxy Agent called rp01, rp02, rp03, rp04 and rp05.

1.2 Workflows

A workflow is aformal definition of a process in terms of activities to be executed, relations between them, criteria that specify the activation and termination and additional information such as the participants, the software tools to be invoked, required inputs and expected outputs and internal data manipulated during the execution [7].

The key aspect of the workflow metaphor is the fact that the ***execution steps as well as their sequencing are made explicit***. This makes it possible to give a ***graphical representation*** of a process defined as a workflow. Such representation is clearly extremely more intuitive with respect to a piece of software code and in general is understandable by domain experts as well as by programmers. Domain experts can therefore validate system logics directly and not only on documents that most of the time are not perfectly up to date. In some cases they could even contribute to the actual development of the system without the need for any programming skill.

Another important characteristic is that, being the execution steps explicitly identified, the workflow engine (i.e. a system able to automatically execute a process defined as a workflow)can trace them. This makes it possible to create ***automatic mechanisms*** to facilitate system monitoring. Typically a workflow engine embeds ready-made mechanisms to trace activities and to create reports on them. Additionally, when processes have to be executed within the scope of a transaction, ***semi-automatic rollback procedures*** can be activated in case of unexpected fault.

Finally, since ***workflows are fully self-documented***, workflow-based development releases the development team of the burden of keeping documentation aligned each time design choices must be revisited to face implementation details or evolving requirements.

1.2.1 Approach

The WADE approach to support the workflow metaphor has two distinguishing characteristics.First of all, unlike the majority of Business Process Management (BPM) tools, it does not provide a single powerful workflow engine. On the contrary it provides an ad-hoc type of agent (called Workflow-Engine-Agent) that embeds a ***micro-workflow engine***. As a consequence, besides normal tasks (JADE behaviours), each Workflow-Engine agent active in a WADE-based applicationis able to execute workflows.

The second important point to highlight is that***the WADE workflow representation formalism is based on the Java language***. That is, a workflow that can be executed by WADE Workflow-Engine agents is expressed as a Java class with a well defined structure. As such WADE workflows can be edited, refactored, debugged and in general managed as all Java classes and can include all pieces of code (methods, fields of whatever types, inner classes, references to external classes and so on) needed to implement the process details. In addition, of course, the execution flow they specify can be presented and modified in a friendly, graphical way. More in details WOLF (the development environment for WADE based applications) is an Eclipse plugin and allows developers to work with a graphical view (suitable to manage the process flow) and a code view (the usual Eclipse Java editor suitable to define execution details) that are kept in synch.

Therefore the WADE workflow engine embedded into Workflow-Engine agents is not an interpreter of a workflow description language, but executes compiled Java code. This on the one hand makes it extremely performing, but on the other hand requires the necessary workflow classes to be available when an agent is requested to execute a workflow. For this reason WADE uses ad hoc Java class loaders to allow deploying new/modified workflows that become immediately executable without the need to turn the system down.

An important consequence of the described approach is that WADE workflows, being Java classes,***can be extended***. That is, it is possible to create new workflows by extending existing ones and just defining the differences.

Finally it must be noticed that WADE does not impose that all system logics are defined as workflows. Developers are free to exploit the workflow metaphor to describe those tasks for which they think it is appropriate and use normal JADE behaviours (or other purely Java patterns) elsewhere. In certain cases one could even decide to create a WADE based application that does not use workflows at all.

1.2.2 Workflow Class Structure

As mentioned a WADE workflow is implemented by a Java class that extends the `WorkflowBehaviour`class of the WADE library. More in details each workflow class includes the `defineActivities()` and `defineTransitions()` methods that specify the activities to be executed and the transitions that connect them. Furthermore each activity corresponds to a method that is invoked when the execution flow reaches that activity. Similarly each transition with an associated condition corresponds to aboolean method that is invoked whenever that condition must be evaluated.

WADE supports several types of activity that differ for the actual operations that are executed when the activity is visited. These include

Execution Activities

- ***Code activities*.** The operations to be executed in a code activity are specified directly by the body of the activity method.

- ***Subflow activities.*** The operationsexecuted in a subflow activity consist in the invocation of another workflow process. The activity method implements the logics to launch the subflow and collect the results if any.
- ***WebService activities.*** The operations executed in a Web Service activity consist in invoking a web service.

Synchronization Activities

- ***WaitEvent activities.*** When the execution flow reaches a WaitEvent activity, the workflow suspends until a given event happens.
- ***WaitWebService activities.*** When the execution flow reaches a WaitWebService activity the workflow suspends until a given operation of a previously exposed Web Service is invoked.
- ***SubflowJoin.*** By default subflows are executed synchronously, i.e. the main workflow blocks until the subflow completes and then goes on. Alternatively it is possible to execute a subflow asynchronously, that is the main workflow proceeds just after launching the subflow. When the execution flow reaches a Subflow Join activity the workflow blocks until a previously launched asynchronous subflow completes.

All information related to the appearance of the workflow are maintained in an ad-hoc annotation of the workflow class called `@WorkflowLayout`.

Figure 4 provides a simple example of how a workflow class may look like.

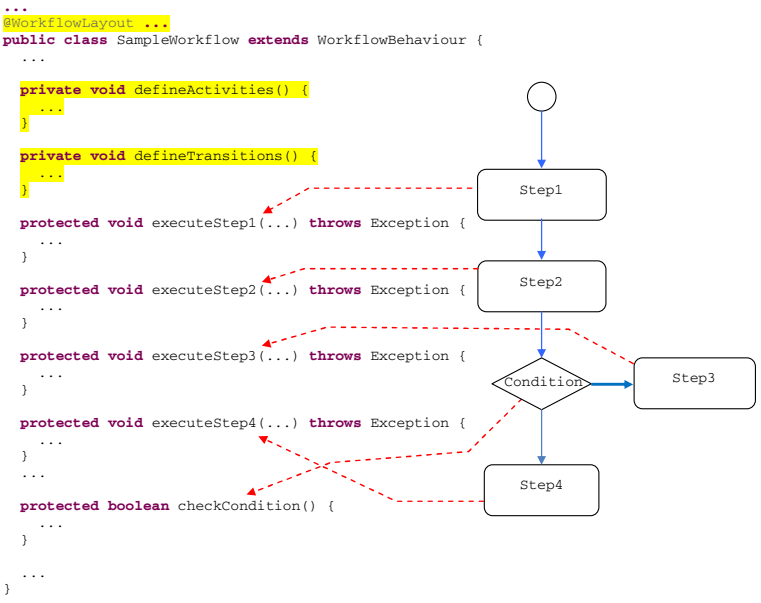

Fig. 4 Workflow class structure

2 The xDSL Domain

In this section the xDSL domain is briefly described highlighting the main phenomena that affect the quality of xDSL services. ADSL (Asymmetric Digital Subscriber Line) and all its variants [8] generically identified with the term xDSL, is a modulation technique that allows transporting data services over traditional copper telephone lines at rates that range from 600 Kbit/s up to 20 Mbit/s. The xDSL segment is therefore the access part (i.e. the part that connects user homes with the closest operator Exchange) of the broadband data network of a telecom operator such as Telecom Italia.

As depicted in Figure 5, when reaching the Exchange, telephone lines are split in two cables. The first one is connected to the traditional telephone network. The second one is plugged in a port of a network element called DSLAM. A DSLAM is basically a multiplexer that receives many flows from user lines and mixes them into a single high speed flow that feeds to the broadband data network. A medium size DSLAM typically has about 800 user ports i.e. 800 connectors where 800 telephone lines can be plugged. At the time of writing, Telecom Italia has about 25000 DSLAMs distributed across the Italian territory from 5 different vendors for a total of about 8.000.000 xDSL lines. The focus in this section is on the new generation of DSLAMs (called IP DSLAMs) whose deployment started in 2006. These are about 6000 appliances and host about 3.000.000 lines. All xDSL lines will progressively be migrated on the new IP DSLAM technology in the next few years.

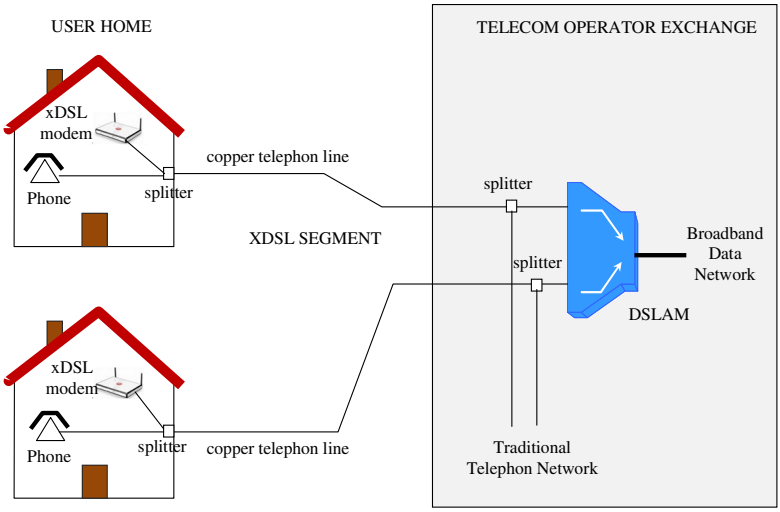

Fig. 5. The xDSL Domain

2.1 xDSL Connectivity

Each port of a DSLAM is equipped with an internal xDSL modem similar to that included in the device at user home. In general an xDSL modem periodically tries to negotiate with its remote peer a suitable transmission rate. Such rate depends on several factors related to both commercial and technical aspects. Commercial aspects (i.e. the contract subscribed by the user) typically define a maximum and a minimum transmission rate. Technical aspects (mainly the signal/noise ratio on the line) determine the actual transmission rate within the range defined by the commercial contract. If the noise is too high to set-up the connection at a rate that is at least the minimum one specified by the commercial contract, the negotiation phase fails. In such case the two modems sleep for a while (typically 20 seconds) and then try negotiating again. When the negotiation phase succeeds the xDSL connectivity becomes UP and the two modems start transmitting at the negotiated rate.

If the user turns off his modem, the DSLAM detects that because no signal is received anymore. In that case the modem in the DSLAM port stops negotiating until the user turn his modem on again.

Finally, by means of a proper management system, a technician can turn the modem in the DSLAM port down. Using the telecom jargon the line is said to be locked. This is typically done when a user withdraws from his contract to be sure that the line will not be used anymore.

Considering the above description xDSL connectivity can be classified into 4 main states:

- WORKING: The two modems are successfully transmitting at a given rate
- NOT_WORKING: The two modems are both on, but,though continuously trying, cannot set-up the xDSL connectivity due to bad line conditions.
- OFF: The modem at user home is off
- LOCK: The DSLAM port has been turned off by a technician.

2.2 Line Quality Management

When the xDSL connectivity is up and the two modems at user home and in the DSLAM port are successfully transmitting data at a given rate, two main problems may affect the line quality.

- ***Transmission errors.*** If the noise on the line increases (this may be due to many factors such as interference with other lines) some bits can be erroneously decoded at the receiving side. These situations are detected by means of CRC checks and the effect is that the packets including wrongly received bits are discarded.
- ***Connectivity leaks.*** As long as the noise increases more packets are discarded. When the number of discarded packets exceeds a given threshold the xDSL connectivity is cut down and the two modems start negotiating a new (lower) rate.

Transmission errors typically have impacts only on realtime services such as IPTV or VoIP. Connectivity leaks on the other hand may severely affect also non-realtime data services such as Internet navigation. In some cases, if the leak is sufficiently long, it is even possible that the computer, set top box and other devices at user home may receive different IP addresses after a connectivity leak.

Many parameters can be adjusted to minimize transmission errors and connectivity leaks according to the line characteristics (line length, cable conditions, noise, services subscribed by the user). Furthermore, considering that line characteristics change over time (e.g. because cable conditions degrades or because another user owning a line close to the first one subscribes to a service that causes interference) such parameters should be periodically re-tuned to keep the quality as high as possible. This practice is known as Dynamic Line Management and, considering that a large telecom operator such as Telecom Italia has millions of xDSL lines, proper automatic mechanisms must be put in place to support it.

3 Agent Based xDSL Monitoring and Management

In this section we focus on Wants Assurance, a large agent-based system recently deployed by Telecom Italia to monitor and dynamically manage all xDSL lines connected to IP DSLAMs. As mentioned at the time of writing the number of these lines is about 3.000.000, but, considering the migration process towards IP DSLAM technology, it will progressively increase up to 8.000.000 and more.

Wants Assurance is fully based on WADE and exploits both its distributed agent oriented nature (inherited from JADE) to achieve scalability and the support for workflows to implement dynamic line management processes that, as will be described later, can be quite long and articulated. The very name Wants derives from Workflows and AgeNTS. The term Assurance on the other hand indicates the process that the system supports. All operations and procedures in the network management domain in facts, are typically classified into three main processes [9]:

Fullfilment– Focusing on setting up services subscribed by customers. This includes inserting new appliances in the network, configuring network nodes to set up end to end connectivity, delivering and installing equipment at customer premises and so on.
Assurance – Focusing on ensuring that a previously set up service keeps on working correctly. This includes fault management, troubleshooting, performance monitoring and so on
Accounting – Focusing on tracking service usage to support billing.

When Wants Assurance was initially conceived in 2008 another mission critical WADE based system had just been successfully deployed. That system, called simply Wants, was (and continues to be) responsible for all configurations on network elements that are necessary to activate broadband data services. Considering the success of Wants in terms of performances, scalability and robustness, it was decided to apply the same combination of agents and workflows to the assurance process too. This is where the name Wants Assurance comes from.

3.1 Event Based xDSL Monitoring

Traditional monitoring systems such as [10] adopt a polling based approach. That is they periodically interrogate the DSLAMs to read the current working conditions (status of the connectivity, current bitrate, detected noise level, transmission errors in the last few hours etc) of all lines one by one. Considering that there are millions of lines, this approach is not particularly efficient. If the polling period is too long it is very easy to miss errors, connectivity leaks and other quality related problems. If it is too short the risk is to burden the DSLAMs with unnecessary management load and the network with unnecessary traffic. In facts in the great majority of the cases the DSLAM will reply that the line is working correctly or is even off.

Unlike traditional monitoring systems, Wants Assurance adopts an event-driven approach. As depicted in Figure 6 it reacts to notifications issued by the DSLAMs and only in that cases, if needed, interrogates the network elements to retrieve further information by means of which it can detect how exactly the line conditions have changed. If no notification referring to a given line is received Wants Assurance assumes the conditions on that line remained the same. As a consequence Wants Assurance is able to continuously monitor in real time all xDSL lines and store significant quality problems in a database as long as they occur. This approach brings two main benefits:

a) When a Telecom Italia technician works on a trouble ticket he is aware of the conditions of the xDSL line both at present and in the past. This is extremely important to properly identify and manage intermittent problems that are very frequent in the xDSL domain.
b) Problems on xDSL lines can be detected even in absence of trouble tickets and therefore it is possible to proactively address them before customers complain.

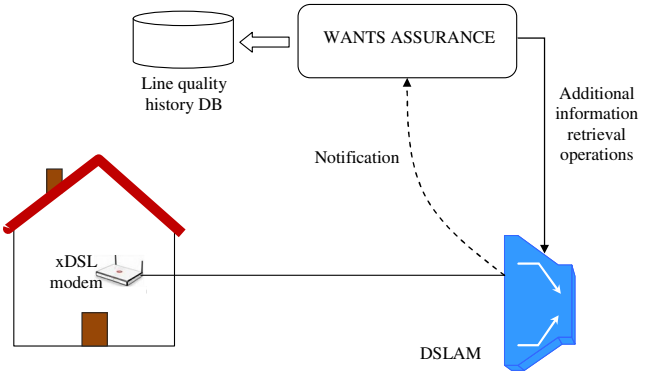

Fig. 6 Wants Assurance monitoring approach

3.1.1 xDSL Connectivity Detection Algorithm

In compliance with the ADSL standard, all DSLAMs issue notifications whenever the operational state of a port changes. When a notification indicating that the state of a port became UP is received, there is no doubt that the xDSL connectivity on the line plugged in that port is WORKING (according to the classification presented in 2.1). Wants Assurance therefore reacts to such notifications simply reading from the DSLAM the bitrate negotiated by the modems in the DSLAM port and at user home. As previously mentioned that bitrate will remain the same until the connectivity is dropped. On the contrary, when a notification indicating that the state of a port became DOWN is received, it is not possible to discriminate whether the xDSL connectivity is OFF, LOCK or NOT_WORKING. In fact the port operational states becomes DOWN both when the user turns off the modem, when a technician locks the line and when actual problems prevent the modems from setting-up the connectivity. Distinguishing between these conditions is of course very important since, while OFF and LOCK are normal situations, NOT_WORKING indicates that there are problems that must be solved. As a consequence Wants Assurance reacts to port-state-DOWN notifications by retrieving additional information from the DSLAM to complete the diagnosis.

3.2 Wants Assurance Internal Architecture

Considering that there are about 3 millions of lines potentially growing up to 8 millions and more, and taking into account that whenever a user turns on or off his modem a port-state (UP or DOWN) notification is issued, it's easy to understand that Wants Assurance must be able to deal with a very large amount of notifications. In particular up to now Wants Assurance manages more than 15 millions of notifications per day that means about 200 notifications per second and in the next few years it is expected to receive up to 500 notifications per second.

Scalability is therefore one of the most important issues that Wants Assurance has to face and this is the main reason why a natively distributed agent-based architecture was adopted. As depicted in Figure 75 main types of agent make up the Wants Assurance system.

Basically all types of DSLAM from all vendors (as the majority of network elements) support the SNMP management protocol [11]. This protocol defines three main primitives.

- GET – The operation by means of which a management system such as Wants Assurance can retrieve information from the network element.
- SET – The operation by means of whicha management system can specify configuration options to the network element and trigger activities.
- TRAP – The primitive used by network elements to notify events to management systems.

GET and SET follow a request-response paradigm. TRAPs on the other hand are one shot SNMP packets and do not have any acknowledgment form.

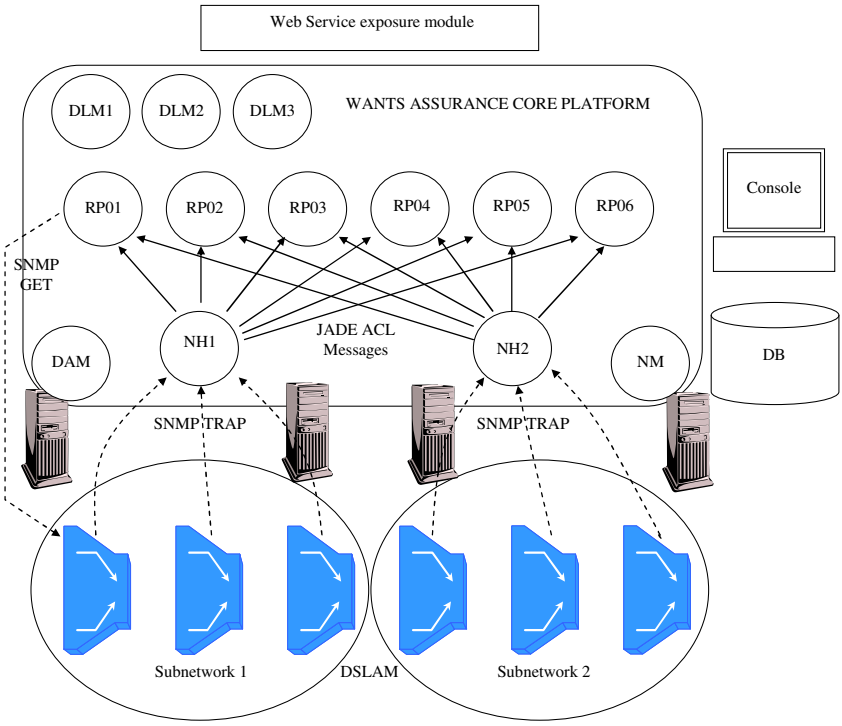

Fig. 7 Wants Assurance internal architecture

Notification Handler Agents (NH) – These are the agents responsible for receiving all notifications (SNMP TRAPs) from the DSLAMs. In order to facilitate network elements pre-configurations,the 5000 IP DSLAMs are logically divided into 5 subsets called sub-networks. All DSLAMs belonging to a given sub-network are configured to send SNMP TRAPs to a given IP address where a Notification Handler agent is ready to receive them.

Resource Proxy Agents (RP) – These are the agents in charge of actually processing notifications and diagnosing xDSL connectivity status variations and transmission errors. At present Wants Assurance is configured with a pool of 200 Resource Proxy agents. At system startup each Notification Handler assigns the DSLAMs belonging to its sub-network to available Resource Proxy agents distributing them according to a round robin policy. After the initial assignment phase a Notification Handler agent will forward all notifications received from a given device to the Resource Proxy agent that device was assigned to. This ensures that notifications referring to the same line are processed in the right sequence even if they are very close in time. Furthermore the fact that the assignment relation between a DSLAM and a Resource Proxy agent is not preconfigured, but is dynamically built at system startup, allows adding new devices or increase the RP pool size at will without taking care of properly modifying the system configurations.

Device Assignment Manager Agent (DAM) – This is the agent that keeps track of device-RP assignments. Whenever a component in the system has to act on a DSLAM it must first retrieve from the Device Assignment Manager agent the Resource Proxy agent that owns the target DSLAM and then pass through it.

Network Manager Agent (NM) – This is the agent responsible for implementing all administration actions such as adding a new device in the system.

Dynamic Line Management Agent (DLM) – These are the agents in charge of the Dynamic Line Management processes that will be described in more details in section 3.3.

The system architecture is completed by an Oracle Database used to keep inventory information of all DSLAMs monitored by the system and to store all xDSL connectivity status variations and transmission errors, a Web based Graphical User Interface and a Web Service exposure module that allows all Wants Assurance monitoring features to be accessed by authorized external systems.

3.3 Dynamic Line Management

As mentioned in section 2.2, the term Dynamic Line Management indicates the practice of modifying configuration parameters that determine how the modems in the DSLAM port and at user home set-up the xDSL connectivity to continuously optimize the transmission quality. More in details the Dynamic Line Management process involves the following main steps.

- ***Trigger*****:** detect quality problems on a line
- ***Measure*****:** if necessary retrieve additional information about the working condition of the line
- ***Diagnosis*****:** on the basis of the quality problems and of the retrieved additional information identify new configuration parameters that can improve the situation.
- ***Action*****:** apply the new configuration on the network element.
- ***Control*****:** compare the quality before and after the action and determine whether or not the effect was beneficial.

Clearly a system like Wants Assurance that continuously monitors the quality of all xDSL lines is a strong enabler for the Dynamic Line Management process as it natively implement both the Trigger and the control steps. As a consequence at the beginning of 2011 it was decided to enhance Wants Assurance with Dynamic Line Management features.

Considering that Dynamic Line Management processes require several steps, can take quite a long time (for instance the Measure step often implies activating a campaign that takes an entire day; similarly the Control step lasts at least 3 days to be sure that the new configuration did not introduce bad side effects) and need to be tracked (for instance to produce reports about general quality improvement of the network), the obvious choice was to implement them exploiting the WADE workflow support. A pool of 20 DLM agents was therefore added to the system. Unlike other Wants Assurance agents, DLM agents extend the WorkflowEngineAgent class of the WADE library and are therefore able to execute workflows.

Figure 8is a snapshot taken from Wolf (the workflow graphical editor of WADE) and represents the main DLM workflow process that is activated when Wants Assurance detects that the quality of a line decreases below a given threshold for at least two days. The first two steps implement process initialization operations and preliminary checks. In particular the customer identifier (typically the telephone number) and the subscribed commercial service are retrieved from the inventory and, in case another DLM process is already active on the same line the current one is immediately aborted to avoid conflicts and duplications. In step 3 the current line configurations are read from the DSLAM and in step 4 proper indicators about the current quality of the line are computed and stored. These indicators will be used both in the successive diagnosis step and, at the end of the process, to evaluate the overall quality variation. Step 5 (EstimateLine) is the core of the process and implements the diagnosis phase. More in details it estimates, by means of sophisticated algorithms whose details are out of the scope of this chapter, the configurations that can optimize the quality of the line (maximize bitrate and minimize transmission errors and connectivity leaks). If an optimal line configuration is determined, the following step interacts with the Activation System to apply that configuration in the network element. The final step monitors the quality of the line in the successive days, compares it with the quality indicators stored in step 4 and produces a report of the actual quality variation. It should be noticed that this step (as well as the EstimateLine one) is implemented itself as a workflow.

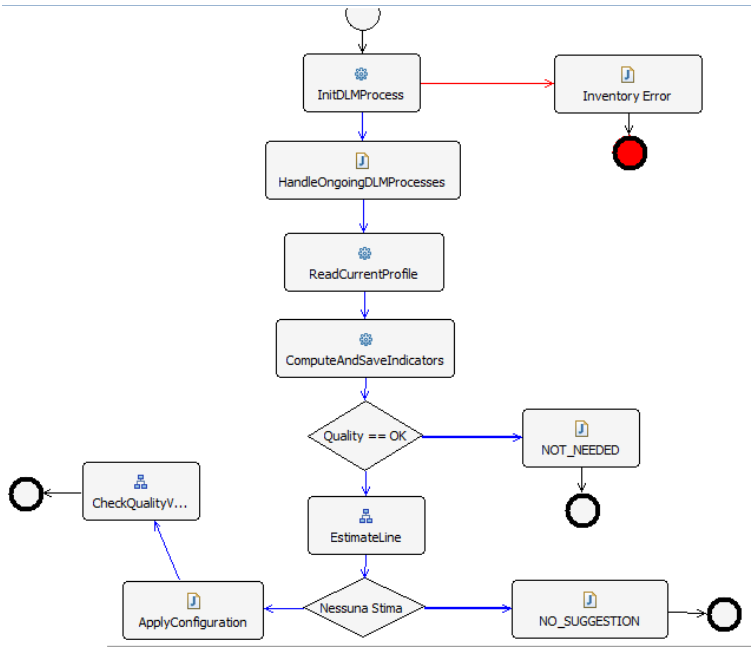

Fig. 8 Main DLM workflow process

4 Conclusions

The Wants Assurance system described in this section is actually the third success case of agent-technology exploitation in Telecom Italia. Other two mission critical systems built on top of WADE are in facts already in use since some years [12]. The first one called "Network Neutral Element Manager" implements a mediation layer between network elements and OSS systems. The second one, known as "Wizard", provides step-by-step guidance to technicians performing maintenance operations in the fields. In all cases WADE has been used as development framework due to its characteristics in terms of

- ***scalability*** – agent oriented systems are by their nature distributed and, as such, provide high degree of scalability
- ***flexibility*** – the workflow metaphor allows deploying new or modified business logics on the fly
- ***maintainability*** – though distributed, WADE-based systems are quite easy to configure, administer, monitor and troubleshoot in case of unexpected problems.

It should be noticed that agent technology provides additional important features such as reasoning, planning and more in general "intelligence" that allow an agent-based system to deal well enough with unforeseen situations. Such features were not used in the three systems developed and deployed by Telecom Italia though.

Up to now this is actually the main lesson learned. Typical characteristics of agent technology start showing their benefits in extremely complex situations where traditional techniques can hardly be applied. However, at least in the Network Management domain, the Industry still focuses on systems that can be fully controlled, whose behavior can be predicted with no uncertainty. Systems that are guaranteed to work properly h24 and to scale well as long as the load increases.

Though been completely agent-oriented, WADE has been designed to take these issues into account first. Therefore it helps decreasing the development effort also in fully known and deterministic contexts and even in prototyping activities.

In the next few years, with the introduction of new broadband technologies such as LTE (Long Term Evolution), the network complexity is expected to sensibly grow. If this trend may lead to situations where agent intelligence can actually make the difference, it will be possible to make Telecom Italia WADE-based evolve to fully exploit the real agent oriented nature of WADE.

References

[1] WADE – Workflow and Agents Development Environment, `http://jade.tilab.com/wade`
[2] JADE – Java Agents Development framework, `http://jade.tilab.com`

[3] Caire, G., Porta, M., Quarantotto, E., Sacchi, G.: Wolf – an Eclipse Plug-in for WADE, http://jade.tilab.com/wade/papers/Wolf_ACEC_2008.pdf
[4] Eclipse, http://www.eclipse.org
[5] Bellifemine, F., Caire, G., Greenwood, D.: "Developing Multi Agent Systems with JADE" – Wiley Series, http://eu.wiley.com/WileyCDA/WileyTitle/productCd-0470057475.html
[6] FIPA – Foundation for Intelligent Physical Agents, http://www.fipa.org
[7] van der Aalst, W., van Hee, K.: Workflow Management: Models, Methods and Systems, http://www.hoepli.it/libro/workflow-management/9780262720465.asp
[8] ADSL2 and ADSL2plus – The new ADSL standards, http://www.broadband-forum.org/marketing/download/mktgdocs/ADSL2_wp.pdf
[9] TMF eTOM – Enhanced Telecom Operations Map, http://www.tmforum.org/BusinessProcessFramework/1647/home.html
[10] Motive Network Analyzer Copper – Alcatel Lucent http://www.alcatel-lucent.com/wps/portal/!ut/p/kcxml/04_Sj9SPykssy0xPLMnMz0vM0Y_QjzKLd4w3dnTRL8h2VAQADYR9IA!!?LMSG_CABINET=Solution_Product_Catalog&LMSG_CONTENT_FILE=Products/Product_Detail_000387.xml&s_cid=smm_tmc0229_bl
[11] SNMP – Simple Network Management Protocol, http://www.pulsewan.com/data101/pdfs/snmp.pdf
[12] Banzi, M., Caire, G., Gotta, D.: WADE: A software platform to develop mission critical applications exploiting Agents and Workflows. In: AAMAS 2008 Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems: Industrial Track (2008)

Chapter 5
The Jadex Project: Simulation

Lars Braubach and Alexander Pokahr

Abstract. Simulation is on the one hand an important application area for multi-agent systems, but on the other hand also a useful tool for building agent applications. This chapter investigates constructs and techniques that foster both usages of simulation in the context of agent technology. The vision for integrating simulation support consists in establishing simulation transparency, i.e. it should be ensured that applications can be built to a large extent without simulation specific parts. First, approaches for dealing with time in simulated and non-simulated agent execution are discussed. Afterwards the role of virtual environments in agent applications is tackled. Both technical topics are illustrated using concrete applications that further represent the different usages of simulation.

1 Introduction

The combination of agents and simulation forms a mutual benefit. Multi-agent-based simulation (MABS) is an approach, that uses the concept of an agent for supporting social simulation. Agents are well suited for e.g. representing realistic human behavior in simulation models, such as pedestrian traffic in a to be constructed train station. Therefore, agents are an accepted technology in the area of simulation. Viewed from the opposite direction, simulation is also a useful technology for supporting the construction of agent applications. In many agent applications, the interaction between the agents is considered to be an important part of the computational algorithm, e.g. in negotiations or decentralized coordination. Building such agent applications often requires fine tuning of parameters and making sure that the application produces suitable results, which can both be achieved by simulating the application behavior for testing and evaluation purposes.

Lars Braubach · Alexander Pokahr
Distributed Systems and Information Systems
Computer Science Department, University of Hamburg
e-mail: {pokahr,braubach}@informatik.uni-hamburg.de

M. Ganzha & L. C. Jain (Eds.): Multiagent Systems & Applications, ISRL 45, pp. 107–128.
springerlink.com

This is one of two chapters describing practical applications built with the Jadex agent framework. This chapter describes techniques and constructs of Jadex, that particularly focus on supporting simulation while establishing *simulation transparency* to a large extent. Simulation transparency means that the functional code of an application must not contain any simulation specific aspects. In this way an easy transistion between simulations and applications can be achieved, e.g. code from an upstream simulation can be directly used to implement an application. It has to be noted that the environment and its connection to the application cannot be made transparent as in many cases a virtual one needs to be replaced by a real one.

Each section starts with a short historical background about why a certain topic was considered important for Jadex, followed by a more general motivation about the relevance of the concept itself. A related work section is presented for each concept, trying to give an overview of the field with pointers to other relevant works in the area. Afterwards the approach as implemented in Jadex is covered in detail and further illustrated by example applications that have been built. Each section closes with a short summary.

In this chapter, the following topics are tackled. The simulation of agent systems for supporting analysis and testing of applications is examined in Section 2. A useful supplement for simulation, but also a relevant topic in itself is the concept of virtual environments as described in Section 3. Finally, in Section 4 a conclusion and critical reflection on the described topics is given.

2 Simulation Clocks

One important aspect of simulation is how time passes during simulation runs. E.g. event-driven simulation allows executing scenarios "as fast as possible", because computation only happens for the relevant time points. Using timed execution, simulation helps comprehending system activities and interrelationships from a global perspective and in a timely condensed fashion. Analysis of system behavior can be done in different ways. On the one hand hypotheses about the system behavior can be tested using simulation experiments and on the other hand conclusions can be drawn from experimentation under different setups e.g. comparing alternative strategies. Finally, verification of system behavior is related to hypotheses testing but more concerned with ensuring that the simulation model complies in its behavior to some system design specification.

Yet, the connection between simulation and application construction is often not well established. In many cases simulation is considered on its own as technique for experimentation. If a simulation is part of a real world application development project, in many cases simulation is used to analyze and verify the expected real world system behavior. Hence, the simulation

model determines the design and implementation of the real world application, which is typically built from scratch, after simulation model validation has been taken place. This kind of throw away system construction is problematic not only for resource wastage reasons and the increased effort due to double development but also with respect to the preservation of validated system properties. These cannot be easily guaranteed for the newly built application if some of the model assumptions are implicit, e.g. hidden as implementation detail.

2.1 Related Work

Corresponding to the two viewpoints, agents for simulation vs. simulation for agents, solutions can be broadly categorized according to agent-based simulation toolkits and agent platforms with support for simulations.

The first group includes approaches systems like Repast [2], NetLogo[1] and SeSAm [6]. Most of these kinds of systems use a simple time-driven simulation clock advancement mechanism, which assumes the time passes in fixed steps. All agents are notified when a new round begins, typically in a sequential one by one manner, by invoking a behavior method. The processing of an agent is finished in a round when its behavior method is done. As communication between agents is handled in an indirect way by using the environment there are no negotiation based interrelationships between agents that need to be considered for end of processing determination. The whole round ends when all agents have finished their executions. In addition, most simulation frameworks assume a very simple agent architecture so that simulation scenarios with many simple agents are supported best. The time-driven clock mode has the advantage of being easy to understand and but disadvantage of being inefficient when activities of agents are not equally distributed over time.

Typical representatives in the area of agent platforms with simulation support are Cybele[2], Brahms[3] and PlaSMA [12]. PlaSMA is an extension for the JADE agent platform allowing it to be used as simulation runtime environment. In order to control the JADE agents PlaSMA uses a conservative time scheduling that resembles the time service protocol introduced in Section 2.2.1. The protocol is hidden from the user by using a simulation agent base class, which automatically notifies the clock when the agent's processing is done. On the other hand, OpenCybele as well as Brahms employ infrastructure support based on clocks similar to the approach described in Section 2.2.2.

[1] http://ccl.northwestern.edu/netlogo/
[2] http://www.i-a-i.com/cybelepro/
[3] http://www.agentisolutions.com/

2.2 Approach

An important aspect of the solution consists in understanding that time advancement control is the key concept of simulation environments. The time advancement mechanism determines the temporal way the application is executed, e.g. in real time or in an event driven mode. The general idea is to use a clock abstraction for encapsulating the logic of time advancement separated from the rest of the infrastructure. The only simulation related activity in application code, which is valid for real-time applications as well, consists in issuing wait actions that interrupt processing until the specified time point has been reached. This allows executing the same application in different modes just by switching the underlying clock type. In general there are two different ways for realizing such a clock as a generic reusable component. It can either be intimate part of underlying runtime infrastructure or it can be added at the application layer. The first option has the advantage that the programming model for applications needs not to be touched at all as the clock interaction can be integrated in the API (application programming interface). On the other hand, it is challenging because it has to be anchored at the heart of the runtime platform. In contrast, the second option is non-invasive with respect to the infrastructure but requires the programmer to explicitly handle a time protocol for clock interaction in addition to the existing platform API. In the following a closer look will be taken at both solution paths, starting with the time service as incarnation of an application layer solution.

2.2.1 Time Service

The time service concept [1] assumes that a global coordinator is responsible for time management. Time clients have to contact the coordinator in case they have finished their activities or want to wait for a specific point in time. Adapting the notion of synchronous process-oriented simulation, the participants send to the coordinator a *Passivate* message to indicate that they have no scheduled activities, or a *Hold(t)* message with the time of the next activity that needs to be scheduled. The coordinator acts in a round based fashion by waiting for messages of each notified participant. It uses the messages to update its global list of announced time points. A round is finished when all participants have sent their decision. The coordinator then iteratively removes the next entry from this list, advances the clock, and informs the corresponding participants that the time point is reached. Then the coordinator will wait until all processes have answered again.

Fig. 1 shows an UML interaction protocol of the time service protocol. The four different interaction cases are denoted by the characters *a* to *d*. The initialization phase (*a*) is used by participants to register at the coordinator using a *Register* message and will receive an *agree* message, if they are not already registered. When a participant terminates it requests a *Deregister*

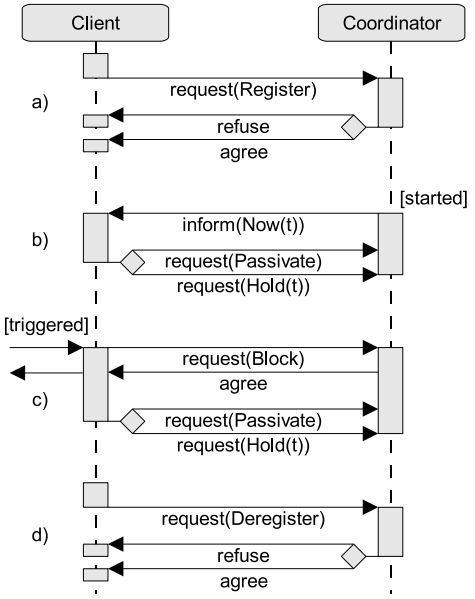

Fig. 1 Time Service Protocol (from [1])

(*d*), receiving a *refuse* if the participant was not registered. The other two interactions (*b* and *c*) may happen repeatedly during simulation runs.

The initial time (*Now*) is sent to all participants after the simulation has been started (*b*). New participants that register while the simulation is running will immediately receive the *Now* message with the current simulation time. While the simulation is running a participant will continuously receive so called wake-up calls whenever its registered point in time is reached. After receiving the wakeup the participant is supposed to execute its current activity and sends back a message afterwards. Either it announces the time point (t) for its next activity by submitting a *Hold(t)* request, or it currently has no activities to be scheduled and therefore submits a *Passivate* request. If the participant does not answer in a pre-defined timeout period the simulation continues with the next event excluding the non-answering candidate.

The participant that has been woken up may interact freely with all available other participants. Thus, waiting participants can react to messages received from other participants. In addition, a waiting participant may receive new information leading him to reconsider its activation time point (*c*). It can decide to activate itself at the current point in time by sending a *Block* request to the time service. The service removes the original time entry and acknowledges this by sending an *agree* message. It has to wait until all active participants declare that they are finished by sending a *Passivate* or *Hold(t)* request.

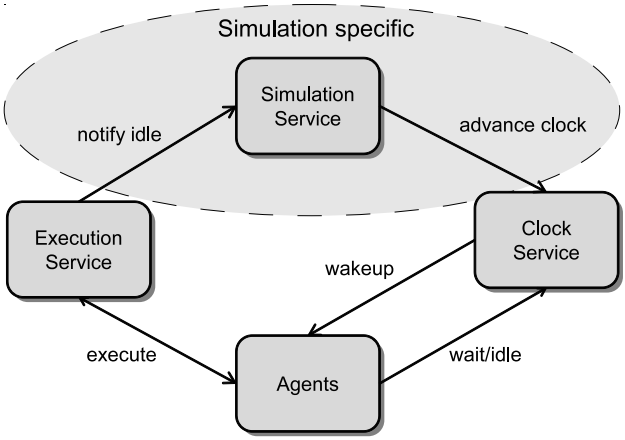

Fig. 2 Infrastructure components

2.2.2 Infrastructure Clock Support

Infrastructure clock support allows completely hiding the timing mechanism that is needed to control application execution. For this purpose the clock has to be coupled with the execution machinery of the infrastructure. The general setup of such an infrastructure is depicted in Fig. 2. It consists of three interacting services, namely the *execution*, *simulation* and *clock service*. These services control the execution of the application by activating agents at specific time points. Concretely, the clock service manages an ordered list of time points which have been announced by the agents calling *wait* or *idle* (if they do not wait). Whenever the simulation clock is instructed by the simulation service to *advance* the time, it removes the next time point from the list and calls *wakeup* on the corresponding agents. The agents are subsequently *executed* by the execution service as long as they wish to perform tasks at the current point in time. The execution service monitors overall agent execution activities until no agent wants to be executed any more. If this is the case, it *notifies* the simulation service about the reached quiescence. The simulation service acts as connecting link between execution and clock service. It allows controlling exactly when the clock is advanced and normally instructs the clock to advance the time whenever it receives notifications of the execution services. In some situations it can also defer the clock notification e.g. if the system is running in stepped mode and requires a human user to trigger the next clock step.

In addition to simulations, Figure 2 also shows how normal applications are executed. In this case the simulation service does not exist and there is no connection between the execution and clock service. This works because normal clocks are active by themselves, i.e. in contrast to simulations time advances automatically and the clock does not need an external trigger. From

the agent's perspective the execution works in the same way as before by announcing wait and idle commands so that they can be built agnostic with respect to the execution mode.

Infrastructure clock support currently is limited to simulations running on the same platform. In order to support also distributed simulations on infrastructure level, the clocks of the participating platforms would have to adhere to a protocol that ensures that a virtual global simulation time is used. Following a conservative approach, one could employ a master slave approach, in which the participating clocks first use an election algorithm to decide about the master role, and afterwards use the master to annouce timing events in a similar way as in the time service protocol from the last section. Such a solution is transparent for the agents on the platform as they need not to be aware of how the clock service derives its current time.

2.3 Applications

In the following two example applications will be presented, for which simulation is a necessary prerequisite. The first, called MedPAge, deals with appointment scheduling in hospitals and the second, named SodekoVS, tackles software engineering with self organization.

2.3.1 MedPAge

Within the "MedPAge" hospital logistics project (cf. the other Jadex chapter in this book), an important requirement was the ability to benchmark different kinds of hospital appointment scheduling algorithms against each other. This was needed to better understand the quantitative advantages and problems of the new decentralized, agent-based approaches with respect to the established mechanism within hospitals. In order to efficiently execute the MedPAge application many time with varying parameters and underlying scheduling mechanisms simulation techniques are required. Using event-driven simulation instead of real-time execute allows conducting the experiments as fast as possible, i.e. computing resources are the only determining factor for experiment execution time.

The second crucial requirement within MedPAge was the ability to use the system to perform benchmarks and to run it as application prototype within a real hospital environment. Typically, this would require a huge effort as simulation and execution platforms rely on a different set of concepts and it is not easily possibly to adapt code written for one type of platform to the other. Furthermore, in most cases agent simulation toolkits focus on large number of simple agents and do not support negotiation protocols and intentional agent concepts. Hence, from those MedPAge project requirements a runtime infrastructure should naturally include general simulation support and allow programming of simulations and applications with a consistent programming model for both.

The infrastructure clock support described above has been integrated into the Jadex platform in order to simplify the development of simulation applications and especially enable the creation of simulations whose code can be kept for subsequent application development. This capability proved useful in the context of the MedPAge project. In an earlier project phase the simulation capabilities of Jadex could be exploited to isolate the most promising appointment scheduling algorithm among several approaches in the hospital domain. In a later project phase the application was extended in the direction of an assistance system that concrete helps with deciding which patient should be called next to a functional unit for treatment. For this purpose the appointment scheduling mechanism was kept as is and an additional user interface was added to the system. Using the system clock instead of a simulation clock the application could directly be tested within its target environment [14].

The time service approach from Section 2.2.1 was an integral part for simulation control in the so called Agent.Enterprise [3] and Agent.Hospital [5] initiatives. They were part of the German priority research programme SPP 1083 and served as integration approaches for the numerous subprojects within the SPP, including MedPAge. The general idea was to create a complex enterprise or hospital application scenario, in which the projects work cooperatively together to fulfill higher level objectives. Concretely, the Agent.Enterprise scenario is an inter-enterprise multi-level supply-chain scenario, including process planning and SCM scheduling as well as tracking and tracing of supply chains. Agent.Hospital builds on a model with numerous different healthcare actors and consists of detailed partial models of the healthcare domain. It enables the examination of modeling methods, configuration problems as well as agent-based negotiation strategies and coordination algorithms. The Agent.Enterprise and Agent.Hospital applications have been realized as so called multi-multi-agent systems, i.e. a multi-agent system that is composed of further multi-agent subsystems that bring about specific functionalities. The role of the time service was to timely coordinate the simulation within the overall multi-multi-agent system by managing the execution order of the different subsystems.

2.3.2 SodekoVS

"SodekoVS" [10] is a DFG-funded[4] project that aims at making self-organization techniques usable as part of the normal software engineering process. In many application areas non-functional requirements like fault tolerance and adaptability play an important role. Examples include urban transport systems consisting of many small vehicles, low cost satellites that are able to perform a mission together as well as monitoring and automatic reconfiguration of server farms in case of changing customer demands. From

[4] Deutsche Forschungsgemeinschaft (German Research Council): http://www.dfg.de

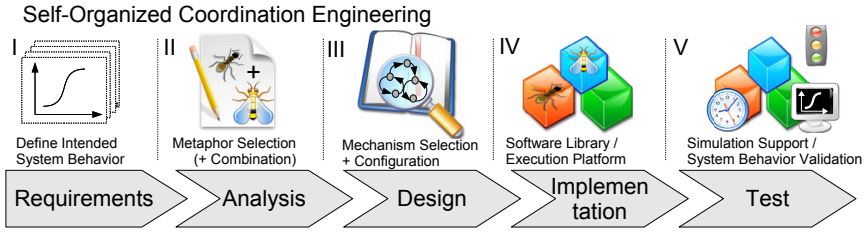

Fig. 3 SodekoVS development process (from [10])

these examples it becomes apparent that it is a key requirement that single entities may fail at any point in time, e.g. a hardware error occurs in a server, and these error must not disturb the overall system functionality. Furthermore, the examples highlight that a completely decentralized infrastructure is assumed in which a multitude of autonomous entities act and interact to bring about the system objectives. No superordinated entity exists, which on the one hand avoid a single point of failure but on the other hand demands novel software concepts to realized coordination as a function of peers.

Methodology

The SodekoVS project aims at providing a development process as well as a middleware for constructing applications with self organization features. The SodekoVS middleware is based on the Jadex agent framework and especially depends on the integrated simulation support. In Fig. 3 the proposed development process is shown. It can be seen that it shares the typical development phases with traditional processes and adds additional self organization tasks in each phase. At the heart of the approach distributed control loops are used to describe the expected macroscopic behavior in terms of role interactions. The control loops describe coordination behavior in terms of system state variables and causal relationships between them denoting the rates of change. Starting from the requirements phase the intended system behavior is elicitated. In the following analysis phase it has to be decided which coordination metaphor fits best the application needs, examples include pheromone approaches inspired by ant colonies and waggle dances of bee societies (cf. [11]). Afterwards a catalogue of ready-to-use self organization mechanisms, in the spirit of software engineering patterns, can be inspected in the design phase to find a suitable coordination mechanism. The patterns represent implemented coordination strategies for common use cases and can be directly integrated into an application. A developer has to configure the mechanism according to the system variables to be used and their update rates. In the implementation phase the binding between these variables and the agent states has to be defined. In this respect it has to be concretized in which situations agents play specific roles according to the macroscopic model and

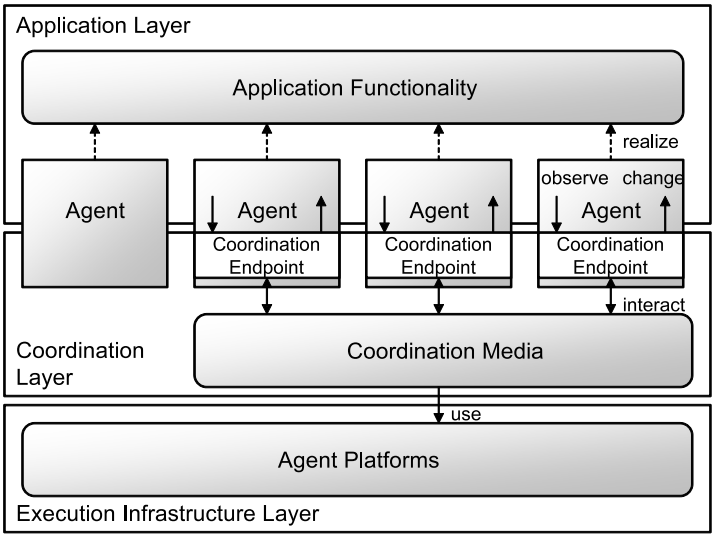

Fig. 4 SodekoVS middleware (from [10])

how transitions between such roles occur, i.e. it has to be defined how single agent behavior causes coordination actions. After the system has been implemented its behavior especially with regard to the coordination behavior has to be tested and validated. For this purpose the system is run in simulation mode in various different scenarios. Often self-organization mechanisms require parameter adjustment to function as expected. Using simulation these tasks become manageable and executable in a comparably short amount of time (compared to real-time application configuration). In case validation is completed successfully, the system can be deployed in the target environment and operates in real time.

Middleware

The architecture blueprint of the SodekoVS middleware is depicted in Fig. 4. The figure shows a layer model with three layers. At the bottom the execution infrastructure layer is located. In this layer the agent platforms are placed, which are responsible for providing fundamental services to the upper layers, e.g. agent execution and management on possible different network nodes. The topmost application layer realizes the application functionality by a set of agents. Between these layers, SodekoVS adds a coordination layer, which has the task to rather transparently realize the modeled self organization. For this purpose the concepts of coordination endpoints and coordination media are introduced. Each coordination endpoint registers itself with a coordination medium and is this way a network of endpoints is created. A coordination endpoint is part of an agent that independent of its original behavior and is

used for two purposes. On the one hand, the endpoint observes the agent's state and forwards relevant changes to its associated coordination medium. On the other hand, the endpoint receives information updates from the coordination medium and influences the agent behavior if those updates are relevant to the entity. The coordination medium itself realizes the dynamics of information processing and distribution by relying on specific decentralized coordination mechanisms. The explicit distinction between endpoints and media reflects the conceptual separation of local entity adaptations and coordination based information exchanges.

The SodekoVS middleware has itself been used for the development of several self-organized simulations and applications. Most notably, the approach was employed in the logistics domain to optimize parcel routing via heavy goods vehicles (HGV) between redistribution centers. In this scenario a market-based negotiation strategy was applied that allowed parcels to bid for transportation by an HGV with a virtual currency. The HGVs transport parcels between redistribution centers and try to optimize their own profit by serving different routes and negotiating prices with goods. More details about the approach can be found in [7].

2.4 Summary

In this section the usefulness of simulation itself and simulation techniques as part of application development have been discussed. It has been highlighted that time advancement is of crucial importance for simulation infrastructures and that it is possible to factor out time management and provide solutions that operate independently of the programming model. This leads to a consistent agent programming for simulations and applications using the same concepts. From the programming perspective one is unaware of the time mode the application is run with and can use a clock type that fits to the scenario needs.

Two conceptually different approaches have been introduced for externally and internally controlling time advancement. The first introduces an infrastructure service called *time service*, which represents a global clock that is used by the agents to coordinate their execution according to the simulation time. Concretely, the service manages a list of time points announced by the agents. In case a time point is due the corresponding agent is awakened and starts processing, which may involve communication with arbitrary other agents that may also start processing. After the activated agent has finishing processing it needs to notify the time service so that the clock can advance and the next agent is activated. All agents may decide to change their registered time point at any time if new information becomes available.

The second approach is based is tightly integrated with the runtime infrastructure and uses the interplay of three services for bringing about simulation in a completely transparent way. The clock service manages again the list of

registered time points. In this case the execution service, which monitors the agent activities, triggers the advance of the clock (indirectly via the simulation service) whenever the agents have finished their processing. In contrast to the time service solution also the interaction with the clock is completely hidden. This is achieved by making the agent to clock interaction part of the normal platform level application programming interface.

It has further been shown that both approaches can be used within different context beneficially. The time service allows creating simulations within heterogeneous and possibly distributed environments, in which no direct control about the execution infrastructure can be exerted. On the other hand, simulation clocks allow constructing simulations and applications only within one platform but with much less effort because the agent programmer does not have to care about simulations and can build its application as if it were a normal application.

3 Virtual Environments

Virtual environments have a number of typical and less common use cases. Obviously, virtual worlds form an integral part of many computer games, and similar technologies can as well be found in training applications. Also for the teaching of agent concepts, virtual worlds are often employed, as the idealized settings simplify the understanding of the complex concepts. But even for the development of more conventional (e.g. business) applications, virtual environments can be a helpful tool. During implementation it can be helpful to execute parts of the later system in a controlled environment. In this case, the virtual environment would represent external systems or subsystems. Explicitly modeling this environment allows observing the behavior of implemented components in certain situations. This approach can be regarded as similar to mocking techniques as found in software testing, where e.g. special mock objects are built for replacing parts of a real software environment during testing. Especially for agent systems, that exhibit pro-active, autonomous and adaptive behavior, setups based on virtual environments are helpful for testing and debugging the complete application during the implementation phase. A virtual representation of the external environment is also paramount when using simulation as described in Section 2. During application development, simulation can fulfill a number of different purposes. On the one hand, it allows intensive testing of an application prior to putting in into productive use. On the other hand, one can benchmark different implementations in the same environment or test implemented system behavior in changing environments. Finally, one may consider the virtual environment as part of a deployed application in the sense of augmented reality. For example, a virtual environment could be used for representing digital pheromones as part of an ant-like path-finding algorithm for a transport logistics application.

3.1 Related Work

In line with the use cases for virtual environments mentioned in the previous section, at least two different strands of research related to agents and virtual environments can be identified. The first concerns specialized simulation toolkits that often include simple agent frameworks for easy definition of the behavior of simulated entities. Typical examples are NetLogo[5] and Repast Simphony[6]. These toolkits are well-suited for agent-based simulation, e.g. for teaching or analysis purposes. In this respect, simulation toolkits usually offer rich facilities for statistical evaluation. Also visualization, e.g. as 2D virtual worlds are a typical strength of these systems. Some simulation toolkits, such as SeSAm [6], even offer graphical tools for specifying simulation behavior making them usable even by non-programmers. On the other hand, simulation models developed in these systems are not meant to be part of deployed applications. Thus unlike middleware platforms like JADE, simulation toolkits do not provide a communication infrastructure or interfaces to external systems.

Another strand of research investigates the explicit modeling of environments for agent-based applications. E.g. Weyns et al. argue in [13] that an explicit representation of an environment in agent applications can be useful as a part of the application itself, e.g. for a coordination layer. The idea is that a clean separation between agent and environment implementation simplifies the development and leads to better maintainable code. On specific model for such an explicit environment is the A&A model, which considers an application to be composed of agents and artifacts [9].

