Pre-Proceedings of AOSE'2011

The 12th International Workshop on Agent-Oriented Software Engineering

---

AAMAS Workshop, 2 May 2011

Editors:

Danny Weyns

Jörg P. Müller

Preface

Since the early 1990s, multi-agent system researchers have developed a large body of knowledge on the foundations and engineering principles for designing and developing agent-based systems. The 11 past editions of the AOSE workshop had a key role in this endeavor. For 2011, the workshop organizers and the steering committee decided to organize a special edition of AOSE. In particular, the goal is to wrap up the previous editions of the workshop with a discussion of the state of the art in the key areas of AOSE and based on that outline the future of the field. This way, we aim to find a way out of the increasing fragmentation and fuzziness on software engineering in multi-agent systems.

The workshop program consists of a number of invited papers complemented with accepted papers from the call for papers. Renowned researches and engineers have contributed with invited papers in different sub-areas of the field, including agent-oriented methodologies (Jorge J. Gomez Sanz), coordination infrastructures for multi-agent systems (Juan Antonio Rodriguez), programming agents and multi-agent systems (Mehdi Dastani), engineering multi-agent organizations (Virginia Dignum), engineering self-organizing systems (Van Parunak), agents and services (Munindar Singh). In addition, we accepted five regular papers from the nine submissions. The accepted papers cover a broad scope of topics in AOSE, from dynamic BDI architectures to an assessment of practical agent applications.

The workshop organizers will edit a special issue in The Knowledge Engineering Review on Challenges in Agent-Oriented Software Engineering based on a selection of revised workshop papers. Additionally, revised accepted papers are planned to be published in a volume of the Lecture Notes for Computer Science series.

We thank all the authors for submitting their work to AOSE. We are grateful to the members of the AOSE 2011 PC for their valuable reviews. We hope that the presented papers will stimulate researchers and engineers to outline a future for agent-oriented software engineering.

Danny Weyns
Jörg P. Müller

Taipeh, May 2011

Workshop Chairs

Danny Weyns	KU Leuven, Belgium
Jörg P. Müller	TU Clausthal, Germany

Program Committee

Carole Bernon	IRIT, Universit Paul Sabatier, France
Juan Antonio Botia Blaya	Universidad de Murcia, Spain
Massimo Cossentino	Italian National Research Council, Italy
Scott Deloach	Kansas State University, USA
Ruben Fuentes	Universidad Complutense de Madrid, Spain
Alessandro Garcia	PUC-Rio, Brazil
Aditya Ghose	University of Wollongong, Australia
Holger Giese	University of Potsdam, Germany
Paolo Giorgini	University of Trento, Italy
Adriana Giret	Technical University of Valencia, Spain
Marie-Pierre Gleizes	IRIT, France
Laszlo Gulyas	Aitia International, Inc., Hungary
Jorge J. Gmez Sanz	Universidad Complutense de Madrid, Spain
Brian Henderson-Sellers	Sidney University of Technology, Australia
Jeffrey Kephart	IBM T.J. Watson Research Center, USA
Mark Klein	Software Engineering Institute, Carnegie Mellon, USA
Joao Leite	Universidade Nova de Lisboa, Portugal
Philippe Mathieu	University of Lille, France
Frédéric Migeon	IRIT, Universit Paul Sabatier, France
Simon Miles	King's College London, UK
Haralambos Mouratidis	University of East London, UK
Flavio Oquendo	European University of Brittany UBS/VALORIA, France
H. Van Dyke Parunak	Jacobs Technology, Jacobs Engineering, Ann Arbor, USA
Michal Pechoucek	Czech Technical University Prague, Czech Republic
Anna Perini	Fondazione Bruno Kessler, IRST, Italy
Gauthier Picard	SMA/G2I - Ecole des Mines de Saint-Etienne, France
Alessandro Ricci	University of Bologna, Italy
Fariba Sadri	Imperial College London, UK
Onn Shehory	IBM Haifa Research Lab, Israel
Michael Winikoff	University of Otago, New Zealand
Eric Yu	University of Toronto, Canada

Additional Reviewers

Jana Görmer	TU Clausthal, Germany
Christopher Mumme	TU Clausthal, Germany

Table of Contents

Software Engineering for Self-Organizing Systems

H. Van Dyke Parunak and Sven A. Brueckner

Vector Research Center, Jacobs Engineering Group
3520 Green Court, Suite 250
Ann Arbor, MI 48105
{van.parunak, sven.brueckner}@jacobs.com

Abstract. Self-organizing software systems are an increasingly attractive approach to highly distributed, decentralized, dynamic applications. In some domains (such as the Internet), the interaction of originally independent systems yields a self-organizing system *de facto*, and engineers must take these characteristics into account to manage them. This review surveys current work in this field and outlines its main themes, identifies challenges for future research, and addresses the continuity between software engineering in general and techniques appropriate for self-organizing systems.

Keywords: software engineering, self-organization, distributed systems, decentralized computing, emergent behavior

1 Introduction

A few decades ago, the idea of self-organization was an intriguing option in the design of a computer application, and its proponents could engage in spirited debate with more classical views of software structure. Today, in many domains (particularly those based on computer networks), the question is no longer whether to use self-organization. Real-world open systems with thousands of autonomous components do in fact organize themselves, for better or for worse. The challenge before us is to understand this dynamic and learn how to manage it [123].

There is no lack of activity around software systems that in one way or another control themselves without direct human intervention. In an attempt to focus this review, we distinguish three kinds of systems: autonomous, self-adaptive, and self-organizing.

An *autonomous* system is one that senses and responds to its environment. The vast research world of agent-based and multi-agent systems is concerned with such systems. Self-organizing systems are made up of autonomous systems, but not every autonomous system is self-organizing.

A *self-adaptive* system is an adaptive system that responds to change without intervention by its creator (thus the "self"). The change may be in the environment, or it may be within the system itself (for example, a fault condition). The difference between a self-adaptive system and an autonomous system is a matter of perspective. If

the environment were static, there would be nothing for an autonomous system to respond do, so every autonomous system in fact adapts in one way or another to environmental change. When we call a system *self-adaptive*, we imply that the change with which the system must cope is unusually large and potentially disruptive.

The *self-organizing* system is a special kind of self-adaptive system. We emphasize two points of refinement.

First, as the use of the term "organize" suggests, a self-organizing system consists of multiple components that can change their interrelations. A single agent could be self-adaptive, but we would not call it self-organizing. Definitions of self-organization often invoke the notion of disorder or "entropy" across the population of elements [51, 100].

Second, we are particularly interested in systems whose response to change does not require centralized reflection. Much work on self-adaptive software requires the system to have an internal representation of its goals [60], or a model of its own architecture [87], or a set of explicit policies ("in case of X, do Y") [40, 48], to guide its adaptation. We are focused on systems that require neither such explicit representations nor a central module to manage the change in the system in response to disruption. Because of this distinction, a hierarchical feedback control system, while composed of many different parts, would still be considered self-adaptive rather than self-organizing.

While this distinction is important and useful [83], we will consider some work that does not completely meet this objective. After all, we are dealing with software *engineering*, not software *science*, and progress often depends on drawing inspiration from many sources [40, 142].

In Section 2, we review the state of the art in software engineering for self-organizing systems. Section 3 summarizes some major trends that we see in current practice. Section 4 outlines directions for future research. Section 5 summarizes how this particular flavor of software engineering relates to the broader field, and Section 6 concludes.

2 State of the Art

In this section, we begin by reviewing some of the immense literature in this field, then survey applications of self-organizing systems and some of the main mechanisms that they employ.

2.1 Literature

Our focus here is on survey articles or programmatic discussions. Later sections of this review will consider more specific studies.

While we distinguish self-organization from self-adaptation, we stand on the shoulders of extensive work in autonomous and self-adaptive software. The classic notion of a feedback control loop can be traced back to the nineteenth century [82], and it was natural for the idea to be applied to computer programs, largely under the

inspiration of Norbert Wiener [146]. The flavor of adaptive control in robotic and manufacturing systems was captured in NIST's Real-time Control System (RCS) reference architecture [1, 2]. Later, IBM's Autonomic Computing Initiative [69, 73] sought to apply these techniques to purely informational systems.

Autonomous systems are the focus of much robotic research, and application concerns have led to recent efforts to define a scale of autonomy [67] and develop methods to test a system's autonomy [70].

Self-adaptive software has been the object of two recent seminars at Schloss Dagstuhl [29, 52], and a special issue of the Journal of Systems and Software is in preparation on this topic [145]. The topic is the object of a careful review article [114], whose approach (focusing on the functions of monitoring, detecting, deciding, and acting) very clearly captures the reflective nature of self-adaptation as opposed to self-organization.

The design and control of self-organizing software per se was the focus of four editions of the ESOA (Engineering Self-Organising Applications) workshop [21, 24, 25, 41], and is treated in Gershenson's recent dissertation [50], and a wide range of shorter studies will be identified in later sections of this review. The areas of self-adaptive and self-organizing systems together are the focus of the ongoing IEEE International Conferences on Self-Adaptive and Self-Organizing Systems (SASO) [116].[1]

2.2 Applications

Self-organization has been applied to a wide range of problems. As noted in the introduction, self-organization is unavoidable in distributed systems, especially open ones, such as networks [15, 61, 64, 123] and water distribution [42], and highly desirable in managing large numbers of robots [53-55, 118, 120] and in agile manufacturing settings [19, 94, 109, 132], where it competes with hierarchical control systems, including holonic schemes [26, 133] that we would consider self-adaptive but not self-organizing. In purely informational settings self-organization has been used to coordinate multiple theorem provers [36], to enable documents to organize themselves [104] and find likely users [20, 62], and to reassign tasks among agents [31, 88]. Mechanisms inspired by wasps and termites have been demonstrated for self-organized construction of physical systems [140].

2.3 Mechanisms

A wide range of instances of self-organization in nature have been isolated and characterized to the point that they can be applied in artificial systems [14, 27, 93]. These derive mostly from social animals (pheromone systems [72, 99, 108, 109, 117, 119, 134, 137], stimulus-based load balancing [140], insect clustering [58, 59, 76, 84, 104, 138], firefly synchronization [130]), but markets [32, 106, 107] and physical systems

[1] Not all studies that take the name "self-organizing" satisfy our definition of the field as distinct from self-adaptation." We would class some of the work reported in venues devoted to "self-organizing software" as in fact only self-adaptive.

such as potential fields [46, 80, 81, 128, 143] have also been invoked. While these mechanisms can be characterized in terms of feedback control, in their natural settings they are highly decentralized and do not rely on explicit models of the structure, goals, or policies of the overall system, thus qualifying as self-organizing and not just self-adaptive.

2.4 Reflection on the State of the Art

While self-organizing solutions have been widely explored, they tend to have two limiting characteristics [142]. First, most applications demonstrate the capabilities of a single mechanism, and do not consider the potential interaction of a toolkit of mechanisms. Second, among software engineers there is relatively little work on the theoretical foundations of these mechanisms. We will return to these themes when we outline directions for future work.

3 Outline of Main Trends

Several general trends are apparent from this brief and highly selective review: decentralization, openness, imitation of nature, and reliance on simulation. To a large degree, these characteristics reflect our definition of "self-organization," but they are common enough in practice to justify focusing on systems that exhibit them.

3.1 Decentralization

Since we distinguish self-organization from self-adaptation partly by the multi-component decentralized nature of the former, decentralization is not surprising, but it is worth refinement and reflection.

Decentralization is not a black-or-white dichotomy. It is useful to distinguish three levels, and in a highly populous system with heterogeneous members, one can imagine gradations and combinations along this spectrum.

At one extreme, and outside our purview, are centralized systems, in which all decisions are made at a single location. We include here not only monolithic systems, but also hierarchical feedback control systems [1]. Holonic systems were originally motivated by emergent dynamics over a hierarchy that defines scale rather than control [75], but engineered holonic applications often look very much like hierarchical control [16, 26, 113]. We can view such systems as centralizing two kinds of information: declarative information about the current state of the system, and imperative information that determines next steps.

At the other extreme are systems in which each entity interacts only with those in its local vicinity. Its neighborhood defines its view of the state of the world (declarative information), and it can only act within that narrow purview (imperative information).

At an intermediate level, the state of the system (declarative information) is collected centrally and made available to all components, but all action is taken locally.

In some cases, one can detect movement along this cline. For example, one team began working with multiple interacting theorem provers in a centralized setting [37], but then revised the system to use global information but only local decisions [36]. In recent work in other domains, their design has moved to a distribution of both declarative and imperative information [42]. A motivation for this movement is the increasing need for real-time response [38], which can be hindered if multiple layers of hierarchy need to be queried to make a decision [144].

There is a limitation to complete localization of interaction: it limits look-ahead. "It only functions acceptably when the (recent) past is representative for the (near) future" [131]. Predictive mechanisms have been proposed to address this problem [30]. A particularly interesting class of problems uses a second-level self-organizing system to make these predictions through a model of the world in which interactions are spatially local but are allowed to evolve faster-than-real-time into the future [63, 97, 102].

Market systems represent an interesting segment of the centralized-decentralized spectrum. Classical Walrasian markets depend on posting bids centrally so that agents can make local decisions [32], thus embodying our midpoint of global declarative information and local imperative information. However, an alternative form of market, Edgeworth barter [4], allows agents to interact pairwise, and still guarantees global convergence. This form of market is completely distributed, and has been applied to problems of distributed constraint optimization [106, 107].

3.2 Openness

In building a self-organizing system from the ground up, one can impose homogeneity on the elements. However, the kinds of self-organization that are being imposed on us (say, through the internet) force us to deal with systems whose elements do not conform to a single blueprint.

Openness greatly increases the complexity of a system. Any single element needs to be prepared to interact with everything outside of its own boundary, which now includes not only other elements that are like itself, but also technical, geographic, political, social and economic realities [39, 123]. Because we cannot predict all of these influences in advance, the line between the preparation of the system (its specification, design, implementation, and testing) and its operation is greatly blurred, a distinction to which we shall return.

In a closed system, entities can be designed to interact directly with each other. The need to cope with an open system has led researchers to focus on a common framework or infrastructure. Any agent that can interact with this infrastructure can be included in the system. At the most primitive level, the physical world is the infrastructure, and agents must have physical sensors and actuators to deal with it (an approach exploited in the axiom that "the world is its own best model" [18]. In the natural world, animals sometimes use the physical world to hold arbitrary markers (for instance, insect pheromones), which is one form of stigmergy [56] (the other being functional changes in the world). Disembodied agents require a computational

framework, and a major line of research [3, 135] is focused on designing such frameworks and their component mechanisms [90-92, 136].

The framework approach to openness imposes a "lowest-common denominator" on all interacting components. There is a trade-off between the simplicity of the common interface to the framework and the range of entities that can interact. A very simple interface supports the widest range of entities, but also limits the amount of information that the entities can exchange [131]. For example, a market is a framework that permits open interaction among a wide range of economic actors by reducing all considerations to a single scalar, price, discarding much detailed information along the way. This consideration has led to the development of relatively sophisticated interaction languages, such as tuple spaces [28, 78] and highly structured symbolic "pheromones" [109].

Openness has implications for the security of a system, in two opposing directions. On the one hand, the more open a system is, the fewer restrictions are imposed on an element that seeks to participate in it, and the easier it is for malicious elements to insert themselves into the system's operation. On the other hand, the more decentralized and localized a system's decisions are, the harder it will be for a malicious element to understand and manipulate the overall state of the system. Roughly, open systems are easier to infiltrate than closed ones, but tend to limit the extent of damage that can be done. On this subject, engineering of self-organizing systems needs to draw extensively on work on cyber-security and trust.

3.3 Imitation of Nature

We have already observed (Section 2.3) that mechanisms for self-organizing systems tend to be drawn from nature, and in particular from biological systems. This tendency can be traced directly to the problem of openness, which organisms must confront in order to survive. The more sophisticated the organism, the more structure it can impose on its own environment, and the less open that environment becomes to other entities. A parade example is the rich linguistic mechanisms that humans use to coordinate with one another. Computational mechanisms modeled on human consciousness and linguistic interaction are the holy grail of AI research, but still beyond our grasp. Artificial versions of cognition have been described as autistic [131] and schizophrenic [65, 125], "idiot savants" with focused capability but lacking adaptability. This may lie behind the preference for simpler insect models in self-organizing software [131], though in fact a more careful analysis suggests humans often use the same kind of simple mechanisms that insects do [95].

3.4 Simulation

Simulation, rather than formal analysis, plays a prominent role in the engineering of most current self-organizing systems [148]. The complexity of these systems makes the development of formal models difficult [65]. In fact, a set of even very simple agents interacting with one another has the computational power of a Turing machine

[44], or perhaps even more [139], and by Rice's theorem [112], any non-trivial feature of such a system is formally undecidable.

Some proponents of simulation argue that a simulation, being a computer program, is a partial recursive function, and thus refuse to recognize any distinction between simulation and formal analysis [45]. The issue is not one of formal structure, but of insight. A computer program, while every bit as formal as a proof, has a very different structure. Most program structures are algorithmic: first do X, then do Y, and then do Z.[2] Such a structure does not lend itself to determining properties such as whether the system will halt, how rapidly it converges, how thoroughly it explores the space of possible behaviors, and whether its equilibria are stable or unstable. The results of Edmonds and Bryson [44] warn that in general such characterizations are unattainable, but as with many formal results, there are special cases where formal methods can support the engineering of self-organizing systems, as we shall see in the next section.

4 Challenges for Future Research

The themes of current systems highlight a number of opportunities for future research. These opportunities are not unexplored, but represent the cutting edge of current work in this field. We consider first the problem of composing more complex systems, then the challenge of characterizing and controlling an existing system, and finally the objective of understanding self-organizing systems formally.

4.1 System Composition

In Section 2.4, we observed that most current applications focus on a single mechanism or phenomenon. Weyns [141] demonstrates the integration of multiple mechanisms in an industrial application, but such hybrid approaches are the exception rather than the rule. Beal suggests that "the composition of phenomena into a larger complex system is rather understudied" [9], and identifies three areas that must be pursued.

First, self-organizing phenomena must be reduced to primitives with well-characterized properties and interfaces. The idea of method fragments [111] is to decompose an approach into fragments using SPEM [89] as the underlying formalism, so that they can be reused and combined with each other. A small but growing circle of activity in defining self-organization mechanisms as software design patterns [35, 47, 63, 72] is also a step in this direction.

Second, we need means of composition that allow self-organizing phenomena to be combined with predictable results. Current efforts that emphasize the centrality of frameworks [3] and architectures [141] are seeking to address this problem, but the need for "predictable results" awaits advances in formal analysis (Section 4.3).

Third, we need means of abstraction that allow details of a complex self-organizing system to be hidden when engineering or analyzing larger subunits. This characteris-

[2] Declarative languages are an exception, and represent an important research topic.

tic, identified by Simon as critical to artificial systems [126], is also likely to depend on further formal insights.

The imitation of nature that is so common in identifying individual mechanisms for self-organization holds promise here as well, if we shift our focus from the individual organisms or species to the level of the ecosystem [9, 19, 71, 74, 105, 115, 137]

4.2 System Characterization and Control

People build systems to perform some task, and need to be able to characterize their behavior and control them.

At design time, we need to understand a range of trade-offs that self-organizing mechanisms impose. These include [38] locality vs. optimality, optimality vs. flexibility, scalability vs. efficiency, efficiency vs. centralization, centralization vs. decentralization, exploration vs. exploitation, and greediness vs. purposefulness. (All of these trade-offs presume that we have well-defined measures of each property, itself a major research challenge). Depending on the requirements of the application, certain regions of each of these scales may qualify as faulty behavior, and techniques of safety engineering can be adopted to identify and avoid them [39].

As the system is operating, we need ways to characterize its behavior. Observing and analyzing the series of events that it generates is one way to gain this insight [68]. One important challenge in this task is that while the nature of the system's behavior as acceptable or unacceptable manifests at the system level, our self-organizing agenda requires us to focus on locally observable phenomena. Information theoretic measures such as the entropy over agent options [22] or over signals passing between agents [64, 66] have proven a promising local window into global system behavior.

Like behavior characterization, behavior control is difficult in a decentralized setting, and is not widely explored [142]. A system of local constraints with attributes defined over component interfaces [49] is one promising way forward. Another is to deploy a control swarm in parallel with the functioning swarm [83]. In some cases centralization may be unavoidable, and a fruitful avenue of exploration is how to combine centralized control where necessary with local control most of the time [40].

4.3 Formal Analysis of Self-Organizing Systems

Attempts to gain a formal purchase on self-organizing systems usually involve one or more of three critical dimensions: a vertical dimension ("emergence") that relates lower-level and higher-level behaviors, a horizontal dimension ("organization") that relates entities at a single level to one another, and a temporal dimension ("dynamics") that explores how the system develops through time.

Emergence.—Perhaps the most widely recognized problem in dealing with self-organizing systems is emergence [147], which we define [103] as system-level behavior that is not explicitly specified in the individual components. Abstracted from software, the problem has a long history, forming the central focus of the discipline of

statistical mechanics (which seeks to relate the observed characteristics of materials at human scale to the interactions of atoms and molecules). This perspective allows the application of concepts such as entropy [57, 64, 100, 122], phase shifts [23, 121, 124], master equations [13, 79], and universality [101] to multi-agent systems. There are further insights to be gained from this approach. For example, the renormalization group [12] has the potential to illuminate understand discontinuities in the behavior of a self-organizing system. By considering the system as it approaches certain limits (for example, low agent density and high number of agents, allowing the use of a gas model [5]), we can place bounds on system characteristics of interest, offering "thermodynamic guarantees" [123] of system behavior.

The mapping from micro to macro behaviors is not symmetrical. To derive the macro behavior from the micro, we run simulations, or (in the appropriate limits) apply techniques from statistical mechanics, and these techniques are useful in system verification. Earlier in the design process, given a specified macro behavior, we need to find micro behaviors that will yield it. The best approach to this problem that we know consists of various forms of generate and test, such as synthetic evolution, which has been applied successfully to define local agent behaviors satisfying a macro specification [17, 96, 117]. This approach requires a system architecture whose representation lends itself to such evolutionary search [19].

An interesting facet of the vertical problem is the level at which goals are satisfied. Individually selfish agents may not yield good results at the group level. We need to develop ways to define and achieve "group-selfish behavior" [131], in which the system as a whole pursues objectives that may not be optimal from the point of view of the components. Insights into this objective may come from biology. The notion of the gene, rather than the individual, as the focus of natural selection [34] can be viewed as a process for favoring a well-defined group of agents (those agents possessing the gene) as opposed to individuals.

Organization.—The horizontal dimension explores the implications of studying which agents can interact with which other ones on the behavior of the whole system. The resulting graph structure is amenable to a variety of formal tools [86]. For example, useful definitions of autonomy and emergence can be formalized in terms of entropy on signals over the edges in the interaction graph [64], and usability can be defined in terms of similar measures on edges connecting the system to users [66]. It has been suggested [122] that the Laplacian spectrum of a network, which captures aspects of the graph's modular and hierarchical structures, may facilitate formalization of the relation between these structures and dynamical processes such as distributed consensus, decentralized coordination and information dissemination [123].

Temporal.—Self-organization is a process that takes place through time, and an adequate formalization of self-organizing systems must support reasoning about the temporal dimension. In many cases, systems need to predict their own behavior in order to adapt appropriately [30, 63, 97, 102, 131], but the nonlinear nature of component interactions means that trajectories diverge over time, leading to a prediction

horizon [98] beyond which any prediction is essentially random. Estimating this horizon is critical to scoping the predictive activity of a system, and quantifying the uncertainty that is inevitable in a self-organizing system [123].

One approach to formalizing the temporal dimension is in defining formal languages to specify system development [7]. (At this point, one may invoke Epstein's insistence on the formal nature of any computer program [45], since higher-level primitives nominated by a programming language do offer a useful abstraction that can give insight to the behavior of the system programmed in the language.) There are a number of examples that could inspire further work in this area, including languages modeling gene network development [43], term-rewriting systems modeling plant growth [110] and its generalization in MGS [129], Coore's Growing Point Language for interconnect topologies [33], Nagpal's Origami Shape Language [85], Werfel's system for distributed adaptive structure generation [140], and Beal's Proto system for spatial computing [8].

5 Relation to Conventional (Software) Engineering

Engineering of self-organizing systems has drawn much from the engineering of conventional software. In this section, we highlight some of the points of continuity and contrast.

Let's begin with **engineering in general**. We have already noted the inappropriateness of the feedback control metaphor for a decentralized approach to self-organization. Nevertheless, the engineering of physical systems has a great deal to teach us. One example is how one handles **noise**. Engineering of physical systems, unlike conventional software engineering, devotes much attention to modeling and quantifying noise in the interfaces between components. Traditional software engineering assumes that noise (i.e., errors) can be eliminated, while other disciplines recognize that it is unavoidable and seek to damp it or provide for graceful degradation [9, 10]. Another example is the adaptation of methods for safety engineering to increasing the robustness and dependability of self-organizing software [39].

The notion of an **architecture** is a powerful way to engage the challenge of system composition and openness, providing a framework for algorithms and identify complementarities and system-level issues [141, 145]. Research on frameworks to provide interaction environments for components [3, 135] is a way to instantiate insights from an architectural approach.

There has been a historical shift in system analysis away from functional analysis and toward **object-oriented** system decomposition. Self-organizing systems benefit from this shift: system functions are usually distributed over many components, and even if some group of components specializes to support a function, that association happens dynamically, rather than being specified in the design [131].

The notion of **design patterns** provides a useful way to abstract individual self-organizing mechanisms so that they can subsequently be recombined in novel ways [127]. The approach has been applied to a number of mechanisms, including market based control [35], gradient fields [35, 72], predictive swarms [102, 144], replication,

collective sort, evaporation, aggregation, and diffusion [47]. It is instructive to observe that the last three patterns are sub-components of a pheromone approach to constructing gradient fields, highlighting both the value of this approach and the need for further development.

Closely related to this work is the application of **SPEM** [89] to facilitate the isolation and integration of **method fragments** [111], illustrated by isolating fragments from Adelfe, CUP, MetaSelf, General Methodology, and SDA, and then recombining them using PASSI.

Finally, engineers of self-organizing systems can take advantage of recent advances in **iterative and incremental development** [9, 77]. It is impossible to anticipate in advance the states accessible to a self-organizing system as it evolves in an open environment. As a result, the line between development and operation inevitably blurs [6]. The system must be specified in terms of desired performance and means of incrementally correcting deficiencies [7], constructing systems that grow and react rather than being constructed and controlled [122]. The need for this perspective is particularly strong in the verification and validation (V&V) of a system. Traditionally, successful completion of V&V is necessary before a system is deployed. Self-organizing systems require mechanisms for "run-time V&V" [11] that can continuously monitor the system's performance as it reorganizes itself in response to unanticipated conditions. It is an open question whether run-time V&V can in fact be done in a fully decentralized manner, or whether some reference to an explicit model of system objectives is necessary. That is, a system can be engineered to organize itself to meet system objectives without carrying a model of those objectives, but it may be the case that it cannot report whether or not it is in fact meeting the objectives unless it has such a model, since the latter task is intrinsically reflective.

6 Conclusion

The engineering of self-organizing software is a challenging domain that has attracted a wide range of creative talent. In spite of the difficulty of the problem and the wide range of approaches, there are consistent themes and well-defined problems to focus future research. As the information universe becomes more distributed and decentralized, the difference between the engineering of self-organizing systems and that of other software will shrink, and the themes that are beginning to manifest themselves in the self-organizing community will be increasingly recognized as staples of software engineering in general.

7 Acknowledgments

This review relies heavily on many colleagues who were kind enough to share their observations on the field, and their own work, with us.[3] We particularly appreciate the

[3] Alphabetically by last name: Bernhard Bauer, Jake Beal, Olivier Buffet, François Charpillet, Jörg Denzinger, Regina Frei, Kurt Geihs, Arnaud Glad, Nicolas Höning,

detailed reviews of the field that several respondents contributed [9, 38, 65, 123, 127, 131, 135, 142]. Even though these reviews are not publicly available, we have borrowed extensively from their ideas and in some cases their wording, and have cited them in order to give appropriate credit. Naturally, we are responsible for how we have combined the ideas that they have so generously shared with us. Think of this exercise as an example of an "open system," in which the components, in this case the contributions of our informants, are allowed to interact in ways that they perhaps did not anticipate. We provide the "infrastructure" for the interaction, and as is often the case in self-organizing systems, the infrastructure makes a great deal of difference in the overall outcome.

In selecting the studies that we cite, we draw heavily on the suggestions of our informants, so our citations should be understood as examples and make no claim to be exhaustive. We look forward to revising this study for publication, and invite the nomination of other work that should be included.

8 References

[1] J. S. Albus. RCS: A Reference Model Architecture for Intelligent Control. *IEEE Computer*, 25(5):56-59, 1992.

[2] J. S. Albus. The NIST Real-time Control System (RCS): an approach to intelligent systems research. *J. Exp. Theor. Artif. Intell.*, 9(2-3):157-174, 1997.

[3] aliCE. aliCE (agents, languages and infrastructures for Complexity Engineering) Home. Bologna, Italy, 2008. http://alice.unibo.it/xwiki/bin/view/aliCE/.

[4] R. Axtell and J. Epstein. Distributed Computation of Economic Equilibria via Bilateral Exchange. Brookings Institution, Washington, DC, 1997.

[5] J. Bachrach, J. Beal, and J. McLurkin. Composable continuous space programs for robotic swarms. *Neural Computing and Applications*, 19(6):825–847, 2010.

[6] L. Baresi, N. Bencomo, B. Cukic, A. Gorla, P. Inverardi, O. Nierstrasz, S. Park, D. Smith, T. Vogel, R. de Lemos, and J. Andersson. Dagstuhl Group C: Process. Dagstuhl, 2010. Available at http://www.dagstuhl.de/Materials/Files/10/10431/10431.SWM12.Slides.ppt.

[7] J. Beal. Functional Blueprints: An Approach to Modularity in Grown Systems. In *Proceedings of the Seventh International Conference on Swarm Intelligence (ANTS 2010)*, 2010.

[8] J. Beal. MIT Proto. MIT, Cambridge, MA, 2010. http://stpg.csail.mit.edu/proto.html.

[9] J. Beal. Software Engineering for Self-Organizing Systems. 2011.

[10] J. Beal and T. F. Knight Jr. Analyzing Composability in a Sparse Encoding Model of Memorization and Association. In *Proceedings of the Seventh IEEE International Conference on Development and Learning (ICDL 2008)*, 2008.

[11] B. Becker, G. Karsai, S. Mankovskii, H. Müller, M. Pezze, W. Schäfer, J. P. S. L. Tahvildari, G. Tamura, N. M. Villegas, and K. Wong. Dagstuhl Group A: Towards

Practical Run-Time V&V (For Self-Adaptive Systems). Dagstuhl, 2010. Available at http://www.dagstuhl.de/Materials/Files/10/10431/10431.SWM10.Slides.ppt.

[12] J. J. Binney, N. J. Dowrick, A. J. Fisher, and M. E. J. Newman. *The Theory of Critical Phenomena - An Introduction to the Renormalization Group*. Oxford, UK, Clarendon Press, 1992.

[13] E. Bonabeau. Agent-based modeling: Methods and techniques for simulating human systems. *Proceedings of the National Academy of Sciences*, 99 (Supplement 3):7280-7287, 2002.

[14] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. New York, Oxford University Press, 1999.

[15] E. Bonabeau, F. Henaux, S. Guérin, D. Snyers, P. Kuntz, and G. Theraulaz. Routing in Telecommunications Networks with "Smart" Ant-Like Agents. In *Proceedings of Second International Workshop on Intelligent Agents for Telecommunications Applications (IATA98)*, Springer, 1998.

[16] L. Bongaerts. *Integration of Scheduling and Control in Holonic Manufacturing Systems*. Thesis at K.U. Leuven, Department of PMA, 1998.

[17] L. Booker. Learning Tactics for Swarming Entities. In *Proceedings of Swarming: Network Enabled C4ISR*, ASD C3I, 2003.

[18] R. A. Brooks. Intelligence Without Representation. *Artificial Intelligence*, 47:139-59, 1991.

[19] S. Brueckner. *Return from the Ant: Synthetic Ecosystems for Manufacturing Control*. Thesis at Humboldt University Berlin, Department of Computer Science, 2000.

[20] S. Brueckner, E. Downs, R. Hilscher, and A. Yinger. Self-Organizing Integration of Competing Reasoners for Information Matching. In *Proceedings of ECOSOA Workshop at SASO 2008*, 2008.

[21] S. Brueckner, S. Hassas, M. Jelasity, and D. Yamins, Editors. *Engineering Self-Organising Systems*. Lecture Notes on Computer Science, Springer, 2007.

[22] S. Brueckner and H. V. D. Parunak. Resource-Aware Exploration of Emergent Dynamics of Simulated Systems. In *Proceedings of Autonomous Agents and Multi-Agent Systems (AAMAS 2003)*, pages 781-788, ACM, 2003.

[23] S. Brueckner and H. V. D. Parunak. Information-Driven Phase Changes in Multi-Agent Coordination. In *Proceedings of Workshop on Engineering Self-Organizing Systems (ESOA, at AAMAS 2005)*, pages 104-119, Springer, 2005.

[24] S. A. Brueckner, G. Di Marzo Serugendo, and D. Hales, Editors. *Engineering Self-Organising Systems*. Lecture Notes in Computer Science, 2006.

[25] S. A. Brueckner, G. Di Marzo Serugendo, A. Karageorgos, and R. Nagpal, Editors. *Engineering Self-Organising Systems*. Lecture Notes in Computer Science, 2005.

[26] H. V. Brussel, J. Wyns, P. Valckenaers, L. Bongaerts, and P. Peeters. Reference Architecture for Holonic Manufacturing Systems: PROSA. *Computers In Industry*, 37(3):255-276, 1998.

[27] S. Camazine, J.-L. Deneubourg, N. R. Franks, J. Sneyd, G. Theraulaz, and E. Bonabeau. *Self-Organization in Biological Systems*. Princeton, NJ, Princeton University Press, 2001.

[28] M. Casadei, M. Viroli, and L. Gardelli. On the Collective Sort Problem for Distributed Tuple Spaces. *Science of Computer Programming*, 74(9):702-722, 2009.

[29] B. H. C. Cheng, R. de Lemos, H. Giese, P. Inverardi, and J. Magee, Editors. *Software Engineering for Self-Adaptive Systems*. Lecture Notes in Computer Science, 5525, Heidelberg, Springer, 2009.

[30] S.-W. Cheng, V. V. Poladian, D. Garlan, and B. Schmerl. Improving Architecture-Based Self-Adaptation through Resource Prediction. In B. H. C. Cheng, R. de Lemos, H. Giese, P. Inverardi, and J. Magee, Editors, *Software Engineering for Self-Adaptive Systems*, vol. 5525, pages 128-145. Springer, Heidelberg, 2009.

[31] V. A. Cicirello and S. F. Smith. Wasp-like Agents for Distributed Factory Coordination. *Journal of Autonomous Agents and Multi-Agent Systems*, 8(3 (May)):237-266, 2004.

[32] S. H. Clearwater, Editor. *Market-Based Control: A Paradigm for Distributed Resource Allocation*. Singapore, World Scientific, 1996.

[33] D. Coore. *Botanical Computing: A Developmental Approach to Generating Inter connect Topologies on an Amorphous Computer*. Thesis at MIT, 1999.

[34] R. Dawkins. *The Selfish Gene*. Oxford University Press, 1976.

[35] T. De Wolf and T. Holvoet. Design patterns for decentralised coordination in self-organising emergent systems. In *Proceedings of the Fourth International Workshop on Engineering Self-Organising Applications (ESOA) at AAMAS 2006*, pages 28-49, Springer, 2007.

[36] J. Denzinger and D. Fuchs. Cooperation of Heterogeneous Provers. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI 1999)*, pages 10-15, Morgan Kaufmann, 1999.

[37] J. Denzinger, M. Fuchs, and M. Fuchs. High Performance ATP Systems by Combining Several AI Methods. In *Proceedings of IJCAI-97*, pages 102-107, Morgan Kaufmann, 1997.

[38] J. Denzinger, H. Kasinger, and B. Bauer. Software Engineering for Self-Organizing Systems. 2011.

[39] G. Di Marzo Serugendo. Robustness and Dependability of Self-Organising Systems – A Safety Engineering Perspective. In *Proceedings of the 11th International Symposium on Stabilization, Safety and Security of Distributed Systems (SSS 2009)*, pages 254–268, Springer, 2009.

[40] G. Di Marzo Serugendo, J. Fitzgerald, and A. Romanovsky. MetaSelf—An Architecture and Development Method for Dependable Self-* Systems. In *Proceedings of the 25th Symposium on Applied Computing (SAC 2010)*, 2010.

[41] G. Di Marzo Serugendo, A. Karageorgos, O. F. Rana, and F. Zambonelli, Editors. *Engineering Self-Organising Systems*. Lecture Notes in Computer Science, 2004.

[42] F. Dötsch, J. Denzinger, H. Kasinger, and B. Bauer. Decentralized Real-time Control of Water Distribution Networks Using Self-organizing Multi-Agent Systems. In *Proceedings of the 4th IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2010)*, pages 223-232, IEEE, 2010.

[43] R. Doursat. The growing canvas of biological development: Multiscale pattern generation on an expanding lattice of gene regulatory networks. *InterJournal: Complex Systems*:1809, 2006.

[44] B. Edmonds and J. J. Bryson. The Insufficiency of Formal Design Methods: The Necessity of an Experimental Approach for the Understanding and Control of Complex

MAS. In *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004)*, pages 938-945, IEEE 2004.

[45] J. M. Epstein. *Generative Social Science*. Princeton, NJ, Princeton University Press, 2006.

[46] F. Flacher and O. Sigaud. Spatial coordination through social potential fields and genetic algorithms. In *Proceedings of the Seventh International Conference on Simulation of Adaptive Behavior (From Animals to Animats)*, MIT Press, 2002.

[47] L. Gardelli, M. Viroli, and A. Omicini. Design Patterns for Self-organizing Multiagent Systems. In *Proceedings of the 5th International Central and Eastern European Conference on Multi-Agent Systems (CEEMAS 2007)*, pages 123-132, Springer, 2007.

[48] J. C. Georgas and R. N. Taylor. Policy-Based Architectural Adaptation Management: Robotics Domain Case Studies. In B. H. C. Cheng, R. de Lemos, H. Giese, P. Inverardi, and J. Magee, Editors, *Software Engineering for Self-Adaptive Systems*, vol. 5525, pages 89-108. Springer, Heidelberg, 2009.

[49] I. Georgiadis, J. Magee, and J. Kramer. Self-organising software architectures for distributed systems. In *Proceedings of the First workshop on Self-healing systems (WOSS '02)*, ACM, 2002.

[50] C. Gershenson. *Design and Control of Self-organizing Systems*. Thesis at Vrije Universiteit Brussel, 2007.

[51] C. Gershenson and F. Heylighen. When Can we Call a System Self-organizing? 2003. http://arxiv.org/pdf/nlin.AO/0303020.

[52] H. Giese, H. Müller, M. Shaw, and R. d. Lemos. Abstracts, Dagstuhl Seminar 10431, Software Engineering for Self-Adaptive Systems. 2010. http://www.dagstuhl.de/Materials/index.en.phtml?10431.

[53] A. Glad, O. Buffet, O. Simonin, and F. Charpillet. Self-Organization of Patrolling-ant Algorithms. In *Proceedings of the International Conference on Self-Adaptive and Self-Organizing Systems (SASO09)*, pages 61-70, 2009.

[54] A. Glad, O. Buffet, O. Simonin, and F. Charpillet. Influence of Different Execution Models on Patrollin Ant Behaviors: from Agents to Robots. In *Proceedings of the Ninth International Conference on Autonomous Agents and Multiagent Systems (AAMAS'10)*, pages 1173-1180, 2010.

[55] A. Glad, O. Simonin, O. Buffet, and F. Charpillet. Theoretical Study of Ant-based Algorithms for Multi-Agent Patrolling. In *Proceedings of the Eighteenth European Conference on Artificial Intelligence (ECAI'08)*, pages 626-630, 2008.

[56] P.-P. Grassé. La Reconstruction du nid et les Coordinations Inter-Individuelles chez *Bellicositermes Natalensis et Cubitermes sp.* La théorie de la Stigmergie: Essai d'interprétation du Comportement des Termites Constructeurs. *Insectes Sociaux*, 6:41-84, 1959.

[57] S. Guerin and D. Kunkle. Emergence of Constraint in Self-organizing Systems. *Nonlinear Dynamics, Psychology, and Life Sciences*, 8(2):131-146, 2004.

[58] A. Hamdi, V. Antoine, N. Monmarché, A. Alimi, and M. Slimane. Artificial Ants for Automatic Classification. In N. Monmarché, F. Guinand, and P. Siarry, Editors, *Artificial Ants: From Collective Intelligence to Real-life Optimization and Beyond*, pages 265-290. John Wiley and Sons, Hoboken, NJ, 2010.

[59] J. Handl, J. Knowles, and M. Dorigo. Ant-based clustering: a comparative study of its relative performance with respect to k-means, average link and 1d-som. TR-IRIDIA-2003-24, IRIDIA, 2003. http://wwwcip.informatik.uni-erlangen.de/~sijuhand/TR-IRIDIA-2003-24.pdf.

[60] W. Heaven, D. Sykes, J. Magee, and J. Kramer. A Case Study in Goal-Driven Architectural Adaptation. In B. H. C. Cheng, R. de Lemos, H. Giese, P. Inverardi, and J. Magee, Editors, *Software Engineering for Self-Adaptive Systems*, vol. 5525, pages 109-127. Springer, Heidelberg, 2009.

[61] M. Heusse, S. Guérin, D. Snyers, and P. Kuntz. Adaptive Agent-Driven Routing and Load Balancing in Communication Networks. *Advances in Complex Systems*, 1:234-257, 1998.

[62] R. Hilscher, S. Brueckner, T. C. Belding, and H. V. D. Parunak. Self-Organizing Information Matching in InformANTS. In *Proceedings of Self-Adaptive and Self-Organizing Systems (SASO07)*, pages 277-280, 2007.

[63] T. Holvoet, D. Weyns, and P. Valckenaers. Delegate MAS Patterns for Large-Scale Distributed Coordination and Control Applications. In *Proceedings of EuroPlop*, 2010.

[64] R. Holzer, H. de Meer, and C. Bettstetter. On Autonomy and Emergence in Self-Organizing Systems. In *Proceedings of Intern. Workshop on Self-Organizing Systems (IWSOS)*, Springer, 2008.

[65] N. Höning. Comments on Software Engineering for Self-Organizing Systems. 2011.

[66] N. Höning and H. La Poutre. Designing comprehensible self-organising systems. In *Proceedings of the Fourth IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2010)*, pages 233-242, IEEE Computer Society, 2010.

[67] H.-M. Huang, K. Pavek, B. Novak, J. Albus, and E. Messina. A Framework For Autonomy Levels For Unmanned Systems (ALFUS). In *Proceedings of AUVSI Unmanned Systems 2005*, 2005.

[68] J. Hudson, J. Denzinger, H. Kasinger, and B. Bauer. Efficiency Testing of Self-Adapting Systems by Learning of Event Sequences. In *Proceedings of the 2nd International Conference on Adaptive and Self-adaptive Systems and Applications (ADAPTIVE 2010)*, pages 200-205, IARIA, 2010.

[69] IBM. An Architectural Blueprint for Autonomic Computing. IBM, 2006. http://www-03.ibm.com/autonomic/pdfs/AC_Blueprint_White_Paper_4th.pdf.

[70] ITEA. Agenda In *Proceedings of the Developing And Testing Autonomy (DATA) Workshop*, International Test and Evaluation Association (ITEA), 2010.

[71] M. A. Janssen, Editor. *Complexity and Ecosystem Management: The Theory and Practice of Multi-Agent Systems*. Cheltenham, UK, Edward Elgar, 2002.

[72] H. Kasinger, B. Bauer, and J. Denzinger. Design Pattern for Self-Organizing Emergent Systems Based on Digital Infochemicals. In *Proceedings of EASe 2009*, pages 45-55, 2009.

[73] J. O. Kephart and D. M. Chase. The Vision of Autonomic Computing. *Computer*, vol. 36, pages 41-50, 2003. Available at http://www.research.ibm.com/autonomic/research/papers/AC_Vision_Computer_Jan_2003.pdf.

[74] J. O. Kephart, T. Hogg, and B. A. Huberman. Dynamics of Computational Ecosystems. *Physics Review*, 40A:404-421, 1989.

[75] A. Koestler. *The Ghost in the Machine*. 1967.

[76] P. Kuntz and P. Layzell. An Ant Clustering Algorithm Applied to Partitioning in VLSI Technology. In *Proceedings of Fourth European Conference on Artificial Life*, pages 417-424, MIT Press, 1997.

[77] C. Larman and V. Basili. Iterative and Incremental Development: A Brief History. *IEEE Computer*, pages 2-11, 2003. Available at http://www.craiglarman.com/wiki/downloads/misc/history-of-iterative-larman-and-basili-ieee-computer.pdf.

[78] M. Lejter and T. Dean. A Framework for the Development of Multiagent Architectures. *IEEE Expert*, 11(December):47-59, 1996.

[79] K. Lerman, A. Martinoli, and A. Galstyan. A Review of Probabilistic Macroscopic Models for Swarm Robotic Systems. In S. E. and S. W., Editors, *Swarm Robotics Workshop: State-of-the-art Survey*, pages 143-152. Springer-Verlag, Berlin, Germany, 2005.

[80] M. Mamei and F. Zambonelli. *Field-based coordination for pervasive multiagent systems*. Springer, 2008.

[81] S. A. a. M. Masoud, A. A. Constrained Motion Control Using Vector Potential Fields. *IEEE Trans. on Systems, Man, and Cybernetics*, 30(3):251-272, 2000.

[82] J. C. Maxwell. On Governors. *Proceedings of the Royal Society of London*, 16:270–283, 1867.

[83] D. Merkle, M. Middendorf, and A. Scheidler. Swarm Controlled Emergence - Designing an Anti-Clustering Ant System. In *Proceedings of IEEE Swarm Intelligence Symposium*, pages 242-249, 2007.

[84] N. Monmarché. On data clustering with artificial ants. In *Proceedings of AAAI-99 & GECCO-99 Workshop on Data Mining with Evolutionary Algorithms: Research Directions*, pages 23-26, 1999.

[85] R. Nagpal. *Programmable Self-Assembly: Constructing Global Shape using Biologically-inspired Local Interactions and Origami Mathematics*. Thesis at MIT, 2001.

[86] M. E. J. Newman. *Networks: An Introduction*. Oxford, UK, Oxford University Press, 2010.

[87] O. Nierstraz, M. Denker, and L. Renggli. Model-Centric, Context-Aware Software Adaptation. In B. H. C. Cheng, R. de Lemos, H. Giese, P. Inverardi, and J. Magee, Editors, *Software Engineering for Self-Adaptive Systems*, vol. 5525, pages 128-145. Springer, Heidelberg, 2009.

[88] J. J. Odell, H. V. D. Parunak, S. Brueckner, and J. Sauter. Temporal Aspects of Dynamic Role Assignment. In *Proceedings of Workshop on Agent-Oriented Software Engineering (AOSE03) at AAMAS03*, pages 201-213, Springer, 2003.

[89] OMG. Software & Systems Process Engineering Meta-Model Specification. Object Management Group, 2008. http://www.omg.org/spec/SPEM/2.0/PDF.