3.2 Approach

The approach chosen for supporting the development of virtual environments in Jadex was separating the different aspects and allowing a declarative specification of each of them [4]. The resulting model is depicted in Figure 5. The *space* describes the environment itself and is further subdivided into the *domain*, i.e. parts of the environment that are independent from agents, and the *interaction*, which establishes a connection between the domain and the agents that are meant to inhabit the environment. The domain representation of the environment state is used by two further aspects of EnvSupport. The *observer* provides a visual representation of the environment, e.g. as a 2D map, and the *evaluation* component extracts environment data for statistical analysis. All these aspects are described as part of an application XML file and are interpreted by the Jadex platform at runtime. In the following sections, each aspect will be explained in more detail.

[5] http://ccl.northwestern.edu/netlogo/
[6] http://repast.sourceforge.net/

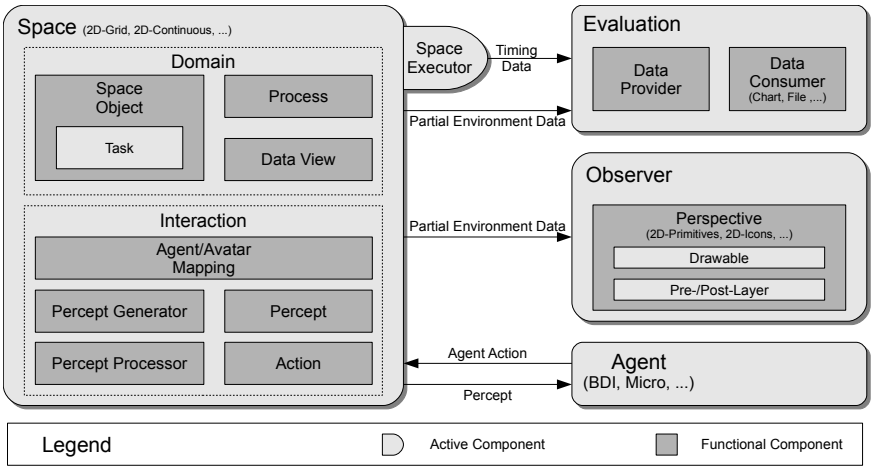

Fig. 5 EnvSupport structure (from [4])

3.2.1 Domain

The underlying assumption regarding the domain is that the state of an environment can be described as a set of objects, so called *space objects*. For each application, the developer can freely define the available object types in the environment, where each type defines a set of properties for describing the object, such as position, size, etc. For static environments it is sufficient to describe the types and instances of all objects. Dynamic environments may change without any actions being performed by agents. Therefore the developer can specify such changes in the environment in two ways. *Tasks* are attached to an object and may continuously change the state of this object (e.g. movement of a car, growth of a plant) until the task is stopped or the object is destroyed. *Processes* are applied to the environment as a whole and may induce changes on all objects as well as destroy some existing objects or create new ones. A typical use case is the creation of new objects according to some predefined stochastic distribution (e.g. arrival of cars at a junction, when the environment should only represent the junction itself).

3.2.2 Interaction

The interaction describes the information flow between the agents and the environment as represented by the state objects. For making each agent situated in the environment, an agent usually has an *avatar*, i.e. a space object that is owned by the agent. In this constellation, the agent represents the brain and the avatar space object represents the body of the situated entity. The interaction is divided into *percepts*, which are environmental states or changes observed by the agent's avatar and forwarded to the agent, and *actions*, that

allow an agent to manipulate the environment state. *Percept generators* can be defined that describe how and when percepts are produced, e.g. by assigning a vision range to an avatar object and generating a percept whenever objects enter or leave the vision range according to the avatars current position in the environment. To simplify dealing with percepts, *percept processors* further describe how a percept enters the internal reasoning process of the agent. E.g. a ready-to-use BDI percept processor allows mapping percepts directly to some belief or belief set of an agent and therefore achieves a seamless integration of EnvSupport with the BDI architecture. For simple micro agents typically custom percept processors are defined by the application developer for triggering appropriate reactive behavior of the agent. Actions are requested by the agent and performed by the *space executor*. The space executor takes care of proper synchronization of agent actions, object tasks and environment processes. Depending on the scenario, the developer may choose a round-based or a continuous time space execution. The first model allows each agent to perform only one action per time step and is especially useful in conjunction with simulation clocks. The second model resembles are natural evolvement of time and only settles conflicts, e.g. if two agents try to pick up the same item at the same time, the executor will make sure that only one of these actions succeeds and the other one produces a failure.

3.2.3 Observer

The purpose of visualization is usually gaining a better understanding of the behavior of the application, either to use the application (e.g. a game or training simulation) or to analyze and debug the application. It largely depends on the structure and properties of the space objects how an environment can be visualized. Typically, space objects are assigned a position in a two-dimensional area. Therefore, common visualizations for 2D maps are readily available in EnvSupport (e.g. continuous areas or discrete grids). In a so called *perspective*, the developer can assign a visual representation called *drawable* to each type of space object. A drawable may consist of an arbitrary number of drawing primitives (geometric shapes, external images, text), which can be further configured using properties of the space object (e.g. using different images according to the age of a plant). *Pre-* and *post-layers* can be added to a perspective to show the image of a map behind other drawables or to paint a grid on top of the visualization. Multiple perspectives can be defined for each application and each perspective can be used to create a visual representation of all objects or a selected subset according to *data views* defined in the domain. Current developments are directed towards extending the observer for incorporating also 3D visualizations based on the JMonkey engine.[7]

[7] http://jmonkeyengine.org/

3.2.4 Evaluation

The observer allows producing an intuitive and highly accessible way of understanding and analyzing application behavior. For the numerical analysis of simulations an evaluation component is provided. It allows keeping track of any space related information during application execution. Just like the observer, the evaluation component takes as input all space objects or a subset as defined in a data view and continuously extracts property values according to the specification of *data providers*. A data provider is a query producing a database table structure, i.e. for each time point of the application execution, the data provider takes the state of the environment and produces a row of data values extracting the relevant information from the space objects. The data is then used in *data consumers* that allow, e.g. writing it to a file for later off-line analysis or plotting it into a chart for real-time observation.

3.3 Agent-Based Simulation: City Bikes

Nowadays, bicycle sharing systems are deployed in many cities, allowing quick and easy 24/7 access to bikes for tourists, commuters, or any other person interested in using a bike for a short period of time. In these systems, the bikes can be checked out and returned at various stations in a more or less dense network of stations. E.g. on her journey to work a commuter can check out a bike near her home location and return it near her work place. An open problem in these systems is the distribution of bikes to the different stations. If too many bikes are at a station, no more bikes can be returned there, but if too few bikes are present, the station might run out of bikes. This problem is typically addressed using dispatchers, which transport bicycles between stations by van to establish a balanced distribution of bikes.

In the context of the *StadtRAD Hamburg*[8] system in Germany, an agent-based simulation model was built [8]. The model served the purpose to test and evaluate different scenarios to determine factors that influence the effectiveness and efficiency of the bicycle sharing system. Two concrete aspects were further investigated in the performed simulation studies. First, it was analyzed how the addition of new stations at certain places would affect the overall bicycle use. Second, several different strategies for dispatching were evaluated. Therefore, the goal of the model was to obtain realistic behavior for the bicycle usage that wasn't based on historical data, but would rather respond to the changes that were made to the environment for the different simulation studies.

3.3.1 Environment Model

As a virtual environment, the network of StadtRAD stations in Hamburg was modelled using EnvSupport. Besides the StadtRAD stations, also the

[8] http://stadtrad.hamburg.de

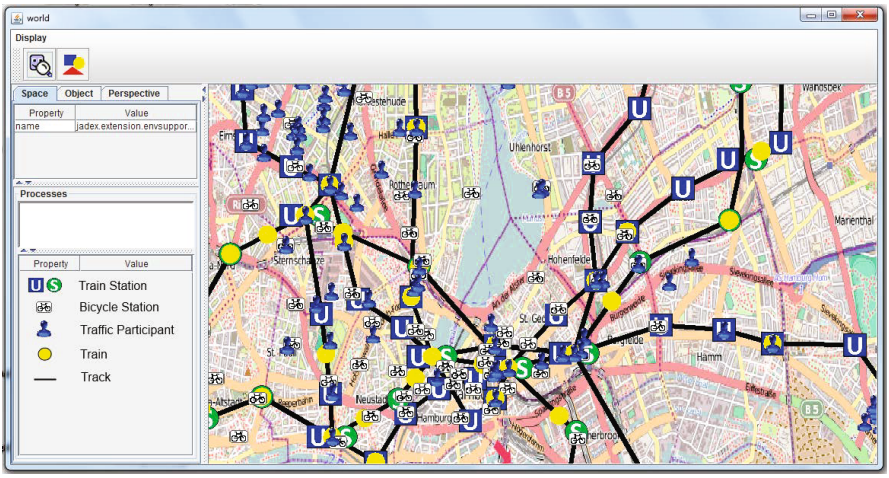

Fig. 6 Screenshot of the simulated StadtRAD environment

public transportation network was modelled, because it was considered that for long distances a combination of subway/urban train and bicycle would be preferred. The simulation model makes use of EnvSupport by defining the domain and interaction aspects in an XML description as follows.

The StadtRAD bicycle stations as well as train stations are represented as domain objects with a fixed location on the map. For the bicycle stations, the number of currently available bikes and the number of total slots[9] have been modelled as properties of the station object. Additionally, traffic participants are domain objects with a dynamic location, i.e. their location changes according to their travels. Each participant is assigned a random mobility value that influences how fast she can travel on foot or with a bike. Furthermore, train schedules have been modelled as domain processes, i.e. the processes encode the logic of moving traffic participants that board/unboard trains at certain locations. When executed, the simulated environment can be visualized as shown in Figure 6. For the visualization, drawable representations such as icon images are assigned to each of the modelled domain objects, such as train and bicycle stations as well as traffic participants.

The behavior of the traffic participants should be controlled by agents. Therefore an avatar mapping is defined that specifies the agent type corresponding to the traffic participant object. As a result, for each traffic participant in the simulation, a corresponding agent instance is automatically created. The agent may use declared actions to interact with the environment. Actions are modelled as Java classes and referred to from the XML environment description. The following actions have been defined in the StadtRAD

[9] At the time the model was built, StadtRAD did only support returning bikes at a station, when there was a free slot. This was changed recently, such that now bikes can also be returned when there are no free slots.

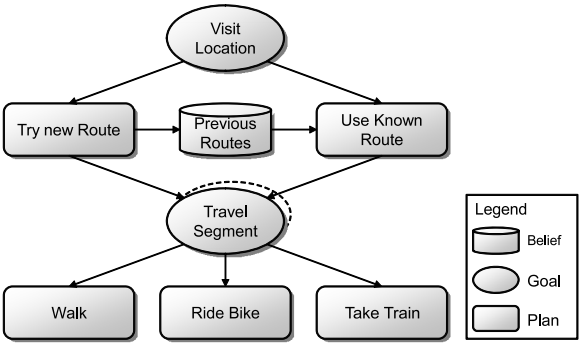

Fig. 7 Goal/plan tree of the traffic participant agent

model. First, the traffic participant may check out or return a bike, if her location matches the position of the station. Similarly, the participant can board/unboard trains at train stations. Finally, unless boarded on a train, a traffic participant can travel by herself to any chosen location, whereby the traveling speed depends on the participant's mobility value and if it currently has checked out a bike.

3.3.2 Agent Behavior

The aim of the simulation model is to achieve realistic bicycle usage behavior for being able to analyze the effect of changes to the StadtRAD system. Therefore, the traffic participants are represented as goal-directed agents that autonomously decide about if and how they would use a bike. The agents are created with a set of recurring goals to visit certain locations for leisure or commuting purposes. Following the BDI model, the agents perform a reactive goal/plan decomposition of their traveling activities (cf. Figure 7). The BDI reasoning starts from the top-level goal: *visit location*. For each target location, the agent decides to choose a previously used route (*plan: use known route*) or to try out a new route (*plan try new route*). For this purpose, the agent has knowledge about its recently travelled routes (*belief: previous routes*). Each route is a sequence of segments, i.e. intermediate locations and corresponding transportation means. As an example an agent might decide to switch to using a bike, instead of following a previous route that included a lot of changes between trains and waiting times. Afterwards the agent would remember the time taken for this new route and depending on its personal preferences, would possibly choose the new route again for future travels.

3.3.3 Simulation Studies

The behavior of the agents depends on their personal preferences, i.e. maximum distances that they would prefer walking, using a bike or taking the

train. Before running the simulation studies, the simulation model was calibrated to more closely match the real user behavior observed in the field. Therefore historical data of the StadtRAD system was compared to results of simulation runs and the parameter distributions for the agents were adapted until the behavior appeared sufficiently realistic. Afterwards the simulation studies were performed by altering the environment and observing how the bicycle usage changes.

The calibration as well as the studies themselves rely on the evaluation features of EnvSupport. By specifying data providers in the environment XML description, various results of the simulation (e.g. mean distance travelled, average number of bicycle checkouts per day) are automatically gathered during execution. Additionally specified data collectors consume the data and provide it to the developer, e.g. as graphical chart views, or export it to files for offline analysis.

The first study was a simple scenario analysis that investigated the effect of introducing an additional station in the StadtRAD network. The simulation allowed estimating the increase of bicycle usages that could be expected by introducing a new station at an important junction point with many train lines ("Schlump"). Most importantly, it could be verified, that the new station would not lead to significantly less bicycle use at other stations in the vicinity.

The second study was much more complex as it involved to comparison of different dispatching strategies. For this study an additional dispatcher agent was introduced, that performed a certain dispatching strategy by moving bicycles from overloaded stations to stations with few bicycles. Three strategies were analyzed that differed in the decision when to start moving bikes between stations. In the first strategy, the dispatcher would become active, when a station runs out of bikes. It would take a certain amount of bikes from the fullest station and transport it to the empty one. The second strategy is an adaptation of the first that introduced a threshold, i.e. already starting to transport bikes if their number drops below a certain value. The last strategy uses historic data of bicycle use and would transport bikes based on previously observed shortages (e.g. from the last day), regardless of the current situation. Simulation results showed that the threshold strategy performed best with respect to achieving the highest value of bicycle usage.

3.4 Summary

The EnvSupport extension allows defining the structure and behavior (domain) of an application environment in terms of objects as well as tasks and processes. Using avatars and actions, the interaction between the environment and the application agents can be clearly defined. This allows testing applications in virtual settings before real deployment. The visualization is further helpful for understanding application behavior either for teaching purposes or for debugging during application development. Furthermore, the

visualization may also be part of the application, e.g. for games or training applications. The evaluation module allows flexible measuring of application performance by observing interesting application values and producing various outputs, such as dynamically updated graphical charts or data files for off-line analysis. In this respect, the evaluation module can e.g. be used for benchmarking alternative implementations of application components.

EnvSupport is currently implemented for local simulations only, even though the principles behind it are general enough to be applied for distributed simulations as well. This requires allowing remote interactions of agents with the environment space. One simple solution to this problem is to create a service interface for the environment and let the agent hosting the environment expose a provided service that the participating agents use to interact with it. Furthermore, to allow also remote observers the world and visualization data of the environment need to made accessible per remote service as well. It has to be noted that such a simple solution may have performance problems due to the high amount of data that needs to be transferred between clients and environment. To avoid this, more advanced but also complex schemes have to be taken into consideration, e.g. by letting the clients perform partial calculation and rendering tasks on their own and synchronize with the environment only at specific rendezvous points.

4 Conclusion and Outlook

Simulation is a very interesting technique in combination with multi-agent systems. First, simulation studies may benefit from a multi-agent perspective as in scenarios with autonomous entities these can be adequately and individually modelled. Second, agent applications may profit from an upstream simulation analysis of specific application aspects before a real deployment is targeted. In the following the lessons learnt regarding simulation support for agent systems is summarized:

- A necessary key technique for supporting simulations is time control. It should be possible to choose the simulation mode that fits best to the simulation task to be performed, i.e. use real-time driven, time-driven or event-driven time advancement. For example if high efficient simulations are necessary due to long periods of time to be simulated or due to extraordinary complexity of the secario a fast-as-possible simulation execution is advantageous.
- An important part of simulations is the simulation environment, which is many cases requires much attention and effort to be built. For this reason, specific support for developing simulation environments should be available. Simulation environments are useful for several reasons. First, they allow describing the boundary of the system and thus its external interface. Second, the environment can help understanding if a system

works properly. Especially, visualizing the environment facilitates a better understanding of the system dynamics.

The guiding principle for simulation support consists in establishing simulation transparency, i.e. the application code should not be polluted with simulation specific aspects. This serves two purposes. On the one hand it enables code reuse, as the implementation of a simulation model that can later serve as basis for the implementation of the target system (with exception of the environment). Furthermore, if the simulation is used to test the system implementation, exact reuse of the code assures that no implementation details of a reimplementation, that would normally have to be created, cause malfunctions. On the other hand, no simulation specific programming language or environment needs to be learnt. Following this principle led to a non-invasive approach towards time as well as environment mechanism realization. Time control has been built in on infrastructure layer in order to hide timing aspects from the agents. The clock abstraction allows for keeping the agent code unaware of timing aspects. Different clocks are supplied which bring about the different simulation modes so that it can be determined at runtime if the application should be executed time-driven, event-driven or real time. Also environment support has been designed to be an optional part of applications. Therefore, the platform supports a general extension mechanism that allows for creating custom functionalities of an agent. The EnvSupport has been designed to follow this extension mechanism and offers its own description model. In general EnvSupport cleanly separates the domain model from its visualization in order to be able to create different views for one application.

References

1. Braubach, L., Pokahr, A., Lamersdorf, W., Krempels, K.-H., Woelk, P.-O.: A generic time management service for distributed multi-agent systems. Applied Artificial Intelligence 20(2-4), 229–249 (2006)
2. Collier, N.: RePast: An Extensible Framework for Agent Simulation. Working Paper, Social Science Research Computing, University of Chicago (2001)
3. Frey, D., Stockheim, T., Woelk, P.-O., Zimmermann, R.: Integrated Multi-agent-based Supply Chain Management. In: Proceedings of the 12th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 2003), pp. 24–29. IEEE Computer Society (2003)
4. Jander, K., Braubach, L., Pokahr, A.: Envsupport: A framework for developing virtual environments. In: Seventh International Workshop From Agent Theory to Agent Implementation (AT2AI-7), Austrian Society for Cybernetic Studies (2010)
5. Kirn, S., Heine, C., Herrler, R., Krempels, K.-H.: Agent Hospital - agent-based open framework for clinical applications. In: Kotsis, G., Reddy, S. (eds.) Proceedings of the 12th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 2003), pp. 36–41. CS Press (2003)

6. Klügl, F., Puppe, F.: The Multi-Agent Simulation Environment SeSAm. In: Kleine Büning, H. (ed.) Proceedings of SiWiS 1998: Simulation in Wissensbasierten Systemen, Technical Report tr-ri-98-194, Universität Paderborn (1998)
7. Pokahr, A., Braubach, L., Sudeikat, J., Renz, W., Lamersdorf, W.: Simulation and implementation of logistics systems based on agent technology. In: Blecker, T., Kersten, W., Gertz, C. (eds.) Hamburg International Conference on Logistics (HICL 2008): Logistics Networks and Nodes, pp. 291–308. Erich Schmidt Verlag (2008)
8. Reichelt, D.: Agentenbasierte Simulation von Fahrradverleihsystemen. Bachelorarbeit, Distributed Systems and Information Systems Group, Computer Science Department, University of Hamburg (December 2011) (in German)
9. Ricci, A., Viroli, M., Omicini, A.: The A&A Programming Model and Technology for Developing Agent Environments in MAS. In: Dastani, M.M., El Fallah Seghrouchni, A., Ricci, A., Winikoff, M. (eds.) ProMAS 2007. LNCS (LNAI), vol. 4908, pp. 89–106. Springer, Heidelberg (2008)
10. Sudeikat, J., Braubach, L., Pokahr, A., Renz, W., Lamersdorf, W.: Systematically engineering self–organizing systems: The sodekovs approach. In: Proceedings des Workshops über Selbstorganisierende, Adaptive, Kontextsensitive Verteilte Systeme (KIVS 2009), p. 12. Electronic Communications of the EASST (March 2009)
11. Sudeikat, J., Renz, W.: Building complex adaptive systems: On engineering self-organizing multi-agent systems. In: Hunter, G. (ed.) Strategic Information Systems: Concepts, Methodologies, Tools, and Applications, pp. 767–787. IGI Publishing (February 2010)
12. Warden, T., Porzel, R., Gehrke, J.D., Herzog, O., Langer, H., Malaka, R.: Towards ontology-based multiagent simulations: The plasma approach. In: Proceedings of the 24th European Conference on Modelling and Simulation (ECMS 2010), pp. 50–56 (2010)
13. Weyns, D., Omicini, A., Odell, J.: Environment as a first class abstraction in multiagent systems. Autonomous Agents and Multi-Agent Systems 14(1), 5–30 (2007)
14. Zöller, A., Braubach, L., Pokahr, A., Paulussen, T., Rothlauf, F., Lamersdorf, W., Heinzl, A.: Evaluation of a multi-agent system for hospital patient scheduling. International Transactions on Systems Science and Applications (ITSSA) 1, 375–380 (2006)

Chapter 6
Agents in Simulation of Cyberattacks to Evaluate Security of Critical Infrastructures

Rafał Leszczyna

Abstract. In the last years critical infrastructures have become highly dependent on the information technologies and exposed to cyberattacks. Because the effects of the attacks can be detrimental, it is crucial to comprehensively asses the security of the infrastructures' information systems. This chapter describes MAlSim – the simulator of malicious software based on software agents, developed for the needs of a testbed for critical infrastructures security. The authors explain the choice of agent paradigm for the development of the toolkit, present main design decisions, overview changes to the project introduced during the implementation, and provide the details of the completed project followed by a brief description of the application of MAlSim to security evaluation of a power plant. The chapter concludes with the discussion of the perspectives for the future of agent technology based on the experiences which came during the course of the project.

1 Introduction

1.1 Cybersecurity of Critical Infrastructures

Critical Infrastructure means those assets, systems which are essential for the maintenance of vital societal functions, health, safety, security, economic or social well-being of people, and the disruption or destruction of which would have a significant impact on citizens as a result of the failure to maintain those functions [5].

In the last decades the role of the Information and Communication Technologies (ICT) in the critical infrastructures, as in other enterprises and organisations, has significantly increased, very often gaining the leading position. The ICT support of contemporary infrastructures regards practically all business processes at any level

Rafał Leszczyna
Gdańsk University of Technology, Faculty of Management and Economics,
Narutowicza 11/12, Gdańsk, Poland
e-mail: rafal.leszczyna@pg.gda.pl

M. Ganzha & L. C. Jain (Eds.): Multiagent Systems & Applications, ISRL 45, pp. 129–146.
springerlink.com

of the organisational structure. Moreover in the critical infrastructures such as power plants, transportation systems, oil refineries, chemical factories or manufacturing facilities – where strategic processes are orientated towards production – the critical role in the operation of these facilities play dedicated systems called Industrial Control Systems (a.k.a. Process Control Systems) – ICS (PCS).

This intensification of the use of ICT in the modern critical infrastructures resulted in numerous invaluable advantages, such as reduction of costs and enhanced functionality but in return it led to increased exposure to computer network-based attacks.

In the last years critical infrastructures experienced a notable number of ICT incidents, including the large-scale cyberattacks targeting Estonia, Lithuania and Georgia in 2007 and 2008. Additionally, the manifestation of Stuxnet (July, 2010) raised a lot of concerns and discussions due to the specific nature of the attack[1].

As the potential effects of cyberattacks to critical infrastructures may be very detrimental, it is crucial to comprehensively evaluate the ICT security of the infrastructures.

1.2 Cybersecurity Evaluation

There are various approaches to ICT security evaluation, spanning from model-based evaluation techniques, through expert analyses to penetration testing and emulation [44, 20, 39, 34, 33].

In the *model-based approach*, a model of the analysed system is developed and the security analyses are conducted in regard to this model [39]. This approach aims at achieving complete and well-structured results, a 'promise' that if the verification provides a positive result, then it means that the system is really secure. Unfortunately the completeness of the analysis is dependent on the complexity of the model. In order to obtain full guaranties, one would have to develop a model which reflects all the states of the analysed system, which is obviously either infeasible or too costly. In fact, it has been happening very often in the past that the models were lacking the representation of important (– as it resulted only after) parts of the reality. This finished in incidents not detected during the analysis phase. There are multiple techniques utilised in this type of evaluation: combinatorial methods, model-checking, state-based stochastic or simulation and others [34].

Expert analyses are based on the thorough analysis of the documentation, both – provided by the operators and gathered individually in the interviewing process. Their completeness and results depend on the thoroughness of performing them analysts and achieving good result is very resource consuming by means of time or personnel. The experts usually support themselves by security evaluation criteria [44] or the lists of required security components [17, 32].

[1] The analyses of Stuxnet revealed that this is a dedicated attack targeting a specific industrial control system likely in Iran.

Penetration testing, being the *attempting to compromise the network of a company for the purpose of assessing its data security* [47] should provide the most accurate results in the shortest time, but the most serious drawback is that the results of such testing may go in an unpredicted direction. In the most ironic scenario the experiment which aimed at improving security of the critical infrastructure may result in its serious breach. For this reason it is advised not to perform penetration testing directly on site as the events may spun out of control. Instead, it is recommended to perform simulations in an isolated environment, or the penetration test in the reconstructed system (a copy of the original, situated in a secure perimeter). These kinds of configurations are called *testbeds* [20], while the process of imitating the original system based on the their hardware and software is often called *emulation*.

Simulation testing, where the analysed system and the attacks are simulated [33], while still fast and effective, does not suffer from the above mentioned disadvantage of posing the potential threat to the evaluated system, but similarly as in the model-based approach (in fact this is a 'specie' of the model-based approach [34]) it is dependent on the quality of the model. Moreover, the model must be implemented in the form of a simulation environment.

1.3 Need for a Malware Simulator

In our work in the laboratory for the security of Critical Networked Infrastructures (CNI) we have been focusing on the simulation and emulation direction. We have developed a CNI security testbed based on one hundred twenty hosts, the required network equipment (which includes sixteen network switches), as well as some ICS devices [24, 12]. In the testbed we simulate attacks against ICT systems of critical infrastructures in order to detect any vulnerabilities in the systems and consequently to evaluate the security of the systems.

Among the computer attacks, malware attacks belong to the most frequent. In 2010 Symantec registered over 3 milliards malware attacks (including the outstanding Stuxnet) [42]. *Malware* is a malicious software that runs on a computer and makes the system behaving in a way wanted by an attacker [40].

When we wanted to simulate these attacks in our laboratory, we encountered the problem of lack of malware simulation. According to our research, such simulators, enabling simulations of malware in an arbitrary, real, physical network of computers didn't exist.

We were looking for a simulator which would run 'on the top' of the analysed system and interact directly with it. Ideally it should not require any support of an intermediate layer of software such as middleware. We expected that it could simulate various types of malware (worms, viruses, malicious mobile code etc.). It should imitate the behaviour of the malware and all its interactions with the system with

very high fidelity. Yet it should prevent any unwanted effects of these interactions. The simulator should support simulations of attacks on various types of hardware platforms and operating systems, as the critical infrastructures' systems and ICS are very diverse.

At the moment the only available virus simulators were either educational/demonstrative [14], such as Joe Hirst's Virus Simulation Suite [15] or Virlab [9], or dedicated for testing of anti-virus software – Rosenthal Virus Simulator [37]. Apart from being not particularly suitable for our purposes, already at the time the solutions were outdated.

As far as the simulation of worms was concerned, the prevalent work was done on developing mathematical models of worm propagation [38, 43, 7, 50], which base on epidemiological equations that describe spread of real-world diseases. The empirical approaches focused on single-node worm spread simulators [27, 26, 45, 31], which are dedicated to run on one machine. Only few distributed worm simulations were implemented [36, 46, 11]. Still, all these tools aimed at simulation of the whole attack context i.e. the malware as well as the system, and the network where the attack takes place.

Also Trojan Simulator [30] had limited applicability as it was dedicated for the evaluating effectiveness of anti-Trojan software.

It became evident that if we wanted to simulate the malware attacks in our testbed, we needed to develop our proprietary tool.

2 Developing MAlSim

2.1 The Choice of Agent Paradigm

We chose the mobile agents paradigm for the development of the simulator because:

- It supports software mobility (thus should facilitate simulation of malware, the software which is inherently – mobile).
- The Agent Platform has all the characteristics of an isolated environment[2] facilitating safe execution of the programs and preventing unwanted effects to impact the system.
- The Agent Platform with the Agent Management System provide the embedded functions which could be used for facilitating the control over the experiments.
- Most of the agent environments are written in Java, thus per se they offer portability of the software, which was important to us because various system technologies are used in critical infrastructures.

[2] In the information security terminology, this kind of environment, which introduces isolation of the executed software from an operating system and other applications is called a *sandbox*.

Moreover not without an influence was the fact that we had a positive experience with developing agent-based software. Agent systems had allowed us for quickly developing and deploying middle-complexity applications in the past.

2.2 MAlSim Design

During the design phase of MAlSim, first of all we had to choose the agent platform to be used as the middleware and the agent programming environment. For the reasons which follow, the obvious choice for us was JADE. First of all, it is a distributed agent platform written in Java, so it satisfies our requirement for the portability of the environment, and provides the necessary means for facilitating the deployment of applications on various hosts running diverse operating systems. As a result MAlSim can be easily deployed over the hosts participating in the experiments. Second, JADE enables mobility of agents, thus another requirement (see the previous section) for our simulation software was met, namely that it should support mobility of software – since most of malware is able to move across various platforms.

Another factor playing role in this choice was that we already had experience in the development in JADE and we were very familiar with it. If we chose another environment we would have to learn it first. Moreover the advantage of JADE was that there was a relatively big and active community of the developers. Anyone looking for an advice regarding programming JADE applications could count on a very timely response from other developers but also from the authors of the Framework.

Regarding the deployment of MAlSim, we planned to take advantage of the standard means of the application deployment of JADE, it means JADE containers. Similarly, in relation to the control over the experiments, at the initial stages of the project we wanted to utilize JADE functionalities, with the JADE Graphical User Interface in the first place.

When starting the MAlSim project, we already had previous experiences with agent systems, including the above mentioned experience with the development in JADE. This experience, among the others, provided us with the good recognition of the current status of agent systems, the level of maturity of agent development environments, or the boundaries of the the development possibilities. As we already said – this was also not our first application developed for JADE and we were aware of the advantages and the limitations of this framework. Thus the design of MAlSim already took into account these characteristics. For example the fact that all agent applications require the presence of the underlying layer of middleware, forced us to accept that the simulator would run only in this layer and that all direct interactions with the system would be limited. In the context of malware simulations, this posed a significant limitation, but on the other hand, from the point of view of security – this additional layer would create a form of a sandbox – an isolation layer, which renders that any unexpected detrimental effects remain only in this layer and won't affect the system. In the end, when we evaluated all the pros and cons, the pros were in prevalence.

2.3 *Design Changes during the Implementation*

However ultimately during the implementation phase of the project we had to verify our last assumption. It means the one regarding the 'obligation' of running MAlSim only at the level of middleware.

We had to change our approach because the first experiments with MAlSim operating only at the level of JADE were too unrealistic. The middleware layer prevented all direct interactions of the simulated malware with the system. Because these interactions are indispensable for the proper reconstruction of attack scripts, we couldn't reproduce them completely. As the result, the effectiveness of the security evaluation decreased.

As we wrote in the previous section, this characteristic of JADE was known to us before, but only the life experience showed how significant impact it had on the security evaluations. In fact, the evaluations with such reduced efficiency had very limited applicability.

In consequence, because these direct interactions between the simulator and the system are indispensable for realistically reconstructing the attacks to thoroughly evaluate the security of the system, we had to introduce additional, external software modules to our simulation toolkit on the level of the operating system layer (see Figure 1). At this stage of the project we took advantage of the system commands and applications. For instance, to disable a network adapter (when simulating its damage), on Windows we applied a Visual Basic Script, on Linux a shell script. In the future, these external modules will be a dedicated code, written specifically for the demands of the simulation.

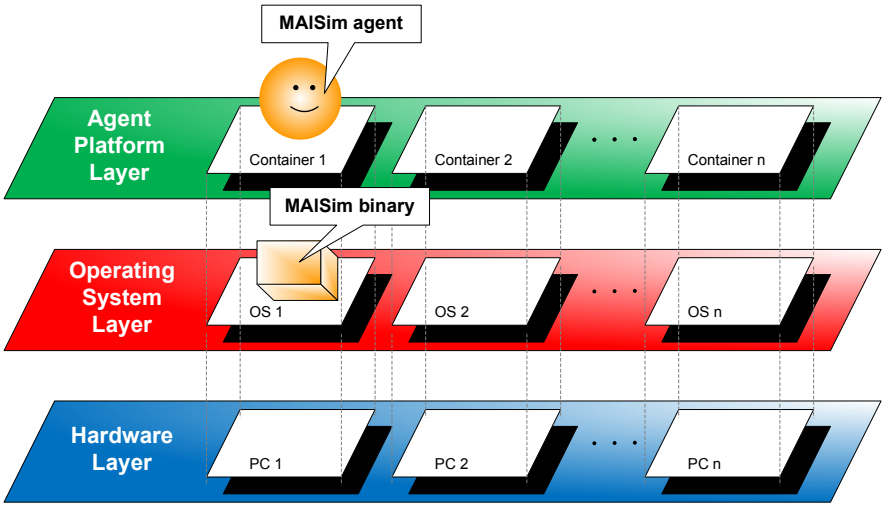

Fig. 1 For the interactions with an operating system MAlSim requires external modules

3 Completed Project

MAlSim consists of software package called *MAlSim Toolkit* and *malware templates* chosen in reference to an applicable *attack scenario*.

3.1 Attack Scenario

An attack scenario is a description of a series of actions and events occurring during an attack. It is written for all the parties involved in the attack (the attacker, the victims, the third parties).

An exemplary attack scenario based on a variant of the W32/Mydoom worm (see Section 3.2) is as follows:

> An employee working in the administrative section of a power plant receives an e-mail informing about a failure of a delivery of 'his' message. For further details he is directed to an attached file. Following this indication, the employee opens the attachment and in this way allows the variation of W32/Mydoom worm to infect his computer. In similar way the worm infects also other computers in the administrative section of the power plant.
>
> Later on another administrative employee wants to verify some data regarding the energy production process using the monitoring system running on the server in the process control area of the power plant. Thus, unaware of the fact that his PC is infected by the malware, opens the VPN connection to a host in process control network. In this moment the worm has an open passage to the critical part of the power plant network. It moves through it and starts infecting the computers in the process control network.
>
> On the fixed date, the worm will launch a Distributed Denial of Service (DDoS)[3] attack against the process control server from all the infected computers.

3.2 Malware Templates

A *malware template* is, as the name indicates, a model of a simulated malware. It specifies the components necessary to simulate particular malicious software and its behaviour.

A fragment of a malware template for W32/Mydoom worm is presented in Listing 1. This virus represents the group of malicious software which create backdoors and perform Distributed Denial of Service Attacks (see the footnote). The template was created based on the descriptions from [8, 41, 29]. The relevant class diagram and sequence diagram are presented in Figures 2 and 3.

[3] A Distributed Denial of Service attack is a distributed version of the Denial of Service Attack (DoS), which is based on the excessive consumption of the resources of the attacked server via sending a large amount of service requests to the server which will result in its inability to provide the service. Since protection methods against the DoS had been developed, attackers introduced a distributed version of DoS in which the requests of service are sent from separate hosts. This is much more difficult to protect, as it is difficult to distinguish malicious service requests from legitimate ones.

Listing 1 Pseudocode of the malware template for simulation of the worm W32/Mydoom.

```
Initial event: Sending e-mail with a malicious attachment.

Trigger: Opening the attachment.

Action 1: Propagating to other computers.

 1. CONNECT(MAlSim)
 2. IF system.date > (stopSpreadingDate) THEN END // propagating only till the
    date indicated within the constant stopSpreadingDate
 3. NEW eMailAddresses[ ] // creating new array in which addresses collected
    from Windows Address Book and local files will be stored
 4. CREATE_FILE("java.exe", windowsFolder)
 5. CREATE_FILE("services.exe", windowsFolder)
 6. "HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run" →"JavaVM"
    = windowsFolder+"\java.exe"
 7. "HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run"
    →"Services" = windowsFolder+"\services.exe"
 8. "HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run" →"JavaVM"
    = windowsFolder+"\java.exe"
 9. "HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run"
    →"Services" = windowsFolder+"\services.exe"
10. REG_CREATE("HKEY_CURRENT_USER\Software\Microsoft\Daemon")
11. REG_CREATE("HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Daemon")
12. c=0
13. WHILE (windowsAddressBook.GET_NEXT(contact) NOT EQUALS NULL)
    address[c++] = contact
    // collecting email addresses from the Windows Address Book
14. fileExtensions = NEW ({".pl*", ".ph*", ".tx*", ".ht*", ".asp", ".sht",
    ".adb", ".dbx", ".wab"})
15. GetAddressesFromFiles(fileExtensions)
    // collecting email addresses from files with particular extensions
16. eMailMessage.to = eMailAddresses
17. messageSenders = NEW ({"Postmaster", "Mail Administrator", "Automatic Email
    Delivery Software", "Post Office", "The Post Office", "Bounced mail",
    "Returned mail", "MAILER-DAEMON", "Mail Delivery Subsystem"})
18. eMailMessage.from = messageSenders[RANDOM(messageSenders.length)]
19. messageSubjects = NEW ({"New Graphic Site", "hello", "hi", "error",
    "status", "test", "report", "delivery failed", "Message could not be
    delivered", "Mail System Error - Returned Mail", "Delivery reports about
    your e-mail", "Returned mail: see transcript for details", "Returned mail:
    Data format error delivered"})
20. eMailMessage.subject = messageSubjects[RANDOM(messageSubjects.length)]
21. eMailMessage.body = "Your message was not delivered due to the following
    reason(s): Your message was not delivered because the destination server
    was unreachable within the allowed queue period. The amount of time a
    message is queued before it is returned depends on local configuration
    parameters. Most likely there is a network problem that prevented delivery,
    but it is also possible that the computer is turned off, or does not have a
    mail system running right now."

22. attachmentNamePrefixes = NEW ({"ATTACHMENT", "DOCUMENT", "FILE",
    "INSTRUCTION", "LETTER", "MAIL", "MESSAGE", "README", "TEXT",
    "TRANSCRIPT"})
23. attachmentNameSuffixes = NEW ({".bat", ".cmd", ".com", ".exe", ".pif",
    ".scr", ".zip"})
24. attachmentName = attachmentNamePrefixes[RANDOM(attachmentNamePrefixes.
    length)] + attachmentNameSuffixes[RANDOM(attachmentNameSuffixes.length)]
25. eMailMessage.attachments[0] = NEW_FILE(this, attachmentName)
26. SEND(newEMailMessage)

Action 2: Setting backdoor access to the computer.

1. OPEN_TCP_PORT(3127)
```

```
2. OPEN_TCP_PORT(3198) // opening TCP ports in order to allow the attacker to
   remotely access the infected computer
3. CONNECT(attackersSite)
   // the constant attackersSite contains the address of the attacker's network
   location
4. DOWNLOAD(attackersProgram)
   // the constant attackersProgram indicates the name of the program located
   on the attacker's location
5. EXECUTE(attackersProgram)
   // executing the downloaded program
6. INFORM(MAlSim)
```

Action 3: Performing Distributed Denial of Service (DDOS) Attack.

```
1. INFORM(MAlSim)
2. IF system.date NOT EQUALS (launchDDOSDate) THEN END // launching the attack
   on the date indicated in the constant launchDDOSDate
3. CREATE_HTTP_GET_REQUEST(httpGetRequest)
4. FOR {c=0; c≤64; c++} SEND(httpGetRequest)
5. WAIT(1000) // wait 1 second (1000 milliseconds)
6. GO TO 2
```

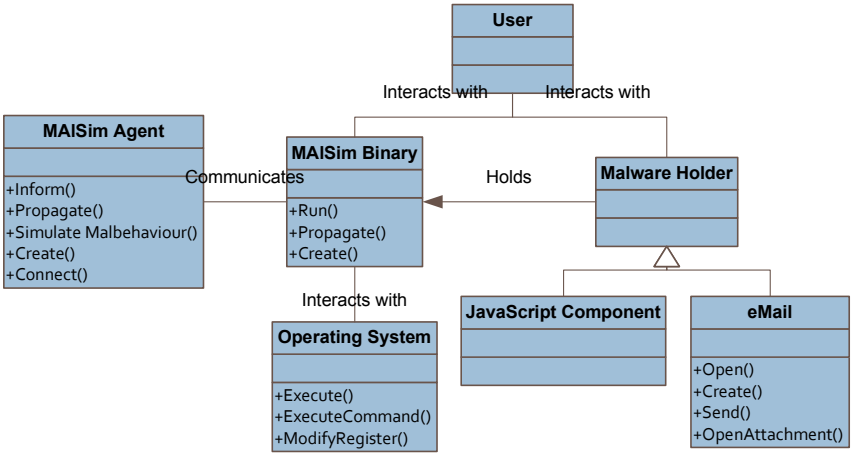

Fig. 2 Class diagram for the W32/Mydoom malware template

We have developed malware templates for viruses and worms: Melissa, Yamanner, W32/Mydoom and W32/Blaster. During the development, various information sources were used. The most popular included [8, 41, 29]. The particular choice of malware was dictated by their 'popularity' (they infected thousands of computers) as well as interesting techniques of propagation and the payload, which was later also used by other malware.

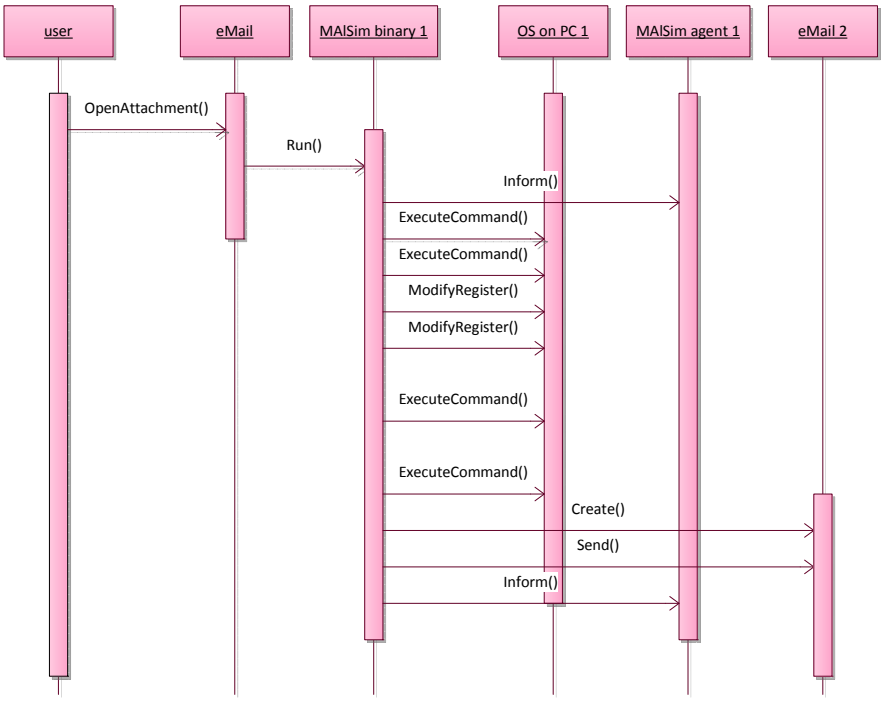

Fig. 3 Sequence diagram illustrating the Action 1: Propagating to other computers of the malware template for W32/Mydoom

As it can be seen on the example of the W32 Blaster template, each template defines:

- *Initial event* of the malware life cycle – a 'birth' of malware.
- *Trigger* – the initial conditions necessary to be satisfied in order to allow the malware to start operate.
- *Malicious actions* – of the simulated malware.

These definitions drive the development of the code of MAlSim agent classes and agent behaviour classes included in the MAlSim Toolkit.

For more details on the development of malware templates see [23, 24].

3.3 MAlSim Toolkit

The software (agent) part of the result of the project is a software package called *MAlSim Toolkit*. MAlSim Toolkit was developed in Java and dedicated for JADE [1].

MAlSim Toolkit consists of:

- Several (Java) classes of MAlSim agent (extensions of JADE `Agent` class).
- Various behavioural patterns implemented as agent behaviours (extensions of JADE `Behaviour` class).
- Diverse migration/replication patterns implemented as agent behaviours (extensions of JADE `Behaviour` class).
- Auxiliary classes for the setup and control of the experiments.

The MAlSim agent class is the basic agent code which implements the standard agent functionalities related to its management on the agent platform, its communication skills and the characteristics related to the nature of simulated malicious software. This code will be propagated across the attacked machines.

The behavioural patterns define agent behaviours aiming at imitating malicious activities of malware but *without their harmful impact*. The patterns include operations such as disabling network adapter, enabling a local firewall to operate in all-block mode or starting a task which extensively consumes processor time etc. They facilitate showing detrimental effects of malware activities but in contrary to their prototypes they are fully controlled. Listing 2 presents the code for imitating the corruption of a network adapter in the behaviour class of a MAlSim agent used in a zero-day virus simulation.

Migration and replication patterns describe the ways in which MAlSim agent migrates across the attacked hosts. The patterns implement malware propagation models as well as user-configured propagation schemas.

The Toolkit includes also auxiliary classes which facilitate setting up and controlling the experiments. An exemplary code of a `StartUp` class used for the setup of a zero-day virus simulation is presented in Listing 3.

Further details can be found in [23].

Listing 2 Java code of `disableNetworkAdapter` method used by MAlSim agents in a zero-day virus simulation.

```
private void disableNetworkAdapter() {
    String os_name = System.getProperty("os.name");
    if (os_name.toLowerCase().lastIndexOf("linux") != -1)
        try { // linux
            String line;
            String cmd = "ifdown eth0";
            Process p = java.lang.Runtime.getRuntime().exec(
                        cmd);
            BufferedReader input = new BufferedReader(
                new InputStreamReader(p.getInputStream()));
            while ((line = input.readLine()) != null) {
                System.out.println(line);
            }
            input.close();
```

```
        } catch (Exception err) {
            err.printStackTrace();
        }
    else // windows
        try {
            String line;
            String command = "cmd /c start DisabilitaLAN.vbs";
            System.out.println(command);
            Process p = java.lang.Runtime.getRuntime().exec(command);
            BufferedReader input = new BufferedReader(
                new InputStreamReader(p.getInputStream()));
            while ((line = input.readLine()) != null) {
                System.out.println(line);
            }
            input.close();
        } catch (Exception err) {
            err.printStackTrace();
        }
}
```