[90] A. Omicini. Towards a notion of agent coordination context. In D. Marinescu and C. Lee, Editors, *Process Coordination and Ubiquitous Computing*, pages 187–200. CRC Press, 2002.

[91] A. Omicini, A. Ricci, M. Viroli, C. Castelfranchi, and L. Tummolini. Coordination Artifacts: Environment-based Coordination for Intelligent Agents. In *Proceedings of 3rd*

international Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004), pages 286-293, ACM, 2004.

[92] A. Omicini and F. Zambonelli. Coordination for Internet application development. *Autonomous Agents and Multi-Agent Systems*, 23(3):251-269, 1999.

[93] H. V. D. Parunak. 'Go to the Ant': Engineering Principles from Natural Agent Systems. *Annals of Operations Research*, 75:69-101, 1997.

[94] H. V. D. Parunak. From Chaos to Commerce: Practical Issues and Research Opportunities in the Nonlinear Dynamics of Decentralized Manufacturing Systems. In *Proceedings of Second International Workshop on Intelligent Manufacturing Systems*, pages k15-k25, Katholieke Universiteit Leuven, 1999.

[95] H. V. D. Parunak. A Survey of Environments and Mechanisms for Human-Human Stigmergy. In D. Weyns, F. Michel, and H. V. D. Parunak, Editors, *Proceedings of E4MAS 2005*, vol. LNAI 3830, *Lecture Notes on AI*, pages 163-186. Springer, 2006.

[96] H. V. D. Parunak. Real-Time Agent Characterization and Prediction. In *Proceedings of International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'07), Industrial Track*, pages 1421-1428, ACM, 2007.

[97] H. V. D. Parunak. Generation and Analysis of Multiple Futures with Swarming Agents. In *Proceedings of the International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2010)*, pages 1549-1550, IFAAMAS, 2010.

[98] H. V. D. Parunak, T. C. Belding, and S. A. Brueckner. Prediction Horizons in Agent Models. In *Proceedings of Engineering Environment-Mediated Multiagent Systems (Satellite Conference at ECCS 2007)*, pages 88-102, Springer, 2008.

[99] H. V. D. Parunak and S. Brueckner. Ant-Like Missionaries and Cannibals: Synthetic Pheromones for Distributed Motion Control. In *Proceedings of Fourth International Conference on Autonomous Agents (Agents 2000)*, pages 467-474, 2000.

[100] H. V. D. Parunak and S. Brueckner. Entropy and Self-Organization in Multi-Agent Systems. In *Proceedings of The Fifth International Conference on Autonomous Agents (Agents 2001)*, pages 124-130, ACM, 2001.

[101] H. V. D. Parunak, S. Brueckner, and R. Savit. Universality in Multi-Agent Systems. In *Proceedings of Third International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2004)*, pages 930-937, ACM, 2004.

[102] H. V. D. Parunak, S. Brueckner, D. Weyns, T. Holvoet, and P. Valckenaers. E Pluribus Unum: Polyagent and Delegate MAS Architectures. In *Proceedings of Eighth International Workshop on Multi-Agent-Based Simulation (MABS07)*, pages 36-51, Springer, 2007.

[103] H. V. D. Parunak and S. A. Brueckner. Engineering Swarming Systems. In F. Bergenti, M.-P. Gleizes, and F. Zambonelli, Editors, *Methodologies and Software Engineering for Agent Systems*, pages 341-376. Kluwer, 2004.

[104] H. V. D. Parunak, R. Rohwer, T. C. Belding, and S. Brueckner. Dynamic Decentralized Any-Time Hierarchical Clustering. In *Proceedings of Proceedings of the Fourth International Workshop on Engineering Self-Organizing Systems (ESOA'06)*, Springer, 2006.

[105] H. V. D. Parunak, J. Sauter, and S. J. Clark. Toward the Specification and Design of Industrial Synthetic Ecosystems. In M. P. Singh, A. Rao, and M. J. Wooldridge, Editors,

Intelligent Agents IV: Agent Theories, Architectures, and Languages, *Lecture Notes in Artificial Intelligence 1365*, pages 45-59. Springer, Berlin, 1998.

[106] H. V. D. Parunak, A. C. Ward, M. Fleischer, and J. A. Sauter. The RAPPID Project: Symbiosis between Industrial Requirements and MAS Research. *Journal of Autonomous Agents and Multi-Agent Systems*, 2:2 (June):111-140, 1999.

[107] H. V. D. Parunak, A. C. Ward, and J. A. Sauter. The MarCon Algorithm: A Systematic Market Approach to Distributed Constraint Problems. *AI-EDAM: Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 13(3):217-234, 1999.

[108] D. Payton, Daily, M., Estowski, R., Howard, M., and Lee, C. Pheromone Robotics. *Journal Autonomous Robots*, 11(3):319-324, 2001.

[109] P. Peeters, H. Van Brussel, P. Valckenaers, J. Wyns, L. Bongaerts, M. Kollingbaum, and T. Heikkila. Pheromone based emergent shop floor control system for flexible flow shops *Artificial Intelligence in Engineering*, 15(4 (Oct)):343-352, 2001.

[110] P. Prusinkiewicz and A. Lindenmayer. *The Algorithmic Beauty of Plants*. New York, NY, Springer-Verlag, 1990.

[111] M. Puviani, G. Di Marzo Serugendo, R. Frei, and G. Cabri. A method fragments approach to methodologies for engineering self-organising systems. *ACM Transactions on Autonomous and Adaptive Systems*, forthcoming, 2011.

[112] H. G. Rice. Classes of Recursively Enumerable Sets and Their Decision Problems. *Trans. Amer. Math. Soc.*, 74:358-366, 1953.

[113] S. Ricketts. Holonic Manufacturing Systems. 1996. Web

[114] M. Salehie and L. Tahvildari. Self-adaptive software: landscape and research challenges. *ACM Transactions on Autonomic and Autonomic Systems (TAAS)*, 4(2):1-42, 2009.

[115] SAPERE. EU Sapere Project (Self-Aware Pervasive Service Ecosystems). 2011. http://www.sapere-project.eu/.

[116] SASO. Self-Adaptive and Self-Organizing Systems. 2011. http://www.saso-conference.org/.

[117] J. A. Sauter, R. Matthews, H. V. D. Parunak, and S. Brueckner. Evolving Adaptive Pheromone Path Planning Mechanisms. In *Proceedings of Autonomous Agents and Multi-Agent Systems (AAMAS02)*, pages 434-440, ACM, 2002.

[118] J. A. Sauter, R. Matthews, H. V. D. Parunak, and S. A. Brueckner. Performance of Digital Pheromones for Swarming Vehicle Control. In *Proceedings of Fourth International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 903-910, ACM, 2005.

[119] J. A. Sauter, R. Matthews, H. V. D. Parunak, and S. A. Brueckner. Effectiveness of Digital Pheromones Controlling Swarming Vehicles in Military Scenarios. *Journal of Aerospace Computing, Information, and Communication*, 4(5):753-769, 2007.

[120] J. A. Sauter, R. S. Matthews, J. S. Robinson, J. Moody, and S. P. Riddle. Swarming Unmanned Air and Ground Systems for Surveillance and Base Protection. In *Proceedings of AIAA Infotech@Aerospace 2009 Conference*, AIAA, 2009.

[121] R. Savit, S. A. Brueckner, H. V. D. Parunak, and J. Sauter. Phase Structure of Resource Allocation Games. *Physics Letters A*, 311:359-364, 2002.

[122] I. Scholtes. *Harnessing Complex Structures and Collective Dynamics in Large Networked Computing Systems*. Thesis at University of Trier, 2010.

[123] I. Scholtes. Thoughts on Engineering Self-Organizing Systems. 2011.

[124] I. Scholtes, J. Botev, A. Höhfeld, H. Schloss, and M. Esch. Awareness-Driven Phase Transitions in Very Large Scale Distributed Systems. In *Proceedings of the Second IEEE International Conferences on Self-Adaptive and Self-Organizing Systems (SASO)*, IEEE, 2008.

[125] P. Sengers. Designing comprehensible agents. In *Proceedings of Sixteenth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1227–1232, Lawrence Erlbaum, 1999.

[126] H. A. Simon. *The Sciences of the Artificial*. Cambridge, MA, MIT Press, 1969.

[127] O. Simonin, F. Charpillet, O. Buffet, and A. Glad. Engineering Self-Organizing Systems. 2011.

[128] O. Simonin, F. Charpillet, and E. Thierry. Collective Construction of Numerical Potential Fields for the Foraging Problem. *ACM TAAS*, (forthcoming), 2011.

[129] A. Spicher and O. Michel. Declarative modeling of a neurulation-like process. *BioSystems*, 87:281–288, 2006.

[130] A. Tyrrell, G. Auer, and C. Bettstetter. Biologically Inspired Synchronization for Wireless Networks. In F. Dressler and I. Carreras, Editors, *Advances in Biologically Inspired Information Systems: Models, Methods, and Tools*, *Studies in Computational Intelligence*, pages 47-62. Springer, 2007.

[131] P. Valckenaers. Self-organizing systems with emergent behavior. 2011.

[132] P. Valckenaers, H. V. Brussel, K. Hadeli, O. Bochmann, B. S. Germain, and C. Zamfirescu. On the design of emergent systems: an investigation of integration and interoperability issues. *Engineering Applications of Artificial Intelligence*, 16:377-393, 2003.

[133] P. Valckenaers and H. Van Brussel. Holonic manufacturing execution systems *CIRP Annals of Manufacturing Technology*, 54(1):427-432, 2005.

[134] M. Viroli and M. Casadei. Biochemical Tuple Spaces for Self-Organising Coordination. In *Proceedings of Coordination Languages and Models*, pages 143-162, Springer, 2009.

[135] M. Viroli and A. Omicini. The "Self-organising Coordination" Paradigm in the Software Engineering of SOS. 2011.

[136] M. Viroli, A. Ricci, F. Zambonelli, T. Holvoet, and K. Schelfthout. Infrastructures for the environment of multiagent systems. *Journal of Autonomous Agents and Multi-Agent Systems*, 13, 2007.

[137] M. Viroli and F. Zambonelli. A Biochemical Approach to Adaptive Service Ecosystems. *Information Sciences*, 180(10):1876-1892, 2010.

[138] B. Walsham. *Simplified and Optimised Ant Sort for Complex Problems: Document Classification*. Thesis at Monash University, Department of School of Computer Science and Software Engineering, 2003.

[139] P. Wegner. Why Interaction is More Powerful than Algorithms. *Communications of the ACM*, 40(5 (May)):81-91, 1997.

[140] J. Werfel. *Anthills built to order: Automating construction with artificial swarms*. Thesis at MIT, Department of CSAIL, 2006.

[141] D. Weyns. *Architecture-based design of multi-agent systems*. Springer 2010.

[142] D. Weyns. Software Engineering for Self-Organizing Systems. 2011.

[143] D. Weyns, N. Boucke, and T. Holvoet. A Field-Based Versus a Protocol-Based Approach for Adaptive Task Assignment. *Journal on Autonomous Agents and Multi-Agent Systems*, 17(2):288-319, 2008.

[144] D. Weyns, S. Dustdar, H. Giese, K. Göschka, V. Grassi, J. Kramer, S. Malek, R. Mirandola, C. Prehofer, R. Schlichting, B. Schmerl, and J. Wuttke. Dagstuhl Group D: From Centralized to Decentralized Control in Self-Adaptation. Dagstuhl, 2010. Available at http://www.dagstuhl.de/Materials/Files/10/10431/10431.SWM9.Slides.ppt.

[145] D. Weyns, S. Malek, J. Andersson, and B. Schmerl. Call for Papers, Special Issue on "State of the Art in Self-Adaptive Software Systems," Journal of Systems and Software (JSS). 2011. Available at http://www.elsevierscitech.com/cfp/CFP-JSS-special-issue-2010.pdf.

[146] N. Wiener. *Cybernetics*. Cambridge, MA, MIT Press, 1948.

[147] T. D. Wolf and T. Holvoet. Towards a Methodology for Engineering Self-Organising Emergent Systems. In *Proceedings of the 2005 conference on Self-Organization and Autonomic Informatics*, pages 18-34, IOS Press, 2005.

[148] T. D. Wolf, T. Holvoet, and G. Samaey. Engineering Self-Organising Emergent Systems with Simulation-based Scientific Analysis. In *Proceedings of the Fourth International Workshop on Engineering Self-Organising Applications*, pages 146-160, 2005.

Programming Multi-Agent Systems

Mehdi Dastani

Utrecht University
The Netherlands
mehdi@cs.uu.nl

Abstract. With the significant advances in the area of autonomous agents and multi-agent systems in the last decade, promising technologies for the development and engineering of multi-agent systems have emerged. The result is a variety of programming languages, execution platforms, and tools that facilitate the development and engineering of multi-agent systems. This paper provides an overview of the multi-agent programming research field by focusing on the aim and characteristics of various multi-agent programming languages and development tools. This overview is complemented with a discussion on the current trends and challenges in this research area.

1 Introduction

Multi-agent systems consist of a set of autonomous and interacting computing systems called agents [92, 42, 97]. Agents are assumed to be autonomous in the sense that they can decide for themselves which actions to perform in order to achieve their individual objectives. Agents interact either with each other through communication or with their environments through their sensors and actuators. As agents may have different architectures being developed by different designers, the global properties of multi-agent systems can be guaranteed if the behaviour of individual agents can be controlled and coordinated. The coordination of agents' behaviours can be done either endogenously or exogenously [2]. In an endogenous approach the coordination models reside within the agents while in an exogenous approach the coordination models reside outside the agents. In particular, in an endogenous approach agents are internally designed to behave in a coordinated manner, while in an exogenous approach agents are coordinated by means of an external entity that controls their behaviours.

Multi-agent systems constitute a promising software engineering paradigm for the development of distributed intelligent systems. The agent-oriented software engineering paradigm provides cognitive and social concepts and abstractions in terms of which software systems can be specified, designed, and implemented. Examples of such concepts and abstractions are beliefs, goals, plans, actions, events, roles, organisational rules and structures, communication, norms and sanctions. In order to develop multi-agent systems in an effective and systematic way, different analysis and design methodologies [10], specification languages [74, 22, 62, 31], programming languages and development tools [14, 15, 13, 94, 69,

50, 57, 17, 78, 60, 43, 45, 79] have been proposed. While most agent-oriented analysis and design methodologies assist system developers to specify and design system architectures in terms of agent concepts and abstractions, the proposed multi-agent programming languages and development tools aim at providing programming constructs and operations to facilitate direct, explicit, and effective implementation of concepts and abstractions involved in multi-agent systems. This is the main challenge of the multi-agent programming research field.

Engineering multi-agent systems requires engineering three different types of entities: individual agents, multi-agent organisations, and multi-agent environments. Although ontological differences between these entities and their implications for programming multi-agent systems have been emphasized during the early ProMAS technical fora [29, 28], the main focus of the multi-agent programming community has been on engineering and development of individual agents. Although multi-agent organisations and environments have been active research areas for many years, resulting in a variety of proposals and models, the need for programming languages that support the implementation of multi-agent organisations and environments has only recently been recognized.

This paper starts with a discussion on the aims and objectives of the multi-agent programming research field from a software engineering point of view. In particular, it presents concepts and abstractions for which multi-agent programming languages aim at providing programming constructs to implement them. The paper presents then an overview of the state of the art in multi-agent programming by focusing on the aims and characteristics of some existing multi-agent programming languages. Of course, there are too many programming languages and frameworks to mention in this paper. We chose programming languages for which an interpreter and an execution platform have been developed. Subsequently, the current trends in this research field will be explained and discussed by means of recent foci and developments. The paper finalizes with a discussion on issues that are currently challenging the multi-agent programming research field.

2 Aims and Objectives

Multi-agent systems can be seen as a development in software engineering which has resulted in a new software development paradigm. Multi-agent systems provide high-level concepts and abstractions to model and develop distributed intelligent systems. Like other software development paradigms, multi-agent systems cover different software engineering phases such as requirement, specification, design, implementation, and testing. Various multi-agent system software methodologies have been proposed (e.g.,Gaia [98], Promethose [67], Tropos [20], INGENIAS [46], and others [10]), each focusing on specific phases of the software development process. For example, Gaia focuses mainly on the analysis and design phases, while Prometheus covers the implementation phase as well. Existing multi-agent system software development methodologies propose concepts and abstractions such as beliefs, goals, plans, events, roles, interaction,

agents, environment, organisation rules, norms, permission, responsibility and capability.

The main aim of the multi-agent programming research field from the software engineering perspective is to propose programming languages that can facilitate direct and effective implementations of multi-agent systems. In particular, the aim of this research field in the context of software engineering is to provide programming constructs that support the implementations of the multi-agent system architectures. In this sense, one can see a multi-agent programming language as a computational specification language for implementing a certain class of multi-agent system architectures, making programming languages dependent on the development methodologies. As different multi-agent development methodologies propose different abstractions and architectures, one may argue that a standard multi-agent programming language cannot emerge as long as different multi-agent system development methodologies do not converge.

Based on concepts and abstractions originated from multi-agent system development methodologies, programming languages for multi-agent systems can be characterised along different dimensions. First, the focus of multi-agent programming languages can be on individual agents, multi-agent organisations, multi-agent environments, or their combinations. Programming languages focussing on individual agents are concerned with issues such as autonomy of agents, reactive behaviours, social awareness, reasoning about norms and organisations, communication and interaction with other agents, and capabilities to sense and act in a shared environment. In order to facilitate the implementation of multi-agent organisations, programming languages should support the implementation of social and organisational issues such as norms (obligation, prohibition, permission) that should be respected or followed by agents, sanctions that should be imposed if norms are violated, roles that the agents can play, delegation of tasks and responsibilities, and the synchronisation of agents' actions. Of course, multi-agent organisations can be implemented either endogenously or exogenously, i.e., either individual agents are implemented in terms of social and organisational concepts, or organisations are implemented as computational entities outside agents controlling their behaviours. Finally, programming languages that support the implementation of multi-agent environments need to provide programming constructs to implement sense and act abilities of agents, tools, artifacts, services, and resources that can be used by agents.

Second, multi-agent programming languages can be characterised based on the language style and their formal or practical foundations. In particular, multi-agent programming languages can be declarative, imperative, or a combination of them. Some multi-agent programming languages are extensions of existing programming languages such as Java or Prolog, while others combine these languages. Declarative languages are often used to represent and reason with concepts such as beliefs, goals, norms, and actions, while imperative languages are often used to implement tasks, services, and processes. Declarative and imperative languages are used to implement various aspects of individual agents, multi-agent organisations, and multi-agent environments. Some multi-agent pro-

gramming languages are abstract and have been proposed as theoretical contribution, while others come with their corresponding development tools and execution platforms. Finally, some multi-agent programming languages come with formal and computational semantics, an implemented interpreter, or both. The existence of formal semantics for multi-agent programming languages is essential for a better understanding of the programming constructs and the verification of multi-agent programs. Without a formal semantics one cannot guarantee the correctness of programs.

Third, multi-agent programming languages can be analysed by means of general programming principles they respect and support. Examples of such principles are modularity, encapsulation, reuse, separation of concerns, recursion, abstraction, exception handling facilities, and support for legacy codes. Of course, the very concept of agent itself supports some of these principles such as encapsulation and reuse. Similarly, the idea of implementing environments and organisations separately support the separation of concerns principle. Multi-agent programming languages can be used in a more efficient and effective manner when they support such principle at different levels. For example, at the individual agent level modularity can be used to support the implementation of different functionalities and roles, recursion can be used to implement complex plans, and exception handling can be used to implement plan failure operations.

Finally, multi-agent programming languages can be evaluated in terms of the functionalities provided by their corresponding integrated development environments. An integrated development environment supports the development of multi-agent programs by means of functionalities such as editing tools allowing easy browsing of codes, debugging tools that help to localize errors and anomalies, and automatic testing tools allowing the automatic generation of test cases for specific part of the programs. The main difficulty for such an integrated development environment is the distributed nature of multi-agent programs, e.g., how to browse through a program that is distributed by means of agents, modules, environment, and organisation programs. Debugging is even harder as it is not clear how to debug one single agent when the execution of the agent depends on the execution of other agent programs, the environment program, and the organisation program.

3 Abstractions in Multi-Agent Programming

In this section, we present key concepts and abstractions that need to be addressed when programming multi-agent systems. The following subsections reflect the highest level of abstraction in multi-agent systems: agents, organisations, and environments. Each subsection discusses the concepts and further abstractions.

3.1 Individual Agents

The focus of most multi-agent programming languages has been on programming individual agents. In these works, multi-agent programs are considered as being composed of a set of individual agent programs that are executed concurrently.

An essential characteristic of individual agents is their autonomy. Without getting into the exact nature of autonomy, we consider an agent as autonomous if it has a decision making component that governs its decisions based on its informational (beliefs/distributions/knowledge) and motivational (desires/utilities/preferences) attitudes. One can argue that any computational system that interacts with other systems (e.g., other agents or environment) can be seen as autonomous, at least from an external point of view. However, the development of an autonomous agent can also be considered from an internal point of view. Such a view requires an explicit decision making component that can be specified, designed, implemented in terms of informational and motivational attitudes. The decision making component should allow a programmer to implement different issues related to an agent's decisions such as decision strategies, resolving decision conflicts, and rationality of decisions. In this sense, programming languages that support the implementation of autonomous agents should provide programming constructs to support the implementation of decision concepts (informational and motivational attitudes) as well as issues related to an agent's decisions.

POMDP [86], BDI [74], and a combination of both [64] can be used as decision models for autonomous agents. POMDP is a quantitative framework that can be used to model a sequential decision process in terms of actions, states, transition probability, observation probabilities, and reward function. In order to determine an agent's decisions, its corresponding POMDP (the agent's decision model) should be solved. However, solving POMDP's are in general computationally intractable such that approximate methods are often proposed to solve POMDP's. Moreover, POMDP's are not suitable to model agents that have complex goals and need to interact with dynamically changing environments [76]. These problems make POMDP not a plausible model to be integrated in a programming language for individual agents.

In contrast, the BDI model has been considered as a qualitative decision model that explains an agent's rational decision in terms of the agent's information about the current state of the world (Beliefs), the states the agent wants to achieve (Desire), and its commitments to already made choices (intentions). The BDI model has proven to be an efficient model for reactive planning and for the agents that have complex goals interacting with highly dynamic environments. The existing BDI-based agent programming languages provide constructs to implement an agent's beliefs, goals, and conditional plans. The conditions that are assigned to plans are specified in terms of beliefs and goals such that a plan can be decided/selected if its belief and goal conditions are satisfied, i.e., an agent can decide a plan if the agent believes the belief condition of the plan and has a goal that entails the goal condition of the plan. The reasoning engine of the BDI agents is often a process that continuously decides a plan to execute. In

some BDI-based programming languages the choice for a plan does also depend on the agent's commitments in the sense that plans are selected if there are no selected plans that aim at achieving a conflicting goal.

Another characteristic of individual agents is their reactive behaviours [63]. The implementation of reactive agents requires an event handling mechanism that generates reactions to the received events. There are many types of events such as messages that are received from other agents, information that are originated from the environment, and the information about the internal working of an agents (e.g., the failure of a plan). Programming languages that support the implementation of reactive agents provide constructs to implement conditional plans where the plan's conditions are defined in terms of events. The reasoning engine of reactive agents continuously check if plans can be selected based on the received events. It should be noted that there is an essential difference between events and goals. In principle, an event causes the generation of a plan and, as soon as the plan is generated, the event is deleted and considered as being processed. Goals are similar to events in the sense that they cause the generation of plans. However, and in contrast to events, goals are not dropped after they have caused the generation of plans. An achieve goal is, for example, dropped if the state denoted by it is achieved, i.e., if the agent believes that the state denoted by the goal is achieved. The relation between beliefs and goals is an essential characteristic of BDI agents which is formulated by means of rationality axioms in the BDI logics [75, 22]. It should also be noted that autonomy and reactive behaviours are two different characteristics and that agents can be both autonomous and reactive, be autonomous without being reactive, or vice versa.