Listing 3 Java code of `StartUp` class used for the setup of a zero-day virus simulation.

```
public class StartUp {
  public static void main(String[] args) {
    String[] defaultArgs = {"-gui", "-detect-main", "false"};
    Boot.main(defaultArgs);
    try {
      Runtime rt = Runtime.instance();
      Profile p;
      AgentContainer ac = null;
      rt.setCloseVM(true);
      String[] containerNames = {"power-plant-pc-1-100",
                                 "power-plant-pc-1-103",
                                 "power-plant-pc-1-104",
       "power-plant-ss-s-030","power-plant-ss-s-031","power-plant-ss-s-032","power-plant-ss-s-035"};
      Object agentArgs[] = {};
      p = new ProfileImpl();
      p.setParameter(Profile.CONTAINER_NAME,
```

```
"power-plant-ts-
                    1-100");
      ac = rt.createAgentContainer(p);
      AgentController mSA = ac.createNewAgent("MalSim",
              "MalwareSimAgent3", agentArgs);
      for (int l=0; l<containerNames.length; l++) {
          p = new ProfileImpl();
          p.setParameter(Profile.CONTAINER_NAME,
                         containerNames[l]);
          ac = rt.createAgentContainer(p);
      }
      mSA.start();
    }
    catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

3.4 *The Life Cycle of the Experiments with MAlSim*

During an experiment, the above described elements go through the following life cycle:

1. Choosing an appropriate *attack scenario*.
2. Selecting a malware template relevant to the scenario (or the development of a new one).
3. Creating a live instance of malware template via MAlSim Toolkit.
4. Performing the experiment and registering its outcomes (all – the intermediate and final).

At the current stage of the project the first three activities are performed manually. The control over the experiments is facilitated by the graphical interface of JADE.

4 Application

MAlSim was applied to the experiments aiming at evaluation of the information security of an existent, fully operative combined cycle electric power plant.

In order to avoid any undesired effects of the simulations, all the evaluations were performed in our Critical Networked Infrastructures security testbed [24, 12] where we reconstructed the hosts and the networks of the power plant with a very high fidelity. All the crucial hardware components were copied, and only the stations

having minor role in the attacks were virtualised. The same software was installed, and the same configurations applied[4].

In this testbed MAlSim was deployed via JADE containers. On each host one container was installed. The control over experiments was effectuated from the JADE main-container, which was located in the dedicated area of the testbed (so called *Threat and Attack Simulator*). From there, the simulated attacks were launched, controlled and monitored.

The simulation led to particular conclusions regarding the security of the evaluated power plant's information system. Apart of the observations regarding the resistance of the system components to cyberattacks, a very important outcome of the experiments was the awareness raising effect obtained by the occasion: after the evaluation we invited the personnel of the power plant to the demonstration of our simulations. The staff could see in person the course of the cyberattack and the damages it caused. And for some of them it was the first occasion to realise the severity of the potential effects of cyberattacks, the importance of complying with the security policy and putting in place the security measures.

For further details see [23].

5 Lessons Learned

When we are looking back at the project now from the perspective of it being finished, we would not change too much the way it was conducted. As we wrote in Section 2.2, the design of MAlSim took into account the characteristics of agent systems and agent programming environments, their strengths and limitations.

The only design decision which we had to revise during the implementation phase was the one regarding MAlSim's operation limited to the middleware layer (see Section 2.3). If we had known before what we learned during the course of the project, we would have incorporated operating system modules in MAlSim from the beginning.

Without them, the introductory experiments with MAlSim executed only under JADE were too unrealistic. The operation limited solely to the middleware layer prevented all direct interactions of the simulated malware with the system, which are indispensable for the proper reconstruction of attack scripts. This led to the decrease in the efficiency of the security evaluations, causing their limited applicability.

6 Perspectives

Our experiences with the development of MAlSim led us to the conclusion that future agent environments should be more integrated with operating systems. Agent must either run directly in the layer of an operating system or if being executed indirectly on middleware – then the intermediate layer must be tightly embedded into the operating system.

[4] For more details of the environment and the evaluation approach see [24, 12, 22, 21].

The security challenge inherent to this integration of agent paradigm [10, 18, 28, 25] would have to be addressed via implementing the security measures proposed by the researchers[5]. It has been agreed that the problem of protecting agent platforms has been addressed sufficiently, it means effective solutions were designed [18, 35, 4]. Also for the protection of agents various techniques have been proposed (for example: [6, 3, 13, 49, 19, 16]), but this area still requires further developments. A promising direction is demarcated by the application of Trusted Computing Modules [48]. In addition to that it must be noted that the protection of agents from malicious platforms is not a new problem to be introduced by the integration of agent platforms with operating systems. Agent systems have been suffering from this issue from the beginning, and it was agreed to be one of the main obstacles preventing the popularisation of agents [10, 18, 28]. Though, to the contrary, the integration of agent platforms with operating system would bring proper attention to the issue, which would significantly increase the chances for it to be resolved.

Depending on security settings controlled by users and administrators, the agent environments should enable flexible control of agents' access to the resources of the platform. In our case, this would allow us for developing an application based exclusively on agents, without the need for creating and executing external programs.

Another reflection is that in general programming environments (Integrated Development Environments – IDEs) used for the development of agents could be more 'agent-friendly' or 'agent-enabled', as for now, the developers of agent applications can use only general purpose programming environments (such as Eclipse, JBuilder).

Features which provide particular support for the agent paradigm and facilitate the development and deployment of agent applications should be introduced to the programming environments. This could be done via extensions to the existent, general purpose environments (our preferred way), or through developing new environments – agents-dedicated. Interesting discussion on this subject can be found in [2]. As it can be seen, the subject is not new.

Finally, our general observation is that agent platforms still leave space for improvements regarding the stability of code execution, error-freeness, the number and level of offered supportive functions, maturity of graphical interface and so on. Improving these aspects should result in higher attention to agent systems and the increased number of successful applications. This would consequently lead to more effort and resources invested into improvement of agent environments, resulting in the advance of their quality, which would impact the number of applications... In our opinion this is a recurring cycle which waits for being properly triggered.

[5] An overview of the security solutions for agents can be found in [25].

References

1. Bellifemine, F.L., Caire, G., Greenwood, D.: Developing Multi-Agent Systems with JADE. Wiley (2007)
2. Bryson, J., Decker, K., Deloach, S.A., Huhns, M., Wooldridge, M.: Panel Summary: Agent Development Tools. In: Castelfranchi, C., Lespérance, Y. (eds.) ATAL 2000. LNCS (LNAI), vol. 1986, pp. 331–338. Springer, Heidelberg (2001)
3. Ceccato, M., Tonella, P., Preda, M.D., Majumdar, A.: Remote software protection by orthogonal client replacement. In: Proceedings of the 2009 ACM Symposium on Applied Computing, SAC 2009, pp. 448–455. ACM, New York (2009)
4. Chess, D., Grosof, B., Harrison, C., Levine, D., Parris, C., Tsudik, G.: Itinerant agents for mobile computing. IEEE Personal Communications 2(5), 34–49 (1995), citeseer.ist.psu.edu/article/chess95itinerant.html
5. Commission, E.: COM(2008) 676 final, proposal for a council decision on a Critical Infrastructure Warning Information Network (CIWIN). Internet (2008)
6. Desnitsky, V., Kotenko, I.: Security and Scalability of Remote Entrusting Protection. In: Kotenko, I., Skormin, V. (eds.) MMM-ACNS 2010. LNCS, vol. 6258, pp. 298–306. Springer, Heidelberg (2010), http://portal.acm.org/citation.cfm?id=1885194.1885223
7. Ellis, D.: Worm anatomy and model. In: WORM 2003: Proceedings of the 2003 ACM Workshop on Rapid Malcode, pp. 42–50. ACM, New York (2003)
8. F-Secure: F-Secure virus description database. Website (2008), http://www.f-secure.com/v-descs/ (last access: January 18, 2008)
9. Faistenhammer, T., Klöck, M., Klotz, K., Krüger, T., Reinisch, P., Wagner, J.: Virlab 2.1. Internet (1993), http://kklotz.de/html/virlab.html (last access: October 29, 2007)
10. Farmer, W.M., Guttman, J.D., Swarup, V.: Security for mobile agents: Issues and requirements (1996), http://gunther.smeal.psu.edu/farmer96security.html
11. Filiol, É.: Franc, E., Gubbioli, A., Moquet, B., Roblot, G.: Combinatorial optimisation of worm propagation on an unknown network. International Journal in Computer Science 2(2), 124 – 131 (2007), http://vx.netlux.org (last access: March 7, 2008)
12. Fovino, I.N., Masera, M., Leszczyna, R.: Security Assessment of a Turbo-Gas Power Plant. In: Critical Infrastructure Protection, pp. 31–40. Springer (2009), http://www.springerlink.com/content/k0137022kw265n08
13. Godoy, G., Tiwari, A.: Invariant Checking for Programs with Procedure Calls. In: Palsberg, J., Su, Z. (eds.) SAS 2009. LNCS, vol. 5673, pp. 326–342. Springer, Heidelberg (2009), http://dx.doi.org/10.1007/978-3-642-03237-0_22
14. Gordon, S.: Are good virus simulators still a bad idea? Network Security 1996(9), 7–13 (1996)
15. Hirst, J.: Virus simulation suite. Internet (1990)
16. Hohl, F.: Time Limited Blackbox Security: Protecting Mobile Agents From Malicious Hosts. In: Vigna, G. (ed.) Mobile Agents and Security. LNCS, vol. 1419, pp. 92–113. Springer, Heidelberg (1998), citeseer.ist.psu.edu/hohl98time.html (last access: May 10, 2006)
17. ISO/IEC: ISO/IEC 27001: 2005(E): Information technology – Security techniques – Information security management systems – Requirements. U.S. Government Printing Office (2005)
18. Jansen, W., Karygiannis, T.: NIST special publication 800-19 - mobile agent security (2000), http://citeseer.ist.psu.edu/jansen00nist.html

19. Karjoth, G., Asokan, N., Gülcü, C.: Protecting the Computation Results of Free-Roaming Agents. In: Rothermel, K., Hohl, F. (eds.) MA 1998. LNCS, vol. 1477, pp. 195–207. Springer, Heidelberg (1998)
20. Leeuwen, B.V., Urias, V., Eldridge, J., Villamarin, C., Olsberg, R.: Cyber security analysis testbed: Combining real, emulation, and simulation. In: Proceedings of the 2010 IEEE International Carnahan Conference on Security Technology (ICCST), pp. 121–126 (2010)
21. Leszczyna, R., Fovino, I.N., Masera, M.: MAlSim – mobile agent malware simulator. In: Proceedings of the First International Conference on Simulation Tools and Techniques for Communications, Networks and Systems (SIMUTools 2008), ICST, France (2008)
22. Leszczyna, R., Fovino, I.N., Masera, M.: Security evaluation of IT systems underlying critical networked infrastructures. In: Proceedings of the First International IEEE Conference on Information Technology (IT 2008), IEEE, Gdansk University of Technology, Gdańsk, Poland (2008)
23. Leszczyna, R., Fovino, I.N., Masera, M.: Simulating malware with MAlSim. Journal in Computer Virology (2008), http://www.springerlink.com/content/k0843hgq60333556 (last access: September 24, 2012)
24. Leszczyna, R., Fovino, I.N., Masera, M.: An approach to security assessment of critical infrastructures' information systems. IET Information Security 5, 135–144 (2011)
25. Leszczyna, R., Kotenko, I.: Security and Anonymity in Agent Systems. In: Essaaidi, M., Ganzha, M., Paprzycki, M. (eds.) Software Agents, Agent Systems and Their Applications, Sub-Series D: Information and Communication Security, vol. 32, pp. 260–285. IOS Press, Amsterdam (2012)
26. Liljenstam, M., Nicol, D.M., Berk, V.H., Gray, R.S.: Simulating realistic network worm traffic for worm warning system design and testing. In: WORM 2003: Proceedings of the 2003 ACM Workshop on Rapid Malcode, pp. 24–33 (2003)
27. Liljenstam, M., Yuan, Y., Premore, B., Nicol, D.: A mixed abstraction level simulation model of large-scale internet worm infestations. In: Proceedings of the 10th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS 2002), p. 109. IEEE Computer Society, Washington, DC (2002)
28. Luck, M., McBurney, P., Preist, C.: Agent Technology: Enabling Next Generation Computing (A Roadmap for Agent Based Computing). AgentLink (2003)
29. McAfee: McAfee virus information. Website (2008), http://uk.mcafee.com/virusInfo/ (last access: January 18, 2008)
30. Mischel Internet Security: Trojan simulator. Internet (2003), http://www.misec.net/trojansimulator/ (last access: October 29, 2007)
31. Moore, D., Shannon, C., Voelker, G.M., Savage, S.: Internet quarantine: Requirements for containing self-propagating code. In: NFOCOM 2003, Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies, vol. 3, pp. 1901–1910 (2003)
32. National Institute of Standards and Technology (NIST): DRAFT Recommended Security Controls for Federal Information Systems and Organizations. National Institute of Standards and Technology (NIST) Special Publication 800-53 Rev. 3. U.S. Government Printing Office (2009)
33. Nicol, D.M.: Modeling and simulation in security evaluation. IEEE Security and Privacy 3, 71–74 (2005), doi:10.1109/MSP.2005.129

34. Nicol, D.M., Sanders, W.H., Trivedi, K.S.: Model-based evaluation: From dependability to security. IEEE Trans. Dependable Secur. Comput. 1, 48–65 (2004), doi: http://dx.doi.org/10.1109/TDSC.2004.11
35. Ordille, J.J.: When agents roam, who can you trust? In: First Conference on Emerging Technologies and Applications in Communications (etaCOM), Portland, OR, March 24 (1996), citeseer.ist.psu.edu/ordille96when.html (last access: March 24, 2006)
36. Perumalla, K.S., Sundaragopalan, S.: High-fidelity modeling of computer network worms. acsac 00, 126–135 (2004)
37. Rosenthal Engineering: Rosenthal virus simulator. Internet (1997)
38. Sharif, M.I., Riley, G.F., Lee, W.: Comparative study between analytical models and packet-level worm simulations. In: PADS 2005: Proceedings of the 19th Workshop on Principles of Advanced and Distributed Simulation, pp. 88–98. IEEE Computer Society, Washington, DC (2005)
39. Singh, S., Lyons, J., Nicol, D.M.: Fast model-based penetration testing. In: Proceedings of the 36th conference on Winter simulation, WSC 2004, pp. 309–317 (2004), http://portal.acm.org/citation.cfm?id=1161734.1161797
40. Skoudis, E., Zeltser, L.: Malware: Fighting Malicious Code. Prentice Hall Professional Technical Reference, Upper Saddle River (2003)
41. Symantec: Symantec security response. Website (2008), http://www.symantec.com/security_response/ (last access: January 18, 2008)
42. Symantec: Symantec internet security threat report trends for 2010. Tech. rep., Symantec Corporation (2011)
43. Symantec Research Labs: Symantec worm simulator. Internet (2005)
44. Takebe, T.: Trend in security evaluation and accreditation. In: Proceedings of the SICE Annual Conference, vol. 2008, pp. 1482–1486 (2008)
45. Wagner, A., Dübendorfer, T., Plattner, B., Hiestand, R.: Experiences with worm propagation simulations. In: WORM 2003: Proceedings of the 2003 ACM Workshop on Rapid Malcode, pp. 34–41. ACM, New York (2003)
46. Wei, S., Mirkovic, J., Swany, M.: Distributed worm simulation with a realistic internet model. In: PADS 2005: Proceedings of the 19th Workshop on Principles of Advanced and Distributed Simulation, pp. 71–79. IEEE Computer Society, Washington, DC (2005)
47. Whitaker, A., Newman, D.: Penetration Testing and Cisco Network Defense. Cisco Press (2005)
48. Wilhelm, U.G., Staamann, S., Buttyán, L.: Protecting the Itinerary of Mobile Agents. In: Demeyer, S., Dannenberg, R.B. (eds.) ECOOP 1998 Workshops. LNCS, vol. 1543, pp. 301–301. Springer, Heidelberg (1998)
49. Yee, B.S.: A sanctuary for mobile agents. In: Proceedings of the DARPA Workshop on Foundations for Secure Mobile Code, Monterey, USA (1997), citeseer.ist.psu.edu/article/yee97sanctuary.html (last access: May 08, 2006)
50. Zou, C.C., Gong, W., Towsley, D.: Worm propagation modeling and analysis under dynamic quarantine defense. In: WORM 2003: Proceedings of the 2003 ACM Workshop on Rapid Malcode, pp. 51–60. ACM, New York (2003)

Chapter 7
Simulated Multi-robot Tactical Missions in Urban Warfare

Peter Novák, Antonín Komenda, Michal Čáp, Jiří Vokřínek, and Michal Pěchouček

1 Multi-robotics in Urban Warfare

Since late 90's of the last century, rapid advances in technology, mechanical engineering, miniaturization, telecommunications and informatics enabled development and routine deployment of sophisticated robots in many real world domains. Besides many applications in assembly industry, e.g., in car, or electronics assembly lines, defense organizations, together with space exploration and mining industries belong to the most demanding and optimistic users of robotic technology [25]. Especially in the military domain we nowadays witness a routine deployment of robotic assets in the field. Some of the popular examples of such robots include unmanned aerial vehicles/systems (UAV/UAS), be it conventional fixed-wing aircrafts (CTOL - a conventional take-off and landing vehicle), various rotorcrafts, such as single, or multi-rotor helicopters (VTOL - a vertical take-off and landing vehicle), autonomous underwater vehicles (AUV), unmanned cars (UGV - a unmanned ground vehicle), or unattended ground sensors (UGS), such as various acoustic, seismic and chemical sensors, or cameras. Various size and equipment classes of such robots are used in tactical, law enforcement or rescue operations for tasks, such as security surveillance of urban areas, firefighting, or providing situational awareness, mapping and exploration of areas stricken by natural disasters, or tasks in dangerous work environments, such as stabilization of damaged nuclear reactors after an earthquake. The robots are usually performing either manipulation tasks, or provide situational awareness to human task forces by information collection, such as continuous video streaming, acquiring static imagery or analysis of chemical, or nuclear hazards [17].

Even though we recently witnessed rapid advances in control of robots in scenarios such as e.g., autonomous cars [26], or service robotics [36, 6], the state of the

Agent Technology Center, Department of Computer Science and Engineering,
Faculty of Electrical Engineering, Czech Technical University in Prague, Technická 2,
CZ-16627 Prague 6 - Dejvice, Czech Republic
e-mail: {peter.novak,antonin.komenda,michal.cap}@agents.fel.cvut.cz,
{jiri.vokrinek,michal.pechoucek}@agents.fel.cvut.cz

M. Ganzha & L. C. Jain (Eds.): Multiagent Systems & Applications, ISRL 45, pp. 147–183.
springerlink.com

art in high-level control of robotic assets still relies mainly on teleoperation. Some of the high-profile examples of such deployed systems used by military and law enforcement are the surveillance drones providing imagery and video streams from operation theaters [32], or robots dealing with relief operations in damaged nuclear power plants [15].

While teleoperation is a very effective method of robot control in many scenarios, it does not scale well with the increasing number of deployed assets. One of the most important problems arising in situations where a larger number of cooperating robots is needed to successfully accomplish a joint mission are the limits of the employed resources. A single operator is capable to directly control only a relatively small number of robots. Thus with increasing number of deployed assets, direct control of robots by humans becomes too costly in terms of human resources. In result, scaling the number of robots requires larger numbers of human controllers, what finally raises also financial and logistical costs of such operations.

One of the natural solutions to the problem of high-level control of multi-robot teams is a significant increase of the level of autonomy of the individual robots, as well as the multi-robot team itself. I.e., instead of resting on a human operator, the tedious lower-level decision making and control (e.g., movement in a terrain, or camera pointing, simple task planning, etc.) should be shifted on board to the robot itself and the human operator should only take care of tasking the multi-robot team and oversee the mission execution. The working hypothesis underlying the solution is that

> *a single operator is capable to task and oversee even a relatively large multi-agent teams comprising of highly autonomous robots which require human intervention only rarely, especially when facing crucial choices in execution of the joint team mission.*

Agent Technology Center (ATG) at the *Department of Computer Science and Engineering* of the *Czech Technical University in Prague* is one of the leading research groups in innovative industrial and research applications of multi-agent systems. Besides other research topics, one of the major realms of its activities is research and proof-of-concept prototyping in the field of mid and large scale simulations of multi-robot systems. In this area, ATG focuses primarily on teams of autonomous aircrafts and ground vehicles. In this chapter we present a cluster of completed projects Tactical AgentFly and Tactical AgentScout, as well as outline the core objectives of an on-going follow-up project AgentFly-In-Air. All these projects aimed at investigation of cooperation and coordination issues in multi-robot teams either carrying out tactical missions in urban warfare scenarios, or providing information collection support to human troops on the ground in such environments. The particular objectives and foci of interest of the projects were organically evolving over time spanning the years 2008–2011. However, the specifications of the individual work-packages and their respective delivered demonstrators provide a set of high-level design requirements on an underlying technological infrastructure.

The main contribution of this chapter is an account of architectural and technological issues related to development of the multi-agent platform and simulation subsystems for the project cluster. The underlying storyline revolves around the ar-

chitectural shifts during the project development caused by gradual extensions of the technology, as well as incorporation of often conflicting application requirements over time. In essence, these can be characterized as a move from a general-purpose MAS platform imposing a specific MAS philosophy towards a more liberal component-based architecture in terms of a toolkit for rapid construction of application-specific fragmentary MAS platforms and applications.

In this respect, firstly, Section 2 provides an overview of the research objectives of the twin projects. Subsequently Section 3 discusses the initial analysis of the approach to design of the underlying multi-agent technological infrastructure. Section 4 gives a more detailed account of the completed projects, the approaches we took to tackle them, as well as the evolution of the underlying technological infrastructure and the involved toolkits. Critical analysis of the technological infrastructure evolution provided in Section 5 highlights the main lessons we learned in the course of the work on the twin projects. Finally, Section 6 concludes the chapter by an outlook to the open issues and challenges for the broader MAS community involved in development and testing of various decentralized algorithms ultimately targeted for multi-robot systems embedded in real hardware.

Sections 3, 4 and 5, the three core sections of the presented chapter are all structured in a similar manner and subsequently discuss the main aspects of the implemented system. Namely the issues of simulation and environment modeling, followed by tackling the problems involved in experimental evaluation of the research algorithms under investigation, together with configuration of simulation scenarios. Subsequently, we discuss topics in mechanisms for agent deliberation and behavior implementation and conclude by treatment of the issues in simulation visualization and user interfaces. The recurrent structure provides scaffolding for the main storyline of the chapter, the evolution of various aspects of the technological platform underlying the Tactical AgentFly and Tactical AgentScout project cluster over time.

2 The Project Cluster: Tactical AgentFly, Tactical AgentScout

In the course of the years 2008–2011, Agent Technology Center was (and still is) involved in a continuous interaction with CERDEC, ONR and AFOSR, the research and development departments of U.S. Army, Navy and Air Force. Over time, these interactions resulted in formulation of several research problems stemming from the real needs of military units carrying out tactical missions on the ground and their usage of advanced robotic and sensory technologies, such as various aerial drones, especially the class of man-portable small unmanned fixed-wing aircrafts, various rotorcrafts and unmanned ground vehicles. In particular, some of the most prominent problem topics included area exploration, surveillance, tracking of mobile targets, patrolling and teamwork coordination in structured heterogeneous multi-agent teams. The work towards investigating these research issues produced a number of technological challenges, which had to be solved and implemented using an

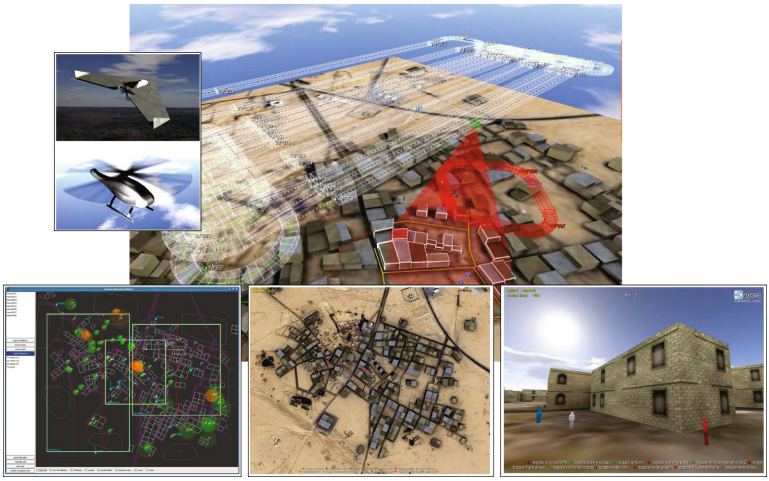

Fig. 1 Visual impression of Tactical AgentFly simulated environment

underlying technological infrastructure providing the basis for development of prototypes demonstrating the proposed research approaches.

From technological perspective, the common character of the here described projects is that the deliverables take the form of executable demonstrators showcasing behaviors of algorithms for coordination of teams of robotic aircrafts, ground vehicles and human troops in synthetic simulated scenarios of tactical urban warfare. Besides the capability to empirically and reproducibly evaluate the performance of the developed algorithms, the simulations must come with rich visualization components. One one hand, rich visualizations plausibly demonstrate that the behavior of the robots complies with their realistic limitations, such as sizes, weights, speeds, physical motion dynamics, and physical properties of the environment. On the other, they also help to facilitate dissemination of the results beyond the particular sponsoring partner organization to non-expert audiences. In a consequence, the form of the projects' deliverables frames the more detailed technological requirements stemming from the research objectives of the projects. In this section we provide an overview of the research issues of the individual phases of the projects Tactical AgentFly and Tactical AgentScout.

2.1 Tactical AgentFly

The objective of the TACTICAL AGENTFLY project was to develop basic agent-based techniques for controlling a group of autonomous UAVs performing information collection in support of tactical missions. The emphasis was on accurate modeling of selected key aspects occurring in real-world information collection tasks, in particular physical constraints on UAV trajectories, limited sensor range and sensor occlusions occurring in spatially-complex environments. The ultimate

goal was to provide a high-level interface through which the operator can control a fleet of UAVs and assign high-level tasks to the multi-robot team. The task allocation to individual UAVs itself, as well as planning of their optimum trajectories was performed automatically by the multi-agent team. The specific objectives of the project were the following:

Formal framework: we had to design a framework for formal specification of the information collection problem, serving as a common reference point for the rest of the project. In particular, we were asked to investigate the concept of an information collection task, including task constraints and task objective functions, which formed a basis for evaluation of the developed algorithms.

Persistent area surveillance: we investigated mechanisms for control of operation of teams of UAVs providing and maintaining an up-to-date operational picture of a designated target area. A key feature of the developed surveillance algorithms is their respect for UAV's motion constraints and the ability to provide full area coverage even in environments affected by sensor occlusions, such as narrow streets between tall buildings. The results of this research track have been published in [30].

Target tracking: we were exploring the mechanisms to control the operation of a UAV providing continuous tracking of one or multiple mobile ground targets, respecting the UAV's motion constraints.

Information collection testbed: an important objective was to develop an extensible software platform for implementing, simulating and evaluating various UAV control mechanisms explored in the project. It had to contain a detailed model of the urban environment, a model of the UAV's on-board camera and a behavioral simulation of several types of ground entities. A simulation of a multi-stage search-and-capture mission was to be prospectively implemented as well in order to enable evaluation of information collection mechanisms on a real-world-like scenario. The testbed had to provide intuitive real-world-like 3D visual output allowing the presentation of the project results outside the strictly technical community.

Command and control (C2) user interface: we had to develop an integrated C2 system for mixed-task information collection. Such a system was to provide an additional level of automation on top of the autonomous surveillance and tracking control mechanisms. It should have consisted of two parts. Firstly, a C2 panel through which the operator can specify information collection tasks and inspect their results, and secondly, an allocation algorithm which optimally allocates a mix of concurrent information collection tasks between a group of UAVs.

After the project's first phase was completed, we were awarded a follow-up project, during which we moved from the basic coordination algorithms for teams of fixed-wing UAVs towards more advanced techniques applied in heterogeneous teams. The main research objectives were the following:

Modeling Vertical Take-off and Landing (VTOL) Assets: we had to extend the existing platform and enable integration of various types of VTOL UAV assets (helicopters, quadrotors, etc.). The second major research objective of the project

was to propose and develop suitable algorithms for trajectory planning of VTOL assets and integrate them with the VTOL model. The resulting algorithms have been presented in [10, 11].

M×N tracking: we focused on investigation of algorithms for tracking larger numbers of mobile targets by relatively small teams of aerial assets. One of the aims was to investigate the methods of maximizing persistence of tracking of the objects and identifying how many assets are needed in different types of tracking scenarios. Concretely, this research led to investigation of techniques for intelligent target tracking task hand-over between multiple UAVs. The results of this research track are discussed in [35].

Coordination for mixed information collection activities: in this workpackage, we aimed at studying mutual interactions between simultaneously performed heterogeneous information collection tasks. For this we had to i) extend the range of considered information collection task types with additional classes, in particular exploration and search, and ii) study the theoretical interactions between mixed/heterogeneous information collection tasks performed simultaneously. In particular, the idea was to investigate the problems arising from automated techniques facilitating transparent switching between different information collection tasks, such as switch from surveillance to target tracking and back.

Mission-centric/oriented information collection: the final objective of the second phase of the Tactical AgentFly project was to further extend the integrated coordination for mixed information collection. That is, the task was to propose techniques for a team of UAVs performing information collection tasks, however taking into account the plans of special operation units carrying out their own mission on the ground in the town, so that the information needs of the mission are optimally covered. The activity aimed at research and possibly prototyping efforts towards considering temporal development and dependencies between the individual information collection tasks as the mission progresses.

2.2 Tactical AgentScout

TACTICAL AGENTSCOUT project started as a branch of TACTICAL AGENTFLY project that aimed at integration of aerial information collection with various types of ground robotic assets. The foci of the project were on multi-agent planning and task-allocation problems. The proposed solutions were to be demonstrated in a simulated tactical scenario taking place in a complex urban environment defined by an *a priori* known street map. Furthermore, the environment included various 3D terrain features, as well as buildings. Finally, the the environment should have contained a number of road blocks on the street map, which however were not *a priori* known to the multi-robot team. The goal was to find a safe and effective path for a convoy to traverse the urban environment. The task of the team of autonomous UGVs was to cooperatively support the convoy by continuous exploration of the area and search for the road blocks on the way of the convoy. The information about the discovered

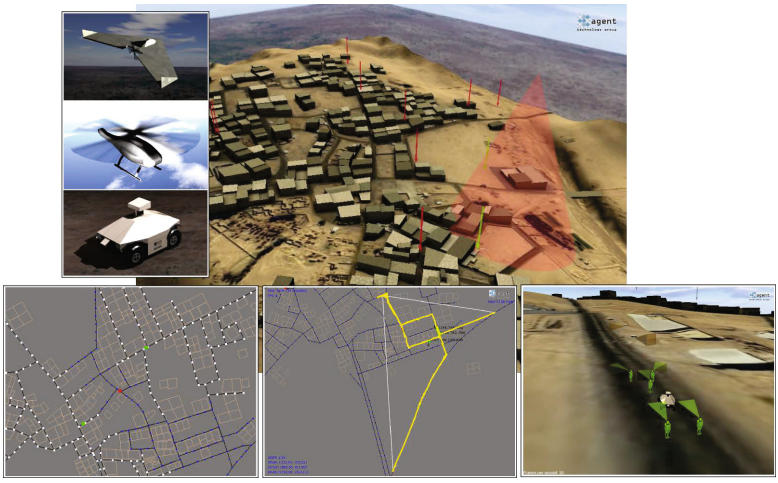

Fig. 2 Visual impression of Tactical AgentScout simulated environment

obstacles is then communicated to the convoy and can be used to update its plan. In particular, the project included the following research topics:

Integration of various ground units: the main planned contribution of the Tactical AgentScout project was to enable integration of various autonomous ground robots into the existing platform. In particular, the challenge was to plausibly model the physical dynamics of the assets within thef simulation.

Distributed dynamic vehicle routing problem (DVRP) solver: the cooperative exploration problem was identified as a variant of the DVRP of the supporting UGVs. Although there are centralized state-of-the-art techniques for solving the DVRP, our goal was to propose a novel multi-agent solver for this class of problems [33].

Planning under uncertainty: the flight trajectory planning algorithms for UAVs as used in Tactical AgentFly assumed reliable execution of the generated plans. In the domain of ground vehicles, we deemed such an assumption too strong. The goal was to develop a path planner for UGVs that is able to efficiently control the vehicle also in dynamic, uncertain environments. To evaluate such a planner, we aimed to simulate the UGVs using physically realistic models that let us generate some of the problematic real-world phenomena such as tire spin, vehicle mass momentum, limited engine power etc. [34].

In a follow-up project we moved towards more advanced techniques studying various aspects of adversarial and cooperative behaviors in dynamic environments. Specifically, the project addressed the following research challenges:

Patrolling of mobile targets: a complementary task to tracking a number of mobile adversary targets is the protection of mobile ground units against attacks from enemy units. The motivating scenario is an urban environment with a number of convoys passing through an area which should be protected by a small

mixed team of aircrafts, e.g., helicopters and small fixed wing aircrafts. In such a scenario, it is vital for the patrol to execute a non-predictable patrolling strategy. The opposite would allow the adversary to optimize with respect to the patrol's strategy and attack the convoy in the worst timepoint, e.g., when the patrol just left the convoy it protects. The solution was the use of randomized strategies which, however, still maintain certain average frequency of visits of each protected convoy. The research objective was to develop algorithms for computation of optimal strategies for protection of mobile targets in adversarial environments. The resulting method has been published in [7].

Smart targets modelling: the mobile target tracking techniques as proposed in the Tactical AgentFly project paid no attention to intelligence of the tracked targets. As an extension, in this project we considered smart targets, i.e., such which actively monitor their environment and optimize their behavior to act with respect to the information they acquire. Smart targets, when being tracked, are aware of that fact and actively try to avoid the tracking unit. Complementary, we consider trackers (be it UAVs, VTOLs, UGVs, or personnel) to be aware of the fact that the tracked targets are aware of their activities and try to act in best response to the whole setup. The objective here was to propose and solve a formal game-theoretical model of pursuit-evasion scenario with heterogeneous teams of agents [24].

Multi-agent re-planning and plan repair: classical-style planning is not robust with respect to unexpected events occurring in dynamic environments. The standard solution, in such cases, is to simply re-plan the agent's behavior from scratch. While decentralized extensions of classical planning can be used to compute activities of the individual team members, full-scale re-planning can become too costly due to the costs of communication in multi-agent teams executing a decentralized planning algorithm. Thus, the objective was to formalize multi-agent plan repair problem and devise and evaluate algorithms for solving it. An initial version of the plan repairing algorithm has been presented in [22, 23].

Coordination and teamwork: reactive planning is an alternative approach to dealing with dynamics of environments. The state-of-the-art techniques of reactive planning, however, do not support implementation of team-level behaviors. The objective here was to extend an existing agent programming framework so that it can accommodate techniques for team-level coordination specification in terms of reactive plans executed jointly by the team members. In particular, we aimed at implementation of the *distributed commitment machines* approach [37] in a chosen agent-oriented programming language.

3 Analysis and Design of the System

The general description of the individual research issues discussed in the previous section and their respective software demonstrators provides a comprehensive overview of the various, often conflicting, requirements on the technological infrastructure for the Tactical AgentFly and Tactical AgentScout project cluster. The

overall goal of the technological side of the whole endeavor, the main topic of this chapter, could be formulated as follows:

> *Develop a set of software tools enabling rapid prototyping of a broad range of simulated missions involving teams of autonomous robotic assets, as well as various situation scenarios in tactical urban warfare the robots are involved in. Furthermore, the toolkit must allow for batch evaluation of performance of the algorithms governing the behavior of the multi-agent teams in a range of configurations of the simulated scenarios.*

In the core, the objective was to facilitate modeling and execution of various types of robotic systems which i) feature heterogeneous physical capabilities, ii) employ heterogeneous control algorithms governing their autonomy, and finally iii) are embodied in simulated, but realistic physical environments. As already discussed in the introductory section, the underlying hypothesis of the research work in the twin projects was that a significant increase of autonomy of the individual robots and the multi-robot team itself will lead to reduction of human resources involved in the operation and monitoring of such teams. The stress on the autonomy of the involved agents, such as various unmanned aerial or ground vehicles, directly induces the remaining requirements usually ascribed to *intelligent agents*[38]. In particular these include reactivity, proactivity and in our context especially social capabilities. In result, the character of the problem directly correlates with the usual assumptions underlying the multi-agent paradigm. Hence the choice of the conceptual and analytical framework for multi-agent systems, as well as the initial tools and platforms for development of multi-agent systems became natural and straightforward. Additionally, our hypothesis was that the inherent decoupling of entities according to the multi-agent systems paradigm should open a way towards future porting and deployment of the developed agent (simulated robot) control algorithms to real hardware. We also hypothesized that the decoupling, i.e., the inherent modularity of multi-agent systems will facilitate rapid prototyping of a broad range of simulated scenarios involving heterogeneous multi-robot teams. It also should have options opened for rich modeling of environments these robots operate in. Finally, the previous industrial development and deployment of multi-agent systems, as also discussed in [3], provided ample argument in favor of future scalability of the developed multi-robot simulations to relatively large numbers of involved agents.

The remainder of this section provides a discussion of the initial analysis and design consideration tackling the technological problem stated above. Subsequently, we provide an overview of the technologies and platforms we initially chose for implementation of the first phases of the project cluster. In Section 4 below we discuss our experience with these technologies and how the evolution of the projects over time led to reconsideration of our initial choices.

3.1 Initial System Requirements

During the initial system analysis step of the first phase of the Tactical AgentFly project, we identified a list of architectural and functional requirements on a

technological infrastructure underlying the project implementation. These can be divided into four groups subsequently discussed below. The structuring along the four aspects of the technological infrastructure also provides a scaffolding for discussion of the system evolution in time and its critical analysis in the subsequent parts of the chapter.

3.1.1 Multi-agent Platform

Given the argument in favor of application of multi-agent paradigm in the projects and the stress on future potential for porting and deployment of the developed algorithms to real hardware assets, there arises a need for a multi-agent platform. I.e., a software supporting modeling of the individual agents, execution and life-cycle management of the multi-agent system as a whole and services of a communication middleware, such as a white pages register and a discovery service. Due to the focus on relatively small and mid-scale simulation scenarios involving dozens, possibly low hundreds of agents and environment entities, the requirements on network and CPU distribution and agent mobility were not an important issues in our projects. Finally, since organizational structuring and coordination of MAS teams was the main research focus on the application level in the projects cluster, we felt that binding to a particular MAS organization philosophy would be rather a burden possibly interacting with the coordination techniques under investigation. Thus, an *a priori* organizational model imposed by the MAS platform was undesirable.

3.1.2 Environment Simulator and Scenario Modeling

A simulated multi-robot system must be embodied in a physical environment which faithfully models a set of relevant aspects of the real physical reality in the simulation. Thus, the technological platform should facilitate rapid development and configuration of instances of simulated environments. First and foremost, such scenarios comprise the physical structure and topology of the environment. In particular, our focus on tactical military operations in urban terrain dictated ability to model landscapes with realistic terrain, traffic infrastructure of small and medium urban areas including buildings, roads and bridges. Finally, due to the focus on robots physically interacting with the environment, such as unmanned ground vehicles, we needed to realistically model their physical interaction with the environment including phenomena of gravity, object masses and rigid body interactions. Due to the need to demonstrate functionality and robustness of the developed coordination mechanisms in various settings, the platform should provide means for straightforward configuration of the physical features of simulated environments in different simulation instances.

The platform should also support clear cut interfaces between agents and the environment. That is, the software interfaces for implementation of robots' sensors and actuators must be well defined so that i) the agents use unified style of such interfaces across the whole application, and ii) in the case of future need, these could be easily bridged to physical hardware sensors and actuators.

3.1.3 Experiments and Configuration

The main objective of the described projects is to investigate various types of coordination mechanisms among agents belonging to teams of cooperative individuals. Thus, besides proposing and developing such algorithms, the main task on the application level is to perform empirical evaluation of their performance and measure it against benchmark cases and alternative state-of-the-art methods. Running such experiments *en masse* requires that the platform is capable of providing means for batch execution of experimental setups in different configurations and enables to collect the resulting data sets in a straightforward manner. Furthermore, to facilitate reproducibility of the experiments, the experiments should be run in a deterministic manner with pre-configured random generators if needed. Finally, since there might be hundreds of such experiment scenario instances, the platform must be able to execute them in parallel, as well as in faster than real-time speed. I.e., it must support both modes of scenario execution: physical-time-speed mode, where one objective second correlates with one second in the simulation; and it should be able to run at the top use of the available resources, such as the CPU speed, operating memory limit, etc.

3.1.4 Agent Behavior Control

On one hand, often in the project cluster experiment setups the individual behavior of the simulated robots was rather rich beyond the particular coordination mechanism under investigation. For instance, while an autonomous ground vehicle must coordinate its plans for traversal of a given town street network with its peers, at the same time it is responsible for local execution of those plans. Besides steering the physical body of the robot and monitoring the progress of the plan execution, this task also involves relatively complex behaviors for dealing with the issues which were abstracted away by the planner. For instance, the plan would prescribe driving through a sharp curve in a narrow street, but due to the physical limits of the vehicle the car could end up stuck in the corner and must maneuver out of the place e.g., by iteratively driving backwards and forwards.

On the other hand, the environment contains also actors, which are not in the focal point of the multi-agent coordination mechanisms. These include agents representing the ground troops carrying out a tactical operation (the blue force), which the robots are supposed to support. Furthermore, there can be also various kinds of adversarial units (the red force), or civilians. Depending on the particular scenario, their behaviors range from relatively simple, such as random movement within a perimeter, to relatively complex ones implementing the mission played by the blue force. From the point of view of the multi-robot team in consideration, such agents model, as well as generate the environment dynamics relevant to the target coordination mechanism under investigation.

Finally, while some simulations would require rather abstract discrete time stepwise simulation of agent's actions, in other scenarios more realistic fully asynchronous time model is needed. Due to uncertainties with respect to the particular

style of the simulations required for evaluation of the broad range of the considered centralized and decentralized coordination algorithms, the simulator shouldn't be strongly bound to a single simulation model.

3.1.5 User Interface and Visualization

To maximize applicability of the algorithms on real hardware robots in the future, one of the requirements of the projects Tactical AgentFly and Tactical AgentScout was that the developed coordination mechanisms must be demonstrated in scenarios featuring believable adversaries, and close-to-real-world physical environments. In result, the underlying technological infrastructure had to support rich visualization interfaces providing both 2D and 3D graphical views on the state of the simulation capable to display the physical dynamics of the simulated entities in real-time. Finally, the platform had to support creation of graphical user interfaces for implementation of real-time simulation and robot-team control interfaces (C2 user interfaces) when needed.

3.2 Initial Technological Infrastructure: AgentFly

In 2008, at the time of starting the first phase of Tactical AgentFly project, timewise the first project of the cluster, the *Agent Technology Center* had already an in-house developed technological infrastructure to start from. In the context of AgentFly project, described below, we developed an advanced technological platform for air traffic management and flight control of unmanned fixed-wing aircrafts. The initial design of the technology used for the Tactical AgentFly and Tactical AgentScout projects heavily relied on re-use and extension of proprietary, in-house developed technologies from the AgentFly project. In the following, we provide a brief overview of the individual software packages planned to be re-used, adapted, or integrated with the newly developed technological platform.

3.2.1 Application Domain Fundamentals: AgentFly Overview

The AgentFly system developed in ATG between 2005–2011[1] is a technological platform facilitating development of mid and large scale simulations of autonomous aircrafts. As of 2008, the system was primarily aimed at investigation of issues in flight trajectory planning of various classes of fixed wing CTOL UAVs and their collision avoidance. I.e., a typical AgentFly scenario instance was configured with a landscape, a number of pre-configured no-flight zones and a number of aircrafts. Each aircraft was initialized with an initial position, space constraints on its flight trajectory envelope, speed vector (or landed on a ground runway) and a set of waypoints it should visit in particular times and fly over in particular speeds and heading vectors. The aircrafts were able to compute optimal flight trajectories spanning the

[1] As of writing this report, the project is still being actively developed, extended, and optimized.

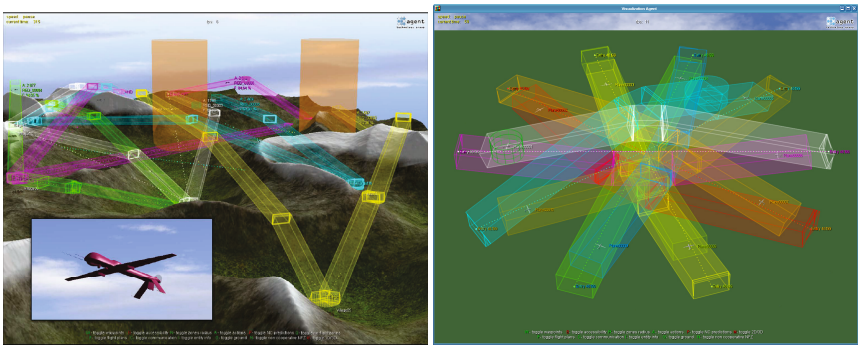

Fig. 3 Snapshot of an AgentFly scenario simulations

waypoints, and at the same time respecting the constraints set up by the environment, together with the time, velocity and heading constraints. At the core of the system lies a decentralized algorithm running on board of the simulated aircrafts which was able to firstly detect conflicts between the trajectories of the planes and subsequently compute solutions to the conflicts in the whole multi-aircraft system. Figure 3 depicts a snapshot of an example scenario running in the AgentFly system.

Initially, the projects of the Tactical AgentFly and Tactical AgentScout cluster were supposed to re-use the existing algorithms for trajectory planning and collision avoidance and extend them so that they could be integrated with algorithms for area surveillance and target tracking.

3.2.2 Multi-agent Platform: Aglobe

The architecture of the AgentFly system relies on a state-of-the-art multi-agent platform Aglobe [31] and its simulation extension AglobeX Simulation (see Figure 4). Aglobe is an award-winning multi-agent platform, similar to *JADE* [4] or *Cougaar* [13], aiming at testing of experimental scenarios featuring agents' position and communication inaccessibility. The focus of the platform is on modeling and development of decentralized multi-agent systems. It provides facilities such as communication infrastructure, data store, directory services, weak migration function, deployment service, etc. Aglobe features an extremely efficient message transport layer and agent life-cycle management and thus offers a very high level of scalability. From the perspective of a single agent, the platform provides two types of interfaces. Firstly, the MAS platform provides core functionalities including intra-agent tasking, message communication, communication visibility, agent life-cycle, and migration. Secondly, it includes and handles a number of services including directory/yellow pages, communication monitoring, and other. Due to its underlying architecture, the service interface is highly modular, as the individual services can be optionally turned on/off.

In Aglobe, agents reside in containers, which can be seen as logical groups of agents acting as a single entity from the perspective of the communication visibility

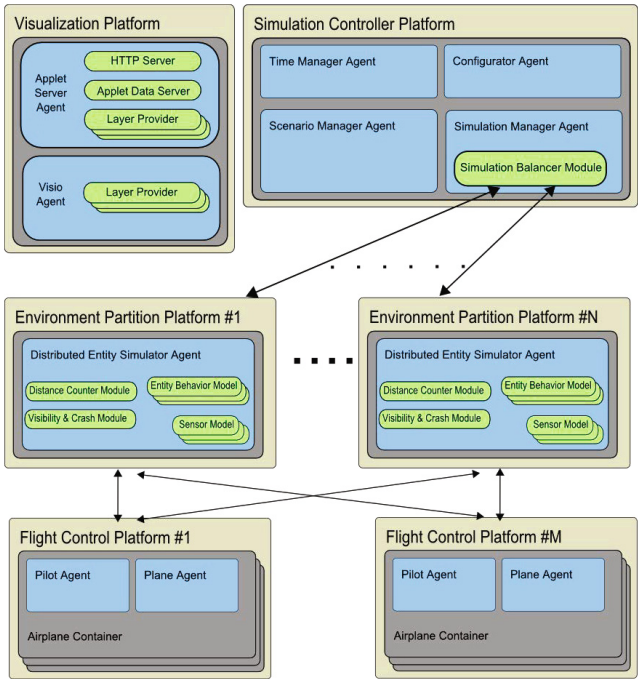

Fig. 4 AglobeX Simulation extension of the Aglobe platform depicted as it was instantiated in the AgentFly project as of beginning of the Tactical AgentFly project

subsystem. The containers run on platforms, where each platform is run within a single *Java Virtual Machine* (JVM), possibly on a single network node. Each agent is running asynchronously in its own computational thread and it is driven by an internal event queue. The events are strictly personalized for each single agent and are used both for self-tasking, as well as for processing external messages. The agents run and communicate in a fully asynchronous manner with respect to each other. The communication subsystem, the message transport layer, is optimized in various respects. All messages are pooled for more efficient creation and destruction of the underlying objects. User of the platform can configure whether the message payloads are sent by serialization or as references (available only within the same JVM). The message delivery mechanism selects an efficient message transport method taking into consideration the platform decentralization, communication capabilities of the underlying network stack and the target platforms the agents reside on.

On the message transport layer, Aglobe employs a publish/subscribe mechanism based on global topic messages. These are used for system communication, as well as widely employed by AglobeX Simulation simulator and other technology subsystems.

3.2.3 Environment Simulator and Scenario Modeling: AglobeX Simulation

AglobeX Simulation (see Figure 4) is a simulator extending the Aglobe multi-agent platform. It aims at modeling and execution of real-world-like 3D simulations including both static, as well as mobile units such as towns and ports on the one hand, as well as aerial and ground vehicles on the other.

The simulator is implemented as a number of interacting Aglobe agents. The responsibilities of the agents are initialization, finalization and execution of the simulation. In the course of a simulation the agents compute the next time-step evolution of the vehicle positions, simulate inputs to their sensors, detect collisions, prepare required information about the vehicles for the communication visibility subsystem, etc. At the time of using the platform in our projects, the core functionality of the AglobeX Simulation component was due to legacy reasons centralized and with respect to other subsystems functioned in a client-server mode. In its current incarnation, the simulator leverages the multi-agent paradigm as well and thus enables the distribution of the entire simulator over a set of network nodes. The simulator's clients are agents or groups of agents residing in a single MAS container controlling the simulated entities, which represent their embodiments within the simulated environment. The agents communicate with the simulation server using the topic messages. In one direction, the agents control their embodiments in the physical simulation and in the other direction, they are perceive the state or changes in the environment through their sensors.

The simulator consist of the following agents:

Scenario manager agent: manages, pre-processes, and serves the configuration data to the simulation (see also Section 3.2.4), manages the simulation scenarios, their initialization, finalization, and monitors their execution.

Time manager agent: provides synchronized time ticks for all the components involved in the simulation.

Simulation manager agent: manages client agents and client containers, creates and removes simulated entities, forwards the initial configuration to the entities (all this based on the configurations from Scenario manager agent).

Entity simulator agent: computes the step-wise evolution of the entity states (positions, orientations, etc.). Each entity is described by its behavior transforming the current state to a new state according to the commands received from the related client agent (agent container). The behaviors together describe mechanics of the environment.

Distance agent: computes and stores distances among all entities in the environment.

Visibility collision agent: processes data from the Distance agent and i) provides them to communication visibility subsystem of Aglobe, and ii) uses them as a basis for entity collision detection.

Sensor agent: represents all monitoring and detection systems of the simulated entities (radars, cameras, etc.) and provides views to the simulation for the simulation agents based on time ticks from the Time manager agent.

All the server-side agents communicate using the topic messages. The agents use the topic messaging to orchestrate and synchronize themselves as a single, centralized or distributed, simulation process.

3.2.4 Configuration, Experiments, User Interface and Visualization

As already discussed above, in AglobeX Simulation, various types and courses of simulation are conceptualized into scenarios. Each scenario describes the initial state of the simulation instance, as well as the initial states of the controlling client agents. Each scenario also contains an initialization process for the related simulation. The scenarios are *Java* programs supported by mechanisms for XML-based configuration processing. A single scenario, together with its configuration initializes and executes a single simulation instance. Besides that, scenarios can also contain components gathering, processing, and storing results from experiments. A scenario can also describe a suite of experiments together with their automated configurations. The experimental results are collected from the simulated processes via topic messages or directly by monitoring the evolution of simulation.

As a full-featured MAS platform, Aglobe further provides various user interfaces (UIs). The core one is the platform management UI. It contains a list of services and agents running on the platform and allows a user to run a new service, stop a service or examine services and agents running within the platform. To facilitate communication monitoring and debugging, there is also a communication sniffer tool. It enables tracking of inter-agent conversations and inspect the content of intercepted messages. Optionally, agent implementations can come with their own application specific UIs. An example is an informational UI provided by the Pilot agent of the AgentFly system.

AglobeX Simulation comes with its simulation-specific user interface. The Scenario manager agent provides dialogs for scenario selection and reset. The Time manager agent allows users to start, or stop simulation time, as well as set the simulation speed. Similarly, there are lists of the simulated entities, their respective agents, logging consoles, and others.

The last subsystem of the initial technological infrastructure is the Visio visualizer for the simulated world populated by agents. Visio is built on *Java3D* library [18] and *JOGL* library [21]. It enables 3D and 2D visualization of the simulated worlds, including the physical environment, simulated entities and additional information used during implementation, debugging phase and demonstrations. The visualized elements are organized in a hierarchical layout (used for the canvas drawing process) and a layered layout (used for turning on/off groups of visual elements). The visualization layers communicate with the rest of the system, i.e., client and simulation agents, by standard Aglobe topic messages.

3.2.5 Agent Behavior Control: POSH

A significant requirement coming out of the initial analysis of the technological infrastructure underlying implementation of the Tactical AgentFly/Tactical

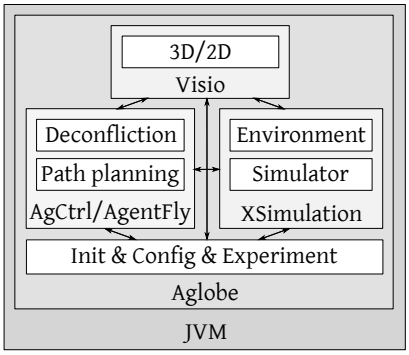

Fig. 5 Initial architecture of the technological infrastructure for the first phase of the Tactical AgentFly project based on AgentFly project (feature framing represent architectural composition, arrows represent coupling)

AgentScout cluster was a need for a framework and an associated toolkit facilitating scripting of agent behaviors. The scenarios modeled in the AgentFly project did not require such complex behavior implementations. Early on, it became clear that for the projects Tactical AgentFly and Tactical AgentScout we need to bring in a new technology facilitating behavior scripting and integrate it with the rest of the system. Our initial idea was to leverage the state of the art frameworks for development of intelligent autonomous agents [5]. We chose to consider the *POSH* (Parallel-rooted, Ordered Slip-stack Hierarchical) reactive planning engine [9, 8]. *POSH* can be seen as a programming language allowing to execute programs encoding reactive action selection based on both the current state of the environment the agent perceives, together with its internal state, which can feature relatively complex data structures (beliefs). *POSH* follows the paradigm of behavior oriented design where behavior modules of a *POSH* agent provide the primitives for formulation of the plans. These are subsequently executed in run-time and represent the agent's deliberation about the particular action to be taken in the next step.

3.3 Initial System Architecture

Figure 5 depicts the initial architecture of the underlying technological platform for the project cluster. It is based on the technologies from the AgentFly project. The platform stack runs within an instance of *Java Virtual Machine* (JVM) and the whole system was written in *Java*. Majority of the subsystems are implemented as agents running in the Aglobe MAS platform container. The Visio visualizer subsystem is partially independent from Aglobe, since it incorporates a standalone part of the visualization engine.

All technological components of the system are configured with a simulation scenario descriptions and their respective XML configurations, initialization programs and experiment descriptions. These parts are depicted as *Init & Config & Experiment*

subsystem. As already noted in the previous subsections, the AglobeX Simulation subsystem (abbreviated as *XSimulation*) runs as an orchestra of Aglobe agents as well. The simulation subsystem consist of the *Simulator* component managing the time and simulation state. The *Environment* component manages simulated entities, sensors, and communication visibility.

The AgentFly system as whole is represented by agents acting as aircraft pilots. A pilot agent implements a particular agent controls mechanism (abbreviated as *AgCtrl*). The component implements two main technologies. Firstly, it is a flight path planner and secondly, a decentralized collision avoidance mechanism, coined also deconfliction. These two components form the intelligent decentralized behavior of the individual aircrafts.

Finally, the visualization subsystem facilitates 3D and 2D visualizations, both based on Visio engine.

4 System Implementation and Experiences

The works on the projects Tactical AgentFly and Tactical AgentScout took part from early 2008 until mid 2011. Both projects were divided into two phases, each taking 12 months duration. Generally speaking, the initial phases of the projects were dedicated mostly to development and feasibility studies of the basic technological framework needed for realizing the main objectives. The successive iterations then focused primarily on the core objectives, namely research, development and empirical evaluation of various advanced multi-agent coordination algorithms.

In Subsection 3.1 we identified and summarized the main requirements on a technological infrastructure of the Tactical AgentFly and Tactical AgentScout project cluster. The remainder of this section discusses the details of their implementation as they were gradually incorporated into the code base. I.e., starting from the technologies underlying the AgentFly project, we discuss how the system architecture evolved by incorporating the individual requirements. During the process, we had to implement completely new modules and libraries and some of the modifications of the core elements of the infrastructure at the time resulted in significant shifts of its underlying philosophy. In result, some of the modifications were not backwards compatible with the original multi-agent platform anymore and led to significant architectural challenges. Alite is an agent simulation toolkit comprising all the general-purpose modules and libraries developed in the course of the works on the here described twin projects. Its gradual implementation could be seen as a consequence of tackling these challenges and, together with the gradual system evolution from a rigid single-philosophy embracing MAS platform to a pragmatic toolkit facilitating both a MAS platform composition, as well as rapid prototyping of the application specific functionalities. Recently, Alite grew out of providing only technological infrastructure for the Tactical AgentFly and Tactical AgentScout project cluster and it is already being used and extended in the context of various other projects ATG is involved in. While the initial design of the twin projects heavily relied on the Aglobe MAS platform and its AglobeX Simulation extension, the final

phase of the Tactical AgentScout project was completely written using Alite toolkit and the libraries it provides. The remainder of this section tells the story of the gradual shift of the underlying architectural philosophy on the background of series of extensions and requirements incorporation into a single system.

The treatment of the individual requirements follows the structure already introduced in the previous sections. I.e., starting with simulation and environment modeling issues, we continue by discussion of execution and configuration of experiments, through requirements and evolution of the agent behavior control mechanisms finally to discuss the implementation of user interfaces and visualization components of the system. Wherever possible, we respect the timeline of the work on the individual requirements.

4.1 Simulation and Environment Modeling

Due to the focus of the here described project cluster on tactical missions in urban areas, one of the first tasks the project team faced was extension of the AglobeX Simulation module with the ability to handle complex urban environments and landscapes. In particular, we extended it to handle 3D models describing buildings and the underlying street map providing the high-level environment abstraction the agents lived in.

With 3D buildings representing a town, agents moving along streets between these buildings and aircrafts flying over and screening the area by use of cameras, the environment modeling also needed to handle occlusions. To tackle this problem, we implemented an environment-specific simulation agent providing an occlusion sensor for the aircraft agents running in the system. As we discuss later, due to the inherent complexity of the AgentFly's AglobeX Simulation architecture, this design decision later turned out to be rather ineffective and after replacing the core simulation component by Alite specific module, the occlusions handling was implemented in a form of pure individual-agent level sensor.

The task of the aircrafts in the Tactical AgentFly project was to perform surveillance and tracking of urban areas with simulated human agents performing either relatively simple adversarial behavior (red force), or representing civilians in the town. The embodiments of the ground entities were added into the environment as a new type of simulated entities and integrated into the common collections of simulated entities. They featured their own mobility models and behavior mechanics controlled by reactive planning engine(s) discussed later in this section.

The Tactical AgentScout project brought a new set of requirements, which led to significant changes of the simulator. The incorporation of simulated unmanned ground vehicles (UGVs), resulted in a need to significantly adapt the model of the general physical dynamics of agents representing physically simulated entities. This adaptation resulted in an incorporation of a 3D physics simulation middle-ware into the system. Execution of trajectory plans by autonomous ground vehicles in physical conditions can result in frequent plan invalidation. Some terrain features, such as slopes, potholes, stones on the ground, or some corners are not traversable by the

vehicle in its current speeds. In result, the autonomous cars have to move around the town by executing, continuous monitoring and adaptation of the trajectory plans. As a part of the solution, we integrated an open source high-fidelity physics simulation engine JBullet [19].

To enable high-level control of the UGVs, which abstracts away the physical reality of the environment, we implemented discrete time movement of simulated ground vehicles on a street graph. In result, the high-level control algorithms steer the cars between waypoints on the street map (e.g., junctions). This development, together with the above described move from proprietary physical environment representation to a state-of-the-art technology for simulated physics led to abandonment of the AglobeX Simulation component from the system. This radical step was reinforced by the fact that AglobeX Simulation supports only simulated continuous space entity motion, however for many of our experiments only rough discrete motion on graph-based representations sufficed. In result, the move to pure Alite-based simulator led to significant decrease of implementation, as well as debugging complexity of the individual experimental scenarios and allowed us to implement also simplified simulation model based on events instead of discrete time ticks (see also below).

To implement aircrafts performing close-up tracking of mobile targets, such as pedestrians and cars resulted in a need to incorporate reactively controlled aircrafts of various types into the system, be it conventional fixed-wing planes (CTOLs), or helicopters (VTOLs). Such UAVs are able to change their flight trajectory in a reaction to changes of movement patterns performed by the ground target. In the case of fixed-wing aircrafts, which cannot stop in mid-air, this problem results in a need to perform relatively complex flight patterns, such as various loops over the target. Together with a need to implement a fine-grained physical dynamic feedback control of helicopters respecting a realistic model of their physical movements, this led to a requirement to adapt the simulator to a much finer grained time resolutions.

Fixed-wing aircrafts feature a much simpler model of their physical dynamics and therefore can afford for longer delays between individual steering points. Due to relying on features such as this, the underlying AgentFly infrastructure turned out to be too rigid for our purposes. An implementation of the required fine time granularity control of physical entities would lead to a significant abuse of the platform and would have result in significant re-implementation of the mechanics integrating high-level planners and the low level physical control of the plane. In result, as already indicated above, we gradually departed from the AglobeX Simulation component and implemented the fine-grained simulator event loop in Alite from scratch. The final implementation proved to be flexible and easily integrable with the other parts of the resulting system.

4.2 Evaluation of Multi-agent Coordination Techniques

The main objective of the twin projects was to develop and most importantly experimentally evaluate various decentralized algorithms for coordination of multi-agent

teams. This objective resulted in two basic requirements. Firstly, the simulator had to be highly configurable in order to allow for high flexibility in terms of both, simulation experiment structure (number of agents, their various types, different initial conditions, etc.) and the executed scenario storyboard, i.e., the particular mission to be executed. Secondly, the experiments comprise of large numbers of executed simulation instances and their runs had to be reliably reproducible.

The Aglobe MAS platform, together with the AglobeX Simulation extension for AgentFly feature an XML-based configuration facility. This however turned out to be too rigid for the purposes of the Tactical AgentFly and Tactical AgentScout projects. The XML-based configuration files are interpreted by the simulator and the MAS platform so that the correct numbers of agents with correct initial conditions are instantiated in the system at the beginning of a run. However, such files allow for configuration of features foreseen during the system development and thus are inflexible with respect to the extremely broad range of scenarios implemented in the here described projects. As a part of departure from the Aglobe and AglobeX Simulation technology, we implemented a flexible configuration facility based on employment of dynamic programming language interpreters of Groovy [16] and Clojure [12]. Interpretation of full-fledged programs in run-time allows to configure any aspect of simulations in a concise and flexible way. In our context, the additional overhead of configuration script recompilation turned out to be relatively small and the corresponding minor slow-down of experiment runs was acceptable, as well.

An important aspect of simulation development is reproducibility of simulations. Large-scale simulations involve various aspects of non-determinism which can lead to non-reproducible simulation runs. Such factors include parallel and random processes, as well as limitations of the underlying hardware, such as CPU scheduling, or memory swapping on the limit of resource utilization, etc. To ensure reproducibility of experimental runs, we carefully considered and implemented the concept of *in vitro* simulation. That is, a simulation which controls all the aspects of the modeled system, or carefully accounts for those, which were abstracted away from. In particular, this means that the simulator has to have an ability to suspend and later resume the simulation process. Furthermore, it should have an ability to speed it up, or slow it down in response to e.g., resource utilization of the underlying hardware, so that race conditions and different results of process scheduling do not affect the simulation outcome. Finally, the random processes involved in the simulation must be also under the simulator's control so that the same sequences of random events are generated in two independent runs of the same simulation.

The vanilla MAS platform Aglobe is based on the assumption of full autonomy of agents which all run asynchronously in a truly decentralized fashion. This feature, while desirable in extremely large-scale simulations of air-traffic, turned out to be difficult to live with due to problems with the inherent non-determinism of the platform asynchronicity and the reliance of the simulation extension on discrete time ticking dynamics. In result, the need to run huge numbers of reproducible experiment runs turned out to hinge on the speed of simulation run execution and ability to make the runs deterministic on demand. To tackle this issue, we departed from the exclusive model of centralized discrete time ticks and implemented event-

based simulation mechanism [2]. This allowed the system to disrespect real-time constraints of the wall clock ticking mechanism and run the simulation as fast as possible given the available computational hardware resources (memory and CPU). However, at the same time the resulting Alite simulator still featured the ability to run at real-time simulation speed for demonstration purposes. In the Tactical AgentFly and Tactical AgentScout projects, the newly implemented event-driven Alite simulator enabled complete synchronization of the simulated processes and thus facilitated high level of control over the simulated environment.

4.3 Scripting and Agent Control

As already mentioned above, realistic evaluation of various coordination mechanisms requires modeling of realistic behavior of the various actors comprising the environment the simulated multi-robot team acts in. As already discussed in Section 3, our initial idea was to employ a state-of-the-art agent programming framework *POSH*. Early on, however, this choice turned out to be difficult due to the fact that the programming system is a research prototype and was not ripe enough for straightforward integration into the project code base. Besides that, our initial feasibility study showed that the use of the framework by inexperienced users, programmers involved in our projects, was rather problematic also due to difficult *a priori* conceptual framework underlying the system.

The initial solution to our problem was implementation of Lightweight Reactive Planner (LRP) [29], a proprietary and relatively simple hierarchical reactive-control engine. While the component served us well in the first iterations of the project, later it also turned out to be inflexible due to its extreme simplicity. In result, we moved to *Belief-Desire-Intention* (BDI) agents implemented in an agent-programming framework *Jazzyk* [27]. *Jazzyk* features a high-level of modularity with respect to integration with 3rd party and legacy systems and allows their straightforward integration into the system in the form of agents' knowledge bases. This allowed us to use heterogeneous knowledge reasoning engines, such as *Prolog*, or object-oriented databases within a single agent system and at the same time maintain a high level of control over agent's behaviors, which were implemented as *Jazzyk* situated reactive plans.

4.4 User Interface and Visualization

The last, but at the same time one of the most important technological parts of the whole mosaic of subsystems are user interfaces and visualizers. Visualization components facilitate various debug, as well as demonstration views on what is going on within the simulation. Graphical 3D demonstrators constituted important deliverables of the Tactical AgentFly and Tactical AgentScout projects. To facilitate vivid and believable presentation of the research results we were using a composition of various 2D and 3D visual representations.

The requirements on simulation visualization can be divided into two groups: i) global overview, and ii) informative details. At the same time, both types of visualizations had to be provided in 2D and 3D as appropriate with respect to the particular purpose. The approach to tackle this issue in the AgentFly project was based on *Java* 2D graphics and a protocol communicating with external 3D engine (namely *CrystalSpace* [14]). With regard to backward incompatibility of changes in newer versions of the 3D engine, this model was dropped. Later, still in the context of the project AgentFly, we implemented an in-house full-featured 3D and 2D visualization engine build on *Java3D* library [18] and later with several features based on Java native connections to *OpenGL* [21]. The engine uses concept of layers optionally showing various information from the running simulation. These layers are connected with data sources in the simulation by the Aglobe asynchronous messaging interface.

In the early phases of Tactical AgentFly and Tactical AgentScout projects, we re-used the working visualization engine from AgentFly. The engine was during the time extended only by additional 3D models and visualization layers for various trajectory and plan types (e.g, for UGVs and VTOLs).

However, with the already discussed departure from use of the AglobeX Simulation simulator, we were given an opportunity to improve upon the visualization technology and replaced the obsolescing visualization engine with *Java Monkey Engine* [20], a state-of-the-art *Java* 3D engine.

Additionally, we have created a simple 2D visualization engine primarily for debugging purposes of the new simulations and environments. The 2D engine was designed considering the common principles of Alite: flexibility, openness and modularity and became a universal tool for 2D visualizations of global overview of simulated environments. Unlike in the AgentFly project, we carefully crafted the system architecture so as to respect the principle of non-interference of visualization with the run of the simulation itself. This is mainly to preserve reproducibility of the *en masse* run experiments, which do not make use of the visualization code at all. The simplicity of the visualization engine also facilitated development of lightweight APIs, so that programmers could make use of it in a straightforward and flexible manner.

4.5 Alite

Alite *[ˈeɪlaɪt]* [1] is a software toolkit aimed at simplifying implementation construction (not only) of multi-agent simulations and multi-agent systems, such as those implemented in the context of the Tactical AgentFly and Tactical AgentScout projects in general. The objectives of the toolkit are to provide highly modular, flexible, and open set of functionalities defined by clear and simple APIs towards easy rapid prototyping and fast implementation. The toolkit does not serve as a predesigned framework or platform for a complex purpose, it rather associates number of highly refined functional elements, which can be variably combined and extended into a wide spectrum of possible systems.

The guiding principles underlying the Alite design are i) modularity, so that the system does not commit a developer to a particular definition of concepts such as *agent*, *environment*, etc., and ii) compositionality, so that the various components of the toolkit can be put together in a rapid and flexible manner. In result, Alite can be seen as an association of highly refined functional elements providing clear and simple APIs, so that relatively complex multi-agent simulation scenarios can be put together rapidly.

Alite agents have access to simple interfaces to the environment (sensors and actuators), while their internal lifecycle is not bound by any *a priori* philosophy. Additionally, they can make use of various types of communication middleware interfaces allowing to model various types of intra-agent communication (synchronous, asynchronous, peer-to-peer, broadcasting, multi-casting, etc.). Alite comes with libraries including various types of planners (reactive, deliberative) and multi-agent solvers (e.g., task allocators, solvers for problems, such as the distributed vehicle routing problem, etc.).

By its compositional nature, Alite provides means for both rapid prototyping, as well as high-level of elaboration tolerance of the implemented systems. E.g., once a simulation scenario, or a functional multi-agent system is put together from various components, application-level customizations and proprietary domain-specific mechanisms, it is very easy to replace one stock planner, or multi-agent solver by another one, as far as they share the underlying assumptions for their use.

Alite addresses the problem of MAS platform resilience in the face of a need to incorporate various *a priori* unknown future requirements by variability in composition of functional elements. The number of possible combinations include wide spectrum of system types, in contrast to a pre-designed frameworks as [31, 4, 13]. As multi-agent application's requirements evolve, the requirements on the agent platform itself are changing. Alite does not provide "a single platform for all", but rather offers an efficient way to build a platform that more precisely fits the particular needs of the MAS application under development. The application can utilize one or more functional elements. As of writing this chapter, Alite provides:

common-event-queue: a general implementation of a temporal event queue and temporal events (can be used for event-based simulations, agent message queues, etc.).

common-entities: a general description of any entity in the system. An entity is defined only by its name (represent agents, simulated embodiements, etc.).

common-capability-register: a general implementation of a simple register of possible capabilities provided by entities (usable for directory services, register of simulation components, etc.).

communication: a component of communication interfaces and basic message transports (includes direct and asynchronous message transport, protocol abstraction, abstraction of communication modes, etc.)

initialization: a component defining basic interfaces for initialization scripts and configuration (includes a config-reader based on Groovy)

environment: a component of interfaces defining basic elements for simulated worlds (includes state storages, and bases for sensors and actuator interfaces).

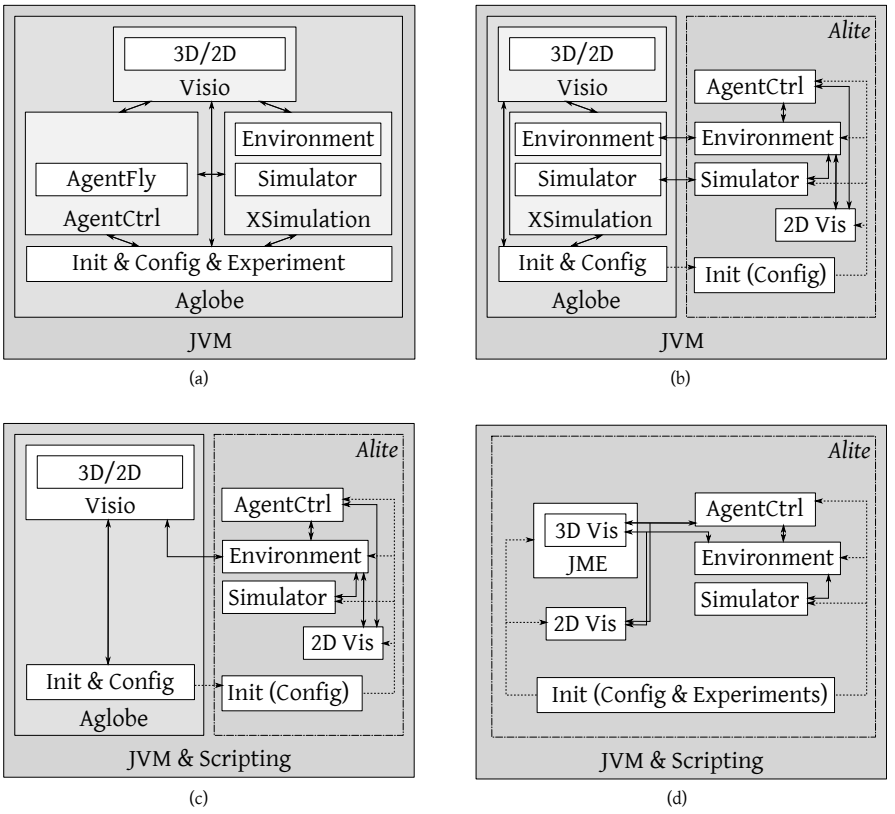

Fig. 6 Changes of the technological infrastructure over time. Initially, the Tactical AgentFly project (a) was fully implemented using the Aglobe platform with AglobeX Simulation (abbrev. *XSimulation*). The initial infrastructure underlying the first phase of the Tactical AgentScout project (b) already partially abandoned some parts of the initial architecture. In the third iteration, namely the second phase of the Tactical AgentFly project, the original Aglobe and XSimulation are employed solely for visualization and configuration purposes. Finally, the second edition of the Tactical AgentScout project was fully implemented using the Alite toolkit. Feature framing represents architectural composition, solid arrows represent coupling, and dashed arrows initialization. Alite does not contain any platform code, thereby it only associates the composition of various utilized technological features (dash-dotted block).

simulation: a component mediating event-based simulation (it is based on the common-event-queue and enriches it by temporal control).

visualization: a set of components containing various visualizers and wrappers to 3^{rd} party visualizing applications (includes 2D visualization, 3D visualization based on JME, wrapper to Google Earth, and others).

4.6 Architectural Changes of the Technological Infrastructure Over Time

In the previous sections, we have discussed changes in particular parts of the infrastructure underlying the project cluster. Figure 6 depicts the overall system architecture as it chronologically evolved in the four iterations of the project cluster (two phases of the two projects).

The initial architecture for first Tactical AgentFly project come out of AgentFly with all technological features based on Aglobe (see Section 3.3).

In the next project, the agent control (*AgentCtrl* in the Figure) mechanisms were replaced by simplified implementation in Alite. The implementation was tested in newly added 2D visualizer, environment and simulation. All these components were initialized by Alite initialization mechanism (abbrev. *Init*) using optionally configurable parts and experiment description. The two systems were integrated by injection of positions from the new environment into the original environment and using time synchronization messages between the two simulators. The original system henceforth acted as a simulator for AgentFly airplanes and 3D visualizer, while the Alite simulator handled the ground mission simulation.

In the third project iteration all parts of the original system excluding the 2D and 3D visualizers were already abandoned. Simulations, environments and agent control, together with tested algorithms were run in the new system. Scripting mechanisms were also introduces to simplify implementation of particular agent control algorithms and mission scripting. They enabled more general dynamic configuration and initialization. The integration of the two systems was based on propagation of the environment information (mainly positions of the simulated entities) into the original 2D/3D visualization through the visualization engine (Visio).

In the last project iteration, a new 3D visualizer based on *Java Monkey Engine* was implemented and allowed full shift from Aglobe-based system to more precisely fitting architecture covered by Alite toolkit.

5 Critical Analysis of the Experience and Lessons Learned

To conclude the report of the previous sections on the implemented agent-based technological infrastructure underlying the Tactical AgentFly and Tactical AgentScout project cluster, let us identify and summarize the main lessons we learned in the process.

The here described project in fact comprises four more or less technologically separate sub-projects: the two thematically related projects Tactical AgentFly and Tactical AgentScout, both implemented in two phases of one year duration each. It can be said that to a large extent almost any software project can be implemented with almost any toolkit, development framework and programming language and following any kind of software engineering methodology. At the same time, however, some tools and methodologies fit some application domains better and lead to more efficient and more natural or straightforward design and implementation pro-

cess. Due to its structure and 4-year duration, we were given a unique opportunity to learn from past experiences, iteratively re-implement parts of the system and experiment with various configurations of its components and thus improve upon the previous experiences. In result, we did not only identify the most important characteristics an ideal technological platform for research of coordination mechanisms in multi-robot missions should feature, but we were also allowed to develop, implement, evaluate and even re-design various components of the technology as well.

From the technological perspective, our work can be seen as a four year long experiment with development of MAS platform, in particular including various simulators for mid and large-scale multi-robot systems comprised of heterogeneous robotic assets, such as CTOLs, VTOLs, UGVs and even simulated humans, different 2D and 3D visualizers, user interfaces and approaches to configuration and *en masse* experiment execution. We have learned that the two most important features around which the design decisions regarding the technological infrastructure revolve are the ability to support *rapid prototyping* and technologies enabling *efficient and fast empirical evaluation* of the implemented research algorithms. Generally, the discussion of architectural changes the platform underwent in the course of the project clearly shows that in time, we moved from an architecture embracing an instance of "one size fits all" philosophy towards open modular design consisting of a number of heterogeneous building blocks, which can be composed into an application-specific design. In the following subsections, we discuss the individual aspects of the system and how the identified requirements on rapid prototyping and efficient empirical evaluation influenced their design.

5.1 Multi-agent Platform

A software platform for implementation and run-time control of the individual agents running within the system is the most important component of the overall technological infrastructure. The project objectives dictated a need to compose various realistic simulation scenarios featuring different types of agents acting in various time paces, etc., so that we are able to test various research algorithms for multi-agent coordination. The sheer range of the scenario variety and at the same time the push towards rapid prototyping and fast research-implementation-evaluation turnover often resulted in clashes with the internal structure and the underlying philosophy of the currently employed multi-agent platform.

The lesson learned on this front is that often learned also in mainstream industrial software engineering. Namely, that in development of multi-purpose technologies, such as the one discussed here, it is often the case that an *a priori* application design philosophy, while being beneficial in the early stages of the project, tends to stand in the way of the development process in later stages. The stream of new requirements, not foreseen at the time of platform design, may sometimes diverge and even contradict some of the underlying principles of the design philosophy of the platform. An example of such was the inherent assumption of the Aglobe MAS platform with its AglobeX Simulation extension that the MAS application components should be

modeled exclusively as agents, which communicate asynchronously. While this optics is well applicable in many applications, this design philosophy has vast consequences on the complexity of the application design, easiness of system debugging and reproducibility of experiment runs. Often implementing the simulator itself is more efficient using plain object-oriented programming principles with simple direct call method invocation, rather than modeling even the system components as agents possibly running on different network nodes. Instead of working around this feature and thus abusing the underlying philosophy, in this case we rather decided to depart from this principle altogether and thus sped up the scenario development process.

In essence, the core lesson described above is that different MAS applications require different philosophies and technological features and the developers should be rather supported in compositional application development. In particular, this means that our resulting MAS technological platform based on the Alite toolkit keeps the MAS design open and comprising a number of complementary building blocks with clear interfaces keeps the door open to swift future redesigns of the application. Such crucial building blocks include various approaches and implementations of features, such as message passing asynchronicity, platform distribution among a number of network nodes, code mobility, agent lifecycle management, etc. Including a particular choice of these features in a single monolithic technological platform and its later extensions often tend to lead to software bloat and design decisions which constrain developers not because of some crucial issue, but due to respecting various interdependencies among the implemented features themselves. The lesson learned in the Tactical AgentFly and Tactical AgentScout cluster is that this situation should be avoided as much as possible. The shift the here described technological infrastructure underwent can be best described as

a move from a relatively rigid general-purpose MAS platform towards a toolkit facilitating rapid application-specific MAS platform construction.

5.2 Environment Simulation and Scenario Modeling

As already discussed in the previous subsection the range of simulation scenarios and modeled missions in our projects was rather large. Different scenarios aimed at evaluation of different multi-agent coordination algorithms often required very different environment features. While at times the environment could be modeled as a coarse grained graph structure with only interpolation of physical movements on the ground in the simulation, often we also needed high-fidelity environmental features including detailed physical landscape and building models. One of the lessons learned in the projects of the Tactical AgentFly and Tactical AgentScout cluster is that the stress on modularity and composability is crucial also with respect to an environment model, as well with respect to the particular model of time the simulator employs.

One of the most important parts of the simulation process is time handling. The underlying philosophy of the AglobeX Simulation simulator is a simulation model

based on synchronous, constant-delay time ticks, which are asynchronously distributed to the simulated agents and other parts of the simulation. There are two common problems with such understanding of time counting in a simulation and both share their roots in the efficiency of the dependent simulation process. One is the temporal homogeneity and the other is causal homogeneity. While for many applications, the fine grained time model is directly employable, in our simulations it turned out to be rather inefficient. In particular, when a simulation contains agents working at various speeds, resp. being idle with different periods of time, the slowest time delay between two simulation ticks must correlate with the fastest agent in the system. This however leads to inefficiencies when an agent which normally exploits the fine grained time ticks becomes idle for a longer period of time. Essentially, nothing happens in the simulation, nevertheless the simulator is still forced to process all the homogeneous minuscule time ticks in between. Furthermore, synchronous time ticking simulations are inefficient for applications comprising large numbers of otherwise causally independent processes (e.g., flight of a UAV and activities of a ground soldier). In discrete time ticking simulations, such processes can become unnecessarily synchronized.

In the course of the projects development, we moved from synchronous time ticking to event-driven simulations. Yet, we were careful enough to maintain the ability of the simulator to switch to discrete time ticking whenever necessary. This allowed us to significantly speed up simulation time of scenarios which can be, without loss of generality, implemented in event-driven simulations. In result, we were able to shorten the research-implementation-evaluation cycle and ultimately also speed up the project completion. Additionally, in our experience, employment of event-driven simulations also simplifies the code required for implementation of agent deliberation and its interaction with the environment.

Finally, the requirement of reproducibility of simulation runs led us to attempts to realization of the concept of *in vitro* simulations. Since we were aware of this problem right from the beginning of the project it did not manifest itself in a significant manner in the course of our work. In simulations of multi-robot systems, the realization of *in vitro* simulators can become of an issue with growing demands on scalability of the system. While having all the aspects of the simulation under control of the simulator in a deterministic and synchronized manner is possible and manageable, the trade-off with growing scale of the system is its worse run-time performance, as well as possibly worse elaboration tolerance issues of the implemented system. The simulator can simply become too large component of the system featuring too many characteristics with underlying assumptions which significantly constrain simple and straightforward implementation of other parts of the system, e.g., agent behavior control.

5.3 Experiments and Configuration

One of the important lessons learned in our experience during the work on the twin projects is that in systems, where the variation of future scenarios of its application cannot be easily foreseen, high level of configurability has to be implemented.

Rather than relying on pre-defined configuration schemes, such as XML files. Our decision to move to integration of full-fledged dynamic language interpreters turned out to be a good one. The cost of run-time configuration compilation at the beginning of the simulation run is negligible with respect to the overall simulation execution time. Furthermore, the gained benefit of practically limitless configurability of the simulation supports the rapid development principle and contributes to high level of modularity of the resulting simulations. These can be essentially constructed and initialized in the configuration scripts. In result, the tasks of implementation of simulation components and scenario configuration become clearly separated, what allowed high flexibility in the development process.

The second important lesson to be learned from the work on the Tactical AgentFly and Tactical AgentScout project cluster is the stress on speeding up the research-implementation-evaluation turn-around. While the increased software platform modularity contributes to speeding up the first components of the cycle, attempts to speed up the runs of experimental setups improves upon the latter parts. It is also important to realize that not only does faster experiments execution shorten the time needed for evaluation of the algorithms under investigation, it speeds up debugging and implementation part of the cycle as well.

In our particular project, the implementation of an event-driven simulation framework led to higher control over execution time of the simulation runs and speeding up the experiments evaluation. This experience however should be considered carefully in the context of our projects. For instance scenarios which would require extremely fine grained synchronous time would probably not benefit from implementation of event-driven simulation loop and also respecting of the requirements on implementation of *in vitro* simulations might become burdensome.

5.4 Agent Behavior Control

In terms of the agent behavior control, again there are two issues to consider. On one hand it is the expressive power of the employed reasoning framework and on the other hand it is the modularity and integrability of its implementation. Furthermore, while some scenarios require quite a heavy-weight deliberation mechanism, in other ones, plain reactive control is sufficient. The lesson learned on this front leads to realization that, if flexibility with respect to future applications is an issue, perhaps rather than employing a relatively heavy-weight agent deliberation framework (e.g., imposing BDI-style agent architecture), it might be more beneficial to use a simpler, but more general toolkit. I.e., it might be more flexible to invest an effort in learning and gaining experience with simple, but extensible deliberation models, such as e.g., finite state machines, rather than commit all future agent deliberation implementations to a particular intelligent agent architecture.

In our case, the choice of the *Jazzyk* language interpreter turned out to be a suitable choice for the more advanced simulation scenarios. The language is extremely simple, but at the same time it allows for compositional programming of agent behaviors. Additionally, it directly supports integration with the underlying simulator, as well as various knowledge representation technologies.

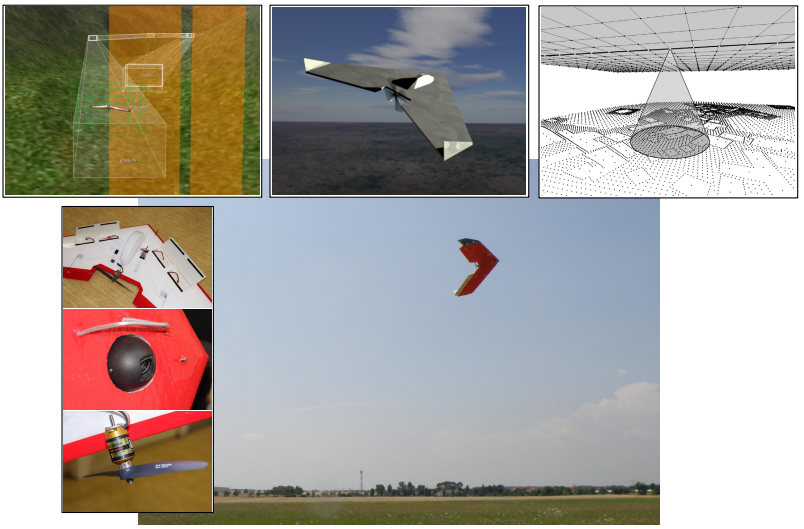

Fig. 7 Mixed-reality in the project AgentFly-In-Air, together with details of a *Procerus Unicorn UAV* test aircraft

5.5 User Interface and Visualization

While user interfaces and visualization do not often stand in the focal point of prototyping in research projects, at this point, we would promote this issue and encourage rich visualization of simulations, especially in robotics research. Not only do realistic 3D scene visualizers provide attractive demonstrations, they also constitute a plausible basis for evaluation of the algorithms by non-experts. In our case, presentation of live 3D demonstrators and video sequences captured from various simulation runs proved to be beneficial in communication with both, the project sponsor, as well as 3^{rd} parties.

The state of the art in open-source 3D scene modeling and animation technologies is at a stage, where straightforward use of the tools by non-expert programmers is easy. Having said that, it is of course important to design the interfaces to the 3D world visualization in a manner, which again stresses rapid prototyping of the resulting scenarios. In our case, integration of various types of 2D and 3D visualization engines in the Alite toolkit turned out to be relatively straightforward and cheap in terms of the involved implementation effort.

5.6 Towards AgentFly-In-Air

After completing the projects Tactical AgentFly and Tactical AgentScout, we collected a significant body of research results, experience and advanced in-house technological platforms for development of mid- and large- scale simulations of multi-robot systems. A natural step along the above described research track was to move

Fig. 8 *Procerus Unicorn UAV* test aircraft with accessories

closer towards deployment of the developed algorithms to real hardware robots. The project AgentFly-In-Air (see Figure 7) aims exactly in this direction. The 18 months long project was started in mid 2011 and as of writing this chapter it is in active development to be completed by the end of 2012.

The foreseen demonstration scenario will involve a number of real, as well as simulated aircrafts performing a continuous surveillance of a pre-defined area and tracking of a number of mobile targets on the ground. Ideally, these will be embodied by real human subjects carrying GPS devices connected to a central simulation, resp. evaluation engine. By this experiment, firstly, we will demonstrate applicability and portability of the multi-agent coordination algorithms developed in the context of the above described projects to real hardware, and secondly, provide a proof-of-concept for the idea of *mixed-reality multi-agent simulation*. I.e., such, that besides a number of simulated agents, it will partly run in reality and will include real world robotic assets.

The imperative to port and deploy a selected subset of the algorithms developed in the context of its precursor projects in real hardware brought new requirements on the design of the overall system and raised new challenges as well. On the front of the application level functionality, the main issues are rooted in decentralization of the coordination algorithms and scalability of the algorithms to the on-board hardware and its resource limitations w.r.t. CPU power, battery usage, communication bandwidth, etc. Complementary to that, availability of only limited number of hardware robots, in our case two *Procerus Unicorn* aircrafts (see Figure 8), requires technology enabling development and execution of mixed simulations. That is experiments, in which parts of the overall system are simulated, but significant parts are implemented in real hardware. In result, as a part of the project, we develop a technology allowing to model multi-agent systems in which all agents are equal w.r.t. their run-time characteristics and interfaces to the physical environment

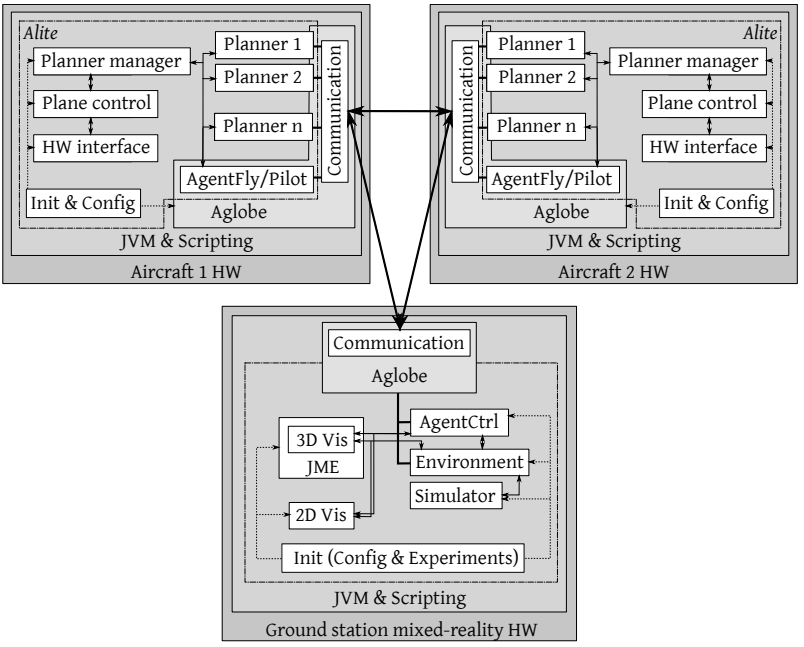

Fig. 9 Planned architecture for the project AgentFly-In-Air. The ground station runs a simulation based on Alite toolkit. Communication with the real airplanes is mediated by Aglobe. The aircraft logic is a composition of new future Alite modules. As planners, the modules of predecessor projects will be used and as a terminal planner the pilot agent of AgentFly system with path planning and collision avoidance will be integrated.

regardless of whether the agents themselves and the environment are simulated, or part of the real world. In result, the system should provide high level of modularity and facilitate gradual steps from fully simulated multi-robotic system through mixed simulation ultimately to pure hardware deployment.

The opportunity to employ the technological infrastructure developed in the context of Tactical AgentFly and Tactical AgentScout in a new setting embedded partly in real hardware, allows us to once again reconsider and critically analyze the use of the above discussed software components. One of the important issues arising from the need to both model, as well as to incorporate true robotic assets in the evaluation scenarios is the already discussed asynchronicity of a message delivery. While in the purely simulated scenarios, for the sake of respecting the *in vitro* simulation principles, we abandoned the real-world asynchronous inter-agent communication model. In the project AgentFly-In-Air, we have to come back to it. In result, we consider to once again employ the Aglobe MAS platform as the underlying technology, but instead of using it to manage also agents' lifecycles, we will treat and use it purely as a decentralized communication middleware (see Figure 9). After all, a fine-grained management of agent lifecycle in terms of control of threads and processes running the logic of the individual agents becomes pointless

in hardware multi-robot systems. We are of course aware, that by this step we will loose the ability to execute evaluation in fully controlled environments akin to *in vitro* simulations, except with hardware robots.

In terms of environment modeling, we will face tasks to integrate real world sceneries into the simulator. It means that to facilitate execution of the foreseen project demonstrator, we will have to be able to model a particular test ground within the simulation, including its landscape, terrain features, traffic network, buildings, etc. Implementation of new components facilitating various aspects of the mixed simulations, such as the landscape modeling, within the Alite toolkit will pose new implementation challenges in the project.

6 Future Perspectives and Final Remarks

With the growing complexity of multi-agent applications and environments, in which they are deployed, there is a need for development techniques that would allow for early testing and validation of application design and implementation. This is particularly true in cases, where the developed multi-agent application is to be closely integrated with an existing, real-world system of multi-agent nature. Our work, in the context of the here described project cluster including Tactical AgentFly, Tactical AgentScout and AgentFly-In-Air projects, aims exactly at this objective.

Our experience in the course of the years 2008–2011 boils down to realization that due to the extremely wide range of scenarios for early testing and validation of algorithms, the scenario development and simulation technological infrastructure should remain extremely modular and feature a compositional architecture.

In the light of this lesson, we feel that rather than in development of special-purpose MAS platforms, the open challenges for the community lie in investigation of programming-framework-independent methodological guidelines for engineering of such multi-agent based software artifacts. Of course we understand that no unified MAS development methodology would fit the requirements of various application domains, however collecting and learning from experience with building such systems is still a realm, in which not enough report are produced. Similarly to the MAS platform lesson highlighted in the critical analysis of the projects implementation, perhaps rather than aiming at a unified MAS development methodology (platform), our goal should rather be a set of rudimentary building blocks out of which such application-specific methodologies can be easily constructed on purpose.

To conclude the chapter, we would like to draw the attention to the notion of *mixed-reality simulation* and the methodological guidelines to be followed in development of such systems. Development and deployment of such complex multi-agent systems is a challenging task. Large numbers of spatially distributed active entities characterized by complex patterns of mutual interaction and feedback links give rise to dynamic, non-linear emergent behaviors, which are very difficult to understand, capture and, most importantly, control. We argue that because of the complexity of

the above-described types of applications, it is no longer possible to develop such systems in a linear, top-down fashion, starting from a set of requirements and proceeding to a fully developed solution. Instead, more evolutionary, iterative methodologies are needed to successfully approach the problem of development of complex multi-agent systems.

In [28], we make first steps towards tackling this open challenge and give a preliminary outline of the *simulation-aided design of multi-agent systems (SADMAS)* approach. SADMAS is a development methodology relying in its core on the exploitation of a series of gradually refined and accurate simulations for testing and evaluation of intermediary development versions of the engineered application. In particular, we propose and argue in favor of using a series of *mixed-mode simulations*, in which the implemented application is evaluated against a partly simulated environment. Over time, the extent of the simulation will be decreasing until the application fully interacts with the target system itself. We argue that this approach could help accelerate the development of complex multi-agent applications, while at the same time keeps risks and costs associated with destruction or loss of the tested assets low. We believe that more research in this area is needed in order to better understand the core problems and issues stemming from deployment and evaluation of embodied multi-agent systems in real world scenarios, be it in industrial settings, or in military scenarios, such as those described earlier in this chapter.

Acknowledgements. The Agent Technology Center and the authors acknowledge the support of the here reported work reported by numerous projects awarded to ATG in the course of the years 2008–2011. The project Tactical AgentFly was supported by the grant W911NF-08-1-052 of U.S. Army CERDEC. The project Tactical AgentScout was supported under the grants BAA 8020902.A/W15P7T-05-R-P209 and W911NF-10-1-0112 by U.S. Army CERDEC. Additionally, the work was also partly supported by the grant MSM6840770038 of the Ministry of Education, Youth and Sports of the Czech Republic.

References

1. Alite project website, http://agents.cz/projects#alite
2. Banks, J., Carson, J., Nelson, B.L., Nicol, D.: Discrete-Event System Simulation, 4th edn. Prentice Hall (December 2004)
3. Belecheanu, R.A., Munroe, S., Luck, M., Payne, T., Miller, T., McBurney, P., Pěchouček, M.: Commercial applications of agents: lessons, experiences and challenges. In: Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS 2006. ACM, New York (2006)
4. Bellifemine, F., Caire, G., Poggi, A., Rimassa, G.: JADE a white paper
5. Bordini, R.H., Dastani, M., Dix, J., El Fallah-Seghrouchni, A.: Multi-Agent Programming: Languages, Platforms and Applications. Multiagent Systems, Artificial Societies, and Simulated Organizations, vol. 15. Springer (2005)
6. Borst, C., Wimbock, T., Schmidt, F., Fuchs, M., Brunner, B., Zacharias, F., Giordano, P.R., Konietschke, R., Sepp, W., Fuchs, S., Rink, C., Albu-Schaffer, A., Hirzinger, G.: Rollin' justin - mobile platform with variable base. In: IEEE International Conference on Robotics and Automation, ICRA 2009, pp. 1597–1598 (May 2009)

7. Bošanský, B., Lisý, V., Jakob, M., Pěchouček, M.: Computing time-dependent policies for patrolling games with mobile targets. In: Proceedings of AAMAS 2011(May 2011)
8. Brom, C., Gemrot, J., Bída, M., Burkert, O., Partington, S.J., Bryson, J.J.: Posh tools for game agent development by students and non-programmers, pp. 126–133. University of Wolverhampton (2006)
9. Bryson, J.J.: The Behavior-Oriented Design of Modular Agent Intelligence. In: Kowalczyk, R., Müller, J.P., Tianfield, H., Unland, R. (eds.) NODe-WS 2002. LNCS (LNAI), vol. 2592, pp. 61–76. Springer, Heidelberg (2003)
10. Chrpa, L., Komenda, A.: Smoothed hex-grid trajectory planning using helicopter dynamics. In: Proceedings of International Conference on Agents and Artificial Intelligence (ICAART), vol. 1, pp. 629–632. SciTePress (2011)
11. Chrpa, L., Novák, P.: Dynamic Trajectory Replanning for Unmanned Aircrafts Supporting Tactical Missions in Urban Environments. In: Mařík, V., Vrba, P., Leitão, P. (eds.) HoloMAS 2011. LNCS, vol. 6867, pp. 256–265. Springer, Heidelberg (2011)
12. Clojure project website, http://clojure.org/
13. Cougaar project website, http://www.cougaar.org/
14. CrystalSpace project website, http://www.crystalspace3d.org/
15. Fukushima robot operator writes tell-all blog (news article), http://spectrum.ieee.org/automaton/robotics/industrial-robots/fukushima-robot-operator-diaries
16. Groovy project website, http://groovy.codehaus.org/
17. High-tech goes into action in disaster zone (news article), http://www.msnbc.msn.com/id/9240563/ns/technology_and_science-science/t/high-tech-goes-action-disaster-zone/
18. Java3D project website, http://www.oracle.com/technetwork/java/javase/tech/index-jsp-138252.html/
19. JBullet project website, http://jbullet.advel.cz/
20. jMonkeyEngine project website, http://jmonkeyengine.com/
21. JOGL project website, http://jogl.dev.java.net/
22. Komenda, A., Novák, P.: Multi-agent plan repairing. In: Decision Making in Partially Observable, Uncertain Worlds: Exploring Insights from Multiple Communities, Proceedings of IJCAI 2011 Workshop, pp. 1–6. AAAI Press (2011)
23. Komenda, A., Novák, P., Pěchouček, M.: Decentralized multi-agent plan repair in dynamic environments. In: Proceedings of AAMAS 2012 (2012)
24. Lisý, V., Bošanský, B., Pěchouček, M.: Anytime algorithms for multi-agent visibility-based pursuit-evasion games. In: Proceedings of AAMAS 2012 (2012)
25. Mining sector embraces sci-fi future (news article), http://finance.ninemsn.com.au/newsbusiness/aap/8259880/mining-sector-embraces-sci-fi-future
26. Montemerlo, M., Thrun, S., Dahlkamp, H., Stavens, D., Strohband, S.: Winning the darpa grand challenge with an ai robot. Artificial Intelligence 21, 982 (2006)
27. Novák, P.: Jazzyk: A Programming Language for Hybrid Agents with Heterogeneous Knowledge Representations, pp. 72–87. Springer, Heidelberg (2009)
28. Pěchouček, M., Jakob, M., Novák, P.: Towards Simulation-Aided Design of Multi-Agent Systems. In: Collier, R., Dix, J., Novák, P. (eds.) ProMAS 2010. LNCS, vol. 6599, pp. 3–21. Springer, Heidelberg (2012)
29. Pěchouček, M., Jakob, M., Semsch, E., Pavlíček, D., Eliáš, V.: Intelligent software agent control of combined UAV operations for tactical missions - final report. Technical report, Agent Technology Center, Department of Cybernetics, FEE Czech Technical University in Prague (2009)

30. Semsch, E., Jakob, M., Pavlíček, D., Pěchouček, M., Šišlák, D.: Autonomous uav surveillance in complex urban environments. In: McGann, C., Smith, D.E., Likhachev, M., Marthi, B. (eds.) Proceedings of ICAPS 2009 Workshop on Bridging the Gap Between Task and Motion Planning, Greece, pp. 63–70 (September 2009)
31. Šišlák, D., Rollo, M., Pěchouček, M.: *A*-Globe: Agent Platform with Inaccessibility and Mobility Support. In: Klusch, M., Ossowski, S., Kashyap, V., Unland, R. (eds.) CIA 2004. LNCS (LNAI), vol. 3191, pp. 199–214. Springer, Heidelberg (2004)
32. US military's UAV missions increasing (news article), http://www.armedforces-int.com/news/us-militarys-uav-missions-increasing.html
33. Vokřínek, J., Komenda, A., Pěchouček, M.: Agents towards vehicle routing problems. In: van der Hoek, W., Kaminka, G.A., Lespérance, Y., Luck, M., Sen, S. (eds.) AAMAS 2010: Proceedings of the Ninth International Conference on Autonomous Agents and Multi-Agent Systems, Toronto, Canada, pp. 773–780. IFAAMAS: Internatioal Foundation for Autonomous Agents and Multiagent Systems (May 2010)
34. Vokřínek, J., Komenda, A., Pěchouček, M.: Cooperative agent navigation in partially unknown urban environments. In: PCAR 2010: The Third International Symposium on Practical Cognitive Agents and Robots. Proceedings of the AAMAS 2010 Workshops, Toronto, Canada, pp. 46–53. IFAAMAS: International Foundation for Autonomous Agents and Multiagent Systems (May 2010)
35. Vokřínek, J., Novák, P., Komenda, A.: Ground Tactical Mission Support by Multi-agent Control of UAV Operations. In: Mařík, V., Vrba, P., Leitão, P. (eds.) HoloMAS 2011. LNCS, vol. 6867, pp. 225–234. Springer, Heidelberg (2011)
36. Willow Garage website, http://www.willowgarage.com/
37. Winikoff, M.: Implementing commitment-based interactions. In: Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS 2007, pp.128:1–128:8. ACM, New York (2007)
38. Wooldridge, M., Jennings, N.R.: Intelligent agents: Theory and practice. Knowledge Engineering Review 10, 115–152 (1995)

Chapter 8
On the Development of Mobile Agent Systems for Wireless Sensor Networks: Issues and Solutions

Giancarlo Fortino and Stefano Galzarano

Abstract. Due to the growing exploitation of wireless sensor networks (WSNs) for enhancing all major conventional application domains and enabling brand new application domains, the development of applications based on WSNs has recently gained a significant focus. Thus, design methods, middleware and frameworks have been defined and made available to support high-level programming of WSN applications. However, even though many proposals do exist, more research efforts should still be devoted to the definition of WSN-oriented methodologies and tools fully supporting the development lifecycle of WSN applications. In this chapter, we promote the use of the mobile agent paradigm for the development of WSN applications and, specifically, describe issues and solutions for the development of mobile agent systems on resource-constrained wireless sensor platforms. In particular we discuss about the design of MAPS (Mobile Agent Platform for Sun SPOTs) and TinyMAPS, our Java-based mobile agent systems for WSNs, which enable agent-oriented development of WSN applications. In particular, while MAPS can run on the capable SunSPOT sensor devices, TinyMAPS is a version of MAPS tailored for more constrained Java-based sensor platforms such as Sentilla JCreate. An analysis of MAPS and TinyMAPS is provided showing analogies and differences among the two platforms. Finally a comparison of MAPS with AFME, another Java-based mobile agent system running on SunSPOT and based on a different architecture and programming model, is presented.

Giancarlo Fortino · Stefano Galzarano
DEIS - University of Calabria,
Rende (CS), Italy
e-mail: g.fortino@unical.it,
sgalzarano@deis.unical.it

M. Ganzha & L. C. Jain (Eds.): Multiagent Systems & Applications, ISRL 45, pp. 185–215.
springerlink.com

1 Introduction

Advances in micro-electro-mechanical systems (MEMS) technology, wireless communications, and digital electronics have enabled the development of low-cost, low-power, multifunctional sensor nodes that are small in size and can communicate over short distances in an ad-hoc manner. Such nodes are, in general, characterized by constrained computing and communication capabilities. A given number of such cooperating sensor nodes can be organized and deployed as a wireless sensor network (WSN) [11]. The development of applications for WSNs requires not only the same middleware/programming support required by conventional distributed applications but also the fulfillment of additional requirements specific to WSNs [5]. In particular, middleware support for conventional distributed applications includes:

- shielding application developers from low-level platform-specific details;
- reusable pattern-oriented frameworks, rather than (re)building software monolithically for each application;
- higher-level network-oriented programming abstractions that better match distributed application requirements;
- a wide array of non-functional services, such as logging, deployment management and security that have been proven necessary to operate effectively in a networked environment.

Moreover, specific WSN features that have to be fully supported are:

- resource-constrained nature of WSNs in terms of processing and memory capabilities, and battery life;
- WSN nodes are usually dynamic and mobile in nature instead of being static and fixed as in traditional distributed systems;
- differently from traditional distributed systems, WSN nodes usually incorporate both application-level functions and the management of low-level aspects related to mobility, routing and security.

Several middleware architectures based on different models (database, macroprogramming, event-based, virtual machine, etc) have been to date proposed to support the development, deployment, execution, and maintenance of sensor-based applications [34]. Nevertheless, considering the commonalities that bind the intrinsic properties of WSNs with those of agents [38], agent-based middleware could be more effective in the context of WSNs than middleware based on other models. It is therefore reasonable to wonder whether agent technology can effectively support the construction of WSN applications. The paradigm of agent-oriented software engineering (AOSE) is argued to be well suited for developing complex software systems in distributed and dynamic environments [28, 23]. In particular, agent technology already offers several approaches for the design and implementation of agent-based sensor applications: (i) the development of a multi-agent application atop non agent-oriented middleware for WSNs [34]; (ii) the development of agent-based middleware that supports agent-based programming [33]; and (iii) the

extension of an existing multi-agent platform with middleware capabilities for WSNs (if possible) [38].

In this chapter, we first provide an overview of mobile agent-based computing in WSNs from several perspectives: network routing, data dissemination and fusion, in-network coordination, programming frameworks, high-level system architectures and applications (see Section 2). In Section 3, we discuss the major challenges for the development of mobile agent systems on resource-constrained sensor platforms and list the main requirements that should be addressed in their development. On the basis of such requirements, the design of the MAPS (Mobile Agent Platform for Sun SPOT) [9] and TinyMAPS [8] frameworks are described in Section 4. A comparison among the current Java-based MASs is discussed in Section 5. Moreover, quantitative performance evaluations between MAPS and TinyMAPS and between MAPS and AFME (Agent Factory Micro Edition) are also provided. In Section 6, both lessons learned and open challenges are delineated. Finally, concluding remarks and future trends in the field are presented.

2 Background and Related Work

Since the mobile agent-based paradigm has fully demonstrated its effectiveness in conventional distributed systems as well as in highly dynamic distributed environments, there are good motivation to prove that they can also effectively deal with the programming and management issues that WSNs have posed. As a confirmation, in the last years agent technology has been successfully used in WSNs at different levels (application, middleware, network) [33]. In particular, the main agent-oriented research efforts have been to date devoted to the following WSN research themes: network routing, data dissemination and fusion, in-network coordination, programming frameworks, high-level system architectures and applications. In the following subsections an overview of some of the main outcomes related to such themes will be presented.

2.1 Network Routing

Several agent-oriented techniques have been defined to support efficient routing in WSNs. Most of them are based on mobile agents that are able to roam across the sensor nodes, performing routing tasks.

An interesting routing technique based on mobile agents is *rumor routing* [13] that allows for routing queries to nodes that have observed a particular event (i.e. a localized phenomenon detected by some sensor node/s). The rumor routing algorithm aims at lower energy consumption than algorithms that flood the whole network with query or event messages. The main idea is that mobile agents *a priori* create paths leading to event nodes as the events occur; later queries are sent on random walks until they find one of the created paths, and then route along the path to event nodes.

In [21] authors propose a solution where mobile agents are created whenever a source node decides to send a data packet to the sink. Agents are then responsible for carrying the data through the network. After reaching the destination, the agent delivers the data to the application and then dies. After arriving at a node, the agent checks a forwarding table available at the node with the possible next hops, including such nodes respective costs and energy levels. Based on that information, agents take a decision. Since energy levels are depleted as agents use a given path, future agents might take more expensive paths that happen to have more energy available, achieving some degree of load balancing. Moreover, agents could negotiate and aggregate their data when they ”meet” in the network, possibly becoming one single agent after such aggregation.

2.2 Data Dissemination and Fusion

Several data fusion and dissemination schemes based on mobile agents have been to date proposed. In [29], authors review and evaluate the most representative mobile agent-based middleware proposals for autonomic data fusion tasks in WSNs, highlighting their relevant strengths and shortcomings. They classify such research proposals in five main categories: single mobile agent-based, multiple mobile agent-based, autonomic data fusion in clustered WSN architectures, hardware based and combined multiple mobile agent/stationary agents-based autonomic data fusion. In [15] mobile agents are used to disseminate data. According to client/server architectures, data at multiple sources is transferred to a destination whereas, according to the mobile agent paradigm, a task-specific mobile agent traverses the relevant sources to gather data and disseminate them according to specific policies. Many inherent advantages (e.g. scalability, extensibility, energy awareness, reliability) of the mobile agent architecture make it suitable for WSNs than the client/server architecture. Mobile agents can be exploited at three levels (node level, task level, and combined task level) to reduce the information redundancy and communication overhead.

2.3 Energy-Aware Coordination

Within application domains involving low-power, wireless devices physically distributed over an environment to acquire and integrate information, one of the main challenges to face with is to coordinate the activities of such devices in order to achieve good system-wide performance. Moreover, several constraints have to be considered: specific constraints on each device (e.g. limited power, communication and computational resources), the limitation for each of them to be able to communicate with only its local neighborhood and the need for a decentralized approach such that there is no central point of failure and no communication bottleneck. The problem of performing decentralized coordination of low-power devices is addressed in [19] by considering the generic problem of maximizing social welfare within a group of interacting agents. Each agent interacts locally with a number of other

agents such that the utility of an individual agent is dependent on its own state and the states of these other agents. In particular, a novel representation of the problem, through a cyclic bipartite factor graph composed of variable and function nodes (representing the agents' states and utilities respectively), is proposed. Such descriptive model allows using an extension of the sum-product algorithm (specifically the max-sum algorithm), which is adopted, along with a local decentralized message passing, to generate approximate solutions to the global optimization problem. It is shown that this approach has a communication cost (in terms of total messages size and, consequently, in energy consumption) that scales very well with the number of agents in the system because the complexity of the calculation that each agent performs depends only on the number of neighbors that it has and not on the total size of the network.

2.4 System Architectures, Services and Applications

Mobile agents have been also exploited to design WSN system architectures and develop services and applications based on WSNs. In [14, 22] authors propose mobile agents for WSN applications and, specifically, decompose the agent design functionality into four components: architecture, itinerary planning, middleware system design, and agent cooperation. This decomposition covers low-level to high-level design issues and facilitates the creation of a component-based and efficient mobile agent system for a wide range of applications. With reference to applications, a measurable bandwidth saving can be obtained either when large amounts of data are locally processed by mobile agents, or when the deployment of a programmable approach enabling task autonomy is required. To this purpose, an efficient design for the core components is required to support the scheme being followed by the agent-based application when dealing with a particular type of problem. Similarly, the WSN application has a direct influence on the type of communications mechanism employed by the mobile agent system to perform its task efficiently. However, their applicability mainly is warranted not only by the overall energy savings they introduce, but also by the extra flexibility they offer when coping with frequent and/or unexpected aspects of the event being sensed that other types of approaches are unable to address efficiently.

In [16] the MAWSN (Mobile Agent-based WSN) architecture for data processing/aggregating/concatenating in a planar sensor architecture is proposed. MAWSN can exhibit better performance than client/server communications in terms of energy consumption and packet delivery ratio. However, MAWSN has a longer end-to-end latency than client/server communications in certain conditions.

Mobile agents have also been applied for location tracking services based on WSNs. The goal is to monitor the roaming path of a moving object through wireless sensor nodes disseminated on an environment. While similar to the problem of location update in personal communication service networks, it is more challenging as there are no central control mechanism and backbone network and the communication bandwidth is very limited. In [37], a mobile agent-based protocol for location

tracking is proposed. Once a new object is detected, a mobile agent is initiated to track the roaming path of the object. The agent follows the object by moving to the sensor closest to the object. Moreover, the mobile agent may invite some nearby sensor agent to cooperatively locate the object and inhibit other irrelevant sensor agents from tracking the object. As a result, the communication and sensing overheads can be greatly reduced.

Finally, in [32] an energy-efficient, fault-tolerant approach for collaborative signal and information processing (CSIP) among multiple sensor nodes using a mobile agent-based computing model, is proposed. The performance of such a model is compared with the classic client/server-based model with respect to the execution time and energy consumption perspectives through both simulation and analytical study. Results indicate that in the context of sensor networks where the number of sensor nodes is very large, the communication bandwidth is considerably low, and the energy resource is contingent, the mobile-agent-based computing model is more suitable for conducting collaborative processing.

2.5 Programming Frameworks

Generally speaking, MASs support mobile agents by basically providing an agent server, an API for mobile agent programming and, sometimes, by supporting programming and administration tools. In particular, the agent server is able to execute agents by providing them with basic services such as migration, communication and resource access. In the last decade, a significant number of MASs for IP-based distributed computing systems have been developed. The majority of them are Java based (e.g. Aglets, Voyager, Ajanta, JADE etc.) and few others rely on other languages (DAgents, ARA etc.). In the context of WSNs it is challenging to develop MASs for supporting mobile agent-based programming [10], due to the currently available resource-constrained sensor nodes, and very few real systems have been to date proposed and concretely implemented. In the following, we first describe the most significant available research prototypes based on TinyOS [4] operating system, and then, we introduce the Java-based agent programming frameworks.

Agilla [20] is an agent-based middleware developed on TinyOS and supporting multiple agents on each node. Agilla provides two fundamental resources on each node: a tuplespace and a neighbor list. The tuplespace represents a shared memory space where structured data (tuples) can be stored and retrieved, allowing agents to exchange information through spatial and temporal decoupling. A tuplespace can be also accessed remotely. The neighbor list contains the address of all one-hop nodes needed when an agent has to migrate. Agents can migrate carrying their code and state, but they cannot carry their tuples locally stored on a tuplespace. Packets used for node communication (e.g. for agent migration/cloning, remote tuple accessing) are very small to minimize messages losses, whereas retransmission techniques are also adopted.

In [35] the authors propose an extension of Agilla to support direct communication based on messages. In particular, to establish direct communications, agents

are mediated by a middle component (named landmark) that interacts with agents through zone-based registration and discovery protocols.

ActorNet [25] is an agent-based platform based on the Actor model. To overcome the difficulties in allowing code migration and interoperability due to the strict coupling between applications and sensor node architectures, actorNet exposes services like virtual memory, context switching, and multi-tasking. Thanks to these features, it effectively supports agent programming by providing a uniform computing environment for all agents, regardless of hardware or operating system differences. The actorNet language used for high-level agent programming has syntax and semantics similar to those of Scheme with proper instruction extension.

In [36] another mobile agent framework is proposed. The framework is implemented on Crossbow MICA2DOT motes. In particular, it provides agent migration and agent interaction based both on local shared memory and network messages.

The above described MASs for WSNs [20, 35, 25, 36] are all implemented for TinyOS-based sensor platforms and use *ad hoc* languages for agent programming (e.g. Agilla uses a micro-programming language, whereas actorNet employs a functional-oriented language). Although some supported operations (e.g. migration) are very efficient, programming complex tasks is not so straightforward and, moreover, developers need to learn another very specific language. The Java language, through which Sun SPOT [3] and Sentilla JCreate [2] sensors can be programmed, due to its object-oriented features, could provide more flexibility and extendibility for an effective implementation of agent-based platforms. Currently, the only four available Java-based mobile agent platforms for WSNs are MAPS [9, 1], TinyMAPS [8], AFME [30], and MASPOT [27]. MAPS and TinyMAPS will be briefly described in Section 4.

The AFME framework, a lightweight version of the agent factory framework purposely designed for wireless pervasive systems and implemented in J2ME, has been recently ported onto Sun SPOT and used for exemplifying agent communication and migration in WSNs. AFME is strongly based on the Belief-Desire-Intention (BDI) paradigm, in which agents follow a sense-deliberate-act cycle. In AFME, agents are defined through a mixed declarative/imperative programming model. The declarative Agent Factory Agent Programming Language (AFAPL), based on a logical formalism of beliefs and commitments, is used to encode an agent's behavior by specifying rules that define the conditions under which commitments are adopted. The imperative Java code is instead used to encode perceptors and actuators. However, AFME was not specifically designed for WSNs and, particularly, for Java Sun SPOT.

MASPOT is a mobile agent system natively designed for Sun SPOTs that, differently from the other Java-based MAS, is able to provide agent's code migration, since it does not rely on the Isolate-based mechanism. In particular, both weak and strong migration are supported. The type of migration is defined for each agent at creation time and cannot change during the agents life cycle. The MASPOT inter-agent communication is based on the tuple spaces model, similar to the one adopted by Agilla. Communication between the base station and the mobile agents requires support for agent mobility. Such a communication is basically established by means

of a chain of references from the base station to the node where an agent currently is. When an agent moves to a new node, it leaves behind a marker indicating the next node to which it has migrated. Furthermore, a specific procedure exists to eliminate circular chains of references that will no longer be used.

3 Requirements for MAS Development on WSNs

Although several research efforts, as discussed in Section 2, have demonstrated that mobile agents are a suitable computing paradigm for supporting the development of distributed applications, services, and protocols on WSNs, the development of flexible and efficient MASs remains a challenging and very complex task due to the currently available resource-constrained sensor nodes. In the following, we first discuss the use of agents in the context of WSN on the basis of the Lange and Oshima research work [26], which delineates seven good reasons for using agents in traditional networks. We then provide an outline on the requirements for MAS development over WSNs.

3.1 On the Use of Mobile Agents for WSN Applications

In their seminal paper [26], Lange and Oshima advertised at least seven good reasons for using mobile agents in generic distributed systems. In the following we describe them with reference to the WSN context.

1. *Network load reduction*. Mobile agents are able to access remote resources, as well as communicate with any remote entity, by directly moving to their physical locations and interacting with them locally to save bandwidth resources. A mobile agent incorporating data processing capabilities can migrate to a sensor node, perform the needed operations on the sensed data and transmit the results to a sink node. This is more desirable, rather than a periodic transmission of raw sensed data from the sensor node to the sink node and the computation of data processing on the latter.
2. *Network latency overcoming*. An agent provided with proper control logic may move to a sensor/actuator node to locally perform the required control tasks. This overcomes the network latency that will not affect the real-time control operations also in case of lack of network connectivity with the base station.
3. *Protocol encapsulation*. If a specific routing protocol supporting multi-hop paths should be deployed in a given zone of a WSN, a set of cooperating mobile agents encapsulating this protocol can be dynamically created and distributed into the proper sensor nodes without any regard for standardization matter. Also in case of protocol upgrading, a new set of mobile agents can easily replace the old one at run-time.
4. *Asynchronous and autonomous execution*. These distinctive properties of mobile agents are very important in dynamic environments like WSNs where connections may not be stable and network topology may change rapidly.

A mobile agent, upon a request, can autonomously travel across the network to gather needed information "node by node" or to carry out the programmed tasks and, finally, can asynchronously report the results to the requester.

5. *Dynamic adaptation*. Mobile agents can perceive their execution environment and react autonomously to changes. This behavioral dynamic adaptation is well suited for operating on long-running systems like WSNs where environment conditions are very likely to change over time.
6. *Orientation to heterogeneity*. Mobile agents can act as wrappers among systems based on different hardware and software. This ability can well fit the need for integrating heterogeneous WSNs supporting different sensor platforms or connecting WSNs to other networks (like IP-based networks). An agent may be able to translate requests coming from a system into specific suitable requests to submit to another different system.
7. *Robustness and fault-tolerance*. The ability of mobile agents to dynamically react to unfavorable situations and events (e.g. low battery level) can lead to a better robust and fault tolerant distributed systems; e.g. the reaction to the low battery level event can trigger a migration of all executing agents to an equivalent sensor node to continue their activity.

An interesting taxonomy about WSNs and their relationships with multi-agent systems can be found in [38]. In particular, the major motivation of using agents over such networks is that many WSNs properties are shared with and can be actually supported by agents and multi-agent systems: physical distribution, resource boundedness, information uncertainty, large scale, decentralized control and adaptiveness. Moreover, as sensors in a WSN must typically coordinate their actions to achieve system-wide goals, coordination among dynamic entities (or agents) is one of the main features of multi-agent systems. In the following, the aforementioned common properties are discussed:

- Physical distribution implies that sensors are situated in an environment from which they can receive stimuli and act accordingly, also through control actions aiming at changing their environment. Situatedness is a main property of an agent and several well-known agent architectures were defined to support such important property.
- Boundedness of resources (computing power, communication and energy) is a typical property both of sensor nodes as single units and of the WSN as a whole. Agents and related infrastructures can support such limitation through intelligent resource-aware, single and cooperative behaviors.
- Information uncertainty is typical in large-scale WSNs in which both the status of the network and the data gathered to observe the monitored/controlled phenomena could be incomplete. In this case, intelligent (mobile) agents could recover inconsistent states and data through cooperation and mobility.
- Large scale is a property of WSNs either sparsely deployed on a wide area or densely deployed on a restricted area. Agents in multi-agent systems usually cooperate in a decentralized way through highly scalable interaction protocols and/or time- and space-decoupled coordination infrastructures.

- In large-scale WSNs, centralized control is not feasible as nodes can have intermittent connections and also can suddenly disappear due to energy lack. Thus, decentralized control should be exploited. The multi-agent approach is usually based on control decentralization transferred either to multiple agents dynamically elected among the available set of agents or to the whole ensemble of agents coordinating as peers.
- Adaptiveness is the main shared property between sensors and agents. An agent is by definition adaptive in the environment in which is situated. Thus, modeling the sensor activity as an agent or a multi-agent system and, consequently, the whole WSN as a multi-agent system, could facilitate the implementation of the adaptiveness properties.

3.2 Requirements and Issues

Although the agent paradigm has great potential to help the development of WSN applications, as demonstrated by all the previously discussed motivations, it is quite clear that the development of MASs for WSNs requires not only the same efforts required by conventional distributed systems but also the fulfillment of additional requirements specific to WSNs [5].

A direct exploitation of generic software agents into sensors is not so trivial, since research in traditional multiagent systems domain does not take into consideration the presence of severe resource constraints that typically arise on sensor nodes. In fact, the management of poor computational and energy resources, leading to many technical limits in designing a practical WSN mobile-agent middleware system, represent the most critical challenge in such a network. Moreover, research often does not opportunely consider that communication might be slow and intermittent and that nodes might be unreliable and failure prone.

Since software agents generally exhibit intelligent behaviors for autonomously coordinate their actions to achieve specific system-wide goals, the complexity of a middleware infrastructure for managing and supporting such agents' properties must be carefully considered with respect to the available resources. This is an extremely important issue because a fundamental requirement is to achieve a good execution performance on each single node for guaranteeing global efficiency and scalability.

Based on our experiences and on the results in literature, we truly believe that for facing with the resource-constrained problem, an agent-based system has to be defined following some design requirements:

- The MAS architecture server must be as lightweight as possible, which implies the avoidance of heavy concurrency models and, therefore, the exploitation of cooperative concurrency control mechanism to run agents.
- A plug-in-like components organization is recommended, in order to dynamically and selectively activate services that are needed while deactivating the useless ones for improving the overall system performance.
- The agent structure must be also lightweight so that agents can be efficiently executed and migrated. This not necessarily implies that agents cannot show

complex and intelligent behaviors, but simply that the mechanisms for defining and encoding their behavioral models have to be simple so that the architecture devoted to agent control and execution is not so resource-hungry.
- Mobile agents must be natively characterized on the basis of the functional layer to which they belong: application, middleware and network layer. They must be also able to locally interact to enable cross-layering.

Despite the actual effectiveness of the aforementioned guidelines, the efforts required for developing efficient MASs may fairly vary on the basis of the features that each single sensor platform provides to the developers. In Fig. 1, a list of the most widely used sensor platforms is shown along with their main characteristics: as it can be seen, each of the sensor type has much less resources than a standard desktop environment.

When possible, the limited resources problem can be overcome by executing heavy software agents, encapsulating computational-intensive functions, into external devices, e.g. components with higher processing capabilities residing outside the WSN. This makes it necessary to properly design and implement MASs by providing the necessary capabilities for allowing a closer interaction between WSNs and traditional networks and distributed systems. If it is not possible to rely on components with much more computational resources, MAS developers have to consider that the execution of advanced agent-based applications is limited by the use of low-overhead techniques and algorithms which necessary have to sacrifice optimality and accuracy, so that agents have to behave as best as possible given the available node resources.

Platform features heterogeneity also brings to an incomparable computing capabilities leading to difficulties in designing a common MAS architecture that could be suitable and efficient for execution on different sensor types. In particular, the need to address such a challenge will occur very soon since WSNs are expected to be deployed into a growing ubiquitous environment, so it is not unlikely to suppose that interaction among different typologies of networks will be a common situation. Although MAS research for WSN has traditionally considered homogeneous sensor node architectures, this assumption is too restrictive if we think about next generation WSN applications.

Another important issue in developing MAS concerns their architecture design approach and related agent definition primitives, so to opportunely satisfy WSN application developers requirements. Most of the research efforts conducted so far are based on a bottom-up approach. On the basis of the sensor nodes hardware, research focuses on how to provide proper software abstractions to assist application agent designers in defining common or advanced tasks, without requiring to deal with low-level details for hardware and networking management. Making the control part of the agents more expressive is the way for achieving a simplification of the agent design while keeping a rapid WSN application re-programming. This approach can also guarantee a reduction of the agent code size (and consequently a general better migration performance), because most of the macro-functionalities are already implemented into the MAS middleware and are directly accessible to agents on each WSN node running the middleware. A problem of such a solution is that often the

	CPU / Microcontroller	RAM	Flash Memory (Program Code)	Flash Memory (Data Storage)	Operating System
Tmote Sky / TelosB	TI MSP430 8MHz, 16bit	10 KB	48 KB	1MB	TinyOS, Contiki
Intel iMote2	Intel/Marvell PXA271 XScale Processor 13 – 416MHz, 32bit	32 MB	32 MB (shared)		TinyOS
Crossbow Mica2, Mica2Dot, MicaZ	Atmel ATmega 128L 8MHz, 8bit	4 KB	128 KB	512 KB	TinyOS, Contiki
Sun SPOT	ARM920T 180MHz, 32bit	512 KB	4 MB (shared)		Squawk JVM
Sentilla JCreate	TI MSP430 8MHz, 16bit	10 KB	48 KB	-	custom JVM CLDC-compliant
Shimmer	TI MSP430 8MHz, 16bit	10 KB	48 KB	up to 2GB (MicroSD slot)	TinyOS, Contiki
Texas Instruments Z-Stack	CC243x-CC253x SoC (Intel 8051, 32MHz, 8bit)	8 KB	32, 64, 128, 256 KB	-	Z-Stack (ZigBee compliant sw stack)
Crossbow IRIS mote	Atmel ATmega 1281 8MHz, 8bit	8 KB	128 KB	512 KB	TinyOS, Contiki
Tyndall mote	Atmel ATmega 128L 8MHz, 8bit	8 KB	128 KB	512 KB	TinyOS
Ember EM250	XAP2 core 12MHz, 16bit	5 KB	128 KB	-	TinyOS, EmberZNet (ZigBee compliant sw stack)
Ember EM351 / EM357	ARM Cortex–M3 12/24MHz, 32bit	12 KB	128 KB / 192 KB	-	TinyOS, EmberZNet
Shockfish TinyNode	TI MSP430 8MHz, 16bit	10 KB	48 KB	512 KB	TinyOS, Contiki
Libelium WaspMote	Atmel ATmega1281 8MHz, 8bit	8 KB	128 KB	up to 2GB (MicroSD slot)	TinyOS, Contiki, ZigBee compliant sw stack
Eistec AB Mulle	Renesas M16C/62P 10MHz	10 KB	128 KB	2 MB	TinyOS, Contiki
Memsic LOTUS	ARM Cortex M3 10-100MHz, 32bit	64KB	512KB	64MB	RTOS

Fig. 1 List of sensor platforms and their characteristics

available high-level abstractions may not be suitable for many applications that necessitate of a more fine-grained control of the node resources, which is generally needed for defining much more efficient tasks. However, although a fine-grained task control is ideal for reaching a more program execution flexibility, it can lead to a potentially bigger, and consequently error-prone, agent code. The alternative, i.e. the top-down approach, is basically based on a first deep understanding of what the primary application requirements are, which become the main driver for the design of the agent-based middleware. In this case, the provided agent programming constructs may not be so straightforward to use, and even may be very application-specific oriented so that a more general use is not possible.

4 MAPS and TinyMAPS

On the basis of the previously discussed basic requirements for developing flexible and efficient agent systems, in the following we briefly present MAPS and TinyMAPS, implemented for supporting rapid prototyping of WSN applications on sensor platforms enabling Java programming. While MAPS is specifically conceived for higher processing-capable sensor devices, Sun SPOTs, TinyMAPS is a version of MAPS tailored for more constrained platforms such as Sentilla JCreate.

4.1 MAPS: Mobile Agent Platform for Sun SPOT

MAPS [9, 1] is an innovative Java-based framework purposely developed on Sun SPOT technology for enabling agent-oriented programming of WSN applications. It has been defined according to the following requirements:

- Component-based lightweight agent server architecture to avoid heavy concurrency and agents cooperation models.
- Lightweight agent architecture to efficiently execute and migrate agents.
- Minimal core services involving agent migration, agent naming, agent communication, timing and sensor node resources access (sensors, actuators, flash memory, switches and battery).
- Plug-in-based architecture extensions through which any other service can be defined in terms of one or more dynamically installable components implemented as single or cooperating (mobile) agent/s.
- Java language for programming mobile agents.

The MAPS architecture (see Fig. 2) is based on components that interact through events and offer a set of services to mobile agents including message transmission, agent creation, agent cloning, agent migration, timer handling, and easy access to the sensor node resources.

In particular, the main components are:

- *Mobile Agent (MA)*, which is the basic high-level component defined by user for constituting agent-based applications.
- *Mobile Agent Execution Engine (MAEE)*, which manages the execution of MAs by means of an event-based scheduler enabling lightweight concurrency. MAEE also interacts with the other service-provider components to fulfill service requests (message transmission, sensor reading, timer setting, etc) issued by MAs.
- *Mobile Agent Migration Manager (MAMM)*, which supports agents migration through the Isolate hibernation/dehibernation feature provided by the Sun SPOT environment [3]. The MAs hibernation and serialization involve data and execution state whereas the code should already reside at the destination node (this is a current limitation of the Sun SPOTs which do not support dynamic class loading and code migration).

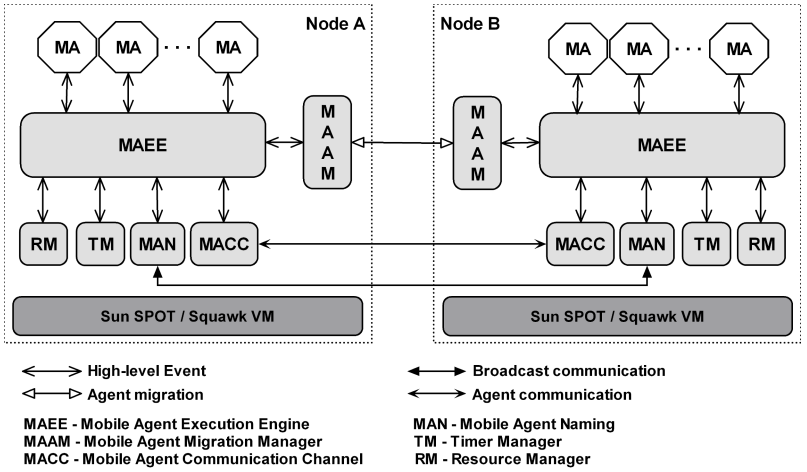

Fig. 2 MAPS architecture

- *Mobile Agent Communication Channel (MACC)*, which enables inter-agent communications based on asynchronous messages (unicast or broadcast) supported by the radiogram protocol.
- *Mobile Agent Naming (MAN)*, which provides agent naming based on proxies for supporting MAMM and MACC in their operations. MAN also manages the (dynamic) list of the neighbor sensor nodes that is updated through a beaconing mechanism based on broadcast messages.
- *Timer Manager (TM)*, which manages the timer service for timing MA operations.
- *Resource Manager (RM)*, which enables access to the resources of the Sun SPOT node: sensors (3-axial accelerometer, temperature, light), switches, leds, battery, and flash memory.

In Fig. 3 the Mobile Agent model is depicted. In particular, the dynamic behavior of MA is modeled as a multi-plane state machine (MPSM). The GV component represents the global variables, namely, the data inside an MA whereas the GF is a set of global supporting functions. Each plane may represent the behavior of the MA in a specific role, so enabling role-based programming, and is composed of local variables (LV), local functions (LF), and an ECA-based automaton (ECAA). This automaton is composed of states and mutually exclusive transitions among states. Transitions are labeled by Event-Condition-Action (E[C]/A) rules, where E is the event name, [C] is a boolean expression based on global and local variables, and A is an atomic action. MAs interact through events that are asynchronously delivered by the MAEE and dispatched, through the Event Dispatcher component, to one or more planes according to the events the planes are able to handle. It is worth noting that the MPSM-based agent behavior programming allows exploiting the benefits deriving from three main paradigms for WSN programming: event-driven programming, state-based programming and mobile agent-based programming.

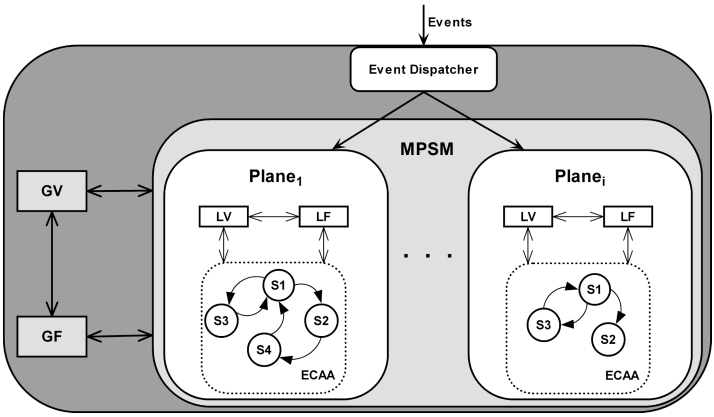

Fig. 3 MAPS agent model

4.2 TinyMAPS

The architecture and the agent programming model of TinyMAPS [8] have been directly derived from MAPS, with proper adaptation to be actually implemented atop the Java-based Sentilla technology [2].

Its architecture is depicted in Fig. 4 and, similarly to MAPS, is based on components interacting through events but offering a more limited set of core-services (agent creation, migration, communication and sensor resource access) to mobile agents. Agent cloning and timer handling are not provided. In particular, the main components are:

- *Mobile Agent (Agent)*, which is the basic component defined by user. It is designed as a simple class that encloses within the behavior (is possible to define only single-plane agents).
- *Mobile Agent Execution Engine (MAEE)*, which manages the execution of MAs by means of a thread that schedules local or remote events according to a FIFO policy. The MAEE also encapsulates others functions:
  - it uses serialization for sending remote events through the inner component *Mobile Agent Event Sender (MAES)*;
  - it implements a system of naming (*Mobile Agent Naming* component, *MAN*) which keeps the WSN nodes along with the active agents running on them;
  - it interacts with the MAER through the *Mobile Agent Migration Manager (MAMM)* for providing a mechanism of migration for mobile agents.
- *Mobile Agent Event Receiver (MAER)*, which is developed as an independent thread that waits for receiving events from remote MAs. After event reception, it delegates the delivering of event to the MAEE.
- *Resource Manager (RM)*, which allows accessing to the resources of the Sentilla node, i.e. a 3-axial accelerometer and LEDs.

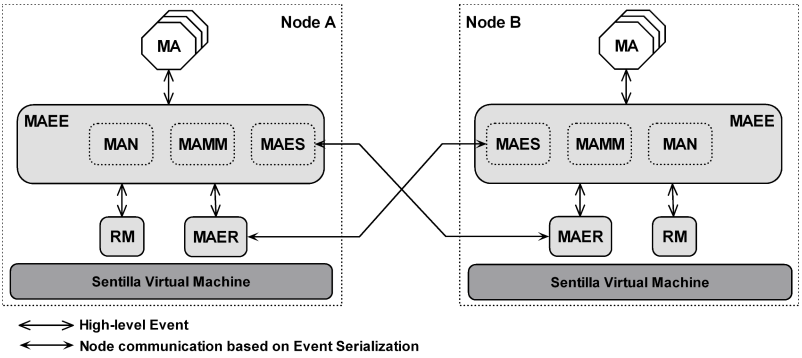

Fig. 4 TinyMAPS architecture

The TinyMAPS mobile agents are defined by following the same multi-plane state machine (MPSM) model previously discussed and depicted in Fig. 3.

5 A Comparison among Java-Based MAS

In the following section we first describe the main characteristics of the Java-based MAS for WSNs and then comparative results of MAPS with both TinyMAPS and AFME are provided.

5.1 Java-Based MASs' Characteristics Comparison

In Table 1, MAPS, TinyMAPS, AFME, and MASPOT are compared with respect to seven characteristics: agent behavior model, intentional agent support, agent behavior definition language, migration type, migration mechanism, agent communication model, and dynamic agent creation.

Both TinyMAPS and MAPS offer similar services for developing WSN agent-based applications. They use finite state machines (FSMs) to model the agent behavior and directly the Java language to program guards and actions, so no translator and/or interpreter need to be developed and no new language has to be learnt. Moreover, differently from TinyMAPS, MAPS is more powerful and fully exploits the release 5.0 "red" of the Sun SPOT library to provide advanced functionality of communication, migration, sensing/actuation, timing, and flash memory storage. Although AFME is based on the same basic programming language, its agent model is different from a finite state machine, since it employs a more complex BDI-like model, which offers support to intentional agents. In particular, it is centered on perceptors, actuators, modules, and services which are developed in Java but have to be strictly correlated to declarative rules provided for modeling the agent behavior. Both approaches are effective for developing agent-based applications even though MAPS is more straightforward as it relies on a programming style based on state

Table 1 Main features offered by the current Java-based MASs for WSNs

	MAPS [9]	TinyMAPS [8]	AFME [30]	MASPOT [27]
Agent Behavior Model	Finite State Machine	Finite State Machine	Belief/Desire/ Intension	No specific model
Intentional Agent Support	No	No	Yes	No
Agent Behavior Definition Language	Java	Java	Java/AFAPL	Java
Migration Type	Strong (but no code)	Weak	Weak	Weak or Strong
Migration Mechanism	Sun SPOT Isolate	Agent descriptor transmission	Agent descriptor transmission	Sun SPOT Isolate + Suite transfer
Agent Communication Model	Message passing	Message passing	Message passing	Tuple spaces
Runtime Agent Creation	Yes	Yes	No	Yes

machines widely known by programmers of embedded systems. Differently from the previous systems, MASPOT does not provide any specific model to facilitate developers, which have to design and implement the agents' behavior without the support of a well defined high-level formalism.

For what concerning the migration support, MAPS offers a "limited" strong migration, since the execution state of the agent is transferred during migration along with the agent data state, but no code migration is supported. In particular, the implementation of mobile agents is based on the Isolate components defined by the Sun SPOT library. Each Isolate represents "process-like" unit of computation isolated from other instances of Isolate and their migration mechanism is directly offered by the SPOT Squawk JVM through their hibernation and serialization. TinyMAPS, instead, supports migration by simply sending an event that contains agent status information and data, which are encapsulated inside the event. Thus, the agent needs to restart its execution on the remote node. In any case, both MAPS and TinyMAPS suffer from the current limitation of the Sun SPOT and the Sentilla JCreate that, as CLDC-compliant devices do not allow dynamic class loading, so preventing from the possibility to support code migration (i.e. any class required by the agent must be already present at the destination node). Similarly to TinyMAPS, AMFE uses a proprietary agent descriptor to capture and transmit agent data and state. At the contrary, MASPOT supports both strong and weak migration and the type of migration is defined for each agent at creation time and cannot change during the agents

life cycle. In particular, along with the migration mechanism based on Isolates, the transmission of Suites (containing a collection of packaged classes and libraries) is employed for migrating the agent code from a central code library situated on the user station (the coordinator computer of the WSN) to a specific Sun SPOT node.

The agent communication model adopted by MAPS, TinyMAPS, and AFME for exchanging information among agents is based on message passing (unicast and broadcast) which is the communication paradigm mostly used in agent-oriented frameworks. The MASPOT inter-agent communication is instead based on the tuple spaces model, similar to the one adopted by Agilla.

The ability to create an agent at runtime could be an important feature for application in which the number of necessary agents cannot be determined "a priori" and simply fixed at compile-time. MAPS, TinyMAP, and MASPOT allow for such capability, so providing more flexibility for the creation of dynamic distributed applications, whereas AFME needs agents to be created only in a static way.

5.2 *Performance Test Comparison between MAPS and TinyMAPS*

To evaluate and compare the performance of MAPS and TinyMAPS, two benchmarks have been defined according to [17] for the following mechanisms:

- *Agent communication*. The agent communication time is computed for two agents running onto different nodes and communicating in a client/server fashion (request/reply). Two different request/reply schemes are used: (i) *data Back and Forward (B&F)*, in which both request and reply contain the same amount of data; (ii) *data B*, in which only the reply contains data.
- *Agent migration*. The agent migration time is calculated for agent ping-pong among two single-hop-distant sensor nodes. Migration times are computed by varying the data cargo of the ping-pong agent.

In Fig. 5 the finite state machines of the agents involved in the two different benchmarks are shown. They are related to both MAPS and TinyMAPS system, since they rely on the same agent modeling formalism.

In the *Sender* agent plane, after the agent creation the *AGN_START* event is automatically signaled bringing the agent to the *IDLE* state and executing the *A0* action for some initialization code. From the *IDLE* state, the transition to the *WAIT_MSG* state is immediately triggered whenever the guard *[msgCount* $<$ *MSG_NUMBER]* holds and, consequently, the *A1* action is executed, consisting in sending a message to the *Receiver* agent. The *Sender* then waits until the reply message is received. If so, the *MSG* event is triggered, the action *A2* is executed (the messages exchange time is evaluated and stored), and the plane returns to the *IDLE* state. If the number of messages exchange reaches *MSG_NUMBER*, before the termination of the Sender agent, the operations included in *A3* are performed (i.e. the communication time average is calculated and a last message is sent to the Receiver for its termination). The *Receiver* agent's behavior is very simple. It waits for a message coming

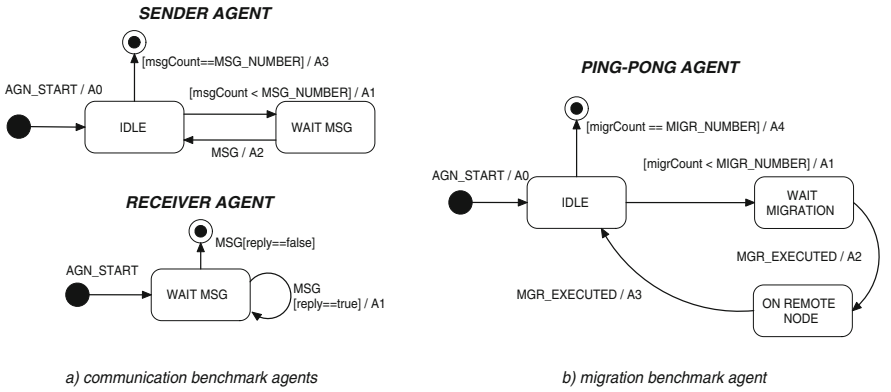

Fig. 5 Planes of the agents employed in the benchmarks

from the *Sender* and on the basis of the value of its *reply* parameter, it will send a message reply (*A1*) or terminate itself.

For the agent migration benchmark, a single *Ping-Pong* agent is employed. Upon agent creation and starting, a request for migration is executed (action *A1*) and the plane transits to the *WAIT_MIGRATION* state, waiting for migration completion, which is signaled with the *MGR_EXECUTED* event. After having moved to the remote node, the agent immediately requests for a new migration for coming back to the origin node (action *A2*). Once the agent is came back to the origin node, the elapsed time is stored and a new round-trip migration starst, unless *MIGR_NUMBER* migrations have been completed. Under such a condition, before terminating, the agent computes the migration time average (action *A4*).

The implementation of the agent planes depicted in Fig. 5 is rather fast, since the basic structure of a generic finite state machine (FSM) is very simple and the main effort for developers is just to insert the code corresponding to the actions of the FSMs, by also making use of the MAPS/TinyMAPS API for accessing to the basic agent management supporting services. An excerpt of the *Sender* agent's plane implementation is shown in the following. In particular, the *eventHandler* method is where the FSM and related actions are encode. For more specific technical details on the design and implementation of MAPS/TinyMAPS agents, readers can refer to [7, 6].

```
.....
public SenderPlane(Agent agent){
      super(agent);      this.currentState = CREATED;
}
.....
public void eventHandler(Event event){
      try {
         switch(this.currentState){
            case CREATED:
               if (event.getName() == Event.AGN_START){   //action A0
                  while(agents.size() == 0){
                     Thread.sleep(200);
                     agents= this.agent.getRemoteAgentsID();
                  }
```

```
                remoteAgentID= (String)(agents.elementAt(0));
                this.msgCount= 0;
                this.currentState = IDLE;
             }
          break;
          case IDLE:
             if(this.msgCount < MSG_NUMBER){   //action A1
                Event msg = new Event(this.agent.getId(), remoteAgentID,
                                                   Event.MSG, Event.NOW);
                msg.setParam("msgPayload", msgPayload);
                msg.setParam("reply", true);
                this.startTime= System.currentTimeMillis();
                this.agent.send(this.agent.getId(), remoteAgentID,
                                                        msg, false);
                this.currentState = WAIT_MSG;
             }else{      //action A3
                compute&printMean(this.commTime);
                Event msg = new Event(this.agent.getId(), remoteAgentID,
                                                   Event.MSG, Event.NOW);
                msg.setParam("reply", false);
                this.agent.send(this.agent.getId(), remoteAgentID,
                                                        msg, false);
                this.agent.terminateAgent();
             }
          break;
          case WAIT_MSG:
             if (event.getName() == Event.MSG ) {     //action A2
                this.commTime[this.msgCount]=
                            System.currentTimeMillis()-this.startTime;
                this.msgCount++;
                this.currentState = IDLE;
             }
       }
    }
    catch(Exception e){LedsManager.error(); e.printStackTrace();}
 }
....
```

In Fig. 6, the comparison results of the agent communication time, with different message payload, are shown, with MAPS performing better than TinyMAPS. Moreover, as message data payload increases, communication time for MAPS is not affected. The tests have been executed by taking into consideration that the Sentilla JCreate platform imposes a maximum message payload size of 78 bytes.

For what concerning the migration benchmark, Fig. 7 shows the obtained results. In particular, the migration times are high due to both the slowness of the JVM operations supporting the migration process and the communication time between two nodes. For agents with low data payload TinyMAPS performs better than MAPS; however, when agent data payload is greater than 58 bytes, MAPS migration mechanism starts performing better. Since TinyMAPS relies on messages for transmitting the agent description, the limitation of 78 bytes still holds. At the contrary, MAPS does not have any agent data payload limitations.

5.3 Performance Test Comparison between MAPS and AFME

The same benchmarks discussed in Section 5.2 have been performed for AFME [30], and the obtained communication/migration performance results are compared with MAPS.

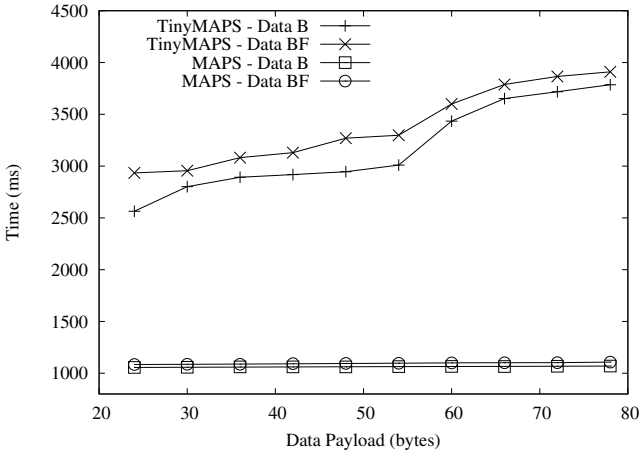

Fig. 6 MAPS vs. TinyMAPS: Agent communication time comparison

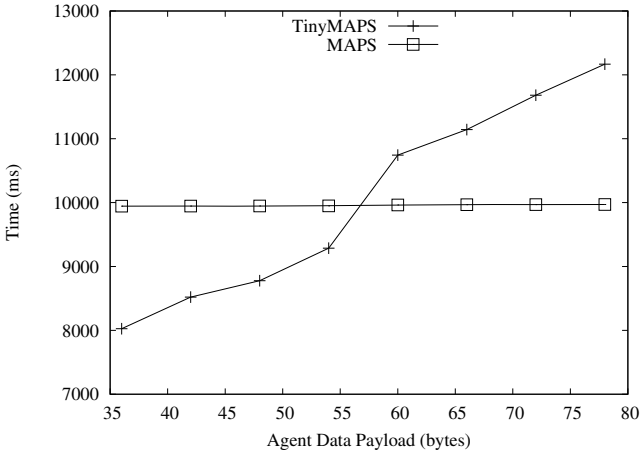

Fig. 7 MAPS vs. TinyMAPS: Agent migration time comparison

Differently from MAPS and TinyMAPS, AFME agents' behavior is defined through AFAPL declarative rules. In the following, the rules of the two agents defined for testing the agent communication time are reported and described.

```
Sender Agent rules:
1. numMsgSent(?msgCount), #?msgCount<MSG_NUMBER > storeStartTime,
                              inform(agentID(receiverAgent),
                                addresses("radiogram://"+receiverNodeAddr));

2. message(inform, sender(receiverAgent, address(?addr))) >
                                    compute&storeCommTime, increaseMsgCount;

3. numMsgSent(?msgCount), #?msgCount==MSG_NUMBER > compute&printTimeAverage;
```

```
Receiver Agent rule:
1. message(inform, sender(senderAgent, address(?addr))) >
     inform(agentID(senderAgent), addresses("radiogram://"+senserNodeAddr));
```

The first rule of the *Sender* agent checks if less than *MSG_NUMBER* messages have been sent. When the *numMsgSent* belief is adopted, it returns the number of messages sent into the *?nSamples* variable, whose value is tested for confirming that a new message has to be sent. If so, the starting time is also acquired (it is in charge of the *storeStartTime* actuator). The second rule computes and stores the elapsed time for the communication upon the reception of the reply message coming from the *receiverAgent* (see the *message* belief), whereas the messages count is incremented (*compute&storeCommTime* and *increaseMsgCount* are the two actuators in charge of performing such operations). Finally, the third rule fires when the number of messages sent so far is equal to *MSG_NUMBER*, so that the final communication time average can be computed and displayed (i.e. the *compute&printTimeAverage* actuators is executed).

The *Receiver* agent has one rule, which simply consists in sending a message reply whenever a message coming from the *Sender* agent is received.

For what concerning the migration benchmark performed, the needed rules for th correct execution of the *Ping-Pong* agent are the following:

```
Startup rule:

1. always(ieeeAddr("\"+ com.sun.spot.peripheral.Spot.getInstance()
                    .getRadioPolicyManager().getIEEEAddress() + \":45")),
   always(destAddr("0014.4F01.0000.07DB:46")),
   always(init);

Ping-Pong Agent rules:

1. init, destAddr(?destaddr), ieeeAddr(?addr) >
     par(
      migrate(?destaddr,null), retractBelief(always(destAddr(?destaddr))),
      retractBelief(always(ieeeAddr(?addr))), retractBelief(always(init)),
                                 adoptBelief(always(couple(?time,?addr))),
                                             adoptBelief(always(migrated)),
                                 adoptBelief(always(ieeeAddr(?destaddr))),
                                     adoptBelief(always(destAddr(?addr))),
                                                        time(?time)
     );

2. migrated, destAddr(?destaddr), ieeeAddr(?addr) >
     par(
      migrate(?destaddr,null), retractBelief(always(destAddr(?destaddr))),
                                   retractBelief(always(ieeeAddr(?addr))),
         retractBelief(always(migrated)), adoptBelief(always(terminated)),
                                 adoptBelief(always(ieeeAddr(?destaddr))),
                                     adoptBelief(always(destAddr(?addr)))
     );