3.2 Multi-Agent Environment

Soon after the emergence of the first generation of the interpreters of agent-oriented programming languages, the need for the implementation of shared environments with which agent programs can interact became apparent. An immediate solution was to model an environment as an external software component with which individual agents could interact. In most multi-agent programming frameworks the environments became simply a software component that were implemented in the same programming language as that of the interpreter of the agent-oriented programming language (e.g., Jave or C++). The state of the software component were considered as the state of the environment while the methods that allow the interaction with the software component were used to implement the effect of the actions that agents could perform in the environment.

One of the first overview papers in the field of multi-agent system environments showed that the concept of environment was originally used with different meanings causing confusions about the exact nature of this abstraction [93, 68]. As argued in this overview paper, some researchers considered multi-agent system environments as equivalent of infrastructures such as message transport system and other infrastructural tools, for example, brokers and management tools, while other researchers considered multi-agent system environments as encapsulating resources, services, and objects. Research on multi-agent system

environments starts receiving popularity by considering environment as a first-class abstraction in multi-agent systems having its own characteristic state and processes.

A multi-agent system environment in [93] is claimed to be used for various purposes, for example to facilitate and coordinate agents' interactions by means of exchanging information through it (blackboard architectures and tuple spaces)[1], or to provide agents the sense and act abilities in order to observe and modify the environment's state, respectively. For example, an environment can provide artifacts or services to allow agents managing their coordination or exchanging information. An environment can also provide various sense and act modalities such as blocking and non-blocking sense operations, event broadcasting, event subscription mechanisms, and synchronous or asynchronous actions.

The implementation of environments requires therefore dedicated programming languages that allow direct and effective implementations of its related concepts and abstractions. Like the development in agent-oriented programming languages, one may expect typical architectures for multi-agent environments. Such architectures would suggest specific concepts, concerns, or components that often need to be implemented when developing a multi-agent system environment. In particular, a dedicated environment programming language should provide programming constructs to implement resources, services, artifacts, processes, several sense and action types and mechanisms. The A&A model [66] has been proposed as a generic paradigm for modelling environments. In the A&A paradigm, an application is composed of agents as well as the so-called artifacts. An implementation of the A&A model is available in form of the distributed architecture and middleware infrastructure CArtAgO [77]. Such an environment architecture consists of a dynamic set of artifacts each of which encapsulates some functions designed by the environment developer.

3.3 Multi-Agent Organisation

The overall objectives of multi-agent systems can be guaranteed by regulating and organising the behaviour of individual agents and their interactions. This can be done either endogenously by integrating the organisation mechanism within the agents themselves, or exogenously by designing the organisation mechanism outside the agents, or a combination of both. An endogenous organisation implies that agents are internally designed to follow, for example, specific interaction protocols, norms or organisational rules, while in an exogenous approach agents are coordinated by means of external artifacts that control the agents' actions according to some interaction protocols, norms, or organisational rules. Generally speaking endogenous coordination mechanisms can be used for the development of closed multi-agent systems where the structure and specification of agents and environments are decided at the design time while exogenous coordination mechanisms can be used for open multi-agent systems where individual

[1] In contrast to direct communication by means of send and receive messages, a shared environment can be used to communicate indirectly by reading and writing information from/to it.

agents may dynamically enter and leave the system. However, from the software engineering perspective an exogenous organisation can also be effective for the development of closed multi-agent systems since such an approach supports the separation of concerns and encapsulation principles.

There have been various proposals for regulating and organising the behaviours of individual agents. Some of these proposals advocate the use of coordination artifacts that are specified in terms of low-level coordination concepts such as synchronization [3, 26]. Other approaches are motivated by organisational models, normative systems, or electronic institutions [41, 40, 47, 55, 56, 30]. In these approaches, the behaviours of individual agents are regulated by means of norms and organisational rules that are either used by individual agents to decide how to behave, or being enforced or regimented through monitoring and sanctioning mechanisms. In these approaches, the social and normative perspective is conceived as a way to make the development and maintenance of multi-agent systems easier to manage. A plethora of social concepts (e.g., roles, groups, social structures, organisations, institutions, norms) has been introduced in multi-agent system methodologies (e.g. Gaia [98]), models (e.g. OperA [37]), specification and modelling languages (e.g. S-MOISE+ [53] and ISLANDER [40]) and computational frameworks (e.g. AMELI [41]).

The implementation of organisations requires programming languages that provides programming constructs to implement social and organisational concepts and abstractions. In particular, the implementation of endogenous mechanisms implies that the agent programming languages provide constructs to allow the representation and reasoning about norms, sanctions, and organisational rules. Such constructs should allow multi-agent programmers to implement agents that make their decisions not only based on their individual goals and beliefs, but also based on the existing norms, sanctions, and other organisational rules. The idea is that individual agents can be implemented in terms of cognitive and social abstractions such that their behaviours are decided upon reasoning about such abstractions. The implementation of exogenous mechanisms requires abilities to monitor and control the behaviours of individual agents. The idea is to have an external organisation software entity that is able to monitor and control the behaviour of individual agents. The question is what should be monitored and how the agents' behaviours can be influenced. As the internals of individual agents cannot be assumed in general, their external behaviours (i.e., communication and interaction with the environment) are the only controllable entities. The organisation software entity can thus observe agents' external behaviour and determine what needs to be done. For example, if the organisation specification disallows certain agents to interact, then the organisation software should be able to block or respond to such interactions. This suggests that the agents' actions (e.g., communication, environment actions including sense actions) should be processed and managed through the external organisation software, i.e., the organisation software intermediates the interaction between agents as well as the interaction between agents and the environment.

4 The State of the Art in Multi-Agent Programming

In this section, we provide an overview of some of the existing multi-agent programming languages together with their development and execution platforms. This overview is by no means complete and does not cover some related and relevant multi-agent programming proposals. The programming frameworks in this overview are chosen because they illustrate different ways to program (some of) the abstractions discussed in previous section, have a development and execution platform, and of course, because of the author's familiarity with the frameworks. Other multi-agent programming frameworks can be found in [14, 15]. This overview will be structured along the focus of the programming languages for individual agents, multi-agent environments, and multi-agent organisations. The programming languages and their corresponding development and execution platforms will be discussed in terms of the concepts explained in section 2.

4.1 Programming Languages for Individual Agents

One of the earliest agent-oriented programming languages is `AGENT-0` [82]. In his seminal paper, Shoham proposes to implement agents in terms of mental components such as beliefs, commitments, capabilities and actions. An agent program in `AGENT-0` consists of an initial belief base, a set of capabilities, a set of commitment rules, together with a repertoire of private actions. Agents can perform different types of actions such as communication, private, conditional and unconditional actions. Agents enter into new commitments by means of commitment rules. A commitment rule consists of conditions on an agent's mental state and the incoming messages. The application of a commitment rule generates a commitment consisting of an action together with the agent identifier towards whom the commitment is made. In fact, the commitments define the actions that an agent have to perform. The execution of an agent is a continuous cyclic process. At each cycle, the received messages are processed, commitments are generated, and actions are performed. `AGENT-0` is undoubtedly one of the first attempts to design an agent programming language that supports the implementation of autonomous agents, i.e., agents that decide actions based on their mental states. However, as indicated in the discussion section of this seminal paper, the state of an `AGENT-0` agent lacks motivational attitudes such as utility, desires, goals, or preferences such that an agent's decisions are based only on events and messages rather than the agent's motivational attitude. Strictly speaking one can therefore argue that the decision behaviour of an agent implemented in `AGENT-0` is not in accordance with the rational decision theories.

Since `AGENT-0` various agent-oriented programming languages have been proposed that extend `AGENT-0` with a larger repertoire of agent concepts and abstractions. The aim of these programming languages is to support the implementation of multi-agent systems, although most of them do not support the implementation of organisational abstractions. Some of these agent-oriented programming languages have an imperative programming style as they extend Java with

agent concepts and abstractions, some languages have a declarative programming style as they extend logic programming languages, and yet other programming languages combine both imperative and declarative styles by integrating for example Java and Prolog. The programming languages that are based on Java have no explicit formal semantics. In the following, we give a brief overview of some of these programming languages.

Imperative Style Agent-Oriented Programming Languages Jade (Java Agent DEvelopment framework) [9] extends Java with a set of multi-agent concepts and abstractions. An agent is created by extending a predefined Jade agent class and redefining its `setup` method. After an agent is created, it will receive an identifier and be registered with the agent management system (a Jade built-in service). The agent is then put in the active state and its `setup` method is executed. The `setup` method is therefore the point where any agent activity should start. Jade agents are behaviour-based in the sense that they can create and execute behaviours. A behaviour can be created by extending the Jade behaviour class via a special construct that adds behaviours (initially in the `setup` method). The created behaviours are added to a behaviour pool. Behaviours are selected for execution from this pool based on a scheduler that constitutes the execution model of the Jade agents. Agents are executed concurrently as different pre-emptive Java threads. The Jade programming framework is developed for practical and industrial applications and comes with a development environment providing a set of graphical tools that support monitoring, logging, and debugging multi-agent programs. The Jade platform is based on a middleware that facilitates the development of distributed multi-agent applications based on a peer-to-peer communication architecture. The Jade execution platform is distributed in the sense that it can run over multiple machines while seen as a whole from the outside world. The Jade platform implements the basic services and infrastructure of a distributed multi-agent application. It supports agent life-cycle, agent mobility, and agent security, and provides services such as white and yellow-pages that can be used by the agents to register their services and search for each other.

Jadex [69] builds on Jade and extend it with programming constructs to implement BDI concepts such as beliefs, goals, plans, and events. It uses XML notation to define and declare an agent's BDI ingredients and Java constructs to implement the agent's plans. Jack [94] extends Java with programming constructs to implement BDI concepts. In both Jack and Jadex a number of syntactic constructs are added to Java to allow programmers to declare beliefsets, to post events, and to select and execute plans. The execution of agent programs in both languages are motivated by the classical sense-reason-act cycle, i.e., processing events, selecting relevant and applicable plans, and execute applicable plans. Beliefs and goals in Jack and Jadex have no logical semantics such that an agent cannot reason about its beliefs and goals. A consequence of this is that a Jack or Jadex agent is not able to generate plans that can contribute to the achievement of its goals, but not necessarily achieve them. Moreover, the con-

sistency and the rational balance of an agent's state in Jack and Jadex, as far as they are defined, is left to agent programmers, i.e., the agent programmer is responsible to make sure that state updates preserve the state consistency and that the rational balance (e.g., between beliefs and goals) is maintained. In these programming languages, an agent's goal is not automatically dropped because it is derivable from the agent's beliefs. Jadex provides a programming construct to implement non-interleaving execution of plans.

Declarative Style Agent-Oriented Programming Languages KGP [57, 17, 78] is based on a model of agency characterized by a set of modules. The model has an internal state module consisting of a collection of knowledge bases, the current agent's goals and plans. The knowledge bases represent different types of knowledge such as the agent's knowledge about observed facts, actions, and communication, but also knowledge to be used for planning, goal decision, reactive behaviour, and temporal reasoning. The KGP agent model includes also a module consisting of a set of capabilities such as planning, reactivity, temporal reasoning, and reasoning about goals. These capabilities are specified by means of abductive logic programming or logic programming with priorities. Another KGP module contains a set of transitions to change the agent's internal state. Each transition performs one or more capabilities, which in turn use different knowledge bases, in order to determine the next state of the agent. Finally, the KGP model has a module, called cyclic theory, that determines which transition should be performed at each moment of time. The KGP model is based on propositional language.

Minerva [60] aims at specifying an agent's state and its dynamics. A Minerva agent consists of a set of specialized sub-agents manipulating a common knowledge base, where sub-agents (i.e., planner, scheduler, learner, etc.) evaluate and manipulate the knowledge base. These subagents are assumed to be implemented in arbitrary programming languages. Minerva gives both declarative and operational semantics to agents allowing the internal state of the agent, represented by logic programs, to modify. Minerva is based on multidimensional dynamic logic programming and uses explicit rules for modifying its knowledge bases.

The family of Golog languages [45, 79] propose high-level program execution as an alternative for controlling the behaviour of agents that operate in dynamic environments with partial observation. In fact, the high-level (agent) program consists of a set of actions, including the sense action (in IndiGolog [79]), composed by means of conditionals, iteration, recursion, concurrency and non-deterministic operators. Instead of finding a sequence of actions to achieve a desired state from an initial state, the problem is to find a sequence of actions that constitute a legal execution of the high-level program. When there is no non-determinism in the agent program, then the problem is straight forward. However, if the agent program consists of actions that are composed only by non-deterministic operators, then the problem is identical to the planning problem. The Golog language family represents the state of an agent as a set of

fluents.The execution of Golog programs is on-line planning (based on situation calculus) and plan execution.

Concurrent MetateM [43] is based on the direct execution of an extension of propositional temporal logic specifications. A multi-agent system in Concurrent MetateM consists of a set of concurrently executing agents with the ability to communicate asynchronously. Each agent is programmed by means of a temporal logic specification of the behaviour that the agent have to generate. In particular, it consists of rules that can be fired when their antecedents are satisfied with respect to the execution history. The consequent of a fired rule, which can be a temporal formula, forms the commitment of the agent that needs to be satisfied. The execution of an agents builds iteratively a logical model for the temporal agent specification. In Concurrent MetateM, the beliefs of agents are propositions extended with modal belief operators (allowing agents to reason about each others' beliefs), goals are temporal eventualities, and plans are primitive actions.

CLAIM [80] is a declarative multi-agent programming language focusing on mobile agents. It comes with a distributed platform called SyMPA that enables the execution of multi-agent programs. A multi-agent system in CLAIM is a set of hierarchies of agents distributed over a network. An agent in CLAIM can be a sub-agent of another one such that the hierarchies determine the parent-child relation between agents. Agents in CLAIM are BDI based and can be programmed in terms of knowledge, goals, capabilities, messages, parent and children. Agents can migrate within a hierarchy as well as between hierarchies by means of the move operation. The migration of agents in CLAIM is a strong migration, i.e., the state of the agent just before the migration is saved, encrypted, and transferred to the destination. At the destination, the agent's state is restored and processes are resumed from their interruption point.

Hybrid Style Agent-Oriented Programming Languages 3APL (An Abstract Agent Programming Language), as originally proposed in [50], is a programming language for single agents. The state of an agent in 3APL consists of declarative beliefs and plans, where plans consist of belief update, test, and abstract actions composed by sequence and conditional choice and iteration operators. This version of 3APL provides only plan revision rules that are applied to revise an agent's plan. The execution of a 3APL agent program is a cyclic process. At each cycle a plan revision rule is selected and applied after which a plan from the plan base is selected and executed. The execution of a plan modifies the belief base of the executed agent program. This original version of 3APL was an abstract programming language which lacked a development and execution platform. This version is extended with declarative goals and a variety of action types [33]. Also, an execution platform is developed for the extended version of 3APL [35].

2APL (A Practical Agent Programming Language) [25] is designed to implement multi-agent systems. It provides two sets of programming constructs to implement multi-agent and individual agent concepts. The multi-agent programming constructs are designed to create individual agents, external environ-

ments, and to specify the agents' access relations to the external environments. In 2APL, an environment (Java object) has a state and can execute a set of actions (method calls) to change its state. At the individual agent level, 2APL agents are implemented in terms of beliefs, goals, actions, plans, events, and three different types of rules. The beliefs and goals of 2APL agents are implemented in a declarative way, while plans and (interfaces to) external environments are implemented in an imperative style. The declarative part of the programming language supports the implementation of an agent's reasoning and update mechanisms. The imperative part of the programming language facilitates the implementation of plans, control flow, and mechanisms such as procedure call, recursion, and interfacing with legacy codes. 2APL agents can perform different types of actions such as belief update actions, belief and goal test actions, external actions (including sense actions), actions to manage the dynamics of goals, and communication actions. Three types of rules are used to generate plans. The first type of rules is designed to generate plans to achieve goals, the second to process (internal and external) events/messages, and the third to repair failed plans. A key feature of 2APL is the distinction between declarative goals and events.

GOALS [49] is a BDI-based programming language designed to implement autonomous agents. It provides programming constructs to implement an agent's knowledge, beliefs and goals declaratively. It also provides programming constructs to implement action selection rules that can be used to select actions based on the agent's current knowledge, beliefs and goals. A characteristic feature of GOAL is the distinction between knowledge and beliefs. Knowledge represents an agent's general information that are not the subject of modification, for example the agent's domain knowledge, while beliefs represents an agent's current information that can be modified during the agent execution, for example by sensing the environment or performing mental actions. Another characteristic feature of GOAL is the absence of plans. The action selection rules generate only atomic actions when they are applied. GOAL provides different types of actions such as user defined actions, built-in actions, and the communication actions. The execution of a GOAL agent is a cyclic process where at each cycle the agent senses the environment, applies action selection rules, and performs the generated actions.

Jason [13] is introduced as an interpreter of an extension of AgentSpeak, which is originally proposed by Rao [73]. Jason distinguishes multi-agent system concerns from individual agent concerns, though it does not allow the specification of agents' access relation to external environments. An individual agent in Jason is characterized by its beliefs, plans and the events that are either received from the environment or generated internally. A plan in Jason is designed for a specific event and belief context. The execution of individual agents in Jason is controlled by means of a cycle of operations encoded in its operational semantics. In each cycle, events from the environment are collected, an event is selected, a plan is generated for the selected event and added to the intention base, and finally a plan is selected from the intention base and executed. A plan rule in Jason indicates that a plan should be generated by an agent if an event is re-

ceived/generated and the agent has certain beliefs. Jason is based on first-order representation for beliefs, events, and plans. Jason has no explicit programming construct to implement declarative goals, though goals can be indirectly simulated by means of a pattern of plans. Moreover, the beliefs and plans in Jason can be annotated with additional information that can be used in belief queries and plan selection process. Finally, plan failure in Jason can be modelled by means of plans that react to the so-called deletion events.

IMPACT [38] is a multi-agent programming framework based on the idea of agentisation, i.e., agents are built around given legacy code. This programming framework comes with a formal semantics and an execution platform. An agent is built around a legacy code by abstracting from the legacy code and describing its main features. In particular, an agent is defined in terms of the set of all datatypes managed by the legacy code, a set of functions over the datatypes allowing external processes to access the datatypes, and a set of composition operators defined on the datatypes generating new composed datatypes. The state of an agent is determined by the state of the data in terms of which the agent is defined. Each agent has a set of actions that it can perform in its environment. An action can have different status such as permitted, obliged, or forbidden. The execution of an agent follows a cycle where messages from other agents are processed (which may in turn change the data and thus its state), the status of each action is determined, the actions that can be executed are determined, and the state is updated accordingly.

4.2 Programming Languages for Multi-Agent Organisations

In the literature on multi-agent systems, there have been many proposals for specification languages and logics to specify and reason about normative multi-agent systems, virtual organisations, and electronic institutions (e.g., [56, 72, 11, 1]). How to develop, program, and execute such normative systems was one of the central themes that were discussed and promoted during the AgentLink technical fora on programming multi-agent systems (see [29, 28] for the general report of these technical fora). In this section, we discuss three proposals for specifying and implementing normative multi-agent systems.

One of the early modelling languages for specifying institutions in terms of institutional rules and norms is ISLANDER [40]. In order to interpret institution specifications and execute them, a computational platform, called AMELI [41], has been developed. This platform implements an infrastructure that, on the one hand, facilitates agent participation within the institutional environment and supports their communication and, on the other hand, enforces the institutional rules and norms as specified in the institutional specification. The key aspect of ISLANDER/AMELI is that norms can never be violated by the agents. In other words, systems programmed via ISLANDER/AMELI make only use of regimentation in order to guarantee the norms to be actually followed. The norms in [41, 44, 83] are related to actions that the agents should or should not perform. In these approaches the issue of expressing more high-level norms concerning a state of the system that should be brought about is ignored. Such

high-level norms can be used to represent *what* the agents should establish — in terms of a declarative description of a system state — rather than specifying *how* they should establish it.

Another approach concerning specification of normative multi-agent systems by means of social and organisational concepts is $\mathcal{M}$OISE$^+$ [54]. This modelling language can be used to specify multi-agent systems through three organisational dimensions: structural, functional, and deontic. In a series of papers, different computational and programming frameworks have been proposed to implement and execute $\mathcal{M}$OISE$^+$ specifications. Examples of such frameworks are $\mathcal{S}$-$\mathcal{M}$OISE$^+$ [53] and its artifact-based version ORG4MAS [55]. These frameworks are concerned with norms that are about declarative descriptions of a state that should be achieved. Following the $\mathcal{M}$OISE$^+$ specification language, $\mathcal{S}$-$\mathcal{M}$OISE$^+$ is an organisational middleware that provides agents access to the communication layer and the current state of the specified organisation. Moreover, this middleware allows agents to change the organisation and its specification, as long as such changes do not violate organisational constraints. In the artifact version of this framework, ORG4MAS, various organisational artifacts are used to implement specific components of an organisation such as group and goal schema. In this framework, a special artifact, called reputation artifact, is introduced to manage the enforcement of the norms.

To summarize, in the work on electronic institutions ISLANDER/AMELI norms pertain to low-level procedures that directly refer to actions, whereas $\mathcal{M}$OISE$^+$/$\mathcal{S}$-$\mathcal{M}$OISE$^+$ are concerned with more high-level norms pertaining to declarative descriptions of the system. However, $\mathcal{S}$-$\mathcal{M}$OISE$^+$ does not allow agents to violate organisational rules and norms by ensuring that they respect organisational specification. This suggests that norms in $\mathcal{S}$-$\mathcal{M}$OISE$^+$ are regimented rather than being enforced by means of sanctions. In the artifact version of this framework, ORG4MAS, the enforcement of norms is assumed to be managed indirectly through a reputation mechanism, but it remains unclear how such a reputation system realizes sanctioning. Another important issue is that AMELI and $\mathcal{S}$-$\mathcal{M}$OISE$^+$ lack a complete operational semantics that capture all aspects of normative systems, including the enforcement of norms. An explicit formal and operational treatment of norm enforcement is essential for a thorough understanding and analysis of computational frameworks of normative multi-agent systems. Also, the computational frameworks related to $\mathcal{M}$OISE$^+$ are not grounded in a logical system such that the soundness and properties of the programmed systems cannot be analysed through formal analyses and verification mechanisms. Finally, it should be noted that ISLANDER/AMELI and $\mathcal{M}$OISE$^+$/$\mathcal{S}$-$\mathcal{M}$OISE$^+$ provide a variety of social and organisational concepts.

`powerJava` [7] and `powerJade` [6] are designed to implement institutions in terms of roles. While `powerJava` extends Java with programming constructs to implement institutions, `powerJade` proposes similar extensions to the Jade framework. In these programming frameworks, an institution is considered as an exogenous coordination mechanism that manages the interactions between participating computational entities (objects in `powerJava` and agents in `powerJade`)

by means of roles. A role is defined in the context of an institution (e.g., a student role is defined in the context of a school) and encapsulates capabilities, also called powers, that its players can use to interact with the institution and with other roles in the institution (e.g., a student can participate in an exam). For an object or an agent to play a role in an institution in order to gain specific abilities, they should satisfy specific requirements as well. In `powerJava` roles and organisations are implemented as Java classes. In particular, a role within an institution is implemented as an inner class of the class that implements the organisation. Moreover, the powers that a player of a role gains and the requirements that the player of the role should satisfy are implemented as methods of the class that implements the role. In `powerJade`, organisations, roles and players are implemented as subclasses of the Jade agent class. The powers that the player of a role gains and the requirements that a player of a role should satisfy are implemented as Jade behaviours (associated to the role).