3. terminated, couple(?time, ?addr) > par(printTime(?time, ?addr),
                                     retractBelief(always(terminated)));
```

The rules defined above are much more complicated with respect to the ones previously defined for the agent communication benchmark, and also much more difficult to read and understand if compared to the simple and clear finite state machine

formalism adopted by MAPS/TinyMAPS and depicted in Fig. 5. In particular, two set of rules are needed for a correct execution of the *Ping-Pong* agent: the rules associated to the agent and representing its running behavior, and a startup rule which is necessary for creating a set of beliefs (*destAddr*, *time*, and *ieeeAddr* along with related values) after the agent creation, but before the agent start, and representing a kind of knowledge initialization. In particular, the *ieeeAddr* belief represents the node address on which the agent is currently running, whereas the *destAddr* belief represents the destination node to which the agent has to migrate.

Upon the agent start, since the aforementioned starting beliefs hold, the first rule fires and the *migrate* actuator is performed for requesting to the AFME middleware the migration of the agent to the destination node, whose address has been previously stored in the *?destaddr* variable by the startup rule. At the same time (the *par* keyword indicate the parallel execution of commitments/actuators), the *ieeeAddr* and *destAddr* need to be retracted and readopted by swapping their related values. Moreover, the time of starting migration is stored and the *migrated* belief is generated so that the agent, on resume, knows that the migration is completed. The second rule is rather similar to the previous one and is in charge of performing the returning migration to the origin node. Once the agent terminates its ping-pong trip, the third rule fires (i.e. the *couple* and the *terminated* beliefs hold) and the elapsed time is finally computed and displayed.

Along with the AFAPL rules, AFME requires also the implementation of proper Java classes, each of which is related to a specific belief and actuator and represents the actual code that is executed on the Sun SPOT nodes. For more specific technical details on the design and implementation of AFME agents, readers can refer to [7, 6].

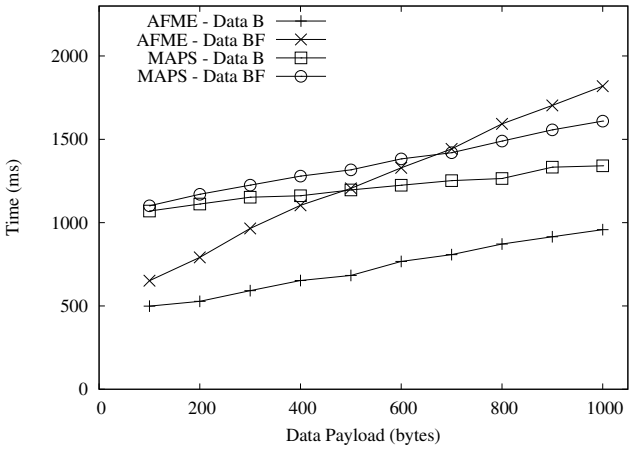

Fig. 8 MAPS vs. AFME: Agent communication time comparison

The results obtained by the two performed benchmarks are described in the following.

Comparison results for the communication time are shown in Fig. 8. For messages with light data payload, AFME performs better than MAPS; however, when the message data payload overtakes 700 bytes, MAPS starts performing better in the case *data BF*.

Comparison results for the migration times are shown in Fig. 9. AFME retains a higher performance migration mechanism, as it is not based on the heavy isolate hibernation/serialization mechanisms of the Squawk VM.

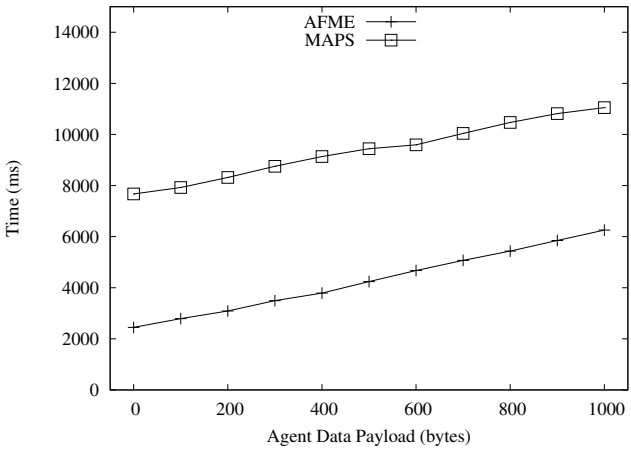

Fig. 9 MAPS vs. AFME: Agent migration time comparison

6 Lessons Learned and Open Challenges

The agent technology has been widely proven (see Section 2) to be an effective paradigm for the development of WSN applications either by providing specific solutions, like software infrastructure layers (i.e. network routing or data dissemination and fusion techniques), or by providing high-level agent-oriented design and programming models able to support developers in building their own distributed applications.

In both cases, we experienced that the best way to fully exploit agents into WSNs is by providing a proper MAS incorporating all the functionalities needed for the execution of agents by managing agent lifecycle, migration, communication, sensing capabilities and sensor resource access. In particular, including all these functionalities into the MAPS and TinyMAPS architectures while maintaining an efficient execution at runtime over a resource constrained environment has been a very complex task. Although the scarce available resources, MAPS and TinyMAPS are able to provide a set of core-services on which developers can define their own agents

through a flexible programming abstraction based on a finite state machine formalism. Moreover, the design requirements discussed in Section 3.2 have been successfully met: lightweight concurrency control mechanism, modular components organization, and lightweight agent structure so that agents can be efficiently executed.

For testing their actual capabilities, both MAPS and TinyMAPS have been employed to develop a real-time human activity monitoring application [9, 8] having strict requirements in terms of sensing timing and processing. The performance results demonstrate that an agent architecture, when properly designed and implemented, can be a comprehensive solution for effectively and efficiently developing real and exigent applications over WSNs.

Unfortunately, several issues are difficult to overcome due to insurmountable technological limitations. As already discussed in Section 5, the SPOT Squawk JVM provides a poorly performing migration mechanism, and does not provide dynamic class loading and code migration. Also, the Sentilla JCreate, based on a different hardware and software architecture, suffers from a much more constrained resources limitations. Moreover, due to incompatible agent migration processes (Isolate-based for Sun SPOT and customized transmission of agent status and data for JCreate), a mobile agent application running over a heterogeneous WSN cannot be defined, unless a kind of software bridge is created with a consequent worsening of agent execution and migration performance. Finally, since a bottom-up approach has been adopted for MAPS and TinyMAPS design and implementation, the provided programming abstractions, defined over the low-level sensor node architecture, guarantee a simple agent definition, but at the cost of a less fine-grained control of the node resources, which is generally needed in many applications for defining much more efficient tasks.

On the basis of the previously highlighted matters, it is clear that many open challenges still remain and further research efforts have to be undertaken.

First of all, much more in-depth research needs to be done for finding a better compromise between easiness in agents definition and efficiency at run-time, because MAS design trade-offs should attempt to satisfy the rapid application development requirement while guaranteeing as much flexibility as possible, depending on the sensor nodes capabilities.

Moreover, despite the efforts made so far in MAS development have been focused on the definition of adequate abstractions for high-level agent modelling, not enough work has been conducted to identify and address others requirements for an effective development of agent-based WSN applications. In particular, the complexity in developing them derives also from the lack of coherent tool chains for application development. In fact, in addition to agent programming, WSN application development involves a series of labor-intensive tasks such as the compilation and verification of program code, configuration of a simulator or of sensor nodes, and deployment/injection of compiled code to nodes. Thus, we believe that a complete agent-oriented methodology addressing the aforementioned issues should be fully integrated with a specifically implemented MAS. Indeed, as such a methodology aims at effectively developing real applications, it should rely on a specific MAS for being supported during the final applications development phases, i.e. implemen-

tation, deployment and execution. Although several agent-oriented methodologies have been conceived so far, they have not been actually fully integrated into a real deployable agent system. Moreover, being WSNs a specific type of distributed embedded system, we believe that the following techniques and methods developed in the research area of embedded computing could be fruitfully exploited for better defining and implement an effective and complete MAS: platform-based design, simulation-driven prototyping, and model-driven development based on domain-specific languages. We therefore think that an effective methodology for agent-based WSN application development, and consequently the development of its related MAS, should integrate method fragments from agent-oriented methodologies with the following WSN-oriented techniques/methods:

- *Platform-Based Design* [24] is a methodology originally developed for the design of embedded systems. According to the PBD, a design is obtained as a sequence of refinement steps that guide the designer from the initial specification all the way down to a physical implementation. To support this process, a set of intermediate abstraction layers and platforms are to be identified. These platforms therefore represent the target system at different levels of abstraction. Each platform is composed on a set of instances. An iterative refinement (mapping) process translates a platform instance to another one of a lower level, until the final implementation is reached. Each refinement step is a design choice taken as the solution of a constrained optimization problem. The cost function is typically the energy consumption (to optimize the system lifetime). The constraints are usually the error rate, latency, and the budget. The PBD methodology has been applied for the system-level design of WSNs [12] to address, through a formal and systematic approach, issues such as reliability and support for heterogeneity that are still one of the main limiting factors to the commercial spread of the WSN technology. In particular, the approach is based on three layers of abstraction and their relative platforms: a service platform at the application layer, a protocol platform to describe the protocol stacks, and an implementation platform for the hardware nodes. In this case, the first refinement step maps the high-level service platform instance to an implementation platform instance, leading to a topology. It is the output of this step that identifies the type, the number, and the location of the physical sensor nodes needed by the application. The communication problem is addressed only later, with a further mapping process that choose the right communication protocol stack (MAC and/or routing) that meets the application requirements and satisfies the energy constraints of the selected physical network infrastructure.
- *WSN simulation techniques and framework*. In the context of WSN, simulation is the most effective technique for supporting the prototyping of applications since mathematical analysis and experimental deployments are not always allowed. This approach is a delicate matter due to the complexity of the WSNs and new aspects inherent in WSN must be included in simulators (e.g., a physical environment and an energy model), leading to different degrees of accuracy against performance [18]. Among simulators specifically designed for WSNs, COOJA [31] allows to simulate WSNs choosing if increasing the accuracy or the performance

giving the possibility to simulate different nodes at different levels. The COOJA simulator is a flexible Java-based sensor network simulator with specific algorithms to simulate entities like the WSN radio channel and battery consumption and capable to emulate microcontrollers. COOJA is the only simulator that has the ability to mix simulations of sensor devices at multiple abstraction levels: (i) Application level, the simulated nodes run the application logic re-implemented in Java (simulating at this level increases performances); (ii) OS level, the nodes use the same code as real nodes, but compiled for COOJA; (iii) Hardware level, the nodes run the same compiled code that can be used in real nodes (simulating at this level increases accuracy). The nodes at different abstraction levels can communicate with each other using the radio channel. COOJA can effectively support the prototyping of WSN applications giving the possibility to simulate high-level code (Java, at the application level) to test the algorithms and then the code can be re-implemented for WSN nodes like TelosB and simulated again at low-level (hardware level).

- *MDD and Domain-Specific Languages for WSNs*. The Model-Driven Development is based on the idea of separating the specification of the operation of a system from the details of the way that system uses the capabilities of its platform. The three primary goals of MDD are portability, interoperability and reusability through architectural separation of concerns. MDD provides a set of guidelines for structuring specifications expressed as models. It defines system functionality using an appropriate domain-specific language (DSL). The MDD approach can be very useful in the WSN domain [39] giving the possibility to overcome the limitation in the programming of heterogeneous WSN due to different platforms and OSs.

In particular, the PBD methodology may be particularly suitable for helping in better defining the MAS architecture on the basis of the constraints of a specific target sensor platform. Moreover, as demonstrated in the software engineering research area, the development of a CASE tool specifically and seamlessly supporting all phases of the methodology from requirement analysis to system deployment and maintenance would promote usability and effectiveness of the methodology.

Finally, as the complexity of the applications grows, the need for proper management and control procedures and techniques is becoming a fundamental requirement. Although a significant number of MAS has been developed, support to self-* properties has not still properly taken under consideration. Such autonomic properties are necessary for allowing MAS-based applications to autonomously achieve system-wide goals despite environment changes and failures.

7 Conclusion

Programming WSN applications is a complex task that requires suitable programming paradigms and frameworks to cope with the WSN-specific characteristics. Several kinds of micro- and macro-programming techniques have to date been proposed. Among them, mobile agent-based programming, which has been formerly

introduced for conventional distributed systems, can be more effectively exploited in the context of WSNs.

In this chapter, we have first introduced the agent-based systems for the WSN context by providing the motivations and the benefits of using agents over such networks. Then, we have focused on the major challenges for the development of mobile agent systems on sensor platforms, and specifically we discussed how the main problems arise when MAS developers have to face with resource-constrained devices and with sensor architectures heterogeneity. In particular, we have presented how we have directly tackled such issues by developing two mobile agent systems, MAPS and TinyMAPS. Although our efforts for designing them around the main requirements needed for a flexible and efficient agent system, it has been shown how some limits cannot be mitigated, especially if they are directly related to the platform characteristics. As an example, the Sun SPOT migration mechanism is an inherently low performance operation, as shown by the quantitative performance evaluation conducted, and also does not allow for agent code migration. Moreover, the difficulties in providing a multi-platform MAS have been exhibited by discussing the needed to properly modify the MAPS architecture for adapting the system to the Sentilla sensor platform.

Although many research works have demonstrated that mobile agents are a suitable technology for supporting WSN applications development, more efforts should still be devoted to the definition of high performance MAS, WSN-oriented methodologies and tools fully supporting the development lifecycle of WSN agent-based applications. In particular, we believe that a full-fledged agent-oriented methodology not only should incorporate useful methods and models derived from available agent-oriented methodologies but also it should include methods and techniques derived from embedded computing such as platform-based design, simulation-driven testing and model-driven development based on domain-specific languages. On-going research activity is therefore focused on such a methodology to support agent-oriented WSN application development based on MAPS/TinyMAPS as implementation platforms in the system implementation phase.

References

1. Mobile Agent Platform for Sun SPOT (MAPS), documentation and software (2011), http://maps.deis.unical.it
2. Sentilla developer community (2011), http://www.sentilla.com/developer.html
3. Sun Small Programmable Object Technology (Sun SPOT), documentation and software (2011), http://www.sunspotworld.com
4. TinyOS web site, documentation and software (2011), http://www.tinyos.net
5. Afzal, S.R., Huygens, C., Joosen, W.: Extending middleware frameworks for wireless sensor networks. In: Afzal, S.R. (ed.) Ultra Modern Telecommunications & Workshops, ICUMT 2009, pp. 1–7. IEEE (2009), https://lirias.kuleuven.be/handle/123456789/261940

6. Aiello, F., Bellifemine, F.L., Fortino, G., Galzarano, S., Gravina, R.: An agent-based signal processing in-node environment for real-time human activity monitoring based on wireless body sensor networks. Eng. Appl. of AI 24(7), 1147–1161 (2011)
7. Aiello, F., Fortino, G., Galzarano, S., Gravina, R., Guerrieri, A.: An analysis of java-based mobile agent platforms for wireless sensor networks. Multiagent and Grid Systems 7(6), 243–267 (2011)
8. Aiello, F., Fortino, G., Galzarano, S., Vittorioso, A.: TinyMAPS: A Lightweight Java-Based Mobile Agent System for Wireless Sensor Networks. In: Brazier, F.M.T., Nieuwenhuis, K., Pavlin, G., Warnier, M., Badica, C. (eds.) Intelligent Distributed Computing V. Studies in Computational Intelligence, vol. 382, pp. 161–170. Springer, Heidelberg (2011)
9. Aiello, F., Fortino, G., Gravina, R., Guerrieri, A.: A java-based agent platform for programming wireless sensor networks. The Computer Journal 54(3), 439–454 (2011)
10. Aiello, F., Fortino, G., Guerrieri, A.: Using mobile agents as enabling technology for wireless sensor networks. In: International Conference on Sensor Technologies and Applications, vol. 0, pp. 549–554 (2008), doi: http://doi.ieeecomputersociety.org/10.1109/SENSORCOMM.2008.101
11. Akyildiz, I.F., Su, W., Sankarasubramaniam, Y., Cayirci, E.: Wireless sensor networks: a survey. Comput. Netw. 38, 393–422 (2002), doi:10.1016/S1389-1286(01)00302-4
12. Bonivento, A., Carloni, L.P., Sangiovanni-Vincentelli, A.: Platform based design for wireless sensor networks. Mob. Netw. Appl. 11, 469–485 (2006), doi: http://dx.doi.org/10.1007/s11036-006-7194-1
13. Braginsky, D., Estrin, D.: Rumor routing algorthim for sensor networks. In: Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications, WSNA 2002, ACM, New York (2002), doi: 10.1145/570738.570742
14. Chen, M., Gonzalez, S., Leung, V.C.M.: Applications and design issues for mobile agents in wireless sensor networks. IEEE Wireless Communications 14(6), 20–26 (2007), doi:10.1109/MWC.2007.4407223
15. Chen, M., Kwon, T., Choi, Y.: Data dissemination based on mobile agent in wireless sensor networks. In: Proceedings of the IEEE Conference on Local Computer Networks 30th Anniversary, LCN 2005, pp. 527–529. IEEE Computer Society, Washington, DC (2005), doi: http://dx.doi.org/10.1109/LCN.2005.44
16. Chen, M., Kwon, T., Yuan, Y., Leung, V.: Mobile agent based wireless sensor networks. Journal of Computers 1(1), 14–21 (2006)
17. Dikaiakos, M.D., Kyriakou, M., Samaras, G.: Performance Evaluation of Mobile-Agent Middleware: A Hierarchical Approach. In: Picco, G.P. (ed.) MA 2001. LNCS, vol. 2240, p. 244. Springer, Heidelberg (2001)
18. Egea-Lopez, E., Vales-Alonso, J., Martinez-Sala, A., Pavon-Mario, P., Garcia-Haro, J.: Simulation scalability issues in wireless sensor networks. IEEE Communications Magazine 44(7), 64–73 (2006)
19. Farinelli, A., Rogers, A., Petcu, A., Jennings, N.R.: Decentralised coordination of low-power embedded devices using the max-sum algorithm. In: Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS 2008, vol. 2. International Foundation for Autonomous Agents and Multiagent Systems, Richland (2008)
20. Fok, C.L., Roman, G.C., Lu, C.: Agilla: A mobile agent middleware for self-adaptive wireless sensor networks. ACM Trans. Auton. Adapt. Syst. 4(3), 1–26 (2009), doi: http://doi.acm.org/10.1145/1552297.1552299

21. Gan, L., Liu, J., Jin, X.: Agent-based, energy efficient routing in sensor networks. In: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS 2004, vol. 1, pp. 472–479. IEEE Computer Society, Washington, DC (2004), doi:10.1109/AAMAS.2004.53
22. González-Valenzuela, S., Chen, M., Leung, V.C.: Programmable middleware for wireless sensor networks applications using mobile agents. Mob. Netw. Appl. 15, 853–865 (2010), doi: http://dx.doi.org/10.1007/s11036-010-0237-7
23. Jennings, N., Wooldridge, M.: Agent-oriented software engineering. In: Handbook of Agent Technology (2001)
24. Keutzer, K., Newton, A.R., Rabaey, J.M., Sangiovanni-Vincentelli, A.: System-level design: orthogonalization of concerns and platform-based design. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 19(12), 1523–1543 (2000), doi:10.1109/43.898830
25. Kwon, Y., Sundresh, S., Mechitov, K., Agha, G.: Actornet: an actor platform for wireless sensor networks. In: Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS 2006, pp. 1297–1300. ACM, New York (2006), doi: http://doi.acm.org/10.1145/1160633.1160871
26. Lange, D.B., Oshima, M.: Seven good reasons for mobile agents. Commun. ACM 42, 88–89 (1999), doi: http://doi.acm.org/10.1145/295685.298136
27. Lopes, R., Assis, F., Montez, C.: MASPOT: A Mobile Agent System for Sun SPOT. In: Proceedings of the 2011 Tenth International Symposium on Autonomous Decentralized Systems, ISADS 2011, pp. 25–31. IEEE Computer Society, Washington, DC (2011), doi:10.1109/ISADS.2011.10
28. Luck, M., McBurney, P., Preist, C.: A manifesto for agent technology: Towards next generation computing. Autonomous Agents and Multi-Agent Systems 9, 203–252 (2004), doi:10.1023/B:AGNT.0000038027.29035.7c
29. Mpitziopoulos, A., Gavalas, D., Konstantopoulos, C., Pantziou, G.: Mobile agent middleware for autonomic data fusion in wireless sensor networks, pp. 57–81 (2009), doi:10.1007/978-0-387-89828-5_3
30. Muldoon, C., O'Hare, G., O'Grady, M., Tynan, R.: Agent migration and communication in WSNs. In: 2008 Ninth International Conference on Parallel and Distributed Computing, Applications and Technologies, pp. 425–430. IEEE (2008)
31. Osterlind, F., Dunkels, A., Eriksson, J., Finne, N., Voigt, T.: Cross-level sensor network simulation with cooja. In: Proceedings 2006 31st IEEE Conference on Local Computer Networks, pp. 641–648. IEEE (2006)
32. Qi, H., Xu, Y., Wang, X.: Mobile-agent-based collaborative signal and information processing in sensor networks. Proceedings of the IEEE 91(8), 1172–1183 (2003), doi:10.1109/JPROC.2003.814927
33. Rogers, A., Corkill, D.D., Jennings, N.R.: Agent technologies for sensor networks. IEEE Intelligent Systems 24, 13–17 (2009), doi: http://doi.ieeecomputersociety.org/10.1109/MIS.2009.22
34. Römer, K., Kasten, O., Mattern, F.: Middleware challenges for wireless sensor networks. SIGMOBILE Mob. Comput. Commun. Rev. 6, 59–61 (2002), http://doi.acm.org/10.1145/643550.643556, doi: http://doi.acm.org/10.1145/643550.643556
35. Suenaga, S., Honiden, S.: Enabling direct communication between mobile agents in wireless sensor networks. In: 1st Int'l Workshop on Agent Technology for Sensor Networks (ATSN 2007), Jointly Held with 6th Int'l Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2007), Honolulu, Hawaii (May 14, 2007)

36. Szumel, L., LeBrun, J., Owens, J.D.: Towards a mobile agent framework for sensor networks. In: Proceedings of the 2nd IEEE Workshop on Embedded Networked Sensors, pp. 79–87. IEEE Computer Society, Washington, DC (2005)
37. Tseng, Y.C., Kuo, S.P., Lee, H.W., Huang, C.F.: Location Tracking in a Wireless Sensor Network by Mobile Agents and Its Data Fusion Strategies. In: Zhao, F., Guibas, L.J. (eds.) IPSN 2003. LNCS, vol. 2634, pp. 625–641. Springer, Heidelberg (2003)
38. Vinyals, M., Rodríguez-Aguilar, J.A., Cerquides, J.: A survey on sensor networks from a multi-agent perspective. The Computer Journal 54(3), 455–470 (2010)
39. Wada, H., Boonma, P., Suzuki, J., Oba, K.: Modeling and executing adaptive sensor network applications with the matilda uml virtual machine. In: Proceedings of the 11th IASTED International Conference on Software Engineering and Applications, pp. 216–225. ACTA Press, Anaheim (2007), http://dl.acm.org/citation.cfm?id=1647636.1647674

Chapter 9
Argumentative Agents for Service-Oriented Computing

M. Morge, J. McGinnis, S. Bromuri, P. Mancarella, K. Stathis, and F. Toni

Abstract. We propose an argumentation-based agent model that supports service and partner selection in service-oriented computing settings. In this model, argumentation is also used to help agents resolve conflicts between themselves, whenever negotiation is required for the provision of complex services. The model relies upon an argumentation framework that is used in a modular architecture where Knowledge, Goals, Decisions and Priorities are manipulated by three specialized modules dealing with decision making, communication and negotiation. We formulate a distributed e-procurement process to illustrate how our agents select services and partners and can negotiate with one another.

Maxime Morge
Université Lille 1, France
e-mail: Maxime.Morge@univ-lille1.fr

Jarred McGinnis
Press Association, London UK
e-mail: Jarred.Mcginnis@pressassociation.com

Stefano Bromuri
University of Applied Science, Western Switzerland
e-mail: Stefano.Bromuri@hevs.ch

Paolo Mancarella
Università di Pisa, Italy
e-mail: Paolo.Mancarella@unipi.it

Kostas Stathis
Royal Holloway, University of London, UK
e-mail: Kostas.Stathis@cs.rhul.ac.uk

Francesca Toni
Imperial College London, UK
e-mail: ft@doc.ic.ac.uk

M. Ganzha & L. C. Jain (Eds.): Multiagent Systems & Applications, ISRL 45, pp. 217–255.
springerlink.com

1 Introduction

Service-oriented computing (SOC) is an emerging inter-disciplinary paradigm for distributed computing, which is changing the way software applications are developed, deployed and utilised. The central theme of service-oriented computing are services that provide autonomous, platform-independent, computational elements that can be described, published, discovered, orchestrated and programmed using standard protocols to build networks of collaborating applications distributed within and across organizational boundaries.

The underlying principles and methodologies of SOC assume a service-oriented architecture (SOA). A basic SOA is often understood as defining an interaction between software agents as an exchange of messages between service requesters (clients) and service providers [50]. Clients are software agents that request the execution of a service. Providers are software agents that provide the service. Agents can be simultaneously both service clients and providers. Providers are responsible for publishing a description of the service(s) they provide. Clients must be able to find the description(s) of the services they require and must be able to bind to them. The basic SOA is not an architecture only about services, it is a relationship of three kinds of participants: the service provider, the service discovery agency, and the service requestor (client).

One of the main issues for SOC applications is how to compose services by assuming an open environment where the relationship of service providers, clients and discovery agencies change over time as new agents enter and/or leave the application dynamically. The issue here is how to establish a possible service composition by negotiating terms and conditions of the participating services for a given duration and according to the requirements and the goals the clients and the providers have to satisfy. The problem here is how to develop mechanisms that allow this negotiation to take place by reducing possible conflicts to support successful service compositions.

To address the issues involved in negotiating complex and composite services, we present an argumentation-based model of agency and its associated architecture to support SOC applications. Argumentation is used to support the agent to reason about services, engage in communicative interaction with other agents to negotiate desired service properties, resolve conflicts and eventually make decisions of which service or partner to select. Argumentation's main strength is that it allows an agent to plead for and against conclusions [52], providing a powerful deliberative and dialectical model for interacting, decision-making agents to assess the validity of received information and to be able to resolve conflicts and differences of opinion. It is an essential ingredient of decision-making [29, 4, 7, 45, 48], inter-agent communication [40], and negotiation [53, 30, 2].

The proposed agent model was developed within ArguGRID, an EU-funded project[1] that started in 2006 and ended in 2009. ArguGRID has also developed a multi-agent systems platform, a peer-to-peer infrastructure and an environment for users to interact with agents and the overall system, by authoring workflows of

[1] http://www.argugrid.eu/

services, hosted upon a Grid platform. While presenting the agent model, we will also emphasise the links between this and the other components of the ArguGRID system.

As shown in Figure 1, argumentation agents in ArguGRID are deployed within a Grid/Service-oriented platform to represent service clients and providers. The platfom consists of four main interacting components.

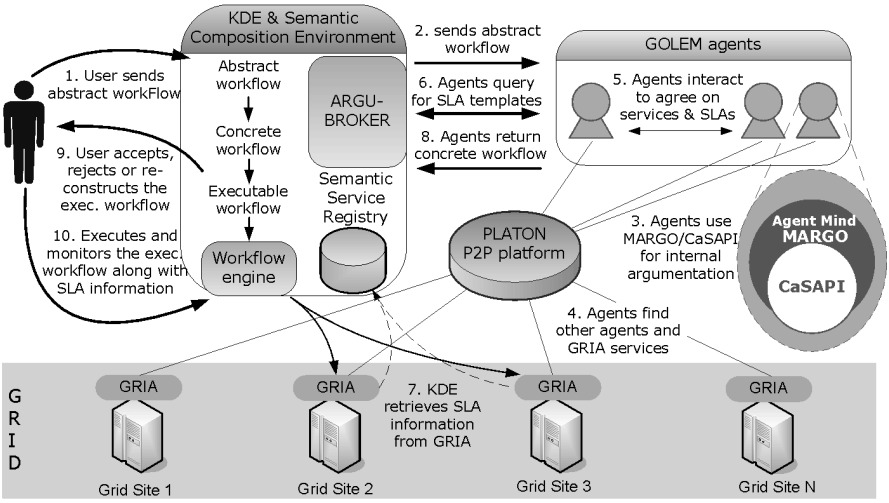

Fig. 1 ArguGRID Platform

KDE is a commercial software tool developed by InforSense Ltd[2]. This system provides facilities to build end user application as remote Web services or Grid services [24]. For the needs of ArguGRID, the KDE system has been extended to support semantic descriptions. In this way, requester goals representing user requirements can be matched with concrete services and can be executed within the Grid infrastructure. KDE was chosen as a supporting technology for the ArguGRID project because InforSense was the main developer of KDE and ArguGRID had several exploitation advantages from the perspective of a company working on workflow management systems.

GRIA[3] is the GRID middleware that ArguGRID has chosen to use to support its scenarios. GRIA is a service-oriented infrastructure designed particularly to support Business-to-Business collaborations (such as the ones required by the ArguGRID scenarios) through service provision across organisational boundaries in a secure, interoperable and flexible manner. In particur GRIA was chosen as it is one of the main technologies to represent Web services on the Grid.

[2] http://www.inforsense.com
[3] http://www.gria.org

PLATON[4] (Peer-to-Peer Load Adjusting Tree Overlay Networks) is a Peer-to-Peer platform supporting multi-attribute and range queries [36] which has been developed in the ArguGRID project to support service discovery mechanisms for load-balancing of peer resources. Load-balancing of peer resources is necessary in order to guarantee logarithmic querying time using any distributed tree-based multi-attribute Peer-to-Peer platform. The reason to include a P2P platform in the project was to allow ArguGRID to scale up to possibly thousands of services. In particular PLATON allows to perform multidimensional queries on our Web services, meaning that we can represent Web services in terms of complex descriptions as we will see later in Section 6.

GOLEM[5] (Generalized OntoLogical Environments for Multi-agent systems) is an agent environment middleware [10] which has been developed in the ArguGRID project. In this chapter, we present how to use GOLEM to host MARGO agents. The deployment of these agents will be described in Section 6. The GOLEM agent platform was chosen as it allows a simple integration between agents and external entities represented as objects in the GOLEM agent environment. In particular, GOLEM has been helpful to represent Web services in terms of objects accessible by the agents of ArguGRID.

As shown in Figure 1, the operation of the platform from the point of view of service requestors can briefly be explained as follows:

1. a user develops an abstract workflow reflecting his/her requirements using the KDE;
2. the abstract workflow is then communicated to the agent that reasons about services and makes decisions, aiding the refinement process of the abstract workflow;
3. the reasoning capabilities of the agent are based on using the MARGO argumentation engine for decision-making, which in turn uses the CaSAPI general-purpose argumentation engine;
4. in order to discover an appropriate agent or a GRIA Grid service, agents are given the capability to use the P2P platform, linking all available agents and GRIA services in a virtual registry that can be queried;
5. an agent can negotiate with other agents to find a concrete workflow whose execution will satisfy the application requirements, as stated in the abstract workflow;
6. agents can query the KDE system about SLA (Service Level Agreement) templates, needed for creating a concrete workflow that satifies user preferences.
7. the KDE, by accesing GRIA nodes, can retrieve information about SLA's;
8. having carried out its mission, the agent representing the user (i.e. the initial agent that received the abstract workflow from the KDE) will return to the KDE the concrete workflow, constituted of a set of GRIA services to be executed in a certain manner/sequence;

[4] http://platonp2p.sourceforge.net
[5] http://www.golem.cs.rhul.ac.uk

9. the KDE shows the suggested workflow to the user. If the user rejects the suggestion, a new workflow would be created;
10. the KDE executes and monitors the resulting workflow. Details about execution and the final results are presented to the user.

Given the ArguGRID context, the main focus and contribution of this work is an argumentation-based agent mind applied to service selection and partner selection in an e-procurement scenario as described in [58]. The e-procurement scenario illustrates how agents support the selection and provision of services for their integration in an open and distributed environment. In this context, a human user requesting a service is only required to specify an abstract description of his needs for these services, possibly with some constraints and preferences about them. The selection of these services, as well as the selection of partners, are tasks delegated to the autonomous agents.

The mind of our agents is composed of three main modules: (i) the *individual decision making* module, allowing to translate the goals, preferences, and constraints provided by the user requesting a service to an internal and abstract representation of the user's needs (ii) the *social decision making* module, allowing to move, by means of negotiation, from these abstract representations to concrete ones, in terms of contracts; (iii) the *social interaction* module, managing the communication given social rules of interaction.

Note that the services available in ArguGRID are encapsulated in Web services and available to the agents deployed in the ArguGRID platform. In this chapter we do not focus on how these services are implemented, but we rather focus on how the agents can select these services, according to their characteristics, by means of argumentation. Thus, we are not concerned with defining semantic Web reasoners or ontologies to describe Web services. We have assumed a fixed ontology that is common to the agents involved in the procurement process and in order to focus on argumentation, communication protocols and the reasoning process.

The remainder of this chapter is organised as follows: Section 2 provides a detailed discussion about the motivating e-procurement scenario; Section 3 discusses the background formalisms and technologies on which we built the agent interaction in our ArguGRID e-procurement scenario; Section 4 provides the architecture of our cognitive model; Section 5 illustrates the agents' deliberation and communication capabilities with a case run of a e-procurement scenario; Section 6 describes implementation issues related to the deployment of our multi-agent system; Section 7 discusses related work; finally, Section 8 concludes by summarising our proposal and discussing our future plans.

2 Motivation: An E-Procurement Scenario

In order to illustrate our model and architecture, we consider an e-procurement scenario, introduced in [58], where a buyer seeks to purchase combined services/products from A-type and B-type suppliers. The e-procurement scenario is particularly suitable for our evaluation as it implies the use of distributed agents representing

distributed services that require to reach an agreement by means of negotiation protocols between buyers and suppliers. The suppliers combine their competencies in order to provide solutions for the buyer. Each agent is representing a user, i.e. a service requester or a service provider. Then, each agent may be responsible for many different Web services. This operation is typically achieved by means of a 6-steps procurement process whereby: a requester looks for potential suppliers (step 1), gathers information in order to evaluate the potential suppliers (step 2), creates a short-list according to this information (step 3). The requester asks the short-listed suppliers to provide a quote for the services (step 4), chooses one of the suppliers (step 5), and finally the requester and the winner negotiate the terms & conditions of the contract, such as the price and the warranty (step 6). If this last step is unsuccessful, the process goes back to the step 5. When no suitable service is found, the requester agent asks its user to reconsider his needs by relaxing the constraints. Figure 2 summarises these steps and indicates the types of dialogues required to support them. These dialogues should conform to suitable protocols.

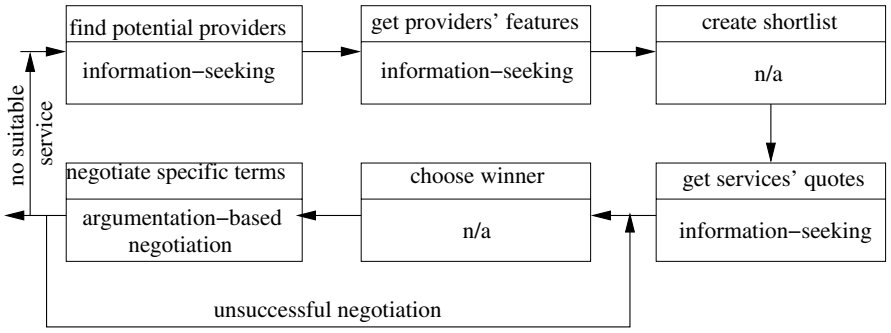

Fig. 2 Deliberative steps for e-procurement

The exchange of offers and acceptances may lead to contracts, i.e. legal relations between providers and requesters that typically force commitments (e.g. obligations) from one agent to another about the provision of services. In this chapter, we will assume that contracts are simple transactions between a provider and a requester, characterised by features of the services provided, and will ignore complex issues such as enforcement of commitments, sanctions, penalties, etc. Our definition of contract is given in Section 4.3. The contracts create virtual organisations (VO, for short) consisting of the buyer, the selected A-type agent and the selected B-type agent.

Within the e-procurement scenario we consider a specific case where a buyer looks for an e-ordering system (S) which is composed of a computer system (S_a) and an Internet connection (S_b). The A-type agents providing S_a are `Al` and `Alice`. While `Alice` is responsible for the concrete instances of services $S_a(a_1)$ and $S_a(a_2)$, `Al` can provide $S_a(a_3)$ and $S_a(a_4)$. The B-type agents providing S_b are `Bob`

and Barbara. Bob is representing the concrete services $S_b(c)$, $S_b(d)$, $S_b(e)$ and $S_b(f)$. On the other hand, Barbara is representing the services $S_b(g)$ and $S_b(h)$. A-type agents are responsible to select B-type agents and provide the combined S to the buyer (cf Figure 3). A possible VO in this case may consist of Bob, Al and the buyer, with two contracts: the first is between the buyer and Al and concerns the combined service S, the second is between Al and Bob and concerns service S_b. These contracts may be negotiated as follows. At first, the buyer plays the role of requester and all A-type agents play the role of potential suppliers in an e-procurement process for service S. In turn, the A-type agents play the role of requesters and the B-type agents play the role of potential suppliers in an e-procurement process for service S_b.

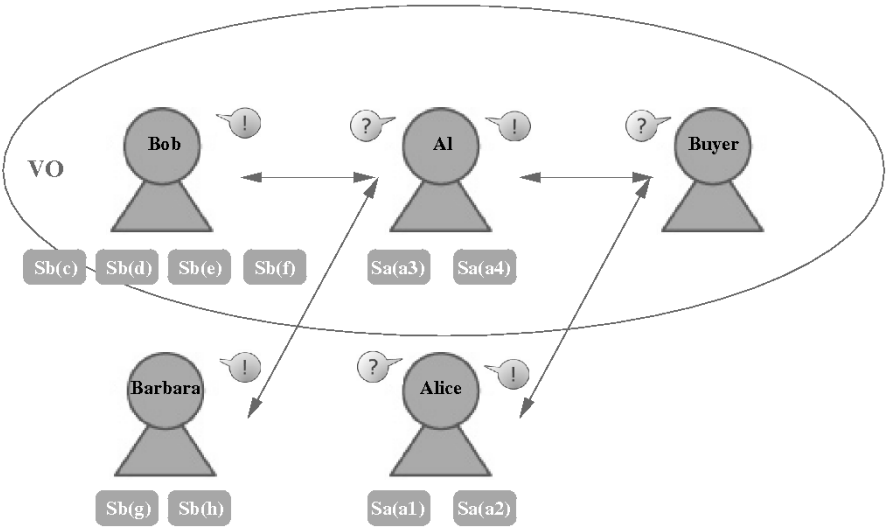

Fig. 3 Use case scenario

In the remainder of this chapter, we will focus on the e-procurement process involving Al as a requester, and Bob as a supplier. Al's goal consists of finding and agreeing to a service S_b provided by a B-type agent. According to its preferences and constraints:

- the cost of the service must be low (e.g. within a budget of 10 euros per month), and
- the quality of the service must be high (e.g. with warranty of a 24 hours assistance).

Taking into account its goal and preferences/constraints, Al needs to solve a decision-making problem where the decisions amount to a service and a supplier for that service. On the other hand, the goal of the supplier Bob consists of a decision-making problem about providing a service. According to Bob's preferences and constraints:

- the cost of the service must be high (e.g. within a budget of 40 euros per month), and
- the quality of the service must be low (e.g. with warranty of a 2 hours daily assistance).

A negotiation is a multi-step interaction between Al and Bob by means of which the agents aim at achieving the common goal to make a deal while resolving their conflicting interests about the cost and the quality of the service. The main strength of the argumentation-based approach we will use is to provide extra information during the interaction. For this purpose, we will assume the agents share a set of common ontologies to describe the services (in particular the intervals in which the price and warranty can be contracted). Moreover, argumentation-based decision making allows agents to identify and resolve the possible conflicts. The two decision making processes of the participants take place in a dynamic setting, whereby information about other agents, the services they require/provide and the characteristics of these services are obtained incrementally within the e-procurement process outlined in Figure 2. As we will see later on, the outcome of this process is the contract obliging Bob to provide a service S_b to Al, with low cost (e.g. 10 euros per month) and low quality (e.g. a warranty of a 2 hours daily assistance).

Within our proposed agent model and architecture, the decision-making process of each agent, as well as the e-procurement negotiation process are supported by argumentation. In the concrete use case, Al, as a requester, uses argumentation to collect information on the available services and on the suppliers. For instance, Al can ask Alice its opinion about Bob and ask to justify it (by providing an argument for it). Al (respectively Bob), as a requester (respectively as a provider), uses argumentation to decide which service it needs (respectively it can provide) taking into account the conflicting preferences/constraints and possibly the inconsistency of information it has gathered. Moreover, through argumentation, the participants provide an interactive and intelligible explanation of their choices. For instance, Al can argue that a service is a good deal when its cost is low. The previous argument will incite Bob to suggest services with a low cost to reach quickly a good deal. Thus, in our framework agents can use argumentation to influence each other.

3 Background on Argumentation and Protocol Language

In this section we discuss about the background formalisms that are necessary to understand the interaction happening in our ArguGRID e-procurement scenario.

3.1 MARGO

MARGO[6] (Multiattribute ARGumentation framework for Opinion explanation), written in Prolog, is the engine developed in the ArguGRID project for service selection and partner selection. We briefly present here the computational argumentation framework for decision-making which has been proposed in [45]. We do not describe the computational counterpart of MARGO (see [46] for more details) but the reader can notice that our framework is computed by the dialectical proof procedure of [19] extended in [23]. Concretely, MARGO is built upon the argumentation engine CaSAPI[7]. So that, we can compute the decisions. Additionally, MARGO models the intuition that high-ranked goals are preferred to low-ranked goals which can be withdrawn. At first, we introduce here argumentation. Then, we define the framework which captures decision making problems. Finally, we define the arguments and their interaction.

3.1.1 Abstract Argumentation

The framework proposed in this chapter is based on Dung's abstract approach to defeasible argumentation [18] which considers *arguments* as atomic and abstract entities interacting through a *defeat* relation over these[8].

Definition 1 (AAF). An *abstract argumentation framework* AAF is a pair $\langle \mathscr{A}, \text{ defeats } \rangle$ where $\mathscr{A}$ is a finite set of arguments and defeats $\subseteq \mathscr{A} \times \mathscr{A}$ is a binary relation over $\mathscr{A}$. We say that an argument $\mathtt{b}$ defeats an argument $\mathtt{a}$ if $(\mathtt{b},\mathtt{a}) \in$ defeats . Moreover, we say that a set of arguments $\mathtt{S}$ defeats an argument $\mathtt{a}$ if $(\mathtt{b},\mathtt{a}) \in$ defeats for some $\mathtt{b}$ in $\mathtt{S}$.

An argument can be viewed as a reason supporting a claim which can be disputed by other reasons defeating it.

According to this framework, Dung introduces various extension-based semantics in order to analyse whenever a set of arguments can be considered as collectively justified.

Definition 2 (Semantics). Let $\langle \mathscr{A}, \text{ defeats } \rangle$ be an AAF. For a set of arguments $\mathtt{S} \subseteq \mathscr{A}$, we say that:

- $\mathtt{S}$ is *conflict-free* iff $\forall \mathtt{a},\mathtt{b} \in S$ $\mathtt{a}$ does not defeat $\mathtt{b}$;
- $\mathtt{S}$ is *admissible* iff $\mathtt{S}$ is conflict-free and $\mathtt{S}$ defeats every argument $\mathtt{a}$ such that $\mathtt{a}$ defeats some arguments in $\mathtt{S}$;
- $\mathtt{S}$ is *preferred* iff $\mathtt{S}$ is maximally (wrt set inclusion) admissible;
- $\mathtt{S}$ is *complete* iff $\mathtt{S}$ is admissible and $\mathtt{S}$ contains all arguments $\mathtt{a}$ such that $\mathtt{S}$ defeats all defeaters against $\mathtt{a}$;
- $\mathtt{S}$ is *grounded* iff $\mathtt{S}$ is minimally complete.

[6] http://margo.sourceforge.net
[7] http://www.doc.ic.ac.uk/ ft/CaSAPI/
[8] Actually, the defeat relation is called attack in [18].

These declarative *semantics* capture various degrees of justification ranging from very permissive conditions, called *credulous*, to restrictive requirements, called *sceptical*. The semantics of an admissible (or preferred) set of arguments is credulous. However, there might be several conflicting admissible sets. That is the reason why various sceptical semantics have been proposed, notably the grounded semantics. Since an ultimate choice between various justified sets of alternatives is not always possible, we consider in this chapter only the credulous semantics.

3.1.2 Decision Framework

The problem of selecting services and partners can be seen as a multi-criteria decision problem with incomplete knowledge. In this chapter, we use influence diagrams to create a model and a representation of this kind of problem.

We assume that users provide, via a Graphical User Interface (GUI), influence diagrams. *Influence diagrams* are simple graphical representations of decision problems [13]. The elements of decision problems (decisions to make, uncertain events, and the value of outcomes) are represented in influence diagrams as nodes of different shapes. These nodes are linked to show the relationship between the elements. The nodes are of the following types: *decision nodes* (represented as squares), *chance nodes* (represented as ovals), and *value nodes* (represented as rectangles with rounded corners). Nodes are connected in a graph connected by arrows, called *arcs*. We call a node at the beginning of an arc a *predecessor* and one at the end of an arc a *successor*. The nodes are connected by arcs where predecessors are independent and affect successors. Influence diagrams which are properly constructed have no cycles. In order to capture multi-criteria decision making, it is convenient to include an additional type of node that aggregates results from predecessor nodes. An *abstract value* node is a special kind of value node represented by a rectangle with rounded corners and double borders. A *concrete value* is specified for every possible combination of decisions and events that feed into this node. By contrast, an abstract value is specified for every possible combination of values that feed into this node. In many cases, decision is a matter of trade-offs between the attributes of the outcomes. In such a case, it is possible to represent explicitly the multiple attributes with a hierarchy of values where the top, abstract values aggregate the lower, concrete values. In addition, the GUI allows the user to communicate user-specific preferences over values and events.

Influence diagrams can be mapped into decision frameworks of the following form:

Definition 3 (Decision framework). A *decision framework* is a tuple $\mathscr{D} = \langle \mathscr{L}, \mathscr{A}sm, \mathscr{I}, \mathscr{T}, \succ \rangle$, where:

- $\mathscr{L}$ is the *object language* which captures the statements about the decision problem;
- $\mathscr{A}sm$, is a set of sentences in $\mathscr{L}$ which are taken for granted, called *assumptions*;
- $\mathscr{I}$ is the *incompatibility relation*, i.e. a binary relation over atomic formulas which is asymmetric;

- $\mathscr{T}$ is the *theory* which gathers the statements;
- $\succ \subseteq \mathscr{T} \times \mathscr{T}$ is a transitive, irreflexive and asymmetric relation over $\mathscr{T}$, called the *priority* relation.

In the object language $\mathscr{L}$, we distinguish six disjoint components:

- a set of *abstract goals*, i.e. some propositional symbols which represent the abstract features that the decisions must exhibit implicitly;
- a set of *concrete goals*, i.e. some propositional symbols which represent the concrete features that the decisions must exhibit explicitly;
- a set of *decisions*, i.e. some predicate symbols which represent the actions which must be performed or not;
- a set of *alternatives*, i.e. some constant symbols which represent the mutually exclusive actions for each decision;
- a set of *beliefs*, i.e. some predicate symbols which represent epistemic statements;
- a set of *names* of rules in $\mathscr{T}$ (each rule has a distinguished name).

The abstract (respectively concrete) goals represent the abstract (respectively concrete) value nodes, the decisions represent the decision nodes, and the beliefs represent the chance nodes in influence diagrams. Since we consider multi-criteria decision problems, goals are structured hierarchically, where the top, abstract goals aggregate the independent lower goals.

We explicitly distinguish *assumable* (respectively *non-assumable*) literals which can (respectively cannot) be taken for granted, meaning that they can (respectively cannot) be assumed to hold as long as there is no evidence to the contrary. Decisions as well as some beliefs can be taken for granted. In this way, a decision framework can capture incomplete knowledge.

Since we want to consider conflicts in this object language, we need some form of negation. For this purpose, we consider *strong negation*, also called *explicit* or *classical negation*, and *weak negation*, also called *negation as failure*. A strong literal is an atomic first-order formula, possible preceded by strong negation $\neg$. A weak literal is a literal of the form $\sim L$, where L is a strong literal of $\mathscr{L}$. $\neg L$ says "L is definitely not the case", while $\sim L$ says "There is no evidence that L is the case". In order to express the mutual exclusion between statements, such as the different alternatives for a decision, we define the *incompatibility relation* (denoted by $\mathscr{I}$) as a binary relation over atomic formulas which is asymmetric. In other words, the incompatibility relation captures the conflicts between decisions, beliefs and goals. Whatever the atom L is, we have $L \mathscr{I} \neg L$ and $\neg L \mathscr{I} L$, while we have $L \mathscr{I} \sim L$ but we do not have $\sim L \mathscr{I} L$. Obviously, $D(a_1) \mathscr{I} D(a_2)$ and $D(a_2) \mathscr{I} D(a_1)$, D being a decision predicate, a_1 and a_2 being different[9] alternatives for D. We say that two sets of sentences Φ_1 and Φ_2 are incompatible (denoted $\Phi_1 \mathscr{I} \Phi_2$) iff there is a sentence ϕ_1 in Φ_1 and a sentence ϕ_2 in Φ_2 such that $\phi_1 \mathscr{I} \phi_2$.

A theory gathers the statements about the decision problem.

[9] Notice that in general a decision can be addressed by more than two alternatives.

Definition 4 (Theory). A *theory* $\mathscr{T}$ is an extended logic program, *i.e* a finite set of rules R: $L_0 \leftarrow L_1,\ldots,L_j,\sim L_{j+1},\ldots,\sim L_n$ with $n \geq 0$, each L_i (with $i \geq 0$) being a strong literal in $\mathscr{L}$. The literal L_0, called the *head* of the rule, is denoted $\texttt{head}(R)$. The finite set $\{L_1,\ldots,\sim L_n\}$, called the *body* of the rule, is denoted $\texttt{body}(R)$. The body of a rule can be empty. In this case, the rule, called a *fact*, is an unconditional statement. R, called the unique *name* of the rule, is an atomic formula of $\mathscr{L}$. All variables occurring in a rule are implicitly universally quantified over the whole rule. A rule with variables is a scheme standing for all its ground instances.

For simplicity, we will assume that the names of rules are neither in the bodies nor in the head of the rules thus avoiding self-reference problems. Since we obtain our decision problems from influence diagrams, we assume that the elements in the body of rules are independent, the decisions do not influence the beliefs, and the decisions have no side effects.

Considering a decision problem, we distinguish:

- *goal rules* of the form R: $G_0 \leftarrow G_1,\ldots,G_n$ with $n > 0$. Each G_i is a goal literal in $\mathscr{L}$. The head of the rule is an abstract goal (or its strong negation). According to this rule, the abstract goal is promoted (or demoted) by the combination of goal literals in the body;
- *epistemic rules* of the form
  R: $B_0 \leftarrow B_1,\ldots,B_n$ with $n \geq 0$. Each B_i is a belief literal of $\mathscr{L}$. According to this rule, B_0 is true if the conditions $B_1,\ldots,B_n$ are satisfied;
- *decision rules* of the form
  R: $G \leftarrow D(a),B_1,\ldots,B_n$ with $n \geq 0$. The head of the rule is a concrete goal (or its strong negation). The body includes a decision literal ($D(a) \in \mathscr{L}$) and a set of belief literals possibly empty. According to this rule, the concrete goal is promoted (or demoted) by the decision $D(a)$, provided that conditions $B_1,\ldots,B_n$ are satisfied.

Considering statements in the theory is not sufficient to make a decision. In order to evaluate the previous statements, other relevant pieces of information should be taken into account, such as the uncertainty of beliefs, the priority between goals, or the expected utilities of the decisions. For this purpose, we consider the *priority* relation $\succ$ on the rules in $\mathscr{T}$, which is transitive, irreflexive and asymmetric. $R_1 \succ R_2$ can be read "R_1 has priority over R_2". $R_1 \not\succ R_2$ can be read "R_1 has no priority over R_2", either because R_1 and R_2 are *ex æquo*, or because R_1 and R_2 are not comparable.

In this work, we consider that all rules are potentially defeasible and that the priorities are domain-specific and extra-logical features, neither determined nor justified by consideration of logic. The priority over concurrent rules depends of the nature of rules. Rules are *concurrent* if their heads are identical or incompatible. We define three priority relations between concurrent rules:

- the priority over *decision rules* comes from the *expected utility* of decisions. The priority of such rules corresponds to the expectation of the conditional decision in reaching the goal literal. For instance, we need to choose a train or a flight in order to go from Pisa to Roma. Even if a direct flight exists, train is cheaper.

The expected utilities of these decisions can be captured by the priority between concurrent decision rules;

- the priority over *goal rules* comes from the *priority* overs goals. The priority of such rules corresponds to the relative importance of the combination of (sub)goals in the body as far as reaching the goal literal in the head is concerned. For instance, we prefer a cheap travel rather than an expensive one which is fast. This preference can be captured by the priority between concurrent goal rules;
- the priority over *epistemic rules* comes from the *uncertainty* of knowledge. The priority of such rules corresponds to the likelihood of the rules. For instance, an Italian travel agency asserts that there is a direct flight between Pisa and Roma, while an international travel agency asserts that this is not the case. If you trust more the Italian agency, the fact that a direct flight exists is likely. The likelihood of a belief can be captured by the priority between concurrent epistemic rules.

3.1.3 Structure of Arguments

In order to turn the decision framework presented in the previous section into a concrete argumentation framework, we need first to define the notion of argument. Since we want that our AF not only suggests some decisions but also provides an intelligible explanation of them, we adopt a tree-like structure of arguments. We adopt here the tree-like structure for arguments proposed in [62] and we extend it with presumptions on the missing information.

Informally, an argument is a deduction for a conclusion from a set of suppositions represented as a tree, with conclusion at the root and suppositions at the leaves. Nodes in this tree are connected by the inference rules, with sentences matching the head of an inference rule connected as parent nodes to sentences matching the body of the inference rule as children nodes. The leaves are either suppositions or the special extra-logical symbol θ, standing for an empty set of premises. Formally:

Definition 5 (Argument). An *argument* is composed of a conclusion, a top rule, some premises, some suppositions, and some sentences. These elements are abbreviated by the corresponding prefixes. An argument $\mathtt{a}$ may be one of the following:

1. a *hypothetical argument* with:

$$\begin{aligned}&\mathtt{conc}(\mathtt{a}) = L,\\ &\mathtt{top}(\mathtt{a}) = \theta,\\ &\mathtt{premise}(\mathtt{a}) = \emptyset,\\ &\mathtt{supp}(\mathtt{a}) = \{L\},\\ &\mathtt{sent}(\mathtt{a}) = \{L\}.\end{aligned}$$

   where L is an assumable belief literal.
   or
2. a *built argument* which may be

   2.1) a *trivial* argument $\mathtt{a}$ built upon a fact f in $\mathscr{T}$ (i.e. $\mathtt{body}(f) = \emptyset$), defined as follows:

$$\begin{aligned}&\texttt{conc}(\texttt{a}) = \texttt{head}(f),\\ &\texttt{top}(\texttt{a}) = f,\\ &\texttt{premise}(\texttt{a}) = \emptyset,\\ &\texttt{supp}(\texttt{a}) = \emptyset,\\ &\texttt{sent}(\texttt{a}) = \{\texttt{head}(f)\}.\end{aligned}$$

2.2) a *tree* argument a built upon the rule r and the set $\{\texttt{a}_1,\ldots,\texttt{a}_n\}$ of arguments, where r is a rule in $\mathscr{T}$ with $\texttt{body}(r) = \{L_1,\ldots,L_j,\sim L_{j+1},\ldots,\sim L_n\}$ and there is a collection of arguments $\{\texttt{a}_1,\ldots,\texttt{a}_n\}$ such that, for each strong literal $L_i \in \texttt{body}(r)$, $\texttt{conc}(\texttt{a}_i) = L_i$ with $i \leq j$ and for each weak literal $\sim L_i \in \texttt{body}(r)$, $\texttt{conc}(\texttt{a}_i) = \sim L_i$ with $i > j$. a is defined as follows:

$$\begin{aligned}&\texttt{conc}(\texttt{a}) = \texttt{head}(r),\\ &\texttt{top}(\texttt{a}) = r,\\ &\texttt{premise}(\texttt{a}) = \texttt{body}(r),\\ &\texttt{supp}(\texttt{a}) = \cup_{\texttt{a}_i\in\{\texttt{a}_1,\ldots,\texttt{a}_n\}}\texttt{supp}(\texttt{a}_i),\\ &\texttt{sent}(\texttt{a}) = \{\texttt{head}(r)\}\cup\texttt{body}(r)\\ &\qquad\qquad \cup_{\texttt{a}_i\in\{\texttt{a}_1,\ldots,\texttt{a}_n\}}\texttt{sent}(\texttt{a}_i).\end{aligned}$$

The set of arguments $\{\texttt{a}_1,\ldots,\texttt{a}_n\}$ is called the set of *subarguments* of a (denoted $\texttt{sbarg}(\texttt{a})$).

The top rule of an argument is the rule whose head is the conclusion of the argument. Notice that the subarguments of a tree argument concluding the weak literals in the body of the top rule are hypothetical arguments. Indeed, the conclusion of a hypothetical argument could be a strong or a weak literal while the conclusion of a built argument is a strong literal. As in [62], we consider composite arguments, called *tree* arguments, and atomic arguments, called *trivial* arguments. Contrary to other definitions of arguments (set of assumptions, set of rules), our definition considers that the different premises can be challenged and can be supported by subarguments. In this way, arguments are intelligible explanations. Moreover, we consider *hypothetical* arguments which are built upon missing information or a suggestion, i.e. a decision. In this way, our framework allows to reason further by making suppositions related to unknown beliefs and over possible decisions.

3.1.4 Interaction

The interactions between arguments may come from the incompatibility of their sentences, from their nature (hypothetical or built) and from the priority over rules. We examine in turn these different sources of interaction.

Since their sentences are conflicting, the structured arguments interact with one another. For this purpose, we define the following attack relation.

Definition 6 (Attack relation). Let a and b be two arguments. a *attacks* b iff $\texttt{sent}(\texttt{a})\ \mathscr{I}\ \texttt{sent}(\texttt{b})$.

According to this definition, if an argument attacks a subargument, the whole argument is attacked.

Since arguments are more or less hypothetical, we define the size of their suppositions.

Definition 7 (Supposition size). Let a be an argument. The *size of suppositions* for a, denoted suppsz(a), is the number of supposition of a: $\texttt{suppsz(a)} = |\texttt{supp(a)}|$.

Namely, the size of suppositions for an argument is the number of decision literals and assumable belief literals in the sentences of the argument.

Since arguments have different natures (hypothetical or built) and the top rules of built arguments are more or less strong, we define the strength relation as follows.

Definition 8 (Strength relation). Let $\texttt{a}_1, \texttt{a}_2$ be two built arguments. $\texttt{a}_1$ is stronger than $\texttt{a}_2$ (denoted $\texttt{a}_1 \succ \texttt{a}_2$) iff

1. either $(\texttt{top}(\texttt{a}_1) \succ \texttt{top}(\texttt{a}_2))$, then $\texttt{a}_1 \succ \texttt{a}_2$;
2. or $(\texttt{top}(\texttt{a}_1) \not\succ \texttt{top}(\texttt{a}_2)) \wedge (\texttt{suppsz}(\texttt{a}_1) < \texttt{suppsz}(\texttt{a}_2))$, then $\texttt{a}_1 \succ \texttt{a}_2$.

An argument is stronger than another argument if the top rule of the first argument has a proper higher priority that the top rule of the second argument, or if it is not the case but the number of suppositions made in the first argument is properly smaller than the number of suppositions made in the second argument.

The two previous relations can be combined.

Definition 9 (Defeats). Let a and b be two arguments. a *defeats* b iff: i) a attacks b and ; ii) $\neg(\texttt{b} \succ \texttt{a})$.

Our notion of argument and this defeat relation can be used within the Dung's seminal calculus of opposition.

The implementation of this argumentation framework, called MARGO[10], is built upon the implementation of [20] in the CaSAPI system [23].

3.2 Protocol Language

In the ArguGRID project, we use the Lightweight Coordination Calculus (LCC) of [55] which is a declarative logic programming language in the style of Prolog, augmented with CCS (a process calculus for communicating systems). LCC allows to drive all social interactions and allows to write once-execute everywhere protocols.

Figure 4 defines the syntax of the LCC protocol language. A protocol consists of a set of agent clauses, $A^{\{n\}}$. An agent clause is the series of communicative actions expected to be performed by an agent adopting the role defined by the agent definition. This agent definition consists of a role (*role*) and unique identifier (*ag*). The roles act as a bounding box for a set of states and transitions. The agent definition is expanded by a number of operations. Operations can be classified in three ways: actions, control flow, and conditionals. Actions are the sending or receiving of messages, a no op, or the adoption of a role. Control flow operations temporally order

[10] http://margo.sourceforge.net

$$
\begin{array}{lll}
\mathscr{P} & \in \text{Protocol} & ::= A^{\{n\}} \\
\mathscr{A} & \in \text{AgentClause} & ::= \lambda :: \text{op.} \\
\lambda & \in \text{AgentDefinition} & \text{agent}(role,ag) \\
\text{op} & \in \text{Operation} & \text{no op} \\
 & & \text{—}\lambda \\
 & \text{(Precedence)} & \text{—(op)} \\
 & \text{(Send)} & \text{—}M \Rightarrow \lambda \\
 & \text{(Receive)} & \text{—}M \Leftarrow \lambda \\
 & \text{(Sequence)} & \text{—op1 then op2} \\
 & \text{(Parallelization)} & \text{—op1 par op2} \\
 & \text{(Choice)} & \text{—op1 or op2} \\
 & \text{(Prerequiste)} & \text{—}(M \Rightarrow \lambda) \leftarrow \psi \\
 & \text{(Consequence)} & \text{—}\psi \leftarrow (M \Rightarrow \lambda) \\
\text{M} & \in \text{message} & ::= \langle m, \mathscr{P} \rangle
\end{array}
$$

Fig. 4 An Abstract Language of the Protocol Language

the individual actions. Actions can be sequentially ordered, performed simultaneously without regard to order, or given a choice point. The definition of the double arrows denote messages M being sent and received. On the left hand side of the double arrow is the message and on the right-hand side is the other agent involved in the interaction.

Constraints can fortify or clarify the semantics of the protocols. Those occurring on the left of the $\leftarrow$ are post-conditions and those occurring on the right are preconditions. The symbol ψ represents a first order proposition. For example, an agent receiving a protocol with the constraint to believe a proposition $\mathtt{s}$ upon being informed of $\mathtt{s}$ can infer that the agent sending the protocol has a particular semantic interpretation of the act of informing other agents of propositions. The operation $(M \Rightarrow \lambda) \leftarrow \psi$ is understood to mean that message M is being sent to the agent defined as λ on the condition that ψ is satisfiable. The operation $\psi \leftarrow (M \Rightarrow \lambda)$ means that once M is received from agent, ψ holds.

4 Agent Architecture

In this section, we outline the mind/body architecture of our agents, focusing on the mind component.

Our agent architecture, pictured in Figure 5, is adapted from the mind/body architecture of [9], extended in GOLEM [10]. The *body* senses what is external to it by using the *Communication Module* (CM), which can access the external world, i.e. the events generated by the *Graphical User Interface* (GUI) and messages coming from other agents as well as the registry used for the partner discovery. Messages received are then stored in the *Incoming Message Queue* (IMQ) until they are treated. Similarly, the messages that the agent wants to send are stored in the *Outgoing Message Queue* (OMQ). The mind and the body can function as co-routines, thus

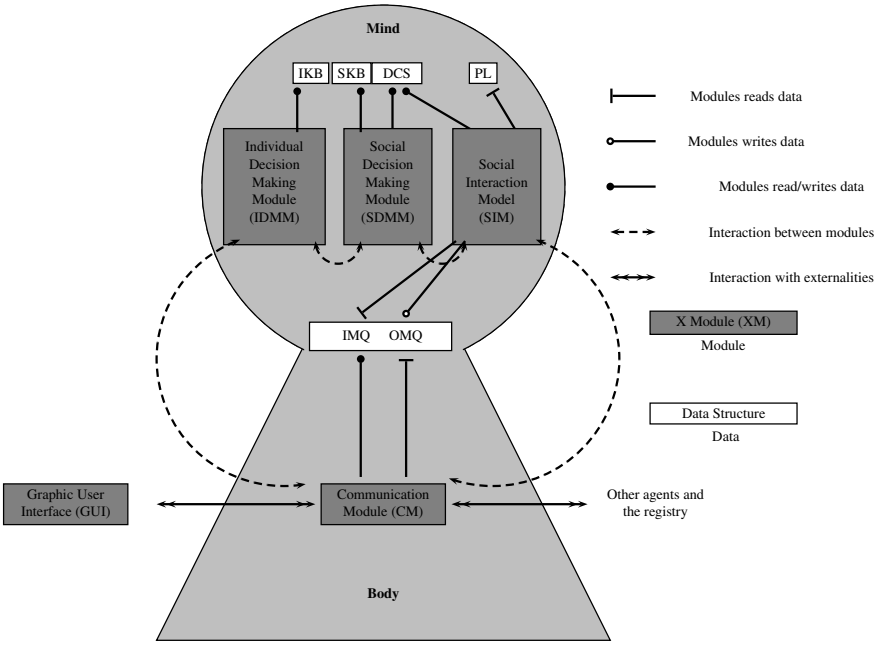

Fig. 5 The modular architecture of agents

allowing the reasoning processes of the mind to be performed concurrently with the body sending and receiving messages.

Our cognitive agents' mind is divided into three modules:the *Individual Decision Making Module* (IDMM), the *Social Decision M aking Module* (SDMM) and the the *Social Interaction Module* (SIM).

4.1 Individual Decision Making

The *Individual Decision Making Module* (IDMM), supports the reasoning about the provided and requested services. The IDMM is supported by the concrete data structures in the *Individual Knowledge Base* (IKB). Decisions are made according to the user's requirements or competencies about the services, the alternative types of services, and the users' preferences and constraints.

Users can provide their requirements to the agents through a GUI to draw influence diagrams, where they can display the structure of the decision problem about the services provided or to be obtained. In addition, the GUI allows the user to communicate user-specific details, in particular preferences and constraints. For instance, Figure 6 gives the influence diagram related to the evaluation of services by an A-type agent, e.g. `Al`. The top, main goal (`provision`) is split into independent abstract sub-goals concerning the cost (`cost`) and the quality of service (`qos`).

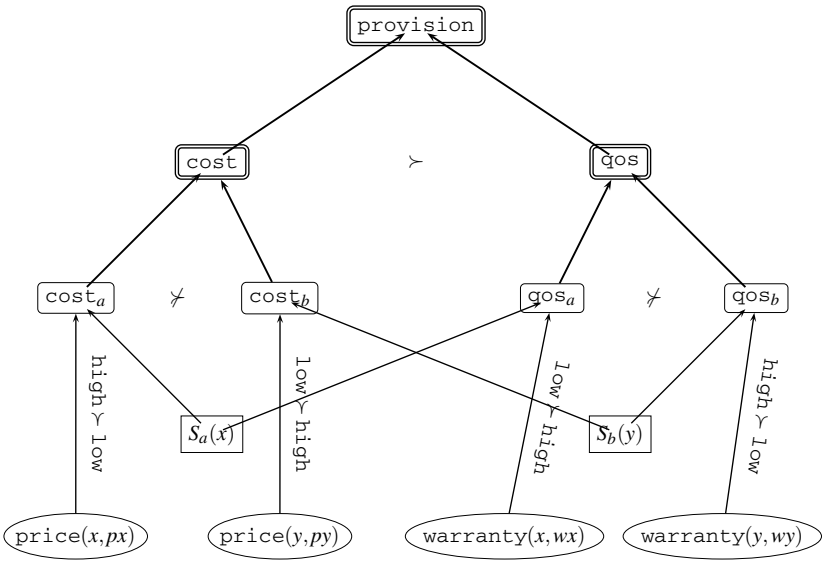

Fig. 6 Influence diagram to structure the decision

These sub-goals are reduced to further concrete sub-goals. For instance, the quality of service depends on the quality of the service S_a ($\texttt{qos}_a$) and on the quality of the service S_b ($\texttt{qos}_b$). While abstract goals just reflect the user's needs, concrete goals provide criteria to evaluate different alternatives.

The main goal, the provision of a composite service S, needs to be addressed by some decisions, e.g. on which concrete S_a and S_b service to adopt (by appropriately instantiating variables x and y [11] in Figure 6). Note that `Al` is a candidate provider of service S_a in our use case, but may be able to provide several instances of this service, namely x may be instantiated in many different ways. Instead, `Al` is not a candidate provider of service S_b, and needs to choose one. These decisions depend on the agent knowledge, namely information about the concrete services provided by `Al` itself and B-type providers (e.g. `price`, `warranty`).

The user provides also his preferences and constraints. For example, the user may specify that `cost` is more important to the user than `qos` as far as reaching `provision` is concerned. *Priorities* are attached to goals, decisions and knowledge in an influence diagram to represent the preferences over goals, the expected utilities of decisions, and the uncertainty of the knowledge.

The structure of the decision problem related to the evaluation of services and the associated priorities are stored within the agent's knowledge bases (in particular the IKB) and reasoned upon by the IDMM. In ArguGRID, the IDMM is

[11] Throughout the chapter we adopt the following convention: variables are in italics and constants are in typescript font.

realised using our argumentation framework for decision making. For instance, the knowledge bases corresponding to the influence diagram of Figure 7 are represented in Table 1. The goal rules are depicted at top, while the decision rules are depicted at the bottom. In this example, there is no epistemic rule. That is the reason why the incorporation of supposition on missing information is essential to perform the individual reasoning of requester agents. A rule above another one has priority over it. To simplify the graphical representation of the rules, they are stratified in non-overlapping subsets, i.e. different levels. The *ex æquo* rules are grouped in the same level. Non-comparable rules are arbitrarily assigned to a level. The goal rules express that achieving `cost` and `qos` is ideally required to reach `provision`, but this can be relaxed: achieving the goal `cost` is enough to reach `provision` ($\mathtt{r}_{012} \succ \mathtt{r}_{01}$). Contrary to the other rules in Table 1, $\mathtt{r}_{01}$ is not in the IKB which reflects the own user requirements represented by `Al` but $\mathtt{r}_{01}$ is the output of the previous interaction between `Al` and the buyer stored in the DCS. $\mathtt{r}_{01}$ reflects the preference of the customer represented by the buyer agent.

$\mathtt{r}_{012} : \mathtt{provision} \leftarrow \mathtt{cost}, \mathtt{qos}$
$\mathtt{r}_{01} : \mathtt{provision} \leftarrow \mathtt{cost}$
$\mathtt{r}_{134} : \mathtt{cost} \leftarrow \mathtt{cost}_a, \mathtt{cost}_b$
$\mathtt{r}_{256} : \mathtt{qos} \leftarrow \mathtt{qos}_a, \mathtt{qos}_b$

$\mathtt{r}_{32}(x) : \mathtt{cost}_a \leftarrow S_a(x), \mathtt{price}(x, \mathtt{high})$
$\mathtt{r}_{41}(y) : \mathtt{cost}_b \leftarrow S_b(y), \mathtt{price}(y, \mathtt{low})$
$\mathtt{r}_{51}(x) : \mathtt{qos}_a \leftarrow S_a(x), \mathtt{warranty}(x, \mathtt{low})$
$\mathtt{r}_{62}(y) : \mathtt{qos}_b \leftarrow S_b(y), \mathtt{warranty}(y, \mathtt{high})$
$\mathtt{r}_{31}(x) : \mathtt{cost}_a \leftarrow S_a(x), \mathtt{price}(x, \mathtt{low})$
$\mathtt{r}_{42}(y) : \mathtt{cost}_b \leftarrow S_b(y), \mathtt{price}(y, \mathtt{high})$
$\mathtt{r}_{52}(x) : \mathtt{qos}_a \leftarrow S_a(x), \mathtt{warranty}(x, \mathtt{high})$
$\mathtt{r}_{61}(y) : \mathtt{qos}_b \leftarrow S_b(y), \mathtt{warranty}(y, \mathtt{low})$

Table 1 The goal rules (at top) and the decision rules (at bottom) representing the users requirements

These concrete data structures (rules and priorities) provide the backbone of arguments. For instance, `Al` can build an admissible argument concluding that the goal related to the cost of the service S_b is reached by choosing $S_b(x)$ if we suppose that the price of service $S_b(x)$ is low.

The IDMM interacts with the other components of the architecture as follows. It interacts with the GUI, through the CM, which uses the GOLEM environment as a mediator to interact with other entities in the system and the user. The IDMM is informed and responds when a service is (or must be) instantiated. The IDMM interacts with the SDMM by asking or by providing the instantiation of the abstract

or partially instantiated service, and by being informed when the provision of a concrete service is (or must be) accomplished. In this way, the IDMM module shifts from the goals and the preferences provided by the user to an abstract representation of atomic services (or composite services, as appropriate). For instance, `Al` can build an admissible argument concluding that the goal related to the provision of the service S is reached if we suppose that the price of the service $S_b(y)$ is low. This is then turned into a concrete representation (choice of y fulfilling the constraint) by the SDMM.

4.2 Social Decision Making

The *Social Decision Making Module* (SDMM) reasons about the concrete instances of services that can be provided/requested. Decisions are made according to user's requirements or competencies, the knowledge about the potential partners and the alternative concrete services, and preferences over them. For this purpose the SDMM is supported by the concrete data structures in the *Social Knowledge Base* (SKB) and in the *Dialogical Commitment Store* (DCS).

The dialogical commitment store is an internal data structure which contains propositional and action suggestions involving the agent, namely with the agent being either the debtor or the creditor. This data structure is shared by the SIM and the SDMM. Concretely, the dialogical commitment store may contain the concrete representation of atomic or composite services and the representation of the partners exchanged during the dialogues, while the SKB contains the concrete representation of atomic or composite services provided by the agent. Moreover, the SKB contains preferences about the services and the partners. The selection (respectively the suggestion) of concrete services is made according to the user's requirements (respectively competencies) about the alternative concrete services, the information about the partners, and preferences over them.

Figure 7 represents the negotiation problem of S_b from an A-type agent's viewpoint, e.g `Al`. The evaluation of the contract (`good_deal`) depends on the provision of the service (`provision`) as considered by the IDMM and depends also on the supplier (`supplier`). The evaluation of the partners depends on their representation (`representation`) and on their performance (`performance`). The supplier's performance is influenced by knowledge about customer testimonials ($\texttt{testimonials}(x,v)$) depending on the number of previous collaborations with these suppliers ($\texttt{previous}(x,n)$) and the average satisfaction in these collaborations ($\texttt{satisfaction}(x,v)$). For simplicity, preferences are not depicted in Figure 7. In the project ArguGRID, the SDMM, like the IDMM, is built upon our argumentation framework for decision making. Statements and priorities are recorded within the agent's SKB, the agent's DCS, and reasoned upon by the SDMM. Alternatively, the user can directly fulfil the SKB and the DCS. The statements corresponding to the influence diagram of Figure 7 are represented in Table 2. The goal rules, the epistemic rules and the decision rules are depicted in the table 2. For instance, the agent whose influence diagram is given in Figure 6 deems the suppliers

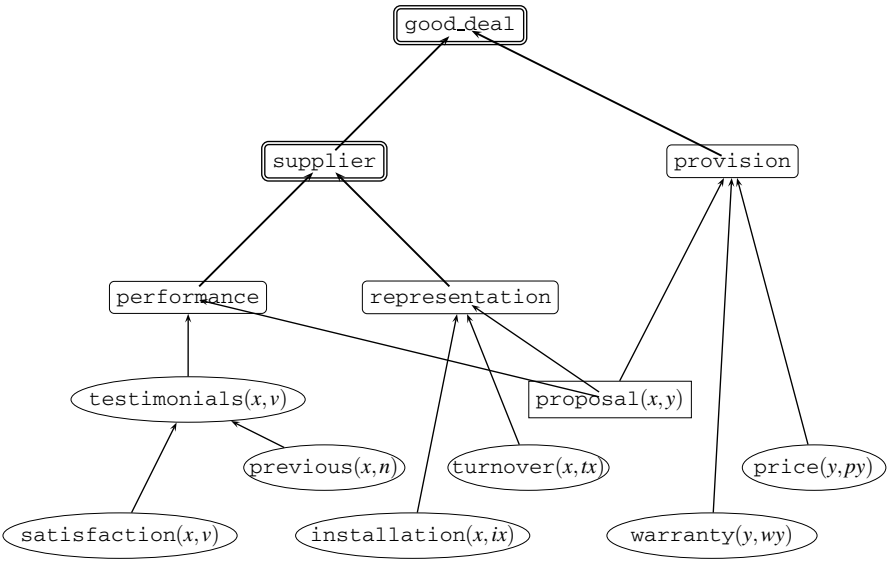

Fig. 7 Influence diagram to structure the negotiation

with an annual turnover greater than two million euros (`turnover`) and at least 50 installations (`installation`). For this purpose, the rules $\mathtt{r}_{31}(x,y)$ and $\mathtt{r}_{32}(x,y)$ are included in the SKB.

The dialogical commitment store of `Al`, which includes the public statements of agents which `Al` is aware of, include the suggestions involving `Al`: either `Al` is the creditor of the suggestion, for instance $\mathtt{commit}(\mathtt{Bob},[\mathtt{good\_deal},S_b(\mathtt{e}),\emptyset])$ is added to the dialogical commitment store when `Bob` suggests it (see the next section); or `Al` is the debtor of the suggestion, for instance $\mathtt{commit}(\mathtt{Al},[\mathtt{good\_deal},S_b(\mathtt{e}),\emptyset])$ is added to the dialogical commitment store when `Al` accepts them (see the next section).

The SDMM interacts with the IDMM by exchanging that abstract service which is (or must be) instantiated and by communicating when a concrete service is (or must be) set up. The SDMM interacts with the SIM (see the next section), by notifying it that the agent needs to play a certain role using a particular protocol, by being informed by the SIM when some offers, some proposals, and some arguments must be evaluated or built and by informing the SIM when the offers, the proposals, the arguments have been evaluated or built.

In this way, the SDMM reasons and takes decision about the proposals and arguments which are exchanged during the dialogues. For instance, `Al` can built an admissible argument concluding that the goal related to the cost of the service S_b is reached since the price of the service $S_b(\mathtt{c})$ is low. This argument is useful for `Al` to justify its choice, $S_b(\mathtt{c})$, in front of `Bob`.

Table 2 The goal theory (at top), the epistemic theory (at middle) and the decision theory (at bottom) corresponding to the social statements

$r_{012} : \texttt{good\_deal} \leftarrow \texttt{supplier}, \texttt{provision}$

$r_{01} : \texttt{good\_deal} \leftarrow \texttt{supplier}$
$r_{134} : \texttt{supplier} \leftarrow \texttt{performance}, \texttt{representation}$
$r_{256} : \texttt{provision} \leftarrow \texttt{cost}_b, \texttt{qos}_b$

$r_{25} : \texttt{provision} \leftarrow \texttt{cost}_b$

$r_{26} : \texttt{provision} \leftarrow \texttt{qos}_b$
$r_{02} : \texttt{good\_deal} \leftarrow \texttt{provision}$

$f_1 : \texttt{testimonials}(\texttt{Bob}, \texttt{high}) \leftarrow$
$f_2 : \texttt{turnover}(\texttt{Bob}, 5) \leftarrow$
$f_3 : \texttt{installation}(\texttt{Bob}, 100) \leftarrow$
$f_4 : \texttt{price}(\texttt{d}, \texttt{high}) \leftarrow$
$f_5 : \texttt{warranty}(\texttt{d}, \texttt{low}) \leftarrow$
$f_6 : \texttt{price}(\texttt{c}, \texttt{low}) \leftarrow$
$f_7 : \texttt{warranty}(\texttt{c}, \texttt{high}) \leftarrow$
$f_8 : \texttt{price}(\texttt{e}, \texttt{low}) \leftarrow$
$f_9 : \texttt{warranty}(\texttt{e}, \texttt{low}) \leftarrow$
$f_{10} : \texttt{price}(\texttt{f}, \texttt{high}) \leftarrow$
$f_{11} : \texttt{warranty}(\texttt{f}, \texttt{high}) \leftarrow$

$r_{21}(x,y) : \texttt{performance} \leftarrow \texttt{proposal}(x,y), \texttt{testimonials}(x, \texttt{high})$
$r_{31}(x,y) : \texttt{representation} \leftarrow \texttt{proposal}(x,y), \texttt{turnover}(x,tx), tx > 2\text{M euros}$
$r_{32}(x,y) : \texttt{representation} \leftarrow \texttt{proposal}(x,y), \texttt{installation}(x,ix), ix > 50$
$r_{51}(x) : \texttt{cost}_b \leftarrow \texttt{proposal}(x,y), \texttt{price}(y, \texttt{low})$
$r_{61}(x) : \texttt{qos}_b \leftarrow \texttt{proposal}(x,y), \texttt{warranty}(y, \texttt{high})$

4.3 Social Interaction

The *Social Interaction Module* (SIM) drives the communications and interactions by the adherence to protocols. These protocol are concrete data structures and are stored in the *Protocol Library* (PL).

Decisions required to conduct the interaction are provided by the SDMM. In practice, the SDMM, once it has reasoned about the services requirements and from whom these services can be requested or to whom these services can be proposed, uses a boot strap mechanism that initiates the required protocol, the role the agent will play in that protocol, and the other participants. From this, the protocol engine in the SIM determines the appropriate message to be sent given those parameters. In cases where there is no ambiguity with respect to the message to be sent, the SIM will automatically send the message to the outgoing message queue from which the agent body will handle the transportation of the message to the recipients' incoming message queues. Where there is a decision to be made either between the choice of two locutions (e.g. whether to accept or reject an offer) to be sent or the instantiation of the content of the locution (e.g. the definition of a proposal), the SIM uses a

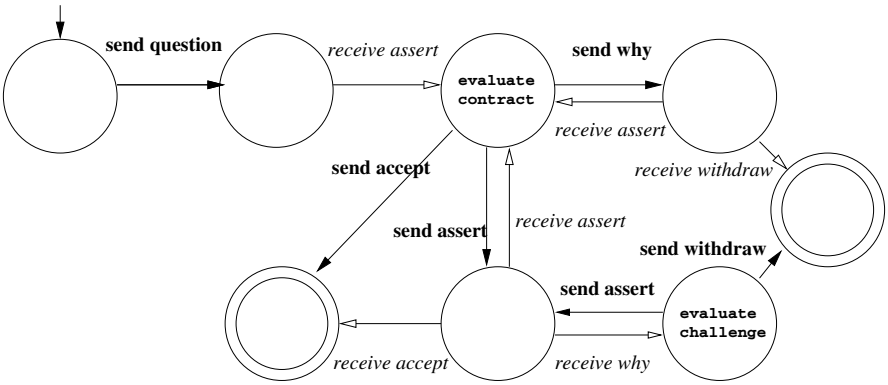

Fig. 8 Negotiation protocol for the requester

precondition mechanism to prompt the SDMM for a solution. Concretely, MARGO interfaces with LCC through the condition mechanism of utterances for a move. Upon the satisfaction of the precondition, the SIM sends the locution to the outgoing message queue. If it is necessary to update the dialogical commitment store of the agent, this can be done with the post condition mechanism which operates in a similar manner.

The agents utter messages to exchange goals, decisions, and knowledge. The syntax of messages is in conformance with a common communication language. We assume that each message:

- has an identifier, M_k;
- is uttered by a speaker (S_k);
- is addressed to a hearer (H_k);
- eventually responds to a message with identifier R_k
- is characterised by a speech act A_k composed of a locution and a content.

In our scenarios, the locution is one of the following: `question`, `assert`, `accept`, `why`, `withdraw` (see Table 3 below for examples). The content is a triple consisting of: a goal G_k, a decision D_k, and knowledge K_k. We will use θ to denote that no goal is given and $\emptyset$ to denote that no knowledge is provided.

In our scenario our agents use two protocols for two types of dialogue: information-seeking and negotiation (see Table 2). Figure 8 depicted our negotiation protocol from the requester viewpoint with the help of a deterministic finite-state automaton. The SDMM interfaces with the SIM's protocol through the condition mechanism to further elucidate the semantics of the protocol being used. For example, at one point in the dialogue the requester is able to send `accept`, `assert` and `why`. The choice of which locution to send is dependant on the SDMM being able to satisfy its precondition.

In the ArguGRID project, the SIM uses LCC to drive all social interactions. Figure 9 represents the set of clauses for the agents in our argumentation-based negotiation protocol. The role being used are:

- the requestor of the service (requestor(g_0, c, K)), g_0 being the main goal to reach, c being the contract, and K being the knowledge. c and K must be instantiated;
- the provider of the service (provider(g_0, c, K)), g_0 being the main goal to reach, c being the contract, and K being the knowledge. c and K must be instantiated;
- the evaluator of the proposal (evaluator(g_0, g_1, c_1, K_1)), g_0 (respectively g_1) being the main (respectively current) goal which is discussed, c_1 being the contract, and K_1 being the knowledge. c_1 and K_1 are related to the proposal;
- the sender of the proposal (proponent(g_0, g_1, c_1, K_1)), g_0 (respectively g_1) being the main (respectively current) goal which is discussed, c_1 being the contract, and K_1 being the knowledge. c_1 and K_1 are related to the proposal.

According to the first clause a(requestor(g_0, c, K), ag_1), ag_1 sends a question to the provider which submits a proposal. The clause for the provider role is the complement of the requestor's one. The second clause a(evaluator(g_0, g_1, c_1, K_1), ag_1) handles the next stage of the argumentation-based negotiation protocol, i.e. the acceptance, the counter-proposal, or the challenge of the proposal. A proposal is accepted, and so recorded in the dialogical commitment store[12], if it is supported by an admissible argument of the SDMM. Indeed, the condition *evaluate_contract*(g_0, c_1, K_1) is satisfied if there is an admissible argument of the SDMM such that the knowledge K_1 and the contract c_1 are in the sentences of the argument and g_0 is the conclusion of the argument. If it is not the case and another proposal, which was not yet suggested, is supported by an admissible argument of the SDMM, then the counter-proposal is asserted, and so recorded in the dialogical commitment store. Otherwise the motivation of the previous proposal is challenged. The clause for the proponent role is the complement of the evaluator's one. An argument which is challenged must be supported by a subargument. Indeed, the condition *evaluate_challenge*(g_1, g_2, c_1, K_1, K_2) is satisfied if: i) there is an admissible argument of the SDMM such that the goal g_2, the knowledge K_1, and the contract c_1 are in the sentences of the argument and g_1 is the conclusion of the argument; ii) there is an admissible argument of the SDMM such that the knowledge K_2 and the contract c_1 are in the sentences of the argument concluding g_2.

The top of the table 3 shows the speech acts exchanged between Al and Bob playing an information-seeking dialogue. This dialogue occurs at the step 4 of the e-procurement process involving Al and Bob (cf Figure 2). Following the protocol for this type of dialogue, the first move is for Al to pose a question to Bob, M_0. This locution seeks the price range for the available services S_b. Bob informs Al with several locutions providing the various services available and their price ranges (M_1, ..., M_4). A similar information-seeking dialogue is played between Bob and Al to inform the latter about the warranty ranges of the available services S_b. According

[12] Actually, the acceptance of a proposal creates an extra-dialogical commitments, i.e. one agent is obligated to provide a service and another is obligated to pay for it. Whether we ignore the task of enforcement, we still make enforcement possible.

a(requestor(g_0,*c*,*K*),ag_1)::=
question(g_0, c, K) $\Rightarrow$ *a*(*provider*(g_0, c, K),ag_2) **then**
commit(ag_2,[g_0,c_1,K_1]) $\leftarrow$ (*assert*(g_0, c_1, K_1) $\Leftarrow$ *a*(*provider*(g_0, c, K),ag_2)) **then**
a(*evaluator*(g_0, g_0, c_1, K_1),ag_2).

a(evaluator(g_0,g_1,c_1,K_1),ag_1)::=
(*accept*(g_0, c_1, K_1) $\Rightarrow$ *a*(*proponent*(g_0, g_1, c_1, K_1),ag_2)) $\leftarrow$
(*evaluate_contract*(g_0, c_1, K_1)) **and** commit(ag_1,[g_0,c_1,K_1])
or
(*assert*(g_0, c_2, K_2) $\Rightarrow$ *a*(*proponent*(g_0, g_1, c_1, K_1),ag_2)) $\leftarrow$
(*evaluate_contract*(g_0, c_2, K_1) **and not**(commit(ag_1,[g_0,c_2,K_2]))) **and**
commit(ag_1,[g_0,c_2,K_2]) **then**
a(*proponent*(g_0, g_1, c_2, K_2),ag_1))
or
(*why*(g_1, c_1, K_1) $\Rightarrow$ *a*(*proponent*(g_0, g_1, c_1, K_1),ag_2) **then**
commit(ag_2,[g_2,c_1,K_2]) $\leftarrow$ (*assert*(g_2, c_1, K_2) $\Leftarrow$ *a*(*proponent*(g_0, g_1, c_1, K_1),ag_2))
and *a*(*evaluator*(g_0, g_2, c_1, K_2),ag_1)) **or**
withdraw(g_1, c_1, K_1) $\Leftarrow$ *a*(*proponent*(g_0, g_1, c_1, K_1),ag_2)).

a(provider(g_0,*c*,*K*),ag_2)::=
question(g_0, c, K) $\Leftarrow$ *a*(*requestor*(g_0, c, K),ag_1) **then**
(*assert*(g_0, c_1, K_1) $\Rightarrow$ *a*(*requestor*(g_0, c, K),ag_1)) $\leftarrow$
(*evaluate_contract*(g_0, c_1, K_1)) **and**
commit(ag_2,[g_0,c_1,K_1]) **then**
a(*proponent*(g_0, g_0, c_1, K_1),ag_2).

a(proponent(g_0,g_1,c_1,K_1),ag_2)::=
commit(ag_1,[g_0,c_1,K_1]) $\leftarrow$ (*accept*(g_0, c_1, K_1) $\Leftarrow$ *a*(*evaluator*(g_0, g_1, c_1, K_1),ag_1))
or
commit(ag_1,[g_0,c_2,K_2]) $\leftarrow$ (*assert*(g_0, c_2, K_2) $\Leftarrow$ *a*(*evaluator*(g_0, g_1, c_1, K_1),ag_1))
then
a(*evaluator*(g_0, g_1, c_2, K_2),ag_2)
or
(*why*(g_1, c_1, K_1) $\Leftarrow$ *a*(*evaluator*(g_0, g_1, c_1, K_1),ag_1) **then**
(*assert*(g_2, c_1, K_2) $\Rightarrow$ *a*(*evaluator*(g_0, g_1, c_1, K_1),ag_1) $\leftarrow$
(*evaluate_challenge*(g_1, g_2, c_1, K_1, K_2)) **and**
commit(ag_2,[g_2,c_1,K_2]) **then**
a(*proponent*(g_0, g_2, c_1, K_2),ag_2)) **or**
withdraw(g_1, c_1, K_1) $\Rightarrow$ *a*(*evaluator*(g_0, g_1, c_1, K_1),ag_1)).

Fig. 9 Representation of the argumentation-based negotiation protocol

Table 3 Information seeking dialogue (top and middle) and negotiation dialogue (bottom)

M_k	S_k	H_k	A_k	R_k
M_0	Al	Bob	question(θ,$S_b(x)$,[price$_b$(x,px)])	θ
M_1	Bob	Al	assert(θ,S_b(c),[price$_b$(c,pc),low $\leq pc \leq$ medium])	M_0
M_2	Bob	Al	assert(θ,S_b(e),[price$_b$(e,pe),low $\leq pe \leq$ medium])	M_0
M_3	Bob	Al	assert(θ,S_b(d),[price$_b$(d,pd),medium $\leq pd \leq$ high])	M_0
M_4	Bob	Al	assert(θ,S_b(f),[price$_b$(f,pf),medium $\leq pf \leq$ high])	M_0

M_k	S_k	H_k	A_k	R_k
M_0	Al	Alice	question(performance,proposal(Bob,y),[testimonials(Bob,z)])	θ
M_1	Alice	Al	assert(performance,proposal(Bob,y),[testimonials(Bob,high)])	M_0
M_2	Al	Alice	why(θ,proposal(Bob,y),[testimonials(Bob,high)])	M_1
M_3	Alice	Al	assert(θ,proposal(Bob,y),[previous(Bob,10),satisfaction(Bob,high)])	M_2

M_k	S_k	H_k	A_k	R_k
M_0	Al	Bob	question(good_deal,$\langle cid$,Bob,Al,$S_b(x)$,[price(x,px),warranty(x,wx)]$\rangle$,$\emptyset$)	θ
M_1	Bob	Al	assert(good_deal,$\langle$c1,Bob,Al,$S_b(d)$,[price(d,high),warranty(d,low)]$\rangle$,$\emptyset$)	M_0
M_2	Al	Bob	assert(good_deal,$\langle$c2,Bob,Al,S_b(c),[price(c,low),warranty(c,high)]$\rangle$,$\emptyset$)	M_1
M_3	Bob	Al	why(good_deal,$\langle$c2,Bob,Al,S_b(c),[price(c,low),warranty(c,high)]$\rangle$,$\emptyset$)	M_2
M_4	Al	Bob	assert(cost$_{\text{Al}}$,$\langle$c2,Bob,Al,S_b(c),[price(c,low),warranty(c,high)]$\rangle$,$\emptyset$)	M_3
M_5	Bob	Al	assert(good_deal,$\langle$c3,Bob,Al,S_b(e),[price(e,low),warranty(e,low)]$\rangle$,$\emptyset$)	M_1
M_6	Al	Bob	accept(good_deal,$\langle$c3,Bob,Al,S_b(e),[price(e,low),warranty(e,low)]$\rangle$,$\emptyset$)	M_5

to this knowledge the A-type agent Al is able to consider which of these services could satisfy its goals.

The middle of Table 3 depicts the speech acts exchanged between Al and Alice playing an information-seeking dialogue. This dialogue corresponds to step 2 of the e-procurement process (cf Table 2). The first move is for Al to pose a question to Alice, M_0. This locution seeks the testimony of Alice about Bob. This testimony is high (cf M_1) and argued by the number of previous experiences and their values.

The bottom of Table 3 depicts the speech acts exchanged between Al and Bob playing a negotiation dialogue. This dialogue corresponds to the step 6 of the e-procurement process (cf Figure 2). They attempt to come to an agreement on the contract for the provision of a service S_b to reach the common goal good_deal. A contract is a tuple $\langle$cid, debtor, creditor, service, terms$\rangle$ where cid is the contract identifier, debtor is the agent providing the service, creditor is the agent requesting the service. A contract is concerned by the provision of a service provided that the list of terms & conditions (denoted terms) are satisfied. With the message M_1, Bob informs Al that it finds out that the terms & conditions of the contract for the provision of the service $S_b(d)$ are justified with respect to the common goal (good_deal). However, Al does not find S_b(d) justified and he proposes S_b(c). Since none of these proposals have been jointly accepted, they should not be considered in the following of the negotiation. Bob attempts to determine the reasons for Al's choice (cf M_3) which is the cost (rather than the quality of service). Given Al's response in M_4, Bob includes the goal provided by Al. Therefore, it finds between the other solutions (S_b(e) and S_b(f)) the one preferred by Al (S_b(e)) and suggest it (M_5). Finally, Al communicates his agreement with the help of an accept (M_6) which closes the dialogue.

5 Case Run

We consider here a case run which illustrates the agent deliberation and communication in the scenario of section 2. This case run involves `Al`, an A-type agent, and `Bob`, a B-type agent, negotiating the provision of a service S_b.

The terms & conditions considered for the evaluation of the contract about S_b during the negotiation (cf bottom of Table 3) are represented at the two axis of the two dimension plot in Figure 10. The acceptability space of the two participants is represented by shaded areas and depends on the price (y-axis) and the warranty (x-axis). Four points reflect the combinations of values: $S_b(\mathtt{c})$ where warranty is high and price is low, $S_b(\mathtt{d})$ where warranty is low and price is high, $S_b(\mathtt{e})$ where both warranty and price are low and $S_b(\mathtt{f})$ where both are high. After the message M_2 (cf left of Figure 3), `Bob` only finds $S_b(\mathtt{d})$ justified and `Al` only finds $S_b(\mathtt{c})$ justified. After the message M_3 (cf center of Figure 10), the acceptability spaces of the two agents have shifted since neither $S_b(\mathtt{c})$ nor $S_b(\mathtt{d})$ have been jointly accepted. Both of the agents make concession since the MARGO mechanism allow to relax the preferences. After the message M_6 (cf right of Figure 10), both agents has identified $S_b(\mathtt{e})$ as a common solution. We can notice that the influence of `Al` on `Bob` avoid to explore the alternative $S_b(\mathtt{f})$ which is not justified from `Al`'s viewpoint. The influence of `Al` on `Bob` is supported by the extra information carry out by the argument.

We have illustrated here the main strength of argumentation-based negotiation which is, as pointed out by [53], that it allows agents to influence each other. Moreover, our agents make concessions when necessary as suggested by [2].

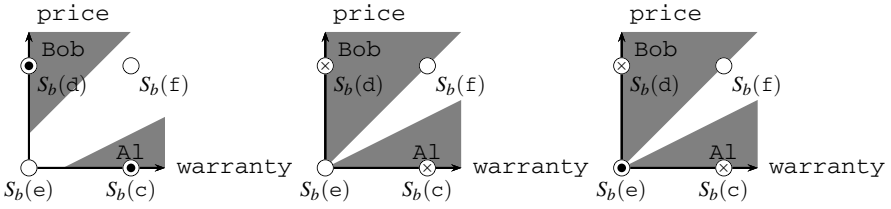

Fig. 10 Acceptability space of participants after the messages M_2 (left), M_3 (center) and M_6 (right)

Figure 11 represents the UML sequence diagram associated with the A-agent `Al`. For brevity, we only describe the internal mechanisms for the negotiations (step 6) of the e-procurement processes involving `Al` and `Bob`. The first transition shown is from the GUI to the IDMM. This is how the user *delegate*s the task of contracting the provision of a service to the agent. Using the techniques described in section 4.1, the user's competency is computed and sent to the SDMM. The SDMM bootstraps the SIM's protocol execution by invoking the protocol, agents it would like to communicate with and the role it will play within the protocol (i.e. the requester). The SIM then drives the interaction by sending the appropriate locution to the CM which

is relayed to the addressed B-type agent. During this exchange, decision must be made, the SIM delegates control to the SDMM to determine the correct course of action. This is shown in the figure 11 as "evaluate". Once the evaluation of the *contract* or *challenge* is performed by the SDMM, it provides the response which the SIM which encapsulates with the correct locution according to the interaction protocol. Finally, through the IDMM and the GUI, the user is notified of the provision of the service. The provider of the service, according to the semantics of the protocol, knows the contract and the provision of the service is accepted by the agent communicating locution, `accept`.

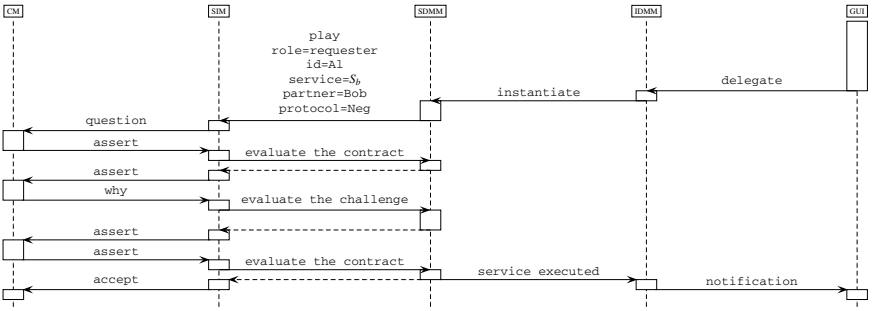

Fig. 11 Sequence diagram of `Al`

6 Deployment and Implementation

Agents in ArguGRID are deployed in GOLEM, a prototype agent platform which can support complex applications of cognitive agents interacting within a distributed environment [10]. GOLEM agents are represented by mean of an agent mind, programmed in Prolog, connected to a Java agent body. The agent body incorporates sensors, to perceive the events happening in the agent environment, and effectors, to produce events in the agent environment. The connection between the Java body and the agent mind is done by using a Prolog-Java bridge, such as InterProlog. In particular in this Chapter we made use of a SWI Prolog agent mind, loading both the MARGO and LCC Prolog libraries, connected to a GOLEM agent body by means of InterProlog. In this section, we also describe how GOLEM has been extended to take into account the specific context of the services.

GOLEM has been integrated with the P2P platform PLATON [36] allowing agents to discover service provider agents. These agents may be deployed in different *GOLEM containers* distributed over a network (cf Figure 12). In order to support agent/service discovery, each container includes two types of registries.

KDE semantic registry: a database which holds semantic descriptions of available services. Such a registry is in charge to perform semantic matchmaking between service queries and concrete available services.

GOLEM registry: a database working as a cache on top of the PLATON P2P platform. Its function is holding information about agents discovered in the GOLEM distributed environment.

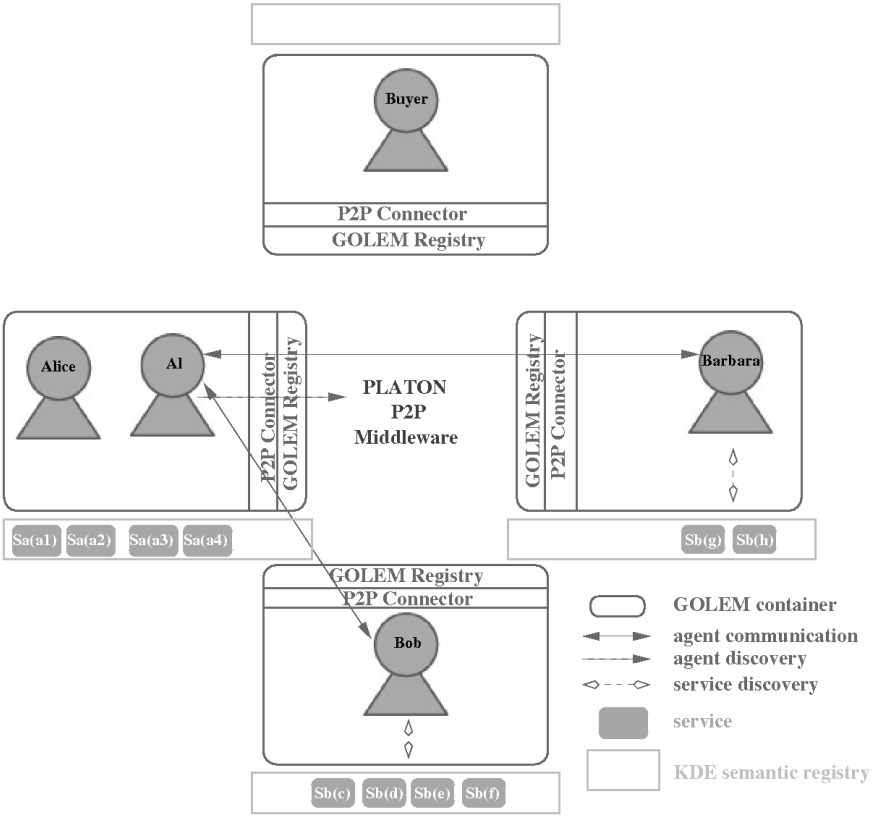

Fig. 12 System Deployment

The *KDE semantic registry* works as a catalogue for the provider agents, holding information about the services provided by the organisation (i.e. user) represented by the agent. In order to support the matching of service descriptions and concrete services, the automated search, the services inside the KDE semantic registry have to carry sufficient information that describes them. Within the agent framework, we abstract away from the particular formalism utilised in KDE to describe the Web services, performing a translation to Prolog terms at the interface between KDE and the GOLEM agent environment. As specified in [10, 11] GOLEM agents can reason about C-logic structures [12]. C-logic is a convenient specification language that has a direct translation to first-order logic and Prolog (see [10, 11] for more details about

the use of C-logic in GOLEM) and that is used in GOLEM to deal with complex structures represented as logical objects. In the context of ArguGRID, the following C-logic structure:

web_service:w1[
service_type ⇒ connection,
domain ⇒ eprocurement,
organisation ⇒ argugrid,
precondition ⇒ creditcard:CCard,
postcondition ⇒ connection:Con[price ⇒ P, warranty ⇒ W]
constraints ⇒ {W= 24*h*, P < 10€}]

represents an instance w1 of class web_service that expresses the requirements for a Web service of type connection. Such a Web service has to be in the eprocurement domain, provided by the argugrid organisation, taking in input objects of class creditcard and providing as output an object of class connection, with a price P and a warranty W that have associated a set of constraints for which the price has to be less than 10 euros per month and a warranty of 24 hours of assistance.

Agents deployed inside a GOLEM container have access to a *P2P connector* interface. This interface wraps PLATON inside GOLEM and allows agents to perform queries to discover other agents and services. The result of such a query is stored inside GOLEM registries for future use. Service provider agents are discovered according to their *affordances* [10], properties that describe the role and competencies of the agent. In our current deployment of ArguGRID we use three properties to describe agents:

- service types, the types of services an agent can represent (e.g. {internet_connection, computer_system});
- organisation, the organisation in which the agent belongs to (e.g. argugrid);
- domain, the domain of knowledge the agent is competent about (e.g. e-procurement).

The above three fields are specified inside the query proposed by the requester, under the parameters of the web_service instances, as previously presented. At the lower level these properties are used by PLATON to create a K-D tree index of the agents; this index ensures that the steps to access a point (an agent) is logarithmic to the number of peers (GOLEM containers) deployed in the distributed network. The details of the algorithm used to describe this process are beyond the scope of this work. The interested reader is referred to [35] for more details.

At the higher level an agent can connect to other agents belonging to different containers by performing physical actions on a connector object (see [10]). Conceptually, the effects of such an action is similar to that of a human manipulating a physical object in a real environment. However, here the effects of the action on the connector make present agents of other containers using PLATON to perform the discovery. When a requester looks for potential suppliers (cf step 1 of the procurement process described in Figure 2 of Section 2), the requester queries the registry

Table 4 Interaction with the Connector Interface

M_k	S_k	H_k	A_k	R_k
M_0	`Al`	`Registry`	`query(supplier,proposal(`x,y`),[`$S_b(y)$`])`	θ
M_1	`Registry`	`Al`	`answer(supplier,proposal(Barbara,`y`),[`$S_b(y)$`])`	M_0
M_2	`Registry`	`Al`	`answer(supplier,proposal(Bob,`y`),[`$S_b(y)$`])`	M_0

and the corresponding connector is involved. Table 4 shows the physical actions exchanged between `Al` and the registry. The first move is for `Al` to pose a query to the registry, M_0. This physical action seeks the suppliers for the available services S_b. The registry reacts to `Al`'s query with several physical actions with inside the description of the various suppliers (M_1 and M_2) in terms of their affordances. According to the new knowledge, the dialogical commitment store of `Al` is updated and this latter is able to perform the next step of the procurement process which consists of collecting information on the available suppliers in order to short-list them. Figure 12 exemplifies what stated above. The A-type agent `Al` queries the connector to find the B-type agents. As a result of these interactions `Al` is notified of the new agents discovered, `Bob` and `Barbara`. Once these agents are discovered, `Al` queries them in such a way that they can check their private KDE semantic registry with an interaction similar to the one presented above with the connector. After the KDE semantic registry provides the services that can match the requester requirements, the agents start the negotiation.

7 Related Work

Combining service-oriented computing and architectures with software agents is an active area of research for intelligent systems [15, 51]. More specifically, current visions of Web-services and the role of agents [14] predict important implications in the engineering of complex distributed systems [27] in general and Grid [22] and ubiquitous [28] computing in particular. A large part of this effort focuses on the service selection problem, where a computational logic approach is playing an important role, for example see McIlraith and colleagues [43, 42], Baldoni et al [5], and Lomuscio et al [34]. In contrast with these efforts, we advocate the automatic discovery of services by agents and how the selection of these services can be made concrete within the context of an argumentation architecture that agents can utilise given abstract specifications of users' goals. The advantage of using argumentation is that agents can provide supporting arguments to select a service, thus being in a position to provide reasons for why a particular service has to be selected instead of another.

The importance of service selection has been studied by Sreenath and Singh in [56]. They explain how services differ from products in terms of how they are being discovered, delivered and evaluated. Sreenath and Singh also provide a general framework for service selection that combines conventional approaches such as

reputation systems, collaborative filtering, and P2P systems with novel techniques from lattice theory.

Maximilien and Singh in [38, 39] argue that current techniques provide no support to actually make rational selections of services, which are key to accomplishing autonomic behaviour in service-oriented computing. They develop a multi-agent system framework based on an ontology for QoS and a new model of trust. The ontology provides a basis for providers to advertise the offered services, for consumers to express their preferences, and for ratings to be gathered and shared. Our work is complementary to the effort of Singh and colleagues in that we provide the logic-based reasoning and social capabilities of an agent to prioritise preferences and select the services that best match the goals of a consumer user. In this context, we also implement these ideas and we can deploy multiple agents that interact and communicate with one another using a platform that can discover these agents over a complex network.

Baldoni and colleagues [6] address the problem of automatic selection and composition of Web services, discussing the advantages that derive from the inclusion, in a Web service declarative description, of the high-level communication protocol, that is used by the service for interacting with its partners, allowing a rational inspection of it. The approach they propose is set in the context of semantic Web by capitalising on existing research in multi-agent systems. Similarly to our work, Web services are viewed as (represented by) software agents, communicating by predefined shareable interaction protocols. A logic programming framework based on modal logic is proposed, where the protocol-based interactions of Web services are formalised and the use of reasoning about actions and change techniques for performing the tasks of selection and composition of Web services in a way that is personalised to a user's request. Like our work, by applying reasoning techniques on a declarative specification of the service interactions allows to gain flexibility in fulfilling the user preference in the context of a Web service matchmaking process. However, this work focuses on discovery and service selection but not on service composition, that we plan to consider in future works.

Bentahar et al. [8] use argumentation implemented via software agents to reason about Web services and improve their performance through the notion of *communities*. A community of Web services is a set of services with similar functionality that grouped together to facilitate discovery. Each community is organised in terms of a *master* Web service (a bit like a broker of composite Web-services) that argues with a set of *slave* Web services (a bit like basic Web services that can participate in composite ones). To persuade a Web service to be part of a composite Web service in a community, master and slave Web services use persuasion and negotiation techniques associated with their argumentation abilities. Web services interact flexibly via dialogue games and are implemented as JACK agents [26]. We differ from this approach in that we use a calculus, LCC, to flexibly handle the interactions. We focus on the use of argumentation to build the decision capabilities of an agent that Bentahar et al abstract away from. We do not need communities to index Web service for discovery but we use GOLEM containers combined with a P2P

framework to enable service discovery. Finally, community for us is a social aspect of the system that emerges via communication, as in [37].

Other MAS research takes a strictly societal or communicative view of agency. It models and formalizes the communication and interaction between agents. The paper [41] describes a formal model for the formation of virtual organisations. The formal model, although accommodates the various technologies described in this chapter, it does not rely upon them. This work is in contrast to other approaches. For example, the authors discuss in [47] as part of the CONTRACT project a contract-based approach to enforcing normative behaviour within multi-agent systems. Numerous works that describe agent systems as electronic institutions [17].

The model and architecture of argumentative agents we have presented, contrary to the abstract negotiation framework of [2], have been implemented and tested with real-world use cases. Our proposal is not the first attempt in this direction. For instance, Kakas and Moraitis [30] provide a framework for effective argumentation-based negotiation. With respect to the latter, we have introduced multi-criteria techniques for the decision making related to the evaluation of proposals by the agents. The framework presented here can be seen to some extent as a specialisation of [2], tailored to service selection. For this purpose, we have proposed: i) an argumentation-based mechanism for multi-criteria decision-making integrating assumptions; ii) an general architecture instantiated with argumentation and logic techniques; iii) the deployment of our architecture with a MAS platform.

We have used here the argumentation-based mechanism for multi-criteria decision-making integrating assumptions proposed in [45]. Contrary to [1, 3, 4], our framework incorporates abduction on missing information, as suggested by [29]. This property is required by the IDMM to build arguments upon suppositions. We can deploy our framework for a number of argumentation semantics by relying on [20], whereas [29] is committed to one such semantics. [48] propose a critical survey of some computational models of argumentation over actions. For instance, [3, 4] have considered several principles according to the different types of arguments which are considered are aggregated. However, contrary to our approach, the potential interaction between arguments is not considered. We have considered the example borrowed from [58] and we have adopted, like [7], an abductive approach to practical reasoning/decision-making which is directly modelled within in our framework.

We have proposed a general architecture which distinguishes the internal reasoning, the social reasoning, and the communication. This architecture has been instantiated with argumentation and logic techniques. The KGP model [31, 33] adopts Knowledge, Goals and Plans as the main components of an agent state. There is no gap between the logical specification of KGP agents and their implementations. Indeed, this model uses computational logic frameworks extending logic programming for specification and realisation purposes. However, this model deals only partially with priorities as required by service selection applications, namely only with preferences between goals [32], but not with uncertainty of knowledge and expected

utilities of alternative services. MARGO[13], i.e. the implementation of our argumentation framework, provided here a revised representation of knowledge, goals and decisions without planning abilities as required by our application. MARGO uses the implementation of [20] in the CaSAPI system [23]. We use LCC for representing and enacting protocols to allow the social norms used by the agents to be verifiable, communicable, inspectable and potentially modifiable. The merits of using a first class protocol are described in [44]. An extension of KGP for service composition has been discussed in [60].

As in [54], we have presented the deployment of an architecture with a MAS platform. The Web-service environment in GOLEM[14] builds upon previous work on deploying MAS with PROSOCS [9]. However, GOLEM extends PROSOCS in many ways. Apart from rationalising the PROSOCS mind/body architecture [57], our implementation uses an argumentative mind component rather than the KGP implementation discussed in [9]. It also generalises the interaction of agents with objects and uses containers as locations [49] of resources, including services. In this sense, GOLEM is part of a growing research and development effort to model situated multi-agent systems [63] without abstracting away the notion of the agent environment as some popular platforms do (e.g. JADE [21] or RETSINA [59]).

8 Conclusion and Future Work

In this chapter, we have described an agent model and architecture using argumentation to automate the selection of services and partners. The modular design of the deliberative and communicative processes bring the well known engineering benefits of modularity as well as create a unique model of agents. The three internal modules are dedicated respectively to decision making (for the IDMM), negotiation (for the SDMM), and communication (for the SIM), and are all realised using argumentation-based frameworks. If the IDMM can be considered as the base appetite of the agent reasoning about how to achieve its individualistic goals and the SDMM can be considered as the social reasoner, conscious of goal solving through collaboration, then the SIM can be considered as its social conscience and filters those impulses by following the social norms of the agent society. For our purposes, the social norms are the rules of dialogues encoded as interaction protocols. In order to test this approach, the architecture is instantiated in the ArguGRID project, by means of the argumentative decision making tool MARGO (for the first two modules) and by the LCC tool for enforcing protocol conformance (for the last module).

We use GOLEM [10] for the deployment of our agents. GOLEM is a multi-agent platform which can support complex application through the deployment of cognitive agents situated in a distributed environment over a network. In particular, GOLEM allows a declarative description of resources in the agent environment, so that such a description is understandable by cognitive agents programmed following the patterns of logic programming. In this way, agents can discover each other

[13] http://margo.sourceforge.net
[14] http://golem.cs.rhul.ac.uk

in a distributed network, as well as reasoning about complex structures representing the requirement of the users in terms of Web services. One possible future work direction, is to include reasoning about semantic descriptions of Web services using OWL [16] in our ArguGRID agents, following an approach like the one described in [25]. The main advantages of having agents reasoning about semantic descriptions, would be that we can decouple the description of the Web services from their implementation and that our cognitive model could be reused in different platforms than ArguGRID.

As mentioned previously, contracts create virtual organisations consisting of debtors and creditors. In this chapter, we have focused on the agent model and architecture and we have ignored virtual organisations. A formal model for virtual organisations using agent negotiation to determine its configuration is described in [41]. The decision making abilities of our agents is useful during the operation of virtual organisations for exception handling, which is labelled as reformation. The additional tasks of monitoring and reportage during the operation and dissolution of the virtual organisations produce data about its performance and form the basis for its evaluation. The execution task during the operation phase is the coordination of the delivery of services.