Finally, a recent programming language that is designed to support the implementation of multi-agent organisations is 2OPL (Organisation Oriented Programming) [30, 89]. This is a rule-based programming language that facilitates the implementation of norm-based organizations. In this programming framework, an organisation is considered as a software entity that exogenously coordinates the interaction between agents and their shared environment. In particular, the organisation is a software entity that manages the interaction between the agents themselves and between agents and the shared environment. 2OPL provides programming constructs to specify 1) the initial state of an organisation, 2) the effects of agents actions in the shared environment, and 3) the applicable norms and sanctions. In 2OPL norms can be either enforced by means of sanctions or regimented. In the first case, agents are allowed to violate norms after which sanctions are imposed. In the second case, norms are considered as constraints that cannot be violated. The enforcement of norms by sanctions is a way to guarantee higher autonomy for the agents and higher flexibility for the multi-agent system. The interpreter of 2OPL is based on a cyclic control process. At each cycle, the observable actions of the individual agents (i.e., communication and environment actions) are monitored, the effects of the actions are determined, and norms and sanction are imposed if necessary. An advantage of 2OPL approach is its complete operational semantics such that normative organisation programs can be formally analysed by means of verification techniques [4]. This organisation oriented programming language is extended with programming constructs that support the implementation of concepts such as obligation, permission, prohibition, deadline, norm change, and conditional norm [89, 87, 88].

4.3 Programming Languages for Multi-Agent Environments

A programming framework for multi-agent environments is CARTAGO (Common ARtifact Infrastructure for AGent Open environment) [77]. This framework is based on the A&A model which proposes a working environment to be used

by agents for supporting their working activities. A working environment is considered as consisting of a set of artifacts organised in workspaces (containers of artifacts). The artifacts are meant to encapsulate specific functionalities and can be added, removed, and organised in the workspaces by agents at runtime. Artifacts can be used by agents through their usage interfaces that allow agents to trigger and control the execution of artifacts' operations and perceiving events from them. Different operations are supported by artifact interfaces. An agent can for example create, remove, or search for artifacts and workspaces. Agents can also execute operations of artifacts and sense the events generated by the artifacts, or inspect the artifacts by retrieving their descriptions. This framework can be distributed in the sense that a working environment can consists of one or more workspaces that can be mapped onto a different nodes of a network. CARTAGO is implemented in Java and has been connected to various agent platforms such as Jason and 2APL.

Beside this generic architecture and programming framework designed for the development of environments, there have been many interesting environments implemented using existing programming languages such as Java or C++. These environments are initially developed in an ad-hoc manner either for one of the existing agent platforms such as 2APL, GOAL, Jadex, and Jason, as a stand-alone application such as Unreal Tournament 2004, or as a simulation environment. The availability of these implemented environments raises the question how they can be (re)used and applied to arbitrary agent platforms. In practice, agent developers rebuild similar environments from scratch. Apart from this duplicating works, the interaction between agents and environments are managed in an ad-hoc manner making the reuse of the environments a dedicated task that depends on the specific agent platform and the environment at hand. This problem has lead to the initiative for creating a generic environment interface standard which provides the required functionalities for connecting agents to environments in a standardized manner [85]. If environments were developed using such a standard, they could be exchanged freely between agent platforms that support the standard and thus would make already existing environments widely available. In order to develop a generic environment interface standard various issues should be addressed. An important issue is the right level of abstraction for modelling the interaction between agents and environments. This generic environment interface standard supports the interaction between agents and environment in two ways. On the one hand, agents can perform actions, including sense action, in the environment. On the other hand, the environment can send events to individual agents. This interface provides constructs to establish and manage the relation between agents with entities (agent bodies) in the environment, the registration of agents by the interface, adding and removing entities from the environment, and performing actions and retrieving percepts from the environment. Several agent platforms such as 2APL, GOAL, and Jason have already integrated the environment interface standard.

5 Current Trends

Existing agent-oriented programming languages are the result of continuous developments. Despite their characteristic differences, these developments and extensions have been quite similar causing the programming languages to converge in the sense of providing programming constructs for the same set of concepts and abstractions. For example, most agent-oriented programming languages provide currently similar types of actions such as actions to modify an agent's state, communication actions, and external actions allowing individual agents to interact with a shared environment. In the logic-based programming languages for BDI architectures, an agent's beliefs and goals are often implemented using Prolog allowing the programmed agent to reason about its beliefs and goals. Most agent programming languages provide constructs to process various types of events by means of generating and executing plans. In order to respect programming principles such as reuse and encapsulation, most agent programming languages provide constructs to support implementation of modules. The similarity between these languages is not only due to similar programming constructs, the underlying semantics of these languages converge as well. Programming languages with declarative beliefs and goals establish rational constraints in their underlying semantics by, for example, requiring that agents should have consistent beliefs and that agents cannot aim at achieving the current (belief) state. Finally, the development and execution platforms corresponding to agent programming languages also converge in the sense that they provide similar functionalities and development tools. Most development platforms provide editors that support the syntax of their corresponding programming languages, different tools to monitor and control the execution of agents, and different platform facilities such as agent management and directory facilitator.

An advance in the field of agent programming languages concerns the concept of goals. Goals are essential for agents with pro-active behaviour [97]. The initial focus of agent-oriented programming languages was on achievement goals, which represent a desired state that the agent aim at reaching. In due course other goal types have been studied, e.g., perform goal (the goal to execute certain actions), test goal (the goal to test an agent's state), and maintain goal (the goal to maintain a state) [18, 36, 39, 52]. In order to allow the implementation of various goal types existing agent programming languages provide a variety of constructs to represent and reason with these goal types. For example, JACK [94] provides programming constructs to implement, among others, test, achieve, insist, and maintain goals, and Jadex [69] has achieve, query, perform and maintain goals. The way in which goals are treated by these programming languages differs. In Jadex goals are represented in XML in terms of a label/name and a number of other parameters while JACK goals are particular types of events. Moreover, neither JACK nor Jadex provide the formal semantics of their goal types. Winikoff et al. [96] provides a survey of existing literature on goal types. A more recent (theoretical) trend in this direction is to go beyond these goal types and to introduce more expressive goal types or even a language for expressing goal types. For example, Dastani et al. [34] proposes six types of multiple state

goals (goals expressing a property that should hold over a number of states), while other approaches propose to take arbitrary Linear Temporal Logic (LTL) formulae as goals [5, 8, 51, 58, 81]. The advantage of the approach proposed by Dastani et al. is their computational setting. The six multiple state goal types can be defined in terms of achieve and maintain goals. This makes it possible to implement these goal types in the agent programming frameworks that already have an operationalization of achieve and maintain goals.

From the software engineering point of view, the ultimate aim of designing a multi-agent programming language is to support practitioners to develop multi-agent systems for industrial applications. To this aim it is important that programming languages satisfy essential principles in structured programming such as modularity. Of course, the separation of concerns at the level of individual agents, organisation, and environment support modularity in multi-agent programming. However, the programming languages for agents, organisations, and environments can be considered as specific programming languages that in turn need to satisfy modularity as well. There have been some proposals for supporting modules in BDI-based programming languages, e.g., [19, 21, 48, 90, 32, 61]. In these proposals, modularization is considered as a mechanism to structure an individual agent's program in separate modules, each encapsulating cognitive components such as beliefs, goals, and plans that together model a specific functionality and can be used to handle specific situations or tasks. However, the way the modules are used in these programming approaches are different. For example, in Jack [21] and Jadex [19], modules (which are also called capabilities) are employed for information hiding and reusability by encapsulating different cognitive components that together implement a specific capability/functionality of the agent. In these approaches, the encapsulated components are used during an agent's execution to process events that are received by the agent. In other approaches [48, 90], modules are used to realize a specific policy or mechanism in order to control an agent execution. More specifically, modules in GOAL [48] are considered as the 'focus of execution', which can be used to disambiguate the application and execution of plans. This is done by assigning a mental state condition (beliefs and/or goals) to each module. The modules whose conditions are satisfied form the focus of an agent's execution such that only plans from these modules are applied and executed. In 3APL [90] a module is a set of planning rules that is associated with a specific goal indicating which planning rules can be applied to achieve the goal. In other words, a module implements specific means for achieving specific goals. In 2APL [32] modules are introduced for encapsulation of different cognitive components that together implement a specific agent functionality. The significant difference with other approaches is that a programmer can perform a wide range of operations on modules. These module-related operations enable a programmer to directly and explicitly control when and how modules are used. For instance, a programmer can create an instance of the module specification, query and update its internals, and execute the updated module instance. An agent that executes a module instance, stops deliberating on its current cognitive state and starts deliberating on a new cog-

nitive state that is encapsulated by the executed module instance. The proposed notion of module can be used to implement a variety of agent concepts such as agent role and agent profile. Recently, a modularization idea for Jason [61] have been proposed. In this proposal, a module encapsulates a subset of an agent's functionalities and consists of cognitive ingredients such as belief, goal, and event bases, a plan library, and a list of exported belief and goal predicates. An agent is then defined as a composition of modules (modules cannot be nested), together with a slightly modified version of the Jason's original interpreter. Finally, it should be noted that the concept of module as used by Novak and Dix [65] is different than other approaches. A module in [65] is considered as one specific cognitive component (e.g., an agent's beliefs) and not as a functionality modelled by different cognitive components. Note also that behaviours in Jade, which can be used to implement an agent's functionality, can be seen as a kind of modular programming.

6 Current Challenges

There are many challenges to meet in the multi-agent programming research field. Examples of these challenges are scalability and automatic code generation. In this overview, we focus on two challenges that require both theoretical and practical investigations. The first challenge is a principle integration of programming languages for individual agents, organisations, and environments. The second challenge is the debugging and testing of multi-agent programs.

6.1 Integration of Programming Languages

Respecting the separation of concerns principle advocates separate programming languages for the implementation of individual agents, environments, and organisations. A multi-agent programming framework for the development of multi-agent systems should therefore facilitate a systematic integration of the corresponding programming languages. Ideally, one should be able to write programs for different components of a multi-agent system separately and integrate these programs either by means of another program that indicates how the programs of different components should interact and executed, or through a development platform that facilitates an integrated execution of all involved programs. It should be noted that this challenge is only relevant when different components of multi-agent systems need to be programmed separately using dedicated programming languages.

One proposal for integrating programming languages for various components of multi-agent systems is based on the integration of 2APL and 2OPL. In this approach, a multi-agent program is implemented by specifying a number of agents programmed in 2APL, one or more environments programmed in Java, and an organisation programmed in 2OPL. An execution of such a multi-agent program is a concurrent execution of the specified individual agents programs, the environment program, and the organisation program. The execution of individual

agents programmed in 2APL may cause agents to interact with each other and with the environment. The resulting actions are not directly effectuated in the environment, but passed to the organisation implemented in 2OPL. The organisation decides the effects of those actions based on the specified organisational norms and sanctions. In particular, the performance of actions by individual agents is effectuated by the organisation program, which allows/disallows actions and realizes the effect of actions in the environment. The organisation will also evaluate the updated state of the environment with respect to the specified organisational norms and impose sanctions when violations are detected. Imposing sanctions is seen as specific updates of the environment state according to the sanctions specified by the organisation. This integration approach views an organisation as an exogenous coordination mechanism.

Another integration proposal is JaCaMo [12]. This approach aims at integrating Jason, Cartago, and Moise to program individual agents, environments, and organisations, respectively. The idea is to integrate on the one hand Moise and Cartago, and on the other hand Jason and Cartago. The integration of Moise and Cartago is by means of organisational artifacts and based on the earlier work on ORA4MAS [59]. The integration of Jason and Caratgo is through artifact operations performed by the Jason agents. In this integrated approach, the organisational infrastructure of a multi-agent system consists of organisational artifacts and organisational agents that together are responsible for functionalities concerning the management and enacting of the organisation. Organisational agents manage organisational activities such as observing and reasoning about organisation dynamics. The violation of norms is detected by organisational artifacts after which organisational agents have to deal with those violations. The organisational artifacts and agents are intended to implement norm regimentation and enforcement by means of sanctions, as originally proposed in [30]. A characteristic of this integration is that the management of organisational activities such as norm enforcement is the responsibility of the so-called organisational agents. It is, however, not clear why such activities should be performed by an agent rather than, for example, by the organisation itself. An agent has by definition its own objectives that is used to motivate its actions. It is the question why norm enforcement should be modelled and programmed in terms of such an agent.

6.2 Debugging and Testing Multi-Agent Programs

Debugging is the art of finding and resolving errors or possible defects in a computer program. In general, there are various types of bugs such as syntax bugs, semantic bugs (logical and concurrent bugs), or design bugs. Design bugs arise before the actual programming and are based on erroneous design of software programs. In contrast to design bugs, both syntax and semantic bugs arise during programming and are related to the actual code of the programs. Although syntax bugs depends on specification of programming languages and are (most of the time) simple typos, which can easily be detected by the program parser (compiler), semantic bugs are mistakes at the semantic level. Because

they often depend on the intention of the developer they can rarely be detected automatically by the program parsers. Therefore, special tools are needed to detect semantic bugs. The ease of the debugging experience is largely dependent on the quality of these debugging tools and the ability of developers to work with these tools.

The main challenge with respect to debugging multi-agent programs are the semantic bugs caused by the execution of autonomous agent programs, and those caused by the interaction between agents, environments, and organisations. The bugs causing by the interaction between agents are often dealt with by means of different types of visualization tools such as sniffer and causality graphs [9, 13, 25, 69, 16, 91]. The visualization tools allow the developer to browse through exchanged messages, inspect the messages, and present them using different visualisation techniques. Debugging the interaction between agents, environments, and organisations are still an unexplored research area. The semantic bugs caused by the execution of individual agent programs are dealt with by a variety of techniques such as breakpoints, assertions, and execution tracers [23, 84, 24, 9, 25, 13]. A breakpoint is a marker that can be placed in the program's code to control the program's execution. When the marker is reached the program execution is halted. Assertions are statements that can be annotated to specific elements of the programming language. When an assertion is evaluated to false, a warning is generated to inform the developer about the agent and the element where the assertion is evaluated to false. The execution tracer is a standard tool that is present in most multi-agent development frameworks. This is a window that enables a developer to view, inspect, and trace an agent's internal state, and to start, stop, and step through the execution of the agent program.

Debugging agent programs that are based on BDI abstractions requires additional tools to monitor and control temporal and cognitive properties of the agent program executions. In [27], a debugging framework is proposed which is based on a specification language to express temporal and cognitive execution properties of multi-agent programs and a set of debugging tools. The expressions of the specification language are related to the proposed debugging tools such that the debugging tools are activated as soon as their associated properties hold for the multi-agent program execution thus far. The specification language is based on linear temporal logic extended with the BDI operators. Given an execution of a multi-agent program, one can check if an agent drops a specific goal when it is achieved, when two or more agents have the same beliefs, whether a protocol is suited for a given task, whether important beliefs are communicated and if they adopted/rejected once they are communicated.

Recent developments in multi-agent programming languages [41, 53, 89, 30] have proposed specific programming constructs enabling the implementation of social concepts such as norms, roles, obligations, and sanctions. Debugging such multi-agent programs requires specific debugging constructs to specify properties related to the social aspects and to find and resolve defects involved in such programs. The presented debugging frameworks assume all agents are developed on one single platform such that their executions for debugging purposes are not

distributed on different platforms. One important challenge and a future work on debugging multi-agent systems remains the debugging of multi-agent programs that run simultaneously on different platforms.

The techniques mentioned above are helpful when errors manifest themselves directly to the developer or user. However, errors in a program do not always manifest themselves directly. For mission and industrial critical systems it is necessary to extensively test the program before deploying it. This testing should remove as many bugs (and possible defects) as possible. However, it is infeasible to test every single situation the program could be in. A testing approach proposed for multi-agent programs is proposed by Poutakidis and his colleagues [71, 70, 95]. Testing is an indispensable part of evaluating multi-agent programs and should therefore be integrated in the existing debugging approaches. This allows the generation of a set of critical test traces which will be the subject of debugging in post mortem mode.

7 Conclusion

The design and development of multi-agent programming languages is still a main issue in the multi-agent programming community. From the software engineering perspective, the aim is to propose programming languages that can support direct and effective implementations of large-scale multi-agent systems by proposing programming constructs to facilitate the implementation of abstractions used in the analysis and design of multi-agent systems. Up until now many multi-agent programming languages have been proposed. They differ from each other in the set of abstractions, programming constructs, and principles they convey. Although these programming languages are developing towards a certain level of maturity in the sense that their programming concepts and operations are well motivated and have profound semantics, a majority of them are still not being employed for the development of large-scale industrial applications. Currently, these programming languages, in particular those that are based on cognitive and social constructs, are mainly considered as research works that aim at designing and prototyping high-level abstract programming patterns.

The incorporation of social and cognitive concepts in the design of multi-agent programming languages requires semantic and computational analyses for these concepts. Moreover, the development of multi-agent programming languages requires formal theories and computational techniques for representing and reasoning about concepts such as beliefs, goals, actions, roles, norms, and sanctions. For these reasons the design and development of multi-agent programming languages have been attractive to researchers from various scientific disciplines such as logic, artificial intelligence, philosophy, cognitive science, and social science. The aim of multi-agent programming research field from this interdisciplinary perspective is to propose computational models of multi-agent systems, which may not necessarily satisfy the software engineering requirement to support building large-scale industrial applications. Within this perspective, multi-agent programming languages based on social and cognitive concepts can

be considered as formal and computational architectures for social organisations and cognitive agents.

The state of the art in the field of multi-agent programming languages shows the development of specialised programming languages for individual agents, organisations, and environments. The main focus of multi-agent programming community has been on the development of programming languages for individual agents. Although research on (formal) models for multi-agent organisations and environments has relatively a long history, the development of programming languages that are designed to support the implementation of multi-agent organisations and environments is a recent phenomenon. The development of programming languages for individual agents show a convergence in the sense that they propose programming constructs for an established set of concepts and abstractions. These programming languages differ from each other as they use different programming styles (declarative, imperative, or both), support different programming principles such as modularity, abstraction, recursion, exception handling, support for legacy codes, and as their corresponding integrated development environments provide different sets of functionalities such as editing, debugging, and automatic generation of codes.

One of the current challenges in multi-agent programming research field is the integration of programming languages for individual agents, multi-agent organisations and multi-agent environments. Although there have been several attempts to integrate specific programming languages, the ultimate goal is a mechanism to facilitate the integration of arbitrary programming languages for individual agents, multi-agent organisations, and multi-agent environments. A possible solution to realize such a goal is to design and develop standard interfaces that can manage the interactions between individual agent programs, multi-agent organisation programs, and multi-agent environment programs. There have already been some initiatives to establish standard interfaces for managing the interaction of individual agent programs and environment programs, but the research in this direction is still in a preliminary phase and needs support and collaboration from the community. Another issue currently challenging the multi-agent programming community is the debugging and testing of multi-agent programs. There is a need for powerful debugging facilities and testing tools that can cope with the distributed nature of multi-agent systems, the autonomy of individual agents, and the interactions between individual agent, multi-agent organisation, and multi-agent environment programs. There have been some initial attempts for enriching debugging tools with expressive specification languages such that tools can be initialized and activated when the execution of multi-agent programs satisfy certain properties, but such attempts ignore multi-agent organisation and environment programs.

As noticed before, this overview is by no means complete. There are still many issues related to multi-agent programming that need to be investigated. Among these issues are mechanisms to deal with plan failure, goal types, reasoning about organisations and environments from an agent's point of view, the integration of concepts such as sensing, planning, acting, learning and emotions in the agent's

deliberation process, the adaptivity of organisation and environment p based on the executions of individual agent programs, and formal verification of multi-agent programs.

References

1. T. Ågotnes, W. van der Hoek, and M. Wooldridge. Robust normative systems. In Padgham, Parkes, Muller, and Parsons, editors, *Proceedings of the Seventh International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, pages 747–754, Estoril, Portugal, May 2008. IFAMAAS/ACM DL.
2. F. Arbab. What do you mean, coordination? In *Bulletin of the Dutch Association for Theoretical Computer Science (NVTI*, pages 11–22, 1998.
3. F. Arbab, L. Astefanoaei, F. de Boer, M. Dastani, J.-J.Ch. Meyer, and N. Tinnermeier. Reo connectors as coordination artifacts in 2APL systems. In *Proceedings of the 11th Pacific Rim International Conference on Multi-Agents (PRIMA 2008)*, volume LNCS 5357, pages 42–53. Springer, 2009.
4. L. Astefanoaei, M. Dastani, J.-J. Ch. Meyer, and F. Boer. On the semantics and verification of normative multi-agent systems. *International Journal of Universal Computer Science*, 15(13):2629–2652, 2009.
5. F. Bacchus and F. Kabanza. Planning for temporally extended goals. *Annals of Mathematics and Artificial Intelligence*, 22(1-2):5–27, 1998.
6. M. Baldoni, G. Boella, M. Dorni, R. Grenna, and A. Mugnaini. powerJADE: Organizations and roles as primitives in the JADE framework. In *In of WOA 2008: Dagli oggetti agli agenti, Evoluzione dell'agent development: metodologie, tool, piattaforme e linguaggi*, pages 84–92, 2008.
7. M. Baldoni, G. Boella, and L. Van Der Torre. Roles as a coordination construct: Introducing powerJava. In *In Proceedings of 1st International Workshop on Methods and Tools for Coordinating Concurrent, Distributed and Mobile Systems*, volume 150 (1), pages 9–29. Electronic Notes in Theoretical Computer Science, 2005.
8. C. Baral and J. Zhao. Non-monotonic temporal logics for goal specification. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 236–242, 2007.
9. F. Bellifemine, F. Bergenti, G. Caire, and A. Poggi. JADE - a java agent development framework. In *Multi-Agent Programming: Languages, Platforms and Applications*. Kluwer, 2005.
10. F. Bergenti, M.-P. Gleizes, and F. Zambonelli (eds.). *Methodologies and Software Engineering for Agent Systems*, volume 11 of Multiagent Systems, Artificial Societies, and Simulated Organizations. Kluwer Academic Publisher, 2004.
11. G. Boella and L. van der Torre. Substantive and procedural norms in normative multiagent systems. *Journal of Applied Logic*, 6:152–171, 2008.
12. R. Bordini, J. Hubner, and A. Ricci. JaCaMo: Jason, Cartago, Moise. `http://jacamo.sourceforge.net/`.
13. R. Bordini, M. Wooldridge, and J. Hübner. *Programming Multi-Agent Systems in AgentSpeak using Jason (Wiley Series in Agent Technology)*. John Wiley & Sons, 2007.
14. R. H. Bordini, M. Dastani, J. Dix, and A. El Fallah Seghrouchni (eds.). *Multi-Agent Programming: Languages, Platforms and Applications*, volume 15 of *Multiagent Systems, Artificial Societies, and Simulated Organizations*. Springer, Berlin, 2005.

15. R. H. Bordini, M. Dastani, J. Dix, and A. El Fallah Seghrouchni (eds.). *Multi-Agent Programming: Languages, Tools and Applications*. Springer, 2009.
16. J.A. Botía, J.M. Hernansaez, and A.F. Gómez-Skarmeta. On the application of clustering techniques to support debugging large-scale multi-agent systems. In *PROMAS*, pages 217–227, 2006.
17. A. Bracciali, N. Demetriou, U. Endriss, A. Kakas, W. Lu, P. Mancarella F. Sadri, K. Stathis, G. Terreni, and F. Toni. The KGP model of agency for global computing: Computational model and prototype implementation. In *Global Computing*, volume 3267 of *Lecture Notes in Computer Science*, pages 340–367. Springer, 2004.
18. L. Braubach and A. Pokahr. Representing long-term and interest BDI goals. In *Programming Multi-Agent Systems (ProMAS)*, 2009.
19. L. Braubach, A. Pokahr, and W. Lamersdorf. Extending the Capability Concept for Flexible BDI Agent Modularization. In *Proc. of ProMAS '05*, pages 139–155, 2005.
20. P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, and A. Perini. TROPOS: An agent-oriented software development methodology. *Journal of Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, 2003.
21. P. Busetta, N. Howden, R. Ronnquist, and A. Hodgson. Structuring BDI Agents in Functional Clusters. In N. Jennings and Y. Lesperance, editors, *Intelligent Agents VI: Theories, Architectures and Languages*, pages 277–289, 2000.
22. P.R. Cohen and H.J. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42, 1990.
23. R. Collier. Debugging agents in agent factory. *ProMAS 2006*, pages 229–248, 2007.
24. K. S. Barber D. N. Lam. Debugging agent behavior in an implemented agent system. *ProMAS 2004*, pages 104–125, 2005.
25. M. Dastani. 2apl: a practical agent programming language. *International Journal of Autonomous Agents and Multi-Agent Systems*, 16(3):214–248, 2008.
26. M. Dastani, F. Arbab, and F.S. de Boer. Coordination and composition in multi-agnet systems. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'05)*, pages 439–446. 2005.
27. M. Dastani, J. Brandsema, A. Dubel, and J.-J.Ch. Meyer. Debugging bdi-based multi-agent programs. In *International Workshop on Programming Multi-Agent Systems (ProMAS)*, 2009.
28. M. Dastani and J. Gomez-Sanz. Programming multi-agent systems. *The Knowledge Engineering Review*, 20(2):151–164, 2006.
29. M. Dastani and J.J. Gomez-Sanz. Agentlink iii technical forum group, programming multiagent systems. `http://people.cs.uu.nl/mehdi/al3promas.html`.
30. M. Dastani, D. Grossi, J.-J. Ch. Meyer, and N. Tinnemeier. Normative multi-agent programs and their logics. In *Post proceedings of the international workshop on Knowledge Representation for Agents and Multi-Agent Systems (KRAMAS 2008)*, volume LNAI 5605, pages 16–31. Springer, 2009.
31. M. Dastani, K. Hindriks, and J.J.Ch. Meyer. *Specification and Verification of Multi-agent Systems*. Springer, 2010.
32. M. Dastani and B. R. Steunebrink. Operational semantics for bdi modules in multi-agent programming. In *Proceedings of the 10th international conference on Computational logic in multi-agent systems*, CLIMA'09, pages 83–101, Berlin, Heidelberg, 2010. Springer-Verlag.
33. M. Dastani, B. van Riemsdijk, F. Dignum, and J.-J.Ch. Meyer. A programming language for cognitive agents: Goal directed 3APL. In *Proceedings of ProMAS03*. LNAI 3067, Springer, Berlin, 2004.

34. M. Dastani, B. van Riemsdijk, and Winikoff. Rich goal types in agent programming. In *Proceedings of the Tenth International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2011)*, 2011.
35. M. Dastani, M.B. van Riemsdijk, and J.-J. Ch. Meyer. Programming multi-agent systems in 3apl. In *Multi-Agent Programming: Languages, Platforms and Applications*. Kluwer, 2005.
36. M. Dastani, M.B. van Riemsdijk, and J.-J.Ch. Meyer. Goal types in agent programming. In *Proceedings of the 17th European Conference on Artificial Intelligence (ECAI'06)*, 2006.
37. V. Dignum. *A Model for Organizational Interaction*. PhD thesis, Utrecht University, SIKS, 2004.
38. J. Dix and Y. Zhang. IMPACT: A multi-agent framework with declarativesemantics. In *Multi-Agent Programming: Languages, Platforms and Applications*, page 6994. Kluwer, 2005.
39. S. Duff, J. Harland, and J. Thangarajah. On proactivity and maintenance goals. In *Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 1033–1040, Hakodate, 2006.
40. M. Esteva, D. de la Cruz, and C. Sierra. ISLANDER: an electronic institutions editor. In *Proceedings of the First International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2002)*, pages 1045–1052, Bologna, Italy, 2002.
41. M. Esteva, J.A. Rodríguez-Aguilar, B. Rosell, and J.L. Arcos. AMELI: An agent-based middleware for electronic institutions. In *Proceedings of the Third International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2004)*, pages 236–243, New York, US, July 2004.
42. J. Ferber. *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*. Addison-Wesley Longman Publishing, 1999.
43. M. Fisher. METATEM: The story so far. In *Proceedings of the Third International Workshop on Programming Multiagent Systems (ProMAS-03)*, volume 3862 of *Lecture Notes in Artificial Intelligence*, pages 3–22. Springer Verlag, 2005.
44. A. Garcia-Camino, P. Noriega, and J. A. Rodriguez-Aguilar. Implementing norms in electronic institutions. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2005)*, pages 667–673, New York, NY, USA, 2005.
45. G. De Giacomo, Y. Lesperance, and H.J. Levesque. Congolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence*, 121(1-2):109–169, 2000.
46. J. Gomez-Sanz and J. Pavon. Agent oriented software engineering with ingenias. In *Lecture Notes in Computer Science*, volume 2691, pages 394 – 403. Springer, 2003.
47. D. Grossi. *Designing Invisible Handcuffs*. PhD thesis, Utrecht University, SIKS, 2007.
48. K. Hindriks. Modules as policy-based intentions: Modular agent programming in GOAL. In *Proc. of ProMAS '07*, volume 4908. Springer, 2008.
49. K. Hindriks. Programming rational agents in GOAL. In *Multi-Agent Programming: Languages and Tools and Applications*, page 119157. Springer, 2009.
50. K. Hindriks, F. De Boer, W. Van der Hoek, and J.-J.Ch. Meyer. Agent programming in 3APL. *Autonomous Agents and Multi-Agent Systems*, 2(4):357–401, 1999.
51. K. Hindriks, W. van der Hoek, and M.B. van Riemsdijk. Agent programming with temporally extended goals. In *Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 137–144. IFAAMAS, 2009.

52. K. Hindriks and M.B. van Riemsdijk. Satisfying maintenance goals. In *Declarative Agent Languages and Technologies (DALT'07)*, volume 4897 of *LNAI*, pages 86–103. Springer, 2008.
53. J. Hübner, J.S. Sichman, and O. Boissier. $\mathcal{S}-\mathcal{M}$OISE$^+$: A middleware for developing organised multi-agent systems. In *Proceedings of the international workshop on Coordination, Organizations, Institutions, and Norms in Multi-Agent Systems*, volume 3913 of *LNCS*, pages 64–78. Springer, 2006.
54. J. Hübner, J.S. Sichman, and O. Boissier. Developing organised multiagent systems using the $\mathcal{M}$OISE$^+$ model: programming issues at the system and agent levels. *International Journal of Agent-Oriented Software Engineering*, 1(3/4):370–395, 2007.
55. J.F. Hübner, O. Boissier, R. Kitio, and A. Ricci. Instrumenting multi-agent organisations with organisational artifacts and agents: Giving the organisational power back to the agents. *International Journal of Autonomous Agents and Multi-Agent Systems*, 20:369–400, 2010.
56. A. J. I. Jones and M. Sergot. On the characterization of law and computer systems. In J.-J. Ch. Meyer and R.J. Wieringa, editors, *Deontic Logic in Computer Science: Normative System Specification*, pages 275–307. John Wiley & Sons, 1993.
57. A. Kakas, P. Mancarella, F. Sadri, K. Stathis, and F. Toni. The KGP model of agency. In *The 16th European Conference on Artificial Intelligence*, pages 33–37, 2004.
58. S.M. Khan and Y. Lespérance. A logical account of prioritized goals and their dynamics. In G. Lakemeyer, L. Morgenstern, and M. A. Williams, editors, *Proc. of the 9th International Symposium on Logical Formalizations of Commonsense Reasoning*, pages 85–90, 2009.
59. R. Kitio, O. Boissier, J. Hübner, and A. Ricci. Organisational artifacts and agents for open multi-agent organisations: "giving the power back to the agents". In *Proceedings of the 2007 international conference on Coordination, organizations, institutions, and norms in agent systems III*, COIN'07, pages 171–186, Berlin, Heidelberg, 2008. Springer-Verlag.
60. J. Leite, J. Alferes, and L.M. Pereira. Minerva — A dynamic logic programming agent architecture. In J.-J.Ch. Meyer and M. Tambe, editors, *Pre-proceedings of the Eighth International Workshop on Agent Theories, Architectures, and Languages (ATAL-2001)*, pages 133–145, 2001.
61. N. Madden and B. Logan. Modularity and compositionality in jason. In L. Braubach, J.-P. Briot, and J. Thangarajah, editors, *Programming Multi-Agent Systems: 7th International Workshop, ProMAS 2009, Budapest, Hungary, May 10-15, 2009. Revised Selected Papers*, number 5919 in LNAI, pages 237–253, Budapest, 2009. Springer.
62. J.-J.Ch. Meyer, W. van der Hoek, and B. van Linder. A logical approach to the dynamics of commitments. *Arificial Intelligence*, 113:1–40, 1999.
63. J.P. Müller. *The Design of Autonomous Agents A Layered Approach*, volume 1177 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 1996.
64. R. Nair and M. Tambe. Hybrid bdi-pomdp framework for multiagent teaming. *Journal of Artificial Intelligence Research*, 23:367–420, 2005.
65. P. Novák and J. Dix. Modular bdi architecture. In *Proceedings of the AAMAS'06*, 2006.
66. A. Omicini. Formal respect in the a&a perspective. *Electronic Notes Theoretical Computer Science*, 175(2):97–117, 2007.
67. L. Padgham and M. Winikoff. Prometheus: A methodology for developing intelligent agents. In *Lecture Notes in Artificial Intelligence*, volume 2585, pages 174 – 185. Springer, 2003.

68. H. Van Dyke Parunak and D. Weyns. Guest editors' introduction, special issue on environments for multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 14(1):1–4, 2007.
69. A. Pokahr, L. Braubach, and W. Lamersdorf. Jadex: A BDI reasoning engine. In *Multi-Agent Programming: Languages, Platforms and Applications*. Kluwer, 2005.
70. D. Poutakidis, L. Padgham, and M. Winikoff. Debugging multi-agent systems using design artifacts: The case of interaction protocols. In *In Proceedings of AAMAS-02*, pages 960–967, 2002.
71. D. Poutakidis, L. Padgham, and M. Winikoff. An exploration of bugs and debugging in multi-agent systems. In *In Proceedings of the 14th International Symposium on Methodologies for Intelligent Systems (ISMIS)*, pages 628–632. ACM Press, 2003.
72. H. Prakken and M. Sergot. Contrary-to-duty obligations. *Studia Logica*, 57:91–115, 1996.
73. A.S. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language. In Rudy van Hoe, editor, *Proceedings of the Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW'96)*, Eindhoven, The Netherlands, 1996.
74. A.S. Rao and M.P. Georgeff. Modeling rational agents within a BDI-architecture. In J. Allen, R. Fikes, and E. Sandewall, editors, *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning (KR'91)*, pages 473–484. Morgan Kaufmann, 1991.
75. A.S. Rao and M.P. Georgeff. BDI agents: From theory to practice. In J. Allen, R. Fikes, and E. Sandewall, editors, *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS'95)*, 1995.
76. G. Rens, A. Ferrein, and E. van der Poel. A bdi agent architecture for a pomdp planner. In *Nineth International Symposium on Logical Formalizations of Commonsense Reasoning*, Toronto, Canada, 2009.
77. A. Ricci, M. Viroli, and A. Omicini. Cartago : A framework for prototyping artifact-based environments in mas. In *E4MAS*, pages 67–86, 2006.
78. F. Sadri. Using the KGP model of agency to design applications. In *CLIMA VI*, volume 3900, pages 165–185. Springer, 2005.
79. S. Sardina, G. De Giacomo, Y. Lespérance, and H.J. Levesque. On the semantics of deliberation in indigolog from theory to implementation. *Annals of Mathematics and Artificial Intelligence*, 41(2-4):259–299, 2004.
80. A. El Fallah Seghrouchni and A. Suna. CLAIM and SyMPA: A programming environment for intelligent and mobile agents. In *Multi-Agent Programming: Languages, Platforms and Applications*, pages 95–122. Kluwer, 2005.
81. S. Shapiro and G. Brewka. Dynamic interactions between goals and beliefs. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2625–2630, 2007.
82. Y. Shoham. Agent-oriented programming. *Artificial Intelligence*, 60:51–92, 1993.
83. V. Torres Silva. From the specification to the implementation of norms: an automatic approach to generate rules from norms to govern the behavior of agents. *JAAMAS*, 17(1):113–155, 2008.
84. J. Sudeikat, L. Braubach, A. Pokahr, W. Lamersdorf, and W. Renz. Validation of BDI agents. *ProMAS 2006*, pages 185–200, 2007.
85. K. Hindriks M. Dastani R. Bordini J. Hubner A. Pokahr T. Behrens, J. Dix and L. Braubach. An interface for agent-environment interaction. In *The Eighth International Workshop on Programming Multi-agent Systems (ProMAS10)*. Springer, 2010.

86. M. Tasaki, Y. Yabu, Y. Iwanari, M. Yokoo, M. Tambe, J. Marecki, and P. Varakantham. Introducing communication in dis-pomdps with locality of interaction. *Web Intelligence and Intelligent Agent Technology, IEEE/WIC/ACM International Conference on*, 2:169–175, 2008.
87. N. Tinnemeier, M. Dastani, and J.-J. Ch. Meyer. Roles and norms for programming agent organizations. In Decker, Sichman, Sierra, and Castelfranchi, editors, *Proceedings of the Eight International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, pages 121–128. IFAMAAS/ACM DL, 2009.
88. N. Tinnemeier, M. Dastani, and J.-J. Ch. Meyer. Programming norm change. In van der Hoek, Kaminka, Lespérance, Luck, and Sen, editors, *Proceedings of the Ninth International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, pages 957–964. IFAMAAS/ACM DL, 2010.
89. N. Tinnemeier, M. Dastani, J.-J. Ch. Meyer, and L. van der Torre. Programming normative artifacts with declarative obligations and prohibitions. In *Proceedings of IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology*, pages 145–152. IEEE Computer Society, 2009.
90. M. B. van Riemsdijk, M. Dastani, J.-J. Ch. Meyer, and F. S. de Boer. Goal-Oriented Modularity in Agent Programming. In *Proceedings of AAMAS'06*, pages 1271–1278, 2006.
91. G. Vigueras and J. A. Botía. Tracking causality by visualization of multi-agent interactions using causality graphs. *ProMAS 2007*, pages 190–204, 2008.
92. G. Weiss. *Multiagent systems. A modern approach to distributed artificial intelligence*. The MIT Press, 1999.
93. D. Weyns, H. Van Dyke Parunak, F. Michel, T. Holvoet, and J. Ferber, editors. *Environments for Multiagent Systems State-of-the-Art and Research Challenges*, volume 3374 of *Lecture Notes in Computer Science*. Springer, 2004.
94. M. Winikoff. JACK^{TM} intelligent agents: An industrial strength platform. In *Multi-Agent Programming: Languages, Platforms and Applications*. Kluwer, 2005.
95. M. Winikoff. Assurance of agent systems: What role should formal verification play? In Mehdi Dastani, Koen V. Hindriks, and John-Jules Meyer, editors, *Specification and Verification of Multi-agent Systems*, pages 353–383. ACM Press, 2010.
96. M. Winikoff, L. Padgham, J. Harland, and J. Thangarajah. Declarative and procedural goals in intelligent agent systems. In *Proceedings of the eighth international conference on principles of knowledge respresentation and reasoning (KR2002)*, Toulouse, 2002.
97. M. Wooldridge. *An Introduction to MultiAgent Systems (second edition)*. Wiley, 2009.
98. F. Zambonelli, N.R. Jennings, and M. Wooldridge. Developing multiagent systems: The Gaia methodology. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 12(3):317–370, 2003.

On the Engineering of Multi Agent Organizations

Virginia Dignum[1], Huib Aldewereld[2], and Frank Dignum[2]

[1] Department of Technology, Policy and Management, Delft University of Technology, The Netherlands
[2] Department of Computer Science, Utrecht University, The Netherlands

Abstract. In this paper we discuss the different approaches in modelling and implementing organizational concepts for Multi Agent Systems. We argue that each approach has its own advantages and disadvantages. It depends on characteristics of the domain which approach suits best. We give a rough sketch of some heuristics of choosing an agent organization approach based on the domain characteristics and also show implementation consequences of choosing a particular approach.

1 Introduction

The increasingly complex requirements of software systems are originating a growing interest in organizational approaches to develop complex and distributed software systems. The use of societal concepts, such as organizations and norms, is commonly accepted as a suitable modeling approach in Multi Agent Systems (MAS) such as described in [14].

Traditionally, MAS considers organisations from an agent perspective, where the focus is on defining the organisational capabilities that agents should possess in order to act within an organization. In this sense, the organisation is *implicit* in the agents' specification. The *explicit* definition of the organisational structure of the domain, as advocated by agent organization approaches (cf. section 2) provides abstractions to describe positions, relationships, and norms or rules, independently from the agents' design. Implicit organisations of agents can be seen as a loosely-coupled network of problem-solvers [26], where each agent must be equipped with the social knowledge necessary to behave in the system. In explicit organisations, the system is capable to coordinate agent activity even when agents themselves are not aware of the social rules of the system. According to the literature, organisational models can moreover be conceived along two main points of view: agent centered (also called individual, internal or emergent perspective) and system centered (also called prescriptive, organisational, external or institutional perspective) [24, 34, 35].

The organisational view mostly originates from work on organisational theory, business process reengineering, and task analysis. This view is particularly relevant to situations where it is needed to control or prescribe the behavior of a society. In short, the organizational stance is meant to control and prescribe systems by describing desired behavior on a high level of abstraction. In most cases,

global processes (objectives, tasks, input / output) will be described explicitly and distributes top-down to guide the collective behavior of individuals. In this view, if individuals are described, they are likely to be individual abstractions (e.g. roles, functions) specified to conform to the organizational objectives.

The individual view originates from social science and anthropology. This view is mainly meant to provide understanding or to describe the behavior of a society. This individual view makes collective behavior emerge bottom-up from the individual's behavior, and describes actual behavior on a high level of detail. Global objectives will be implicit in the practices and activities of individuals.

In this paper, we argue that these two different approaches to organisations are both valuable to develop systems and maintain that the choice for one or another is determined by its fit to the characteristics of the application domain. Taking organizational theory as a starting point we identify a number of domain characteristics such as locus of control, existence of global goals, emergence, dynamicity of the domain, etc. As such, this approach can be used as a starting point in determining what type of MAS should be developed for which type of situation (and thus can co-determine agent methodology, platform, etc.).

The paper is organised as follows. In section 2 we position different existing organization-based approaches to MAS along these organization dimensions. In section 3 we link domain characteristics and organizational approach and in section 4 we discuss the implementation complexity of the different frameworks and the required/assumed agent attitudes towards the framework. Finally, we present our conclusions in section 5.

2 Related Work / Organization vs. Autonomy

Multi-agent organizations have been motivated as a proper way to deal with (coordination) problems that can arise from agent autonomy, especially in open MAS [21], where we do not know what kind of agents will enter into the system [13, 23]. In this context, the organization can be seen as a set of behavioural rules accepted and adopted by a group of agents to facilitate interaction and the achievement of individual and global objectives. Many approaches to organizing agent systems have been proposed in MAS research varying on explicitness of the organizational representation and whether the organization is represented internally or externally. See Figure 1 for an overview of some well-known frameworks.

The horizontal implicit-explicit axis represents the amount of (explicit) representation of the organizational model. Approaches on the far left have little to no representation of the organization. The organization is emergent from the behavior of the agents or is derivable from the concepts used in agents' programming. The approaches on the far right have the organizational model completely represented, separate from the agents. The vertical internal-external axis depicts the level of externality of the representation. Approaches on the bottom require (parts of) the organizational model implemented within the agents, whereas the

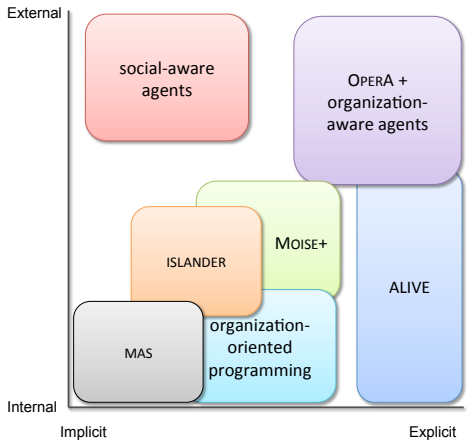


Fig. 1. Classification of organizational frameworks

approaches on the top end have a (common) shared representation of the organization.

In the next section we look at how the position within this diagram influences the implementation of a system in that framework and how this classification can be used to select the best framework for solving a problem domain. First, let us look in a bit more detail to the frameworks in Figure 1 to see why they are in that position.

MAS. 'Traditional' multi-agent systems (e.g., GAIA [36], PROMETHEUS [27]) only use organizational concepts in the mind of the designer; that is, any organizational elements used for the coordination of a traditional MAS are translated by the designer into agent code. Intrinsically, organizational changes or changes to organizational 'structure' (if it can be called as such) require reprogramming of all the agents in the system. It is, in that sense, the absolute cornerstone of our diagram, because the organization is both fully internal (represented in the agents' code) and implicit (all organizational concepts are translated into coordination methods/functions within the agent).

Organization-oriented programming. A recently proposed change in the agent programming paradigm [29] allows for the use of organizational concepts as programming constructs, similar to the idea of using agent concepts as programming constructs (in contrast to object-oriented concepts, for example). This work claims that these changes should make it easier to program agents that are organizationally adapt, since the agents can be developed with programming elements on the same level of abstraction. This makes the approach more explicit than the 'traditional' MAS approaches, yet keeps the organizational definitions internal to the agents, thus still requiring reprogramming the agents to accommodate organizational changes.

ISLANDER. The ISLANDER 'family' of organization specifications is more explicit than the previous frameworks in that it has a single explicit rep-

resentation of the organization (the ISLANDER specification, [17]). It uses roles and norm descriptions to structure the MAS. In [32] the specification was used to generate a specific type of agents suitable to enact the specification, resulting in an internal representation. In the more recent version called AMELI [18], the specification is used as a definition of a platform where agents can, through governors, join and participate in the system. Although the organizational specification is no longer required to be a part of the agents, the governors need to be specifically engineered to accomodate the organization (the specification is now a part of them). Due to the restriction that governors filter all actions performed by a participating agent and only allows the performance of the correct action, the agents still need a substantial amount of knowledge of the organization's inner workings.

MOISE$^+$. MOISE$^+$ [23] uses a similar approach to AMELI in filtering the actions of the agents participating in the organization. AMELI called them governors, MOISE$^+$ calls them OrgBoxes, which facilitates the interactions between the agent and the organization (including interactions between agents in the same organization). We consider MOISE$^+$ to be slightly more explicit than ISLANDER due to the more extensive normative element. ISLANDER captures most organizational norms in its interaction definitions, which, in principle, cannot be violated (due to the use of governors that keep the agents 'on the right track'). MOISE$^+$, on the other hand, has a much more prominent place for deontic norms, which has the benefit of giving the agents more freedom in choosing possible actions (thus increasing the autonomy of the participants).

ALIVE. The ALIVE framework [19] uses an OperA-based organizational model [13, 1] that is used to define and generate the code of the enacting agents. The difference between ALIVE and the approach used in [32] is that the organizational model remains available to the system after the agents have been generated, that is, the model is not just used to generate the code of the agents, but the agents are also enabled to read and use the organizational model at runtime (e.g., the agents used the organizational model at runtime to validate their plans with respect to the organizational norms). Moreover, due to the Model-Driven Architecture approach used in ALIVE, explicit links are available between agent elements and organizational elements. In this sense, the organizational model is more explicit than in the ISLANDER approach. Concerning internal-external representation, the model is used in both ways since it is used to generate the agents (internal representation) as well as used as guideline during runtime (external representation).