We have learned a number of lessons while developing the ArguGRID prototype system. Firstly, the knowledge engineering process can become complex when dealing with multiple and possibly heterogeneous knowledge representation technologies. In ArguGRID we decided to avoid formalisms based on Description Logic, for instance, as the reasoning that we could perform using this type of technology was limited to subsumption mechanisms; MARGO style argumentation combined with the underlying Prolog engine allowed for a computationally more expressive and flexible approach. Secondly, another problem that we discovered is that P2P engines have several limitations due to the impossibility to handle semantic descriptions of the entities, but only flat multidimensional points. Thirdly, if distribution is an important issue and/or access to the communication protocols is predicated on an agent's role then a system like LCC is more appropriate for dealing with agent communication. Alternatively, if agents are permitted to access the whole protocol then a shared memory approach [61] is equally suitable, can limit the amount of messages exchanged and can mediate the protocol of communication. Finally, the definition of a mediation infrastructure, such as the one provided by GOLEM, allows for a better integration of the components involved in the system, as it allows for a better description and representation of the resources.

Acknowledgements. This work was supported by the Sixth Framework IST programme of the EC, under the 035200 ArguGRID project.

References

1. Amgoud, L.: A general argumentation framework for inference and decision making. In: Fahiem Bacchus, T.J. (ed.) Proc. of the 21st Conference on Uncertainty in Artificial Intelligence (UAI), pp. 26–33. AUAI Press, Edinburgh (2005)

2. Amgoud, L., Dimopoulos, Y., Moraitis, P.: A unified and general framework for argumentation-based negotiation. In: Proc. 6th International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS), Honolulu, Hawaii, pp. 963–970 (2007)
3. Amgoud, L., Prade, H.: Comparing decisions in an argumentation-based setting. In: Proc. of the 11th International Workshop on Non-Monotonic Reasoning (NMR), Session on Argumentation, Dialogue, and Decision Making, Lake District, UK, pp. 426–432 (2006)
4. Amgoud, L., Prade, H.: Explaining qualitative decision under uncertainty by argumentation. In: Proc. of the 21st National Conference on Artificial Intelligence (AAAI), Boston, pp. 16–20 (2006)
5. Baldoni, M., Baroglio, C., Martelli, A., Patti, V.: Reasoning about interaction protocols for web service composition. Electr. Notes Theor. Comput. Sci. 105, 21–36 (2004)
6. Baldoni, M., Baroglio, C., Martelli, A., Patti, V.: Reasoning about interaction protocols for customizing web service selection and composition. J. Log. Algebr. Program. 70(1), 53–73 (2007)
7. Bench-Capon, T., Prakken, H.: Justifying actions by accruing arguments. In: Proc. of the 1st International Conference on Computational Models of Argument, pp. 247–258. IOS Press (2006)
8. Bentahar, J., Maamar, Z., Benslimane, D., Thiran, P.: An argumentation framework for communities of web services. IEEE Intelligent Systems 22(6), 75–83 (2007)
9. Bracciali, A., Demetriou, N., Endriss, U., Kakas, A., Lu, W., Stathis, K.: Crafting the mind of PROSOCS agents. Applied Artificial Intelligence 20(2-4), 105–131 (2006)
10. Bromuri, S., Stathis, K.: Situating cognitive agents in GOLEM. In: Weyns, D., Brueckner, S., Demazeau, Y. (eds.) Proc. of the Engineering Environment-Mediated Multiagent Systems Conference (EEMMAS), pp. 76–93. Katholieke Universiteit Leuven, Leuven (2007)
11. Bromuri, S., Stathis, K.: Distributed Agent Environments in the Ambient Event Calculus. In: DEBS 2009: Proceedings of the Third International Conference on Distributed Event-Based Systems. ACM, New York (2009)
12. Chen, W., Warren, D.S.: C-logic of Complex Objects. In: PODS 1989: Proceedings of the Eighth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, pp. 369–378. ACM Press, New York (1989)
13. Clemen, R.T.: Making Hard Decisions. Duxbury Press (1996)
14. Cohen, M., Stathis, K.: Strategic change stemming from e-commerce: Implications of multi-agent systems in the supply chain. Strategic Change 10, 139–149 (2001)
15. Curcin, V., Ghanem, M., Guo, Y., Stathis, K., Toni, F.: Building next generation Service-Oriented Architectures using Argumentation Agents. In: 3rd International Conference on Grid Services Engineering and Management (GSEM 2006), Germany (2006)
16. Dean, M., Schreiber, G., Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D.L., Patel-Schneider, P.F., Stein, L.A.: OWL web ontology language reference. Tech. rep., W3C (2004), http://www.w3.org/TR/2004/REC-owl-ref-20040210/
17. Sierra, C., Dignum, F. (eds.): AgentLink 2000. LNCS (LNAI), vol. 1991. Springer, Heidelberg (2001)
18. Dung, P.M.: On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. Artif. Intell. 77(2), 321–357 (1995)
19. Dung, P.M., Kowalski, R.A., Toni, F.: Dialectic proof procedures for assumption-based, admissible argumentation. Artificial Intelligence 170(2), 114–159 (2006)
20. Dung, P.M., Mancarella, P., Toni, F.: Computing ideal sceptical argumentation. Artificial Intelligence, Special Issue on Argumentation 171(10-15), 642–674 (2007)

21. Bellifemine, F.L., Giovanni Caire, Greenwood, D.: Developing Multi-Agent Systems with JADE. Wiley (2007)
22. Foster, I.T., Jennings, N.R., Kesselman, C.: Brain meets brawn: Why grid and agents need each other. In: 3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004), pp. 8–15. IEEE Computer Society, New York (2004)
23. Gartner, D., Toni, F.: CaSAPI: a system for credulous and sceptical argumentation. In: Simari, G., Torroni, P. (eds.) Proc. of the Workshop on Argumentation for Nonmonotonic Reasoning (ArgNMR), pp. 80–95 (2007)
24. Ghanem, M., Azam, N., Boniface, M., Ferris, J.: Grid-enabled workflows for industrial product design. In: Proc. of the 2nd IEEE International Conference on e-Science and Grid Computing (e-Science 2006). IEEE Computer Society (2006)
25. Grosof, B.N., Horrocks, I., Volz, R., Decker, S.: Description logic programs: Combining logic programs with description logic. In: Proc. of the Twelfth International World Wide Web Conference (WWW), pp. 48–57. ACM (2003)
26. Howden, N., Ronnquist, R., Hodgson, A., Lucas, A.: Jack - summary of an agent infrastructure. In: 5th International Conference on Autonomous Agents (2001)
27. Huhns, M.N., Singh, M.P.: Service-oriented computing: Key concepts and principles. IEEE Internet Computing 9(1), 75–81 (2005)
28. Huhns, M.N., Singh, M.P., Burstein, M.H., Decker, K.S., Durfee, E.H., Finin, T.W., Gasser, L., Goradia, H.J., Jennings, N.R., Lakkaraju, K., Nakashima, H., Parunak, H.V.D., Rosenschein, J.S., Ruvinsky, A., Sukthankar, G., Swarup, S., Sycara, K.P., Tambe, M., Wagner, T., Gutierrez, R.L.Z.: Research directions for service-oriented multiagent systems. IEEE Internet Computing 9(6), 65–70 (2005)
29. Kakas, A., Moraitis, P.: Argumentative-based decision-making for autonomous agents. In: Proc. of the 2nd International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS), pp. 883–890. ACM Press (2003)
30. Kakas, A., Moraitis, P.: Adaptive agent negotiation via argumentation. In: Proc. 5th International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS), Hakodate, Japan, pp. 384–391 (2006)
31. Kakas, A.C., Mancarella, P., Sadri, F., Stathis, K., Toni, F.: The KGP model of agency. In: Proc. of ECAI, pp. 33–37 (2004)
32. Kakas, A.C., Mancarella, P., Sadri, F., Stathis, K., Toni, F.: Declarative Agent Control. In: Leite, J., Torroni, P. (eds.) CLIMA 2004. LNCS (LNAI), vol. 3487, pp. 96–110. Springer, Heidelberg (2005)
33. Kakas, A.C., Mancarella, P., Sadri, F., Stathis, K., Toni, F.: Computational logic foundations of kgp agents. J. Artif. Intell. Res. (JAIR) 33, 285–348 (2008)
34. Lomuscio, A., Qu, H., Sergot, M.J., Solanki, M.: Verifying Temporal and Epistemic Properties of Web Service Compositions. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 456–461. Springer, Heidelberg (2007)
35. Lymberopoulos, L., Bromuri, S., Stathis, K., Kafetzoglou, S., Grammatikou, M.: Towards a p2p discovery framework for an argumentative agent technology assisted grid. In: Proc. of the CoreGRID Workshop on Grid Programming Model, Grid and P2P systems Arhcitectures, Grid Systems, Tools, and Environments, Crete, Greece (2007)
36. Lymberopoulos, L., Papavassiliou, S., Maglaris, V.: A novel load balancing mechanism for P2P networking. In: Proc. of ACM Sponsored Conference GridNets, Lyon, France (2007)
37. Mamdani, E., Pitt, J., Stathis, K.: Connected Communities from the standpoint of Multiagent Systems. New Generation Computing 17(4) (1999)
38. Maximilien, E.M., Singh, M.P.: A framework and ontology for dynamic web services selection. IEEE Internet Computing 8(5), 84–93 (2004)

39. Maximilien, E.M., Singh, M.P.: Toward autonomic web services trust and selection. In: Aiello, M., Aoyama, M., Curbera, F., Papazoglou, M.P. (eds.) Service-Oriented Computing - ICSOC 2004, pp. 212–221. ACM, New York (2004)
40. McBurmey, P., Parsons, S.: Games that agents play: A formal framework for dialogues between autonomous agents. Journal of Logic, Language and Information 11(3), 315–334 (2002), Special Issue on Logic and Games
41. McGinnis, J., Stathis, K., Toni, F.: A formal model of agent-oriented virtual organisations and their formation. Multiagent and Grid Systems 7(6), 291–310 (2011)
42. McIlraith, S.A., Son, T.C.: Adapting golog for composition of semantic web services. In: Fensel, D., Giunchiglia, F., McGuinness, D.L., Williams, M.A. (eds.) Proceedings of the Eighth International Conference on Principles and Knowledge Representation and Reasoning (KR 2002), pp. 482–496. Morgan Kaufmann, France (2002)
43. McIlraith, S.A., Son, T.C., Zeng, H.: Semantic web services. IEEE Intelligent Systems 16(2), 46–53 (2001)
44. Miller, T., McBurney, P.: Using Constraints and Process Algebra for Specification of First-Class Agent Interaction Protocols. In: O'Hare, G.M.P., Ricci, A., O'Grady, M.J., Dikenelli, O. (eds.) ESAW 2006. LNCS (LNAI), vol. 4457, pp. 245–264. Springer, Heidelberg (2007)
45. Morge, M.: The Hedgehog and the Fox. In: Rahwan, I., Parsons, S., Reed, C. (eds.) ArgMAS 2007. LNCS (LNAI), vol. 4946, pp. 114–131. Springer, Heidelberg (2008)
46. Morge, M.: Arguing over goals for negotiation: Adopting an assumption-based argumentation decision support system. In: Jao, C. (ed.) Efficient Decision Support Systems - Practice and Challenges From Current to Future, ch. 12, pp. 211–240. InTech (2011)
47. Oren, N., Panagiotidi, S., Vázquez-Salceda, J., Modgil, S., Luck, M., Miles, S.: Towards a formalisation of electronic contracting environments. In: COIN@AAMAS&AAAI, pp. 156–171 (2008)
48. Ouerdane, W., Maudet, N., Tsoukias, A.: Arguing over Actions That Involve Multiple Criteria: A Critical Review. In: Mellouli, K. (ed.) ECSQARU 2007. LNCS (LNAI), vol. 4724, pp. 308–319. Springer, Heidelberg (2007)
49. Overeinder, B.J., Brazier, F.M.T., Marin, O.: Fault tolerance in scalable agent support systems: Integrating darx in the agentscape framework. In: Proc. of the 3rd IEEE International Symposium on Cluster Computing and the Grid (CCGrid), pp. 688–696. IEEE Computer Society, Japan (2003)
50. Papazoglou, M.P.: Service-oriented computing: Concepts, characteristics and directions. In: 4th International Conference on Web Information Systems Engineering (WISE 2003), pp. 3–12. IEEE Computer Society, Italy (2003)
51. Payne, T.R.: Web services from an agent perspective. IEEE Intelligent Systems 23(2), 12–14 (2008)
52. Rahwan, I.: Argumentation in multi-agent systems. Guest Editorial: Autonomous Agents and Multiagent Systems 11(2), 115–125 (2005)
53. Rahwan, I., Ramchurn, S.D., Jennings, N.R., McBurney, P., Parsons, S., Sonenberg, L.: Argumentation-based negotiation. The Knowledge Engineering Review 18(4), 343–375 (2003)
54. Ricci, A., Buda, C., Zaghini, N., Natali, A., Viroli, M., Omicini, A.: simpa-ws: An agent-oriented computing technology for ws-based soa applications. In: Paoli, F.D., Stefano, A.D., Omicini, A., Santoro, C. (eds.) Proceedings of the 7th WOA 2006 Workshop, From Objects to Agents (Dagli Oggetti Agli Agenti), *CEUR Workshop Proceedings*, CEUR-WS.org, Italy (2006)

55. Robertson, D.: Multi-agent Coordination as Distributed Logic Programming. In: Demoen, B., Lifschitz, V. (eds.) ICLP 2004. LNCS, vol. 3132, pp. 416–430. Springer, Heidelberg (2004)
56. Sreenath, R.M., Singh, M.P.: Agent-based service selection. Journal of Web Semantics 1(3), 261–279 (2004)
57. Stathis, K., Kakas, A.C., Lu, W., Demetriou, N., Endriss, U., Bracciali, A.: PROSOCS: a platform for programming software agents in computational logic. In: Müller, J., Petta, P. (eds.) Proceedings of the Fourth International Symposium From Agent Theory to Agent Implementation (AT2AI 2004 – EMCSR 2004 Session M), Vienna, Austria, pp. 523–528 (2004)
58. Stournaras, T. (ed.): eBusiness application scenarios. Deliverable document D1.2 ARGUGRID (2007)
59. Sycara, K., Paolucci, M., Velsen, M.V., Giampapa, J.: The retsina mas infrastructure. Autonomous Agents and Multi-Agent Systems 7(1-2), 29–48 (2003)
60. Toni, F.: Argumentative kgp agents for service composition. In: Proc. AITA 2008, Architectures for Intelligent Theory-Based Agents, AAAI Spring Symposium. Stanford University, USA (2008)
61. Urovi, V., Stathis, K.: Playing with agent coordination patterns in MAGE. In: Coordination, Organization, Institutions and Norms in Agent Systems (COIN@AAMAS 2009), Budapest, Hungary (2009)
62. Vreeswijk, G.: Abstract argumentation systems. Artificial Intelligence 90(1-2), 225–279 (1997)
63. Weyns, D., Omicini, A., Odell, J.: Environment as a first-class abstraction in multi-agent systems. Autonomous Agents and Multi-Agent Systems 14(1), 5–30 (2007)

Chapter 10
Public Administration Workflows Re-engineering: An Agent-Based M&S Approach

Emiliano Casalicchio and Salvatore Tucci

Abstract. Workflows in Public Administration (PA) can be mainly classified as inter-organization processes and cannot be modeled using standard methods such as Petri-nets or WF-Nets, but need new modeling paradigms to describe: i) the structure of the organization; ii) the factors that influence the execution of the workflow and iii) the actors (humans and IT systems) that interact with the workflow, generating the workload. This chapter describes the experience matured by the authors in the design and implementation of an agent-based modeling and simulation framework to support the re-engineering of Public Administration workflows. The project, started in late 2003, faced the challenge of analyze, evaluate the performance and finally re-engineer a Public Administration process of the Presidency of Counseil of Ministers: the IT infrastructure management and the service provisioning process. The project was developed at the Italian Prime Minister Office for Informatics and Telematics headed by the second Author. From the solution initially developed to solve this specific problem we built a general framework to support Public Administration processes re-engineering. This framework, named Wf-Simulator, has been successfully used in real workflow modeling and simulation. The chapter describes the initial project, the Wf-Simulator framework and three real case studies: service provisioning in PA, day hospital surgery admission and blood examination management.

1 Introduction

The *Business Process Re-engineering* (BPR) theory [17, 11], originally formulated thirty years ago, is today one of the main approach to achieve correct process management, to increase business outcome and to increase process quality of service. At

Emiliano Casalicchio · Salvatore Tucci
Department of Civil Engineering and Computer Science Engineering,
University of Rome Tor Vergata, Via del Politecnico 1 00133 Roma Italy
e-mail: emiliano.casalicchio@uniroma2.it, tucci@TorVergata.it

M. Ganzha & L. C. Jain (Eds.): Multiagent Systems & Applications, ISRL 45, pp. 257–278.
springerlink.com

the beginning of 1990, Davenport and Short [33, 13] defined business process (BP) and of BPR. Their vision emphasize the concept that tasks composing a process are performed by users (humans or IT systems) and that the process involves different organizations. From the Davenport's process vision emerges that a process can be seen as a complex systems composed of different entities interacting and performing tasks to achieve a goal. In some case the interaction can be cooperative in nature or in some other cases entities can have a selfish behavior competing in the use of resources to realize their own objective.

The interaction among entities is based on information exchange, where the activities (that could be managerial or operational) consists in objects manipulation (data and/or goods). The set of tasks composing a BP are organized in a Workflow (Wf) managed by a Workflow Management System (WfMS). Therefore, the design and re-engineering of BP deals with the analysis of the underlining workflow.

As every complex system, BP demands for performance optimization and cost reduction. A key stimulus for re-engineering has been the continuous development and deployment of sophisticated information systems and networks, as well as the affirmation of new business models, enabling paperless environment, and the evolution of the society. In late 2003 we were faced with the challenge of analyze, evaluate the performance and finally re-engineer a Public Administration process of the Italian Presidency of Council of Ministers [1]: the IT infrastructure management and service provisioning processes (SABS – Sistema Acquisto Beni e Servizi) used by all the administrative structures. This practical application stimulates the definition of a more general research project, hereafter referred as the *Wf-Simulator* project, committed to study, design and develop a framework for Pubic Administration business process re-engineering. This Chapter reports our experience in re-engineering Public Administration BPs and will discuss the main issues faced in the Wf-Simulator Project.

The SABS process, as many Public Administration processes, can be classified as an Inter-Organization workflow. Differently from other type of workflow (Production, collaborative, administrative), Inter-organization workflow describes not only humans resources, triggers, alarms and messages but also roles and groups of humans resources, authorization levels, granular access control list and task execution time.

Modeling and simulation is a valid technique to approach BPR, in particular in the validation of Wf properties and in the execution of what-if analysis. Workflow can be modeled with formalisms such as Petri Net [15, 19, 32], YAWL [5], Temporal logic [8, 12, 16], Cuncurrent Transaction Logic [10, 14, 27]. As stated by [6] Petri Net are a valid tool to model workflows because: their formal semantic, the fact that they are state-based instead event-based and the wide range of analysis techniques available. The formal semantic allow an unambiguous specification. Being state-based rather then event based means that state are modeled explicitly (event-based framework only model the transition between states explicitly). Finally, there

[1] http://www.governo.it

exists many analysis techniques for verifying qualitative and quantitative properties of modeled workflow.

Petri-Nets can be evaluated through simulation and, in our opinion, are an effective approach to reason about BPR. However, to model inter-organization workflows, differently from other type of workflows (e.g. production, collaborative, administrative Wf), must be considered many other factors:

- who/what enables a transition
- who/what (humans, the environment, technologies) influences the execution and service time of a transition
- the hierarchical organization of the different actors interacting with the workflow, and
- the authorization levels required to perform a specific action or a set of actions.

In order to comply with these modeling requirements is needed a tool that integrates:

- classical Wf modeling approach (e.g. a Petri-Net)
- a model of the structure of the organization
- a model of the factors influencing the execution of the workflow
- a model of the actors (humans and systems) that interact generate the workload of the workflow.

Among all the modeling and simulation solutions we identify Agent-based Modeling and Simulation (ABMS) as the candidate tool to unravel the public administration business process re-engineering problem.

This paper describes the Wf-Simulator framework that integrates an ABMS engine (the Recursive Porus Agent Simulation Toolkit – RePast [28]) and a workflow management system (Bossa[2]). The Bossa WfMS uses the Petri Net formalism to describe the workflow, in the specific: the Wf topology, the actions that must be performed on the resources manipulated by the process, the set of states, the transition from a state to the next one, the alarms, and all the other characteristics of the workflow. On the other side ABMS is used to model the actors (system and humans) taking part in the process and interacting with the workflow.

For each system or human being, modeled by an agent, were considered: the set of resource it can process; the set of actions allowed on the set of resource it can access; the set of rules that define the behavior and the interaction capabilities. For resources we define the processing time of each action allowed and the change of state due to the processing/use of a resource. Each agent is an independent entity interacting with the workflow.

Figure 1 shows the conceptual architecture of the WfSimulator. The inputs of the framework are a description (models and parameters) of: the workflow, the organization, the workload, and the systems/actors involved. The agent-based simulation engine drives the process evolution and the WfMS emulates the execution of the workflow, like in a real environment. The methodology proposed is independent from the modeling and simulation technologies used in the implementation.

The Wf-Simulator we realized has been successfully applied in two fields:

[2] http://www.bigbross.com/bossa

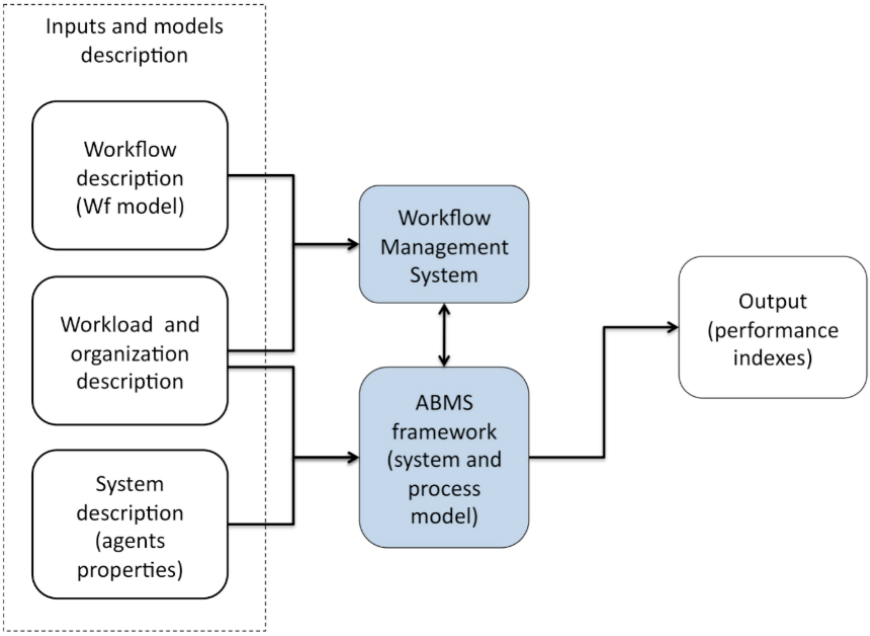

Fig. 1 The conceptual architecture of the Wfsimulator framework

- *Real workflow re-engineering*. Some of the real case we deal width and we reported in this chapter are: the SABS, the health care system for blood analysis management, the day-hospital surgery management.
- *Education*. The Wfsimulator is used in the course of e-Government[3] as an example of inter-organizatin workflow modeling and simulation and it allows students to practice with what-if analysis and re-engineering of real workflow.

This paper describes our experience in the use of agent-based simulation technologies to solve the problem of Business Process Re-engineering. The chapter is organized as in the following. Base concepts of workflow, WfMS and Agent-based modelig and simulation are given in Section 2. Section 3 describes the Wf-Simulator giving an idea of the architecture of the framework. Moreover, here we discuss also how the two technologies, ABMS and WfMS, were integrated and the lessons we learned. Section 3.2 describes the modeling methodology, independent from the case study. Section 4 describes three example of workflows modeled and analyzed using the the Wf-simulator. Finally, section 5 gives perspectives and concluding remarks.

[3] University of Rome Tor Vergata, Faculty of Engineering, Curriculum in Management Engineering.

2 Basic Concepts

In order to better understand the proposed framework, this section briefly described the concepts of Workflow, Workflow Management System, WF-net and Agent Based Modeling and Simulation. The description that follow does not pretend to be neither exhaustive nor formal.

2.1 Workflow and WfMS

The Workflow Management Coalition (WfMC) defines workflow the automation of a business process, in a whole or part, where documents, information or tasks are processed by participants and transferred from one participant to another according to procedural rules [37, 36]. A Workflow Management System defines, creates and manages the execution of workflows through the use of software, running on one or more workflow engines.

As previously mentioned, there are different type of workflows and then of WfMS. *Production* workflows typically manage an high number of similar activities with the goal to maximize the throughput. Production WfMSs minimize the humans intervention, have the capability to manage complex processes and can be integrated with legacy systems. *Collaborative* workflows are based on the workgroup concept and for that reason are often called Groupware. Such WfMSs typically provide a working environment that facilitates collaboration of peoples with common interests or common goals. The main goal is to optimize the collaboration rather than maximize the throughput. *Administrative* workflows, are characterized by the simplicity of the process definition phase. The processes are typically defined using predefined forms. In such systems flexibility is more important then productivity. *Inter-organizational* workflows, of main interest for this work, introduce the concept of roles and access control list. An inter-organizational workflow not only describes humans resources, triggers, alarms and messages but also roles and groups of humans resources, authorization levels, granular access control list and task execution time.

2.2 WF-Net

Workflow nets (WF-nets) [7] are an extension of Petri networks [31], and have been recently introduced to model business processes that are automated by workflow management systems [2] [3]. WF-nets inherit all the properties from Petri-nets and add new features enabling a natural description of workflows.

A WF-net has two special places, a unique source place i and a unique sink place o. The source is such that there are no transitions that share i as output place and the sink is such that there are no transitions that share o as input place. The only exception are strongly connected Petri nets characterized by a transition having o as input place and i as output place.

Another important feature of WF-nets is the concept of *trigger*. A trigger is an external condition which leads to the execution of an enabled task. The execution of a task instance for a specific case starts the moment the task instance is triggered. A task instance can only be triggered if the corresponding case is in a state which enables the execution of the task. We consider four types of tasks: Automatic, User, Message and Time.

An automatic task is triggered the moment it is enabled. This task does not require the intervention of a human being and it is automatically executed by the system when a specific condition is satisfied. Examples of automatic tasks are: to electronically sign or archive a document when it enters in the system; Automatically check the availability in the department back account when a service/goods request is entered in the system.

A user task is triggered by a human participant, i.e., a user selects an enabled task instance to be executed. For example: put a product in a box and send it by ordinary mail to the recipient; check the analysis results and validate the report; remove or insert the blood sample in a clinical machinery, etc.

A Message task is triggered by a specific external event, that is a message arriving in the system. Examples of messages are telephone-calls, fax messages, e-mails or any other example of electronic messages/data exchanged. For example: when arrive a fax/e-mail asking for assistance the process to allocate the resource to assist the customer is activated.

Finally, Time tasks instances are triggered by a clock, i.e., the task is executed at a predefined time. For example a periodical backup of the system database.

The routing capabilities of a Petri net are improved with four operators: AND-split, AND-join, XOR-split and XOR-join. Such constructs allow to model sequential task execution, concurrent task execution, alternate execution of tasks and iterative execution of tasks.

Graphical representation of triggers and routing constructs can be found in many papers, e.g. [4].

Finally, the most important feature of a WF-net is the *soundness* property: a workflow that verify the soundness property is a valid and correct workflow. If the soundness properties is verified, no more token are in the network when the process terminates (this means that there are no pending transitions).

Examples of how to model workflows using WF-nets are given in [21] and [1].

2.3 Agent-Based Modeling and Simulation

Agent-based modeling and simulation (ABMS) is one of the most suitable approaches to model and simulate complex systems and processes, for example: critical infrastructures [**?**, **?**], electricity market and market dynamics in general [29, 38, 35], organization in industry and enterprises [20], emergency management [24], missions control and management [18, 26]. Moreover, ABMS is the more appropriate modeling and simulation approach, to reproduce human interaction [9] and to integrate a workflow model based on WF networks[25, 22].

In this kind of approach, the whole model of a target system or process is obtained considering a population of interacting agents. The key characteristic of an agent is that it exists as an individual entity with location, capabilities, and memory. From the interaction among these agents ”emerge” behaviors that are not predictable by the knowledge of a single agent.

Agent-based modeling is obtained by interconnecting agents that is, independent systems that autonomously elaborate information and resources in order to define their outputs, that become inputs for other agents.

The entity location defines where an agent is located in a physical space (e.g., geographic region) or in an abstract space (e.g., the Internet). This characteristic is not essential to model Inter-organizative workflow but it could help when the environment conditions influence the workflow execution, for example the cost of resources and employees.

What the entity can perform is defined by its capabilities. An agent can modify its internal data representation (perception capability), it can modify its environment (behavior capability), it can adapt itself to environment's changes (intelligent reaction capability), it can share knowledge, information, and common strategies with other entity (cooperation capability), it can execute actions without external intervention (autonomy capability). All this capabilities allow to model humans and systems interacting with the workflow, that is the active entities that manipulate data and goods contributing to the completion workflow case. The capabilities of an agent influences the time to perform an action or task, and could influence the overall throughput of the workflow.

Finally, the experience history (for example, overuse or aging) and data defining the entity state represents the agent's memory. Memory can be used to learn about the past and improve the capability (e.g. the capacity to perform a task/action).

We remark that agents can be defined to be autonomous, problem-solving computational entities which are capable of effective operations in dynamic and open environments. Agents are often deployed in environments in which they interact, and maybe cooperate, with other agents (including both people and software) that have possibly conflicting aims.

3 The Wf-Simulator

In this section we describe the architecture of the Wf-Simulator and we describe the main steps of the Wf modeling process.

3.1 The Architecture

Figure 2 shows the detailed architecture of the inter-organizational workflow simulation framework. The core components are the agent-based simulation framework and the WfMS. As mentioned in the introduction, as agent-based simulation engine we used RePast and as Workflow Management System we used Bossa. The input

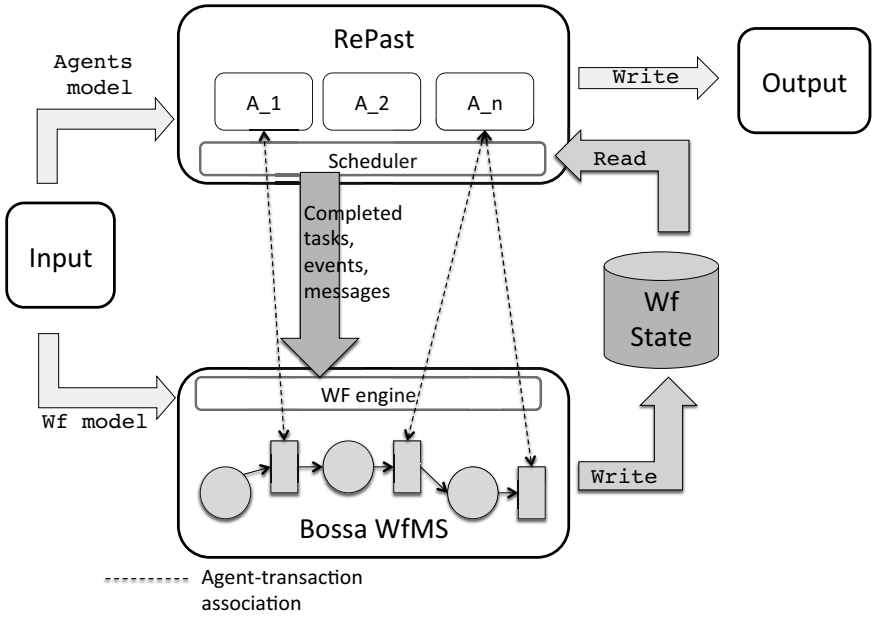

Fig. 2 The detailed framework architecture

model of the framework is a graphical tool, WoPeD (Workflow Petri Net Designer)[4], that produces a Petri Net Markup Language (PNML) description of the Wf model. The output is directly managed by the agent-based simulation framework. Moreover, statistics can be stored in files for post processing.

The WfMS Bossa is used to emulate the execution of the workflow. Bossa uses the Wf-Net formalism to describe and execute the workflow. One of the problem we encountered during the project development phase was that Bossa uses its own language to describe the workflow. Therefore, the first Wf-Simulator release did not allow a graphical description of the workflow but it required to provide directly a textual description processed by Bossa. In the second release of the framework we enhanced Bossa with a plugin to interpret a PNML Wf description. This allowed us to integrate into the framework a graphical Wf designer such as WoPeD. Bossa emulates the execution of the workflow and store the workflow state on the file system.

The Bossa workflow engine offers classes that describe a basic structure of workflow, and that allow to implement and to execute workflows. In the Wf-Simulator we extend the Bossa's resource concept: resources became dynamic agent classes, aware of the environment, with memory capability and with interaction capability, rather then being static data structures.

[4] http://woped.org

RePast, the agent-based simulation framework, allows to model, by means of agents, the actors that interact with the workflow performing actions and manipulating passive resources. Repast is a discrete event simulator that offers: the possibility to specify agents behavior and characteristics; a completely adaptable scheduler that supports both sequential and parallel discrete event operations, and that is responsible for the execution of agents behavior and simulation management task (e.g. updating displays, recording data). The software agents are used to model humans, organized in appropriated hierarchies, and the systems (e.g. software systems) used to manipulate the resources and to produce services. Each agent is associated with the a set of transactions it is in charge to manage. Moreover, the agent behavior can impact directly the execution of transaction associated to user type tasks and indirectly the execution of transactions associated to automatic tasks and message tasks.

The simulation is orchestrated by the RePast simulation engine, through a specific class we designed (the Wf-Simulator class).

The RePast simulation library is extended to invoke the workflow engine library. In the specific Bossa, to manage persistent transactions in the workflow evolution stores all the results on the file system. RePast needs runtime data to simulate the system behavior and to display results at runtime. The workflow engine library is accessed by RePast during the simulation (inside the `inizialize`, `step`, `run` and `pause` RePast methods). Every tick the RePast scheduler get the workflow status and schedules the agents actions. Also external events generated by the resources or by the environment are caught and processed. Summarizing the steps are the following:

1. RePast reads the state of the workflow stored on the file system by Bossa
2. RePast assigns new activities to agents (humans and/or systems) and schedules task execution. The task that must be executed and the resources that must be used are therefore determined by the workflow engine
3. Each agent simulates the execution of the assigned task and when completed, the associated transition is triggered back to Bossa.
4. Bossa fires the transitions enabled by the completion of tasks executed in step 3. Therefore, Bossa determines the new state of the workflow and write it on the filesystem. Now the process goes back to step one

The integration of an agent-based simulation engine and of workflow engine results in a system capable:

- to execute an inter-organizational workflow
- to simulate the behavior of resources involved in the workflow (human beings, computer systems, etc)
- to simulate the environment that influence and that is influenced by the workflow
- to display statistics and the workflow state at realtime.

Finally, RePast is also in charge to manage the output. The agents model can be instrumented with sensors collecting performance indexes such as: number of completed task and pending task (for a single task or for a class of tasks); execution and

queue time for a task (or a class of tasks); number of occurrence for a specific event (e.g. the arrival of a message); number of task assigned to a specific agent or class of agents; number of tasks completed by a specific agent or class of agents; task completion time for a specific user or class of users or a single user. The time series for the collected indexes can be displayed online using the GUI library of Repast. On the contrary, aggregated values (e.g. average, variance, X-percentile) obtained post processing the collected time series must be elaborate off line and plotted using specific tools.

For example, figure 3 shows the GUI of the Wf-Simulator framework and the online plot of some performance indexes. The set of performance indexes is customizable depending on the case study. In the Wf-Simulator GUI there are: a control panel to manage the simulation allowing the load, start, stop and pause operations; a panel to input the parameters (loaded from a file by default); configurable statistic graphs updated at real time and there is a text output displaying the workflow status.

The agents model design process we used were based on UML. A consistent number of proposals deal with using UML to model agents and multi-agent software systems [30] [23] [34].

3.2 The Modeling Approach

In the Wf-Simulator, workflow modeling consist of two main steps: description of the workflow and workflow model parameterization.

The first step of the business process modeling is related to: describe the workflow model using the WF networks formalism; definition of the model for human resources, active resources and passive resource using the agent-based modeling approach. Defined the workflow model, through the verification of properties such as the soundness and the validity (see [7]) it is possible to highlight macro problems such as deadlocks or useless states and transactions. Concerning the resources, humans being and active resources are modeled by agents while passive resources are modelled by passive data structures. For the modeling of human resources can be chosen different formalism and levels of detail. A human resource model depend on the specific case should be modeled, for example: normal behavior defined by process rules and inter-organizational workflow structure; adaptation capability under stress conditions; adaptation capabilities in absence of a predefined set of rules. Neural networks and fuzzy logic are example of human behavior modeling formalism. Active resources, e.g. computer system, computer networks, mechanical and electronic devices, are modeled as queue networks (e.g. a simple queue or a complex network). However, these modeling aspects are out of the scope of this paper.

The collection of simulation model parameters is part of the second step and it is related to: the workflow model, the human resource model, the active and passive resource models. There are no written rules that establish the parameters to be gathered. Indeed, them depends on the target problem and on the level of detail of the model. Input parameters can be obtained measuring the existing system or from historical data.

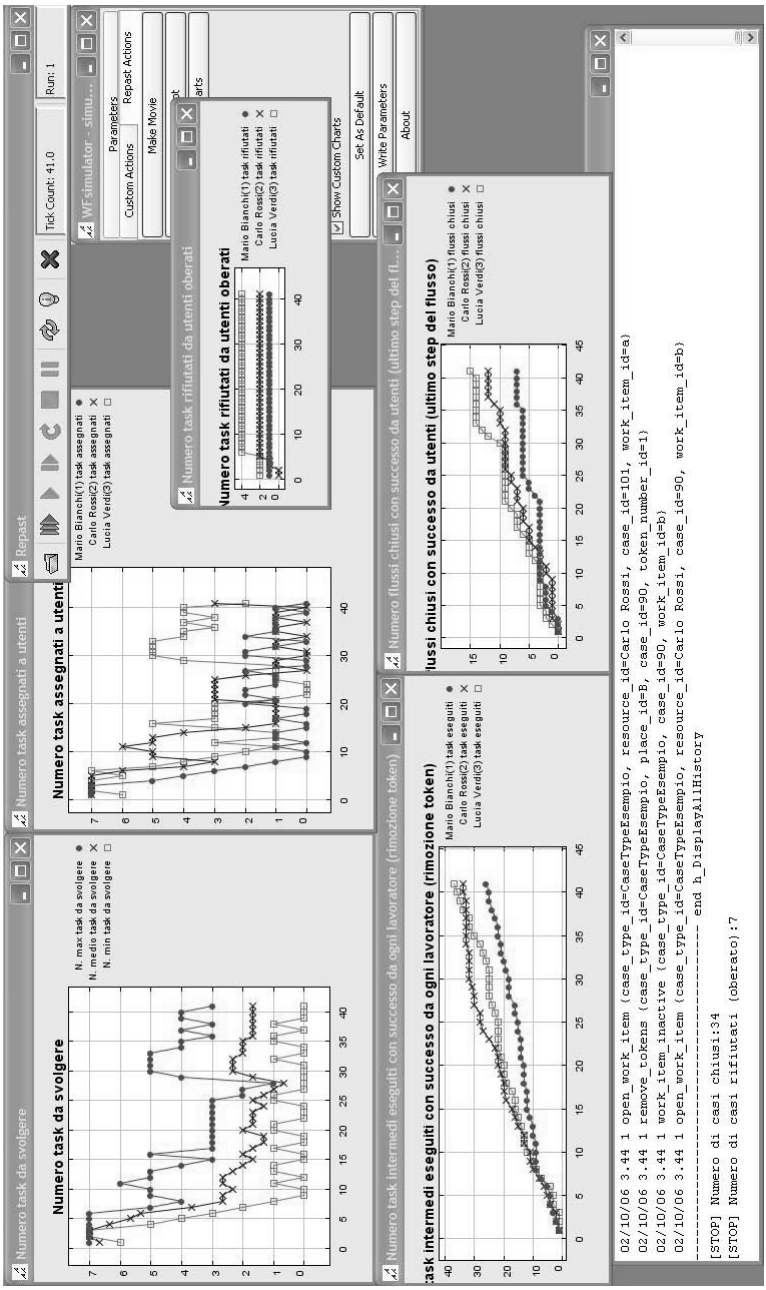

Fig. 3 Screenshot of the Wf-Simulator GUI for the SABS implementation

Concerning the workflow model, the needed parameters are:

- The number of different human resources and users
- The list of group of users and their description
- The list of users, and for each user the roles and the authorizations as well as the mapping to the set of groups
- The list of transaction and relative parameters. This list allows to create the WF and Petri network, determining the places and transitions For each place we need to know the fan-in and fan-out. For each transition we need to know: the fan-in and fan-out, the service time, the type of firing (automatic or triggered by the time, by the messages, by the resources)
- The tasks arrival rate (for the workflow)
- The number of task assigned to a user in a time unit
- The maximum number of concurrent tasks that a user can handle

4 Practical Experiences

As mentioned in the introduction, the proposed framework has been used to study different processes with the purpose to evaluate their performances and to conduct *what-if analysis* preparatory for the process re-engineering.

In the following we present three case studies we dealt with: the SABS process; the Day Hospital Surgery Admission process; the Blood Examination process.

4.1 The SABS Case Study

The workflow that model the SABS is shown in figure 4. The system parameters were characterized analyzing the SABS access logs and the SABS database. In the workflow there are seven different categories of human resources, and eight possible operations to manipulate goods and services. The human resource categories are: Generic user, Call Center operator, Dean, Administrator, Officer, User (Servant), Officer (Servant). Each group of users is authorized to execute only a subset of operations (see table 4.1). Users, depending on the category, can submit a request, can assign a request, can split a request in sub-requests, can complete a request, can generate sub-requests for a partially satisfied request, can negotiate a request change, can negotiate the closure of a request and can remove a request.

The analysis of the workflow properties allowed to discover structural problems, that are: the workflow did not satisfy the soundness and validity properties (see [7]); there was a dead-step; and there was a meaningless transaction (i.e. a transaction that never fire). The action taken was to re-design the workflow.

Figures 5 (part A) and 6 (part B) shown a portion of the re-engineered workflow.

In this case the what-if analysis had different goals, for example: performance optimization (e.g. improve the number of requests successfully served); identification of potential bottleneck in the flow; costs reduction. Due to confidentiality reasons results can not be reported here.

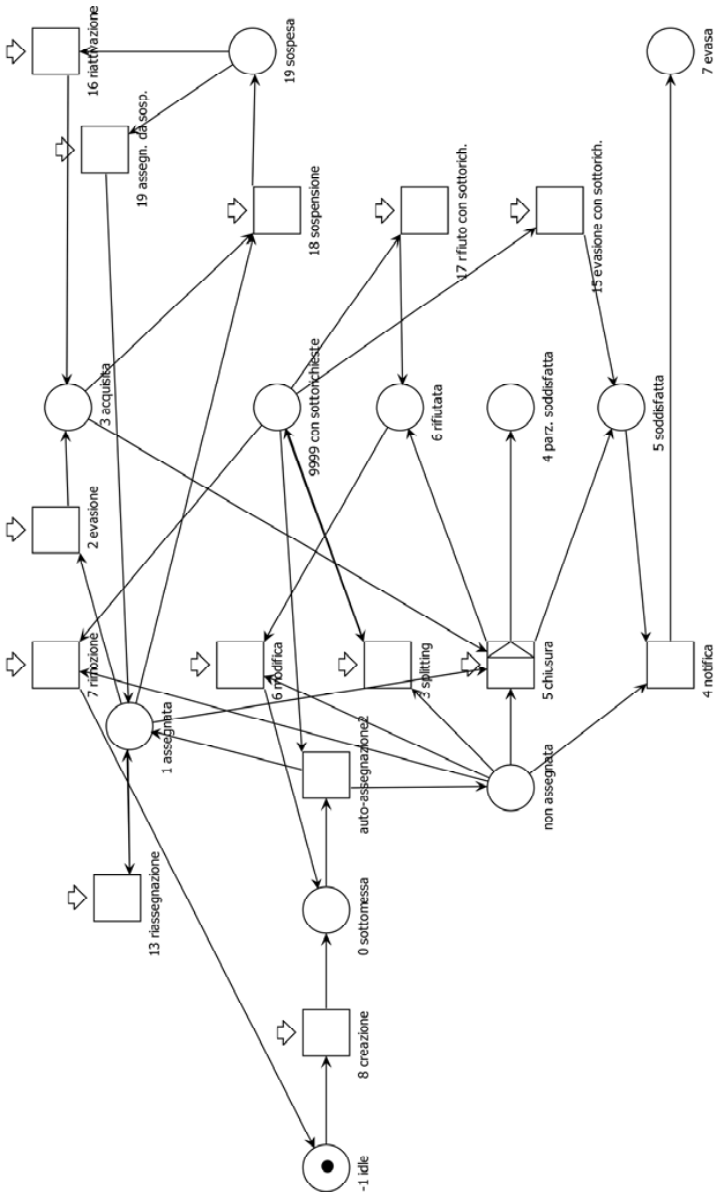

Fig. 4 SABS: The WF-net model

Table 1 The SABS capability list

Activitiy Description	Resources/Roles authorized
Create	All
Assign	Call Center, Dean, Officer, Officer (Servant)
Complete	User (Servant), Officer (Servant)
Split	Call Center, Dean, Officer, Officer (Servant)
Notify	Call Center
Close	User (Servant), Officer (Servant)
Modify	Call Center
Remove	Call Center

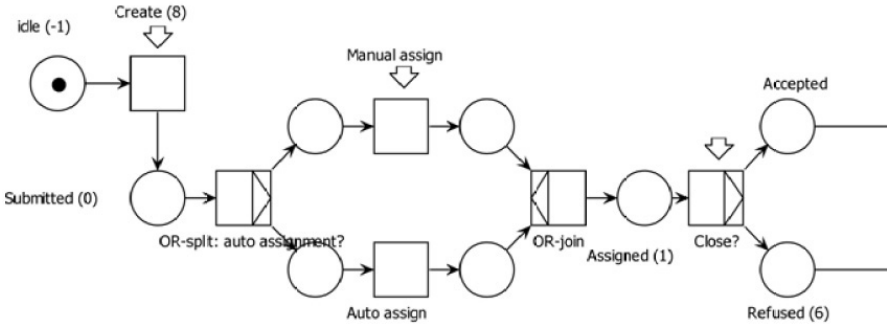

Fig. 5 The WF network model (part A) of the SABS workflow. After the *accept* place, there is the transaction *Re-assign* in part B. After the place *refused(6)*, there is the transaction *close(5)* in part B.

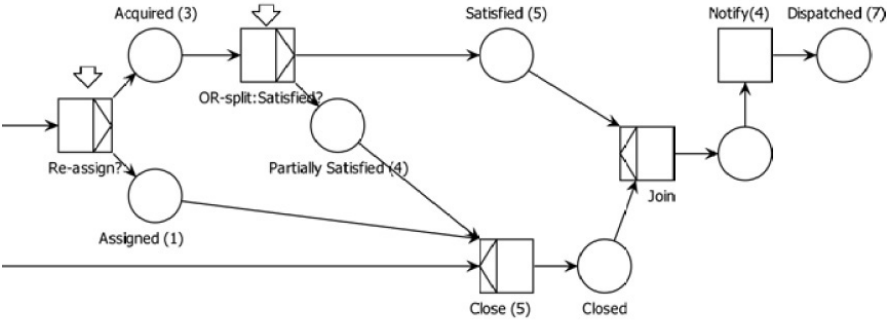

Fig. 6 The WF network model (part B) of the SABS workflow

4.2 The Day Hospital Surgery Admission Case Study

This case study is related to the complex process activated by a patient requesting for a Day Hospital surgery (DH process in the following). The DH process start

with a reservation request issued by the patient to the regional center for health care reservations (Centro Unico Prenotazioni), CUP hereafter. The CUP proposes one or more options for the first appointment and the patient has the possibility to refuse, because, for example, the appointment is too far. We assume that the CUP does not influence the performance of the system because it is a external infrastructure out of our control. If the patient accept the appointment she is examined (on the date) by the surgeon that decide if the surgery is needed or if it is more appropriate to provide an alternative treatment.

If the surgery is really needed the assigned doctor open a case history and assign the appropriate clinic tests that will be performed in the laboratory associated with the Day Hospital ward. Concluded the clinic tests the Laboratory send back the results to the surgeon that has all the elements to decide if the patient can be operated or if is needed some pre-surgery therapy to improve the health level of the patient and therefore the success of the surgery itself. Terminated the treatment period, clinic tests are repeated. When the exams results show an health level of the patient that is appropriate to execute the surgery, the surgeon will book and execute the surgery.

The high level view of this process, hereafter DH process, is described by the workflow shown in figures 7 and 8. Transactions `l`, `e` and `c` fire in places `P`, `G` and `D` (part A) respectively that are the source of tokens for transactions `m`, `f` and `q` (part B) respectively.

Each macro activity correspond to a single task or to a complex process described by a workflow and analyzed separately (see table 2 for some examples).

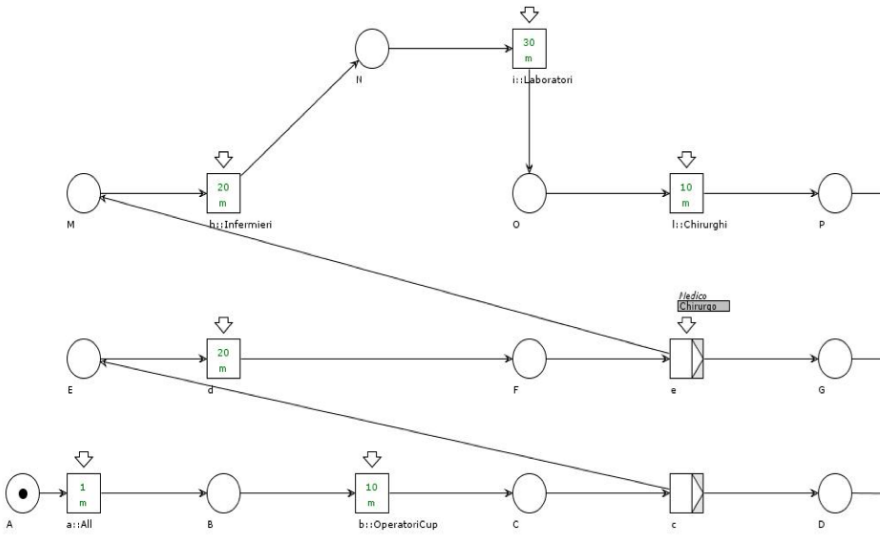

Fig. 7 The Day Hospital surgery workflow (part A)

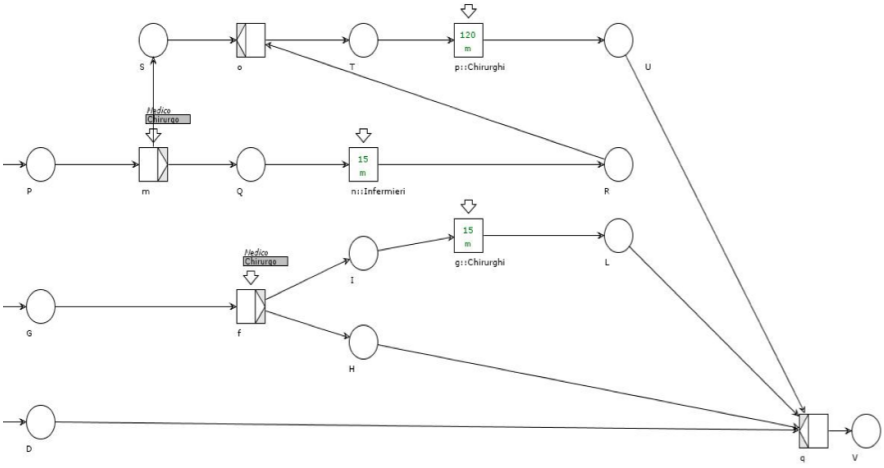

Fig. 8 The Day Hospital surgery workflow (part B)

The goal of the analysis is the optimal planning of resources to maximize the number of patients served by the DH process.

To design the workflow model and to characterize model parameters has been queried different databases such as: the regional CUP database, the Enterprise Resource Planning (ERP) database and the Laboratory database. The CUP database give us information about the number of patients that request a visit. From the ERP database we extracted the organization structure of the ward and the number of human resources, e.g. doctors and nurses. From the Laboratory database has been extracted the organization structure of the laboratory.

The main model parameters are shown in table 3. The resources and relative roles identified are: Surgeon, Nurser, Laboratory, CUP Operator. The laboratory is a complex structure composed of different resources and roles but at this level of abstraction it is consider an atomic resource. In the next use case will be presented a use case that modeled the laboratory workflow.

We decide to consider this to cases to show the hierarchical nature of the modeling approach we propose. Indeed processes can be exploded and refined with dedicated Wf-net as in the case of the laboratory.

One of the main metrics we used to measure the performance of a workflow and in this specific case of the DH process is the *Number of Successfully Closed Cases (NSCC)* computed over a specific time period T. In the DH process $T = 30$ days and the NSCC measures the number of patients that activate the DH process and that are successfully servers, that is they got a surgery or an alternative treatment or a pre-surgery therapy.

The what if analysis we conducted is oriented to identify the more appropriate set of resources that maximize the number of closed cases. The initial set of resources was 17 Surgeon, 17 Nursers, 1 Laboratory and 5 CUP Operators. With this combination were assigned 292 cases, 28 cases were successfully closed and 22 rejected.

Table 2 Main activities composing the DH case study

Activity Description	Transition id
CUP reservation	a
Check on CUP reservation acceptance	b
Pre-Surgery visit	c
Therapy prescription	d
Pre-Surgery tests	e
Exams verification	f
Surgery reservation	g

Table 3 Activities parameters and capability list for the DH case study

Activity Description	Service time	Resource involved	Trigger type
CUP reservation	10min.	CUP operator	User
Check on CUP reservation acceptance	0min	–	Automatic
Pre-Surgery visit	20min	Surgeon	User
Therapy prescription	15min	Surgeon	User
Pre-Surgery tests	30min	Laboratory	User
Exams verification	10min	Surgeon	User
Surgery reservation	0min	–	Automatic

Exploring different combinations of the number of Surgeon, Nursers and Laboratories, we found that the *NSCC* can be improved of 20% increasing of the 11% the number of surgeon and nursers. Indeed the best result were obtained with 19 Surgeon, 19 Nursers, 1 Laboratory and 5 CUP Operators. In this case, over the 291 assigned cases in 30 days, 31 were successfully closed and 20 rejected.

4.3 The Blood Examination Case Study

In this section we consider the execution of clinic tests process in the Internal Laboratory of a specific structure/ward (hereafter LIS process). A typical example of test is blood examination. The LIS process, that is a macro activity of the DH process above considered, is a complex process characterized by specific cases depending on the clinic exams under consideration. In this section we analyze the blood examination request case.

The LIS workflow starts with the collection of a blood sample in a test tube labelled with a bar code. The test tube is accepted after an optical scan of the bar code label. The blood sample is therefore analyzed by means of a clinic instrument that automatically generate the answer. The blood test answer is validated by a doctor. The resource involved in a Laboratory are clinic tools, doctors, clinic technicians

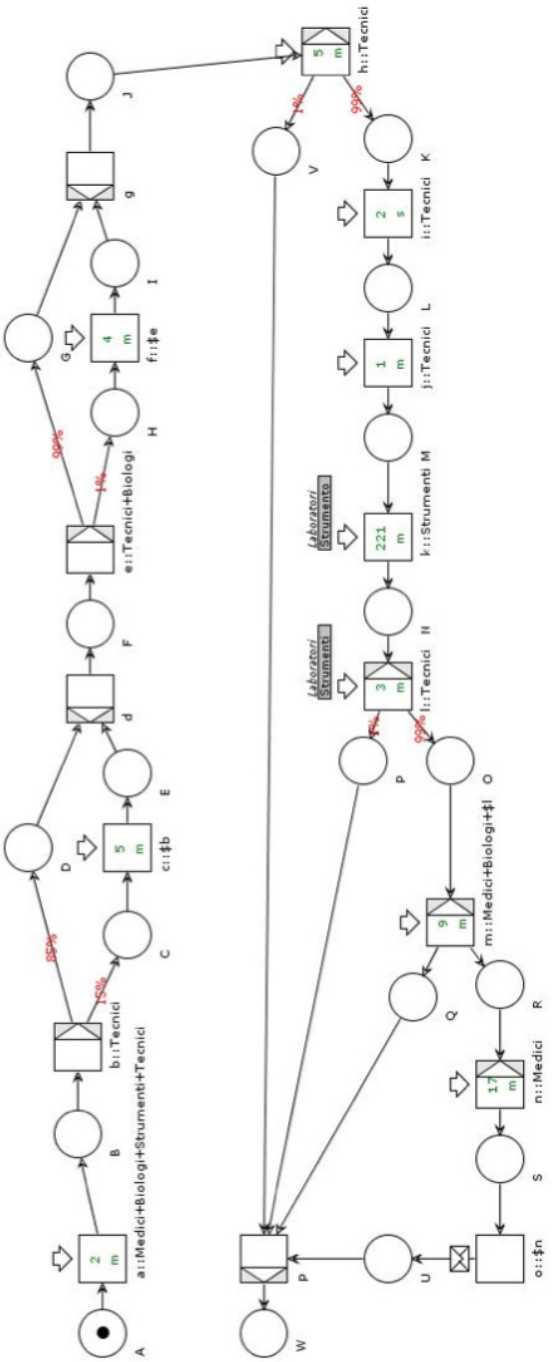

Fig. 9 The blood examination case of the LIS workflow

Table 4 Main activities of the blood examination case in the LIS case study

Activity Description	Transition id
Reception of the blood sample	a
Identification of the patient	b
Registration of patient data	c
Pre-analytic validation	d
Analysis of the blood sample	e
Technical validation	f
Clinical validation	g

Table 5 Activities parameters and capability list for the blood examination case in the LIS case study

Activity Description	Service time	Resource involved	Trigger type
Reception of the blood sample	2min.	All	User
Identification of the patient	0min	Technicians	User
Registration of patient data	5min	Technicians	User
Pre-analytic validation	5min	Technicians	User
Analysis of the blood sample	221min	Tools	User
Technical validation	3min	Tools	User
Clinical validation	9min	Tools, Doctors, Biologists	User

and biologists. Some of the main activities and the related parameters are reported in tables 4 and 5.

Also in this case the what-if analysis we conducted is oriented to find the allocation of resources that maximizes the *NSCC* metric. With the initial setup (1 clinic instrument, 7 doctors, 6 biologists and 8 technicians) the *NSCC* is equal to 124 over 308 requests of analysis in 30 days. The number of rejected cases is 15. Adding two more biologists is possible to achieve an *NSCC* of 145 over 324 assigned cases in 30 days with only 21 rejected cases.

5 Perspectives and Concluding Remarks

The lessons we learned from this experience is the following. First, the majority of case we dealt with, business processes are not designed having in mind performances and other non functional properties. A framework such as the Wf-Simulator is a valid support to the performance by design.

Second, the Wf-Simulator allows to discover that, in many cases, performance improvement can be achieved only dimensioning the proper number of resources, without re-engineering the whole process.

Third, ABMS allows to describe any type of resources and behavior. The main drawback is that the agent model must be customized for each use case. Indeed, also if the Wf-Simulator enable a graphical definition of the workflow and an automatic

translation into the WfMS description language, for each resource must be coded an agent with related rules of behavior. Right now there are no solution to completely automate this process. The use of aspect oriented programming and model driven development can be used in the direction to generalize and to automate agents design and development.

Finally, the methodology we defined is independent from the implementation we provided. Indeed the Wf-Simulator can be implemented using different ABMS frameworks and WfMS. We tried different combinations of Agent-based simulator and WfMS mainly for education purposes. A reasoned comparison has not yet be done.

Acknowledgements. The authors would thanks Ing. Silvia Agatello, that was the main contributor to the first Wf-Simulator release, and all the master and PhD students that contributed to the improvement of the Wf-Simulator framework.

References

[1] Rapid application development toolkit radicore.org (2006), http://www.tonymarston.net/php-mysql/workflow.html
[2] van der Aalst, W.: Putting petri nets to work in industry. Computers in Industry 25(1), 45–54 (1994)
[3] van der Aalst, W.: A class of petri net for modeling and analyzing business processes. Computing Science Reports 95/26, Eindhoven University of Technology, Eindhoven (1995)
[4] van der Aalst, W.: The application of petri nets to workflow management. The Journal of Circuits, Systems and Computers 8(1), 21–66 (1998)
[5] van der Aalst, W., Hofstede, A.H.M.T.: Yawl: Yet another workflow language. Information Systems 30, 245–275 (2003)
[6] van der Aalst, W.M.P.: Three good reasons for using a Petri-net-based workflow management system. In: Navathe, S., Wakayama, T. (eds.) Proceedings of the International Working Conference on Information and Process Integration in Enterprises (IPIC 1996), pp. 179–201 (1996)
[7] Aalst, W.V.D., Hee, K.V.: Workflow Management: Models, Methods, and Systems. MIT Press (2004)
[8] Attie, P.C., Singh, M.P.: Specifying and enforcing intertask dependencies. In: Proceedings of the 19th VLDB Conference, pp. 134–145 (1993)
[9] Bonabeau, E.: Agent-based modeling: Methods and techniques for simulating human systems. Proceedings of the National Academy of Sciences of the United States of America 99 (suppl. 3) (2002)
[10] Bonner, A.J.: Workflow, transactions and datalog. In: Proceedings of the Eighteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 1999, pp. 294–305. ACM, New York (1999)
[11] Champy, M.H.J.: Reengineering the corporation: A manifesto for business revolution. Harper Business, New York (1993)
[12] Chomicki, J., Toman, D.: Temporal logic in information systems. In: Logics for Databases and Information Systems, pp. 31–70. Kluwer Academic Publishers, Norwell (1998)

[13] Davenport, T.: Process innovation: Re-engineering work through information technology. Harvard Business School Press, Boston (1993)
[14] Davulcu, H., Kifer, M., Ramakrishnan, C.R., Ramakrishnan, I.V.: Logic based modeling and analysis of workflows. In: Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, PODS 1998, pp. 25–33. ACM, New York (1998)
[15] Ellis, C.A., Nutt, G.J.: Office information systems and computer science. ACM Comput. Surv. 12, 27–60 (1980)
[16] Eshuis, R., Wieringa, R.: Verification support for workflow design with uml activity graphs. In: Proceedings of the 24th International Conference on Software Engineering, ICSE 2002, pp. 166–176. ACM, New York (2002)
[17] Hammer, M.: Re-engineering work: Don't automate, obliterate. Harvard Business Review, 104–111 (July-August 1990)
[18] Hoogendoorn, M., Jonker, C.M., Schut, M.C., Treur, J.: Modeling adaptive multi-agent organizations for naval missions. In: Proceedings of the 5th WSEAS International Conference on Artificial Intelligence, Knowledge Engineering and Data Bases, AIKED 2006, pp. 470–478. World Scientific and Engineering Academy and Society (WSEAS), USA (2006)
[19] Jablonski, S., Bussler, C.: Workflow Management: Modeling Concepts, Architecture and Implementation. Int. Thomson Press (1996)
[20] Jacques Ferber, O.G., Michel, F.: From Agents to Organizations: an Organizational View of Multi-Agent Systems. Springer (2004)
[21] Salimifard, M.W.K.: Petri net-based modelling of workflow systems: An overview. European Journal of Operational Research 134(3), 664–676 (2001)
[22] K. Sarshar Th. Theling, P.L., Jerrentrup, M.: Integrating process and organization models of collaborations through object petri nets. Contribution to XML4BPM (2006)
[23] Kavi, K., Kung, D.C., Bhambhani, H.: Extending UML for Modeling and Design of Multi-Agent Systems. In: Proc. of Int'l Workshop on Software Engineering for Large-Scale Multi-Agent Systems (may 2003)
[24] Laskowski, M., Mukhi, S.: Agent-Based Simulation of Emergency Departments with Patient Diversion. In: Weerasinghe, D. (ed.) eHealth 2008. LNICST, vol. 1, pp. 25–37. Springer, Heidelberg (2009)
[25] Liebermann, B., Merz, W.L.M.: Using mobile agents to support inter-organisational workflow management. Applied Artificial Intelligence 11(6), 551–572 (1997)
[26] Sierhuis, M., Clancey, W.J.: Agent-based mission modeling and simulation. In: Agent-Directed Simulation Workshop, 2006 Spring Simulation Multiconference, SpringSim (2006)
[27] Mukherjee, S., Davulcu, H., Kifer, M., Senkul, P., Yang, G.: Logic-Based Approaches to Workflow Modeling and Verification. In: Chomicki, J., van der Meyden, R., Saake, G. (eds.) Logics for Emerging Applications of Databases, ch. 5, pp. 167–202. Springer, Berlin (2004)
[28] North, M.J., Collier, N.T., Vos, J.R.: Experiences Creating Three Implementations of the Repast Agent Modeling Toolkit. ACM Trans. Model. Comput. Simul. 16(1), 1–25 (2006)
[29] North, M.J., Macal, C.M.: Managing Business Complexity: discovery strategic solution with agent-based modeling and simulation. Oxford University Press (2007)
[30] Odell, J., Parunak, H., Bauer, B.: Extending UML for Agents. In: Proc. of Agent-Oriented Information Systems Workshop, pp. 3–17 (2000)
[31] Petri, C.: Kommunikation mit automaten. phd thesis. Ph.D. thesis, Institut fur instrumentelle Mathematik, Bonn (1962)

[32] Salimifard, K., Wright, M.: Petri net-based modelling of workflow systems: An overview. European Journal of Operational Research 134(3), 664–676 (2001)
[33] Short, T.D.J.: The new industrial engineering: Information technology and business process redesign. Sloan Management Review 31(4), 11–27 (1990)
[34] da Silva, V.T., Choren, R., Lucena, C.J.P.: A UML Based Approach for Modeling and Implementing Multi-Agent Systems. In: Proc. of 3rd Int'l Conf. on Autonomous Agents and Multiagent Systems, pp. 914–921 (2004)
[35] Takahashi, H., Takahashi, S., Tsuda, K., Terano, T.: Analysis passive investment strategies and asset price fluctuation in financial market through agent. In: Terano, T., Kita, H., Kaneda, T., Arai, K., Deguchi, H., Chen, S.H., Cioffi-Revilla, C., Gilbert, N., Kita, H., Terano, T. (eds.) Agent-Based Simulation: From Modeling Methodologies to Real-World Applications, Agent-Based Social Systems, vol. 1, pp. 144–157. Springer, Tokyo (2005)
[36] Workflow, M.: Coalition: Workflow management coalition terminology and glossary. The Workflow Management Coalition Specification (1999), Document Number WFMC-TC-1011 Issue 3.0
[37] Workflow, M.: Coalition: Wfmc (2006), http://www.wfmc.org
[38] Yamamoto, H., Ishida, K., Ohta, T.: Evolution of Cooperative Behavior in C2C market: Effect of Reputation Management System, vol. 1, pp. 48–57. Springer, Tokyo (2005)

Author Index