Organization-aware agents. Recently an approach has been proposed [30, 31] where the agents are enabled to read and understand organizational models.In this sense, the organization is defined separate from the agents, and makes no assumptions about which agents enact what roles, making it extremely explicit. It is also more external than the other approaches as the organization only exists as a (shared) specification between the agents. The agents need the capabilities to understand the organization specification (e.g. in [30] the OperA model is used), but do not need to be programmed

with parts of the organization model a priori. Moreover, the agents need to be extended with capabilities for role enactment, that is, to determine whether they are capable and/or interested to play a particular role in the organization.

Social-aware agents. The social-aware agents shown in the top-left corner of Figure 1 corresponds to the (philosophical) idea of agents that can determine social context and work in environments of social emergence (for instance, see [9, 10, 12]). The agents of such a system derive their intended role in the 'organization' from their social context, meaning that the organization is more or less emergent and thus not a part of the agents themselves. The organization is also not represented in any form, but emerges from agent interactions (or is derived from the behaviors of the agents). No current implementation of such systems is known (at this time).

3 Deciding on Organization Design

In this section, we discuss the approaches to organization in MAS presented in section 2, their characteristics and aims and their suitability to different problem domains as identified in [15].

An important issue that must be taken into account in order for agent technology to be taken up by the main computer industry is the means to perform domain analysis that informs the decision of using a specific type of agent system and a design methodology fitting with that type of agents.

Several attempts have been made already to give a comprehensive classification of agent types (see e.g. [20]) and of intrinsic characteristics of agent systems, such as autonomy [6] or interaction [28]. However, these classifications are made with the purpose to classify the agent systems based on technical features. E.g. mobile agents versus static agents. Instead of using technical aspects as starting point to distinguish agent types and methodologies we use organizational and human social phenomena [8, 16, 2], because the design of MAS resembles that of human organizations in many respects. In both cases we want to coordinate a number of distributed, autonomous entities in such a way that the system as a whole will optimally function in its environment. In human organizations, design mostly adopts a multi-contingency view [5], which says that an organization's design should be chosen based on the multi-dimensional characteristics of a particular context. These characteristics include structural (goals, strategy and structure) and human (work processes, people, coordination and control) components.

The characteristics of a domain should therefore lead the decision process towards the choice of a system architecture. As such, we posit that the development and analysis of system design should follow similar dimensions to those used in organizational theory (OT). Based on OT research, we group domain characteristics into different design dimensions described in the remainder of this section. Each dimension should be interpreted as a sliding scale where the current (or the desired) situation of an application domain can be plotted between the two extreme values.

In [15], we apply this classification to the analysis of concrete applications developed in practice, identified through a survey and literature research. In the following, we briefly describe the main characteristics of domains that influence the choice of organization approach.

Complexity of a domain refers to the number of factors and their interdependencies that must be considered for operation in a domain. In domains with *low* complexity, only a few factors need to be considered, and they have few interdependencies. In *highly* complex domains, even minor isolated changes can provoke major changes in the total system.
Uncertainty refers to the level of understanding of the environment behavior. *Uncertain* contexts are characterized by a lack of information, limited capabilities, ambiguity and unpredictability. Uncertainty means that forecasting is less accurate and the future is unclear. *Predictable* domains are easier to forecast and the effects of particular courses of action are likely to be understood beforehand.
Environment is the space in which the system exists. *Open* environments pose no restriction on the agents joining the system. On the other extreme, in *closed* environments, only agents whose behavior is fully known can join the system. Agents in closed systems are often explicitly designed to cooperate towards a common goal and are often implemented in combination with the whole system [37].
Emergence is the arising of patterns, structures, or properties that cannot be explained by the system's components and their interactions. *Emergent* systems can be seen as conglomerations of single entities with hardly any fixed interaction or explicit social structure. There is no notion of common goals or plans, and entities are free to enter or not in interaction with others. In *designed systems* structure is determined by organizational design, which is independent of the entities themselves. Such structures implement the idea that interactions occur not just by accident but aim at achieving some desired global goals [16].
Goal Autonomy. Autonomy is often taken as a defining property of agents. However, in many cases, people, groups, and departments are not really completely autonomous, in the sense that they are not free to determine their objectives but only their plans on how to reach those objectives. Castelfranchi has separated autonomy from agenthood [7]. Goal autonomy means that individual entities are able to reason about which goals to adopt and fulfill, and are thus able to determine whether or not to accept a certain position in an organization, by reasoning about the goals associated with that position.
Control activities involve decisions on when to invoke and the amount of effort to put into scheduling and coordinating domain activities. In *centralized* control, control decisions are specific for one or more roles in the organization. It determines different levels of autonomy for agents. Distributed or *decentralized* control means that (all) roles are collectively responsible for decisions which are achieved by collaboration or consensus [3].

Table 1 gives an overview of the domain characteristics associated with different approaches to agent organizations. In this table, we summarize the characteristics of the application domain that require different agent system types.

This approach is initial and will require further evaluation and refinement in order to provide fully methodological guidance for system choice. Note that this only describes 'ideal' situations, when all values of a given domain will fall into the same column in table 1 and the domain is thus a perfect fit to that type of agent system. In reality, most situations will fall into different types. A guideline to the evaluation of misfits is to consider the relative 'importance' of the characteristic for the determination of the system type. Some domain characteristics can be seen as stronger indicators of an agent system type than others.

Domain Features	**Emergent Organization**	**Implicit Organization**	**Explicit Organization**
Complexity	Low	Medium	High
Uncertainty	High	Low	High
Openness	-	Closed	Open
Social Configuration	Emergent	Designed	Variable
Goal Autonomy	- (agents have no self reflection)	No (agent goals include organizational goals)	Yes (negotiate conditions)
Locus of Control	Local	Local	Global
Example frameworks	ADELFE [4] MASS [25]	Prometheus [27] Gaia [36]	OperA [1] MOISE$^+$ [23] AMELI [18]

Table 1. Agent system choice guidelines.

4 Engineering Organizations

In the previous sections we have compared different frameworks with respect to the externality and explicitness of the organization model. We showed that this has an effect on the sort of domains where the framework works best. In this section we look at the implementation complexity of the different frameworks and the required/assumed agent attitudes towards the framework.

4.1 Programming Agents

The traditional ('pre-2000') method of implementing complex MAS with organizational aspects was through enriching the agents' code with the coordination aspects needed to have organization(-like) behavior. A limitation of these approaches (as argumented by, e.g., [33]) is the fact that these traditional methods are typically assuming systems to be closed; that is to say, all the agents in the system are to be designed and implemented either 1) by the same developer, or 2) according to and within fixed specifications that enable the organizational aspects. For that fact, all agents are required to be homogeneous with respect to coordination aspects.

All aspects of the organization (which role the agent plays, how the roles are to interact, what to expect from other roles, etc.) have to be implemented in the agents. Because traditional agent programming paradigms do not include organizational concepts in the programming language itself, all of these mentioned organizational aspects have to be translated to existing language constructs (e.g., beliefs, goals, planning rules).

Since the agents that will enact the organization have to be programmed anyway (regardless of the framework chosen), one can consider the implementational burden of organizational MAS to be rather low. During the "normal" design phase of the agents, the organizational aspects are envisioned and incorporated into the agents' code. The major disadvantage of these approaches becomes apparent when looking at changing the organization at a later stage. All of the organizational aspects are deep-rooted within the agents' code. Moreover, many of the organizational aspects can be hard to locate since they had to be translated to beliefs, goals, etc. Finding and revising the organization in the MAS thus comes close to a major code rewrite of the MAS itself.

4.2 Organization through Interface

In pursuit of more open agent systems the coordination aspects (organization) of MAS was moved outside the agents. Several frameworks propose a interface component, or interface agent, to regulate the activity of agents within the organizations they belong to. Such approaches support external explicit specification of organizations but limit the activities of the agents within the organization to those described and allowed by the organization. When an agent desires to (1) adopt a role, (2) send a message to another agent in the organization, or (3) otherwise interact with the organization, it has to ask this service form the interface. This is the case of the OrgBox in S-Moise$^+$, Teamcore proxy in KARMA, and governor in AMELI.

This interface is used to mediate the participation of an external agent within the institution. This enables operation in open environments where heterogeneous agents can join the organization. Organizations are then composed of interface agents and internal agents. Interface agents can communicate with external agents while the rest of roles in the organization are not accessible to external agents. In order for agents to communicate with their interface, they are solely required to be capable of opening a communication channel. Interface agents control the activity of external agents in the organization allowing or blocking agent requests as they meet or not organizational aims. As such agents are not free to decide on the violation of organizational norms. The interface agent can also respond to information requests, monitor agents' obligations and inform about the events the agent must be aware of within the organization.

While each of these tries to move the organization outside of the agents, they each still require that the agent has large amounts of knowledge about the inner-organizational workings to function within the organization at all [11]. Typically, incorrect behavior and erroneous interactions are ignored by the middleware. This ranges from interactions outside the scope of the organization to behavior

that leads to the violation of norms and even interactions that could be correct but are not expected by the specification at that time.

In that respect the implementation requires the agents to have sufficient knowledge about the organizational working. The agents need to know exactly what is expected of them, and at which time, to (successfully) participate in the organization. While this is to be expected to an extend, the implementations typically are too restrictive and provide outside agents with too little information to function correctly. While the control over the agents in the organization has shifted to the middleware, the organization itself (the objectives, the roles, the role interactions) are still very much a part of the agents.

The change to organizations in the middleware has benefits over the traditional approaches since changes to the organization are easier to make. The organization is represented more externally in the middleware and usually in organizational concepts. This makes finding the right components to change easier. However, since the organization is also embedded in the agents, these have to be changed as well! The organizational middleware thus gained us a more explicit representation (useful to see design choices at a later stage), but also increased the implementational burden (one now has to design and maintain both the agents and the middleware).

4.3 From Explicit Organization to Agents

In figure 1 we discussed two approaches with an explicit modeling of the organization; ALIVE [19] and organization-aware agents [30]. In the following we take the OperA framework as example of explicit modelling of organization. OperA assumes a clear separation between the agents in the system and the roles in the organization. In this way, the organization abstracts from the actual participants and only describes the aims and concerns of the organization with respect to the social system. The two approaches mentioned earlier differ on the externality of the organization; we look at each in more detail below.

Generating Agents. The ALIVE approach [19] builds on principles from Model Driven Engineering (MDE) by using meta-models and model transformations to generate agent-code based on organizational specifications. The defining characteristic of MDE is the use of models to represent the important aspects of the system, be it requirements, use cases, or implementation-level artifacts such as code. The Model Driven Development promotes the automatic transformation of abstracted models into specific implementation technologies, by a series of predefined model transformations.

Following the MDE approach, depicted in the left of Figure 2 a transformation is defined between the organizational meta-model (the meta-model of OperA in the ALIVE approach) and the meta-model of a specific MAS architecture. Using this transformation and a domain specific organizational model, a MAS can be generated that complies to the particular MAS meta-model and implements the organizational model defined in the OperA model.

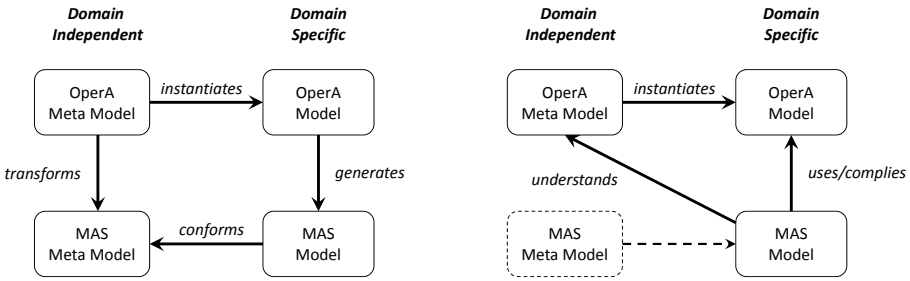


Fig. 2. MDE to generate agents (left) and for use by organization-aware agents (right).

A limitation of this approach stems from the high-level of abstraction of the OperA model used. OperA only describes the aspects of the system relevant to the organization, often leaving out details about concrete interactions (protocols, plans) and domain specific elements (actions, capabilities). In order to have a complete implementation, these details have to be added (to the MAS, or to an intermediate model). In a sense, one is modeling both the organization (in OperA) and the MAS separately, thus increasing the implementational complexity and burden.

Moreover, since the OperA model is transformed into a MAS, changes to the organization require a rewrite of the MAS (like in the traditional approaches, all details will be deep-rooted in the agents' code). However, due to the use of MDE, the organization and all the links between the organization and the MAS have been explicitly modeled, making the rewrite quite simple; after making changes to the organization, a re-transformation of the organizational model to the MAS should make most of the revisions required.

Organization-Aware Agents. The other approach is by using organization-aware agents [30, 31]. This approach requires agents that are able to understand the OperA meta-model such that they can enact roles in an organization defined by a domain-specific model (cf. Figure 2 (right)). Agents who want to enter and play roles in an organization are expected to comprehend and reason about the organizational specification, if they are to operate effectively and flexibly in the organization. This implies that the agents should have reflective capabilities with respect to their own goals, beliefs, perceptions and action potential. In [30] it is investigated how GOAL [22] agents can determine whether they have the necessary capabilities to play roles in an (OperA) organization.

Since no implementation currently exists, it is hard to name all the advantages and disadvantages of this approach compared to those previously mentioned. A major concern could be that the development of agents that are organization-aware takes a lot of effort, but on the other hand it can be raised that these agents should be extremely flexible and should be able to be reused in various different organizations after very few changes.

5 Conclusions and future work

In this paper, we have explored how the concept of organization can be used to engineer MAS. An advantage of considering organizations as a first order concept, is that we can study the global aspects of system independently from the particulars of the individuals involved, i.e. without having to give formal accounts of the specific way an individual is designed and motivated.

We have also indicated the different approaches to representing organizations for MAS. Rather than advocating a particular approach as being the most desirable, we have shown that different kind of domains call for different approaches of organization based MAS. Each approach has its own advantages and disadvantages. It depends on the domain which of these weigh more heavily.

Finally, we have discussed some consequences for the implementation of organization-based MAS. Also here we see that each approach requires a large implementation effort. However, the effort is located in different aspects. Thus it depends which aspects are most likely to change or can be reused whether the implementation effort of a certain approach is cost effective. If an organization is not expected to change maybe it is best to implement the organization implicit in the agents. However, if agents enter and leave many (interrelated) agent organizations one might opt for organization aware agents that require little effort to be adapted for each organization (but have a large start up implementation cost).

The ideas presented in this paper show several directions for further research. Most important is to come up with a well founded methodology to determine which type of organization approach is best suited for which type of domain. A first step has been presented, but rigorous experiments should be performed in practice to verify the assumptions.

The idea of organization aware agents has also recently been presented but no implementation has been given. This is necessary to evaluate the overhead of building such agents and whether this might not proof too much for the average application.

References

1. Huib Aldewereld and Virginia Dignum. OperettA: Organization-oriented development environment. In *Proceedings of the 3rd International workshop on Languages, Methodologies and Development Tools for Multi-agent Systems (LADS2010@Mallow)*, 2011.
2. A. Artikis and J. Pitt. A formal model of open agent societies. In *Proc. Autonomous Agents*, pages 192–193. ACM Press, 2001.
3. K. S. Barber and C. E. Martin. Dynamic reorganization of decision-making groups. In *Proceedings of the 5th Autonomous Agents*, 2001.
4. C. Bernon, V. Camps, M. Gleizes, and G. Picard. Engineering adaptive multi-agent systems: The adelfe methodology. In *Agent-Oriented Methodologies*, pages 172–202. Idea Group, 2005.
5. R. Burton, G. DeSanctis, and B. Obel. *Organizational Design: A step by step approach*. Cambridge University Press, 2006.

6. C. Carabelea, O. Boissier, and A. Florea. Autonomy in multi-agent systems: A classification attempt. In M. Nickles, M. Rovatsoso, and G. Weiss, editors, *Autonomy-2003*, volume 2969 of *LNAI*, pages 103–113. Springer, 2004.
7. C. Castelfranchi. Guarantees for autonomy in cognitive agent architecture. In *ATAL'94*, volume 980 of *LNAI*. Springer, 1995.
8. C. Castelfranchi. Engineering social order. In A. Omicini, R. Tolksdorf, and F. Zambonelli, editors, *Enginering Societies in the Agents World*, LNAI 1972, pages 1–19. Springer, 2001.
9. Cristiano Castelfranchi. Formalising the informal? dynamic social order, bottom-up social control, and spontaneous normative relations. *Journal of Applied Logic*, 1:47–92, 2003.
10. R. Conte and F. Dignum. From social monitoring to normative influence. *Journal of Artificial Societies and Social Simulation*, 4(2), 2001.
11. F. Dignum, V. Dignum, J. Thangarajah, L. Padgham, and M. Winikoff. Open agent systems??? In L. Padgham and M. Luck, editors, *Agent-Oriented Software Engineering (AOSE'07)*, volume 4951 of *LNAI*, pages 75–89. Springer, 2007.
12. F. Dignum, D. Morley, E. Sonenberg, and L. Cavedon. Towards socially sophisticated BDI agents. *International Conference on Multi-Agent Systems*, page 0111, 2000.
13. V. Dignum. *A Model for Organizational Interaction: based on Agents, founded in Logic*. SIKS Dissertation Series 2004-1. Utrecht University, 2004.
14. V. Dignum, editor. *Handbook of Research on Multi-Agent Systems: Semantics and Dynamics of Organizational Models*. Information Science Reference, 2009.
15. V. Dignum and F. Dignum. Designing agent systems: State of the practice. *International Journal on Agent-Oriented Software Engineering*, 4(3), 2010.
16. V. Dignum, F. Dignum, and J.J. Meyer. An agent-mediated approach to the support of knowledge sharing in organizations. *Knowledge Engineering Review*, 19(2):147–174, 2004.
17. M. Esteva, D. Cruz, and C. Sierra. Islander: an electronic institution editor. In *Proceedings of the International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2002)*, 2002.
18. M. Esteva, B. Rosell, J. A. Rodrìguez-Aguilar, and J. Ll. Arcos. AMELI: An agent-based middleware for electronic institutions. In *AAMAS'04*, pages 236–243. ACM Press, 2004.
19. European Commission FP7-215890. ALIVE, 2009. `http://www.ist-alive.eu/`.
20. S. Franklin and L. Gasser. Is it an agent, or just a program?: A taxonomy for autonomous agents. In J. Müller et al., editor, *Intelligent Agents III*, pages 21–35. Springer-Verlag, 1997.
21. C. Hewitt. Open information systems semantics for distributed artificial intelligence. *Artificial Intelligence*, 47:79–106, 1991.
22. Koen V. Hindriks. Programming rational agents in GOAL. In Rafael H. Bordini, Mehdi Dastani, Jürgen Dix, and Amal El Fallah Seghrouchni, editors, *Multi-Agent Programming: Languages, Tools and Applications*. Springer, Berlin, 2009.
23. J. Hübner, J. Sichman, and O. Boissier. S-Moise+: A middleware for developing organised multi-agent systems. In O. Boissier et al., editor, *COIN I*, volume 3913 of *LNAI*, pages 64–78. Springer, 2006.
24. J. Hübner, J. Sichman, and O. Boissier. Developing organised multi-agent systems using the moise+ model: Programming issues at the system and agent levels. *International Journal on Agent-Oriented Software Engineering*, 1(3/4):370–395, 2007.

25. M. Ivanyi, L. Gulyas, R. Bocsi, V. Kozma, and R. Legendi. The multi-agent simulation suite. In *Emergent Agents and Socialities: Social and Organizational Aspects of Intelligence (AAAI Fall Symposium Series 2007)*. AAAI, 2007.
26. G. O'Hare and N. Jennings. *Foundations of Distributed Artificial Intelligence*. Wiley, 1996.
27. Lin Padgham and Michael Winikoff. Prometheus: a practical agent oriented methodology. In B. Sellers and P. Giorgini, editors, *Agent Oriented Methodologies*, pages 107–135. Idea Group, 2005.
28. V. Parunak, S. Bruekner, M. Fleitscher, and J. Odell. A design taxonomy of multi-agent interactions. In P. Giorgini, J. Muller, and J. Odell, editors, *AOSE III*, volume 2935 of *LNAI*. Springer, 2003.
29. Nick Tinnemeijer. *Organizing Agent Organizations*. SIKS Dissertation Series 2011-02. Utrecht University, 2011.
30. Birna van Riemsdijk, Virginia Dignum, Catholijn Jonker, and Huib Aldewereld. Programming role enactment through reflection. In *Proceedings of the Joint International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-2011)*. 2011.
31. M. Birna van Riemsdijk, Koen V. Hindriks, and Catholijn M. Jonker. Programming organization-aware agents: A research agenda. In *Proceedings of the Tenth International Workshop on Engineering Societies in the Agents' World (ESAW'09)*, volume 5881 of *LNAI*, pages 98–112. Springer, 2009.
32. W. Vasconcelos, J. Sabater, C. Sierra, and J. Querol. Skeleton-based agent development for electronic institutions. In *Proceedings of AAMAS02, First International Conference on Autonomous Agents and Multi-Agent Systems*, pages 696–703. ACM Press, July 2003.
33. J. Vázquez-Salceda. *The Role of Norms and Electronic Institutions in Multi-Agent Systems. The HARMONIA framework*. Whitestein Series in Software Agent Technology. Birkhäuser Verlag, 2004.
34. J. Vzquez-Salceda, H. Aldewereld, and F. Dignum. Norms in multiagent systems: Some implementation guidelines. In *EUMAS*, 2004.
35. G. Weiss, M. Nickles, M. Rovatsos, and F. Fischer. Specifying the intertwining of cooperation and autonomy in agent-based systems. *International Journal of Network and Computer Applications*, 29, 2006.
36. M. Wooldridge, N. Jennings, and D. Kinny. The Gaia methodology for agent-oriented analysis and design. *Journal of Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, 2000.
37. F. Zambonelli, N. Jennings, and M. Wooldridge. Organizational abstractions for the analysis and design of multi agent systems. LNAI 1957, pages 235–251. Springer, 2001.

Institutions as a Basis for Service Engagements

Munindar P. Singh

Department of Computer Science
North Carolina State University
Raleigh, NC 27695-8206, USA

singh@ncsu.edu

Despite many advances, IT today fails to adequately address some of the most basic needs of business. The expansion of the web and the ready availability of information have raised expectations that today's approaches are ill-suited to address. Specifically, businesses need to initiate and participate in service engagements for virtually any business function.

For example, consider an enterprise that needs to create and launch a new product in the marketplace. Realizing this basic business function involves a slew of services ranging from market analysis to requirements to design to manufacturing to delivery to customer relationship management. In today's business environment, typically, most if not all of these services would be provided to the enterprise by external parties. In turn, such parties might initiate additional service engagements. The mutual business relationships are complex and dynamically changing. The net effect is that a complex business ecosystem comes into being.

We consider the problem of effectively modeling and enacting service engagements involving two or more autonomous, heterogeneous entities. These entities are best thought of as offering business services, in contrast with the technical web or grid services, which have garnered most research attention from computer scientists. Such service engagements arise commonly in today's information environments, yet conventional techniques are not adequate for handling them.

We propose a new interaction-oriented approach that addresses how to administer business service engagements. Our approach is based on the idea of institutions, inspired by the study of human organizations and institutions, and formalized in terms of the relationships among the participants in such settings. Doing so enables us to provide a clear and natural (to stakeholders) way to specify service engagements, highlighting the interactions among the participants. We consider practical use cases demonstrating the flexibility of our approach.

Engineering Coordination: Selection of Coordination Mechanisms

René Schumann

National Institute of Informatics,
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan
schumann@nii.ac.jp

Abstract. Reuse of code and concepts is a driving force of agent-oriented software engineering (AOSE). In the field of AOSE the probably most recognized types of reuse are agent frameworks and the FIPA standards. To support developers of multiagent systems it is also necessary to foster reuse of mechanisms like coordination. In particular we address in this article the selection of effective and efficient mechanisms for the coordination of plans among autonomous agents. We will detail research done in the field of AOSE concerning the reuse of concepts and focus on the selection of suitable mechanisms. The selection for coordination mechanisms is, up to now, not covered in AOSE sufficiently. Therefore, we present the ECo-CoPS approach that defines a structured process for the selection of coordination mechanisms for autonomous planning systems, where the local autonomy, as well as, the existing planning systems can be preserved. A case study is presented to detail how the ECo-CoPS approach can foster the selection process.

1 Motivation

Reuse of code and concepts is a driving force in software engineering in general and in AOSE in particular. In the field of AOSE the probably most recognized types of reuse are agent frameworks and the FIPA standards [7]. In this article we address the reuse of concepts on a higher level. In particular we are focusing on the reuse of coordination mechanisms for plans among autonomous agents. Each agent is representing a particular sub-plan, that needs to be coordinated with the other sub-plans. Within each plan the future actions of an agent, or the activities of the entity the agent represents, are fixed. Typically the generation of each of these sub-plans is characterized by its computational complexity and the need for information concerning goals and resources of the particular agent. The reuse of coordination mechanisms has been supported by mediating agent infrastructures like TuSCoN [16]. In these infrastructures coordination mechanisms are embedded into the environment as coordination artifacts that can be used by the agents. This is a valuable approach for reusing coordination mechanism, but some prerequisites have to be fulfilled to use these artifacts. In fact, it is necessary for the agent to reveal all planning relevant information to the infrastructure, to allow for an externalized coordination, and ensure that the

global, as well as, the local plans are feasible. This can be a serious limitation for the application of coordination artifacts. It limits the autonomy of the agents, because a significant aspect of an autonomous agent is its ability to determine its future actions, and so its local plan. Thus coordination artifacts cannot be applied to scenarios where autonomous planning agents have to coordinate their local plans to archive a joint global goal, as they have to maintain their local autonomy and they are responsible to maintain their local goals.

The field of coordination mechanisms among autonomous agents and their plans has attracted numerous researchers. Therefore, a variety of different coordination mechanisms have been proposed. Unfortunately, if a particular situation is given for which we have to select an appropriate coordination mechanism, AOSE cannot provide any guidance and therefore the decision mainly relies on the background of the developer. We have surveyed the proceedings of the previous AOSE workshops and other AOSE related literature. Moreover, we have surveyed specific reuse centered research in the field of software engineering, like the proceedings of the International Conference of Software Reuse[1]. The reuse specific research in software engineering has addressed specifically the selection of commercial off-the-shelf software. The field of AOSE has not been recoginized by those reseacher, up to now. In the field of AOSE ifself, the work published addressing the selection of existing concepts for reuse is considerable small.

In the following we will discuss work we have identified as relevant. Then we will present the ECo-CoPS approach (Section 3), which has been developed to select an effective and efficient coordination mechanism for autonomous agents, that have to coordinate their plans. The ECo-CoPS approach support developers by providing a structured decision making process and offers tooling supporting the process. In Section 4 we present an example how the process can be used. Finally we summarize our findings and outline future research.

2 Reuse of concepts in AOSE: An overview

An established way to reuse concepts in software engineering is to define and use patterns, e.g., the well-known design patterns by Gamma et al. [8]. For the field of AOSE Lind [15] suggested a format for agents oriented patterns and presents an architectural and an interaction protocol as examples. Architectural patterns, like Broker, Moderator, and Wrapper, have also been presented by Heyden et al. [10]. Interaction patterns, like patterns for Subscription or Call for proposals have been discussed by Kolp et al. [14], as well. Those authors termed these patterns *social patterns*. The authors present a framework for describing those patterns in a unified way, that has been specialized for agent-based development. Those architectural and interaction patterns have been standardized by the FIPA [7]. A wider scope of patterns in AOSE has been proposed by Sauvage [17]. Sauvage distincts between MetaPatterns, that describe abstract constructs for the design of agent-based systems. He introduces organizational schemes,

[1] For an overview of the proceedings see `http://www.isase.us/pastconferences.htm`, Accessed: 02/04/2011.

like organizations and roles, and protocols as two meta-patterns. The second group of patterns are so-called metaphoric patterns. Sauvage mentioned marks, like pheromones, and influences as two metaphoric patterns. The third class of patterns are architectural patterns, addressing the architecture of agents.

Design patterns for self-organizing systems have been summarized by Gardelli et al. [9]. The patterns are collected from the design of nature-inspired self-organizing systems. For instance, patterns addressing the evaporation, aggregation, and diffusion of pheromones are presented. The identification of particular design patterns for the coordination in self-organizing systems is addressed in the article by de Wolf and Holvoet [20]. The authors present two design patterns for coordination. These techniques are gradient fields and market-based control.

An idea for reusing proofs within the validation of multiagent systems based on the idea of the component-based verification has been proposed by Brazier et al. [2]. Thus, only the proofs for components that have changed have to be updated. If it is possible to proof that these subsystems stay within their previously defined specification, the proof of the overall systems specification remains valid. Hilarie et al. [11] argue that to facilitate reuse of agents or components of agents, it is necessary to formally specify these components and then proof the compliance of components to their specification. This can foster reuse, as those components can become the building blocks for future systems. For that reason the authors present a formal notation combined out of Objective-Z and statecharts. The focus of the authors is on the design of the formal notation and on the prove of compliance. A quite similar approach for the reuse of organizations has been proposed by Jonker et al. [12]. In their paper the authors present a formalism to describe organizational structures and they assign properties to these organizations. They propose to build a library of organizational structures. An organizational designer then should be able to place queries to the library and retrieve possible organizational structures that might suit his requirements. The authors propose different aspects for indexing, e.g., by group functionality, environment assumptions or realization constraints [12]. The retrieved organization descriptions might be adapted to the given situation at hand.

Bartolini et al. [1] argue that the current representation of interaction protocols is not sufficient, as only the sequence of messages are fixed. But typically more information is required, e.g., to generate a valid bid in an English auction. This kind of information has to be implicitly encoded by the agent designer. Thus the authors present a framework for specifying negotiations, based on rules for encoding the negotiation protocol. Agent should be able to reason about those protocols and apply them autonomously. The reuse is thereby on emphasizing a more precise and complete form of specification for negotiations.

2.1 Reuse of coordination mechanisms

An idea, already mentioned, for the reuse of coordination mechanisms are coordination artifacts [16]. Within a coordination artifact a coordination mechanism is embedded that can be used by the agents. These artifacts can be reused. As

already discussed a significant drawback of these artifacts for autonomous planning agents is that the agents must reveal planning relevant information and loose partly their autonomy about their future activities.

The need for an easier retrieval for reusing interaction protocols has been identified by Bussmann et al. [4]. Therefore they focus on the selection process of interaction protocols. To be applicable an interaction protocol has to respect the existing dependencies of the current situation. Therefore, the authors suggest to classify interaction protocols according to a number of criteria. These characteristics are: the number of agents involved, the computability of constraints and preferences, the number of agent roles, the role assignment, the number of joint commitments, and the size of joint commitment as criteria. An agent designer should specify its requirements according to these criteria and then identify an interaction protocol that might be suitable for the given situation.

As one can see in the work addressing reuse of concepts by Bussmann et al. [4] and Jonker et al. [12] the idea of building repositories that can be browsed for content with specific characteristics can be an useful approach, that has been adopted for identifiying possible suitable coordination mechanisms.

3 The ECo-CoPS approach

The main idea of the ECo-CoPS approach is, that existing planning (sub)systems should not be replaced or changed, to enable the coordination among the agents. Each agent can be the representative of a planning entity, like a company for instance. The agents can manipulate the input of the local planning system and gather information from the output of the planning system.

The goal of the ECo-CoPS approach is to guide the selection process to find a coordination mechanism for inherently distributed autonomous planning systems. This selection is guided by the ECo (**E**ngineering **Co**ordination) process that is detailed in the following. An important step of the ECo process is the prototypical implementation of possible candidate solutions. The implementation step of the ECo process is supported by the CoPS (**Co**ordination of **P**lanning **S**ystems) process and framework. Both guide and ease the implementation of a coordination mechanism and will briefly described in the following. Even though the CoPS process and the CoPS framework have been designed to support the ECo process, they are optional for the ECo process. A detailed description of the ECo-CoPS approach can be found in [18].

3.1 The ECo process

The ECo process comprises of five steps that can be executed in an iterative manner. These steps are: model the coordination problem, elicit coordination requirements, select appropriate coordination mechanisms, implement selected approaches, and evaluate candidate mechanisms to identify the best one. The process is outlined in Figure 1.

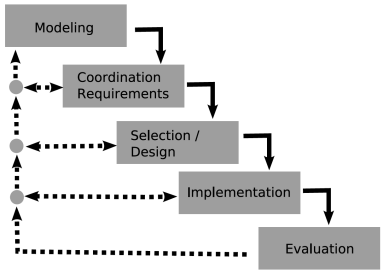


Fig. 1: The ECo process model

In the modeling phase the coordination problem and each planning problem is modeled with a specific level of detail to describe the necessary criteria that the local plans are feasible and aspects that should be optimized in the particular planning process. Moreover, the global perspective, defined by the dependencies between the planning problems, has to be modeled, as well.

In the elicitation step requirements are identified that have to be fulfilled by a coordination mechanism to be applicable for the given coordination problem. These requirements can, for instance, characterize under which conditions the planning systems are coordinated. These requirements can be formally described, using the terms and concepts introduced in the modeling step.

The third step is the selection phase. Coordination mechanisms have to be identified that can satisfy the coordination requirements. This steps results in a set of candidate mechanism that can effectively coordinate the planning systems. If this set is empty a suitable mechanism has to be designed.

To evaluate the effectiveness of these candidates they have to be implemented. Implementation is supported by the CoPS process and the CoPS framework, both discussed in the following.

If prototypical implementations exist, the candidate solutions can be evaluated with real-world like data, to find the most efficient coordination mechanism.

3.2 The CoPS process

The CoPS process is a sub-process of the ECo process. It structures the decision making during the implementation of a coordination mechanism. The CoPS process addresses decisions on the global level, i.e. among all entities, and on the local level, for each entity individually. The CoPS process is shown in Figure 2. The global process step is the definition of commonly accepted conversation protocols. It is global in the sense that all agents have to agree on the same conversation protocols to allow for an effective coordination. All other steps of the CoPS process have to be done by each entity by itself; therefore they are referred here as local. First each entity has to define its conversation policy. A conversation policy is a ”restrictions on communication based on the content of the communicative act" [13]. Within a conversation strategy a planning entity has to encode which concessions it is willing to make to whom, for instance.

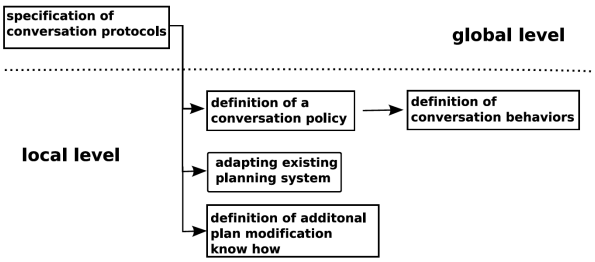


Fig. 2: Overview of the CoPS process

A conversation strategy is implemented in the conversation behaviors. A conversation behavior is executed in a particular state of the conversation, i.e. a state in the conversation automaton of one of the participants of the conversation.

The access to the local planning system can either be done directly, if the planning system is part of the agent, or by using integration techniques, like web-services, for instance.

It might be useful to add local planning-relevant knowledge to the agent, so that the agent can modify the input data of the planning system in a meaningful way. This could lead to reduced interaction times between the planning system and the agent, as the agent can modify the input data in a way that allows the planner to operate more efficiently.

3.3 The CoPS framework

The CoPS framework supports the implementation phase of the ECo process. The CoPS framework aims to facilitate the implementation. Within the CoPS framework the abstract implementation of a *planning authority agents*, the agent that represents a planning authority, and a coordination agent is provided. A coordination agent represents a network of agents, that needs to coordinate their activities and performs some management and bookkeeping actions for the entire network. The agents of the framework have to be instantiated and concrete strategies for conversations have to be implemented, as well as, the access to the planning system and additional knowledge how the planner should be used. In particular the definition and reuse of conversation protocols and their localization for each planning entity are supported by the CoPS framework.

4 Applying the ECo-CoPS approach: A case study

The ECo-CoPS approach has been applied to different case studies from the fields of logistics [18] and ambient intelligence [19]. Here we present a case study from the field of logistics. We use a simple setting within a manufacturing process of a company. First goods have to be produced, then they have to be packed and finally shipped to the customer. The work flow is detailed in Figure 3. A

Fig. 3: Work flow of a the production and distribution example

number of orders have to be satisfied. Each orders specifies a type of product, a destination for the shipment and a due date. In the beginning all orders are released and a plan to satisfy all orders has to be computed. During the production process a scheduling problem has to be solved, which has been taken from the literature [3, 5]. To compute a packing plan a 3-D bin packing problem has to be solved. Finally, to plan the shipment a vehicle routing problem has to be solved. Each of these problems is known to be a computational hard problem. We have developed three independent planning systems, each responsible for computing a valid sub-plan. An order is completed if all products are shipped to the customers. If the delivery date is later than the specified due date a penalty per time unit of lateness is imposed. We present here a compressed version of the case study, the complete case study can be found in [18].

The overall modeling of this problem is done using a set-constraint based approach, which is omitted here. During the modeling phase all relevant concepts are defined, which are necessary for the definition of the coordination requirements and to define measurements for the overall performance of the company. The coordination requirements that are of particular interest in this case study are that for each sub-problem a feasible plan exists and the overall global plan is feasible. This requires that the planning sequence is correct for all items, and that all items are produced, packed and shipped requested in the orders. As an global objective function we use the overall costs, compromising the costs of packaging, the costs for transportation, and eventually penalties for lateness.

The selection step contains two stages. A first identification step to shrink down the number of candidates and a qualitative evaluation as a second step to analyze if the candidate mechanisms satisfy the coordination requirements. In the identification step we take advantage of the idea of building repositories of mechanism description, which are annotated with relevant characteristics. Therefore we have build up a repository of different types of coordination mechanisms and classified them according to coordination specific characteristics. In brackets we point out the characteristic the scenario requires. The characteristics are presented in form of binary questions and are the following: Does an allocation problem exists? (No); Are the local objective functions comparable? (No); Are the planning systems homogeneous or heterogeneous? (heterogeneous); Does a common objective function exists? (Yes); Is information hiding necessary? (No); Do cyclic dependencies exist? (No). More details of the classification can be found in [18]. By browsing the repository of different classes of coordination mechanisms, according to the needs of the current scenario, we can restrict the number of coordination mechanisms that have to be investigated in depth for the applicability for the given scenario. As a result of the first step we identify

the following coordination approaches as possible candidates: plan merging, decentralized planning for a centralized plan, result sharing, and negotiation. Note that for this simple case the approach of decentralized planning for a centralized plan [6] is equivalent to result sharing. Different planners compute partial solution and pass them to the next planner which is then generating his part of the overall plan. This, in fact, is result sharing. The sequence of the planning systems computing their partial plan is given by the work flow presented in Figure 3. The plan merging approach requires an additional entity that collect all local plans and is capable of integrating them and, if necessary, propose plan modifications to ensure consistency. This requires planning knowledge to compute plan modifications that ensure a feasible global plan, as well as, feasibility of the local plans. Therefore this solution is similar to a complete centralized planner, which is not in the scope of this research.

The class of negotiations as coordination means cover a wide field. In this case study a key problem is that most costs are fixed in the last planning step, where the least flexibility of planning decision exists. Ideally a backward oriented planning would be more appropriate. But this approach makes it more complex to ensure feasibility of the overall plan, as the execution sequence of the planning systems would be directly inverse to the sequence imposed by the dependencies among the planning problems. A solution to this problem can be a mechanism that facilitates the exchange of requirements towards the local plans, and plan suggestions that tries to satisfy the requirements and still ensuring feasible local plans. Such a coordination approach would result in a sequence of exchanges of requirements to, and suggestions of plans. This corresponds to a negotiation, trying to minimizing the total costs. By starting with the parts of the planning process where most of the costs are fixed requirements can be identified that lead to an overall solutions with lower costs. Previous planning stages have to identify what requirements are possible to fulfill and offer those to the subsequent planning entity.

In the implementation step the remaining two major concepts, result sharing and negotiations, are implemented. Note that we have only to derive the agents from the CoPS framework, implementing the particular coordination mechanism, and enable them to use the existing planning systems. Therefore, the efforts for implementing these coordination systems are considerable low. The result sharing approach is referred here as sequential planning, as the planning steps are done sequentially, following the dependencies among the planning problems. The agents access their planning systems using web services. In the evaluation phase we compare both approaches using randomly generated problem instances of different size. First, we analyze how both approaches scale with the problem size. Second, we perform a detailed analysis for specific problem sizes. For the first analysis we consider scenarios from 1 up to 30 orders. The resulting costs for both approaches are shown in Figure 4. Note that the scales of the subfigures are not identical. We do so, to allow the reader to see the differences also between scenarios with few orders. For one order both methods are equivalent and generate the same plan. In all other scenarios the improved, negotiation-

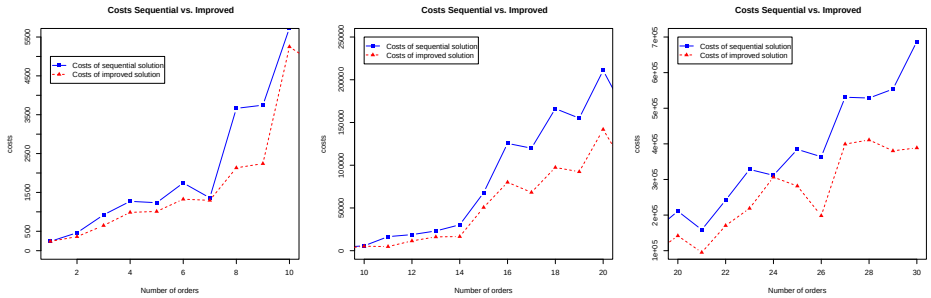


Fig. 4: Scaling of both coordination approaches with different problem sizes (1–30 orders)

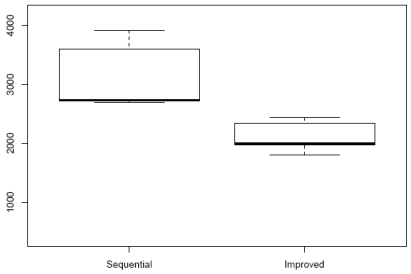
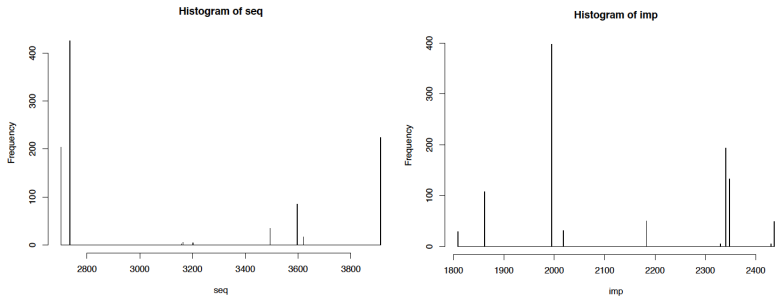


Fig. 5: Box plots for instance 2 comparing costs of the sequential and improved coordination approach

based, approach performs better than the sequential planning. Even though this data does not allow to draw a conclusion, as the number of instances is too small, it shows a clear indication. Moreover, we can see that about 7 orders the costs increase drastically. With about seven orders the first penalties have to be paid as not all orders can be performed in time. The second drastic increase can be seen at approx 15 orders. Then the system goes in an overload situation, where nearly all orders cannot be performed in time, and the penalties rise dramatically.

Based on these results we investigate particular problem sizes in more detail. In total we created 10 different scenarios consisting of the same number of orders and compute 1000 replications for each scenario. Here we present the results obtained with a scenario with five different orders. If we compare the results between both approaches we can summaries, for this scenario that the improved, negotiation based, coordination approaches, leads to a better overall performance and a more stable result, as the spread of the results is lower, than the result sharing approach. We present in Figure 5 the box plots comparing the mean costs and the spread obtained in different runs. As typically for planning systems the spread results from the fact that a few different solutions are computed over and over again. The resulting histograms for both approaches are shown in



(a) sequential (b) improved

Fig. 6: Histogram for the sequential and improved approach for instance 2

Figure 6. Summarizing this evaluation we can now select an effective and efficient coordination mechanism, as a result of applying the ECo process. For the given situation at hand the negotiation based approach has to be selected.

5 Summary and Outlook

In this paper we argued that it is necessary to provide methods for identifying effective and efficient concepts during the design of multiagent systems to support developers. We have exemplified this, addressing the selection process of coordination mechanisms for autonomous planning agents. Up to now, this problem mainly occur in logistic scenarios. With the ongoing trend towards ubiquitous intelligent system the task to coordinate intelligent planning systems is going to spread into various domains. Therefore, we have discussed the application of the ECo-CoPS approach also in the field of ambient intelligent systems [19]. We have detailed that research in the field of AOSE has covered this field not sufficiently. For that reason we have presented the ECo-CoPS approach that defines a structured process for the selection of coordination mechanisms for autonomous planning systems, where the local autonomy of the agents, as well as, the existing planning systems can be preserved. A case study has been presented to detail how the ECo-CoPS approach can support the selection process. Moreover, we think that the ECo process can be used as a blueprint for the selection of other concepts in multiagent systems, as well.

As the ECo-CoPS approach presents a process for handling specific problems its assessment becomes more sound by multiple iterations of the process. This allows for analyzing if additional tailoring of the process or the definition of additional supporting sub-processes might be useful. Therefore we strive to apply the process in more case studies from different domains. It turned out that the modeling step of the ECo process can become time intensive. Therefore we want to investigated the usage of different modeling techniques for the coordination problems. To take more advantage of the efforts in the modeling phase we

want to generate more synergies between the modeling and the implementation step. Therefore we are considering to use specific UML profiles and the object constraint language (OCL) for modeling, as this way of model might offer additional value during the implementation phase. Therefore the CoPS process and in particular the CoPS framework might have to be adapted.

Acknowledgment

This work was been supported by a fellowship within the Postdoc-Programme of the German Academic Exchange Service (DAAD).

References

1. Claudio Bartolini, Chris Preist, and Nick R. Jennings. Architecting for reuse: A software framework for automated negotiation. In John Mylopoulos, Michael Winikoff, and Nick R. Jennings, editors, *Agent-Oriented Software Engineering III Proc. of the Third International Workshop, AOSE 2002*, volume 2585, pages 88 – 100, Bologna, Italy, 2003. Springer.
2. Frances M. T. Brazier, Frank Cornelissen, Rune Gustavsson, Catholijn M. Jonker, Olle Lindeberg, Bianca Polak, and Jan Treur. Compositional design and verification of a multi-agent system for one-to-many negotiation. In *Proceedings of the Third International Conference on Multi-Agent Systems, ICMAS'98*, pages 49 – 56. IEEE Computer Society Press, 1998.
3. Robert W. Brennan and William O. A simulation test-bed to evaluate multi-agent control of manufacturing systems. In *WSC '00: Proceedings of the 32nd conference on Winter simulation*, pages 1747–1756, Orlando, Florida, 2000. Society for Computer Simulation International.
4. Stefan Bussmann, Nick R. Jennings, and Michael Wooldridge. Re-use of interaction protocols for agent-based control applications. In Fausto Giunchiglia, James Odell, and Gerhard Weiß, editors, *Agent-Oriented Software Engineering III Proc. of the Third International Workshop, AOSE 2002*, volume 2585 of *Lecture Notes in Computer Science*, pages 73 – 87, Bologna, Italy, 2003. Springer.
5. Sergio Cavalieri, Luc Bongaerts, Marco Macchi, Marco Taisch, and Jo Weyns. A benchmark framework for manufacturing control. In *2. International Workshop on Intelligent Manufacturing Systems*, pages 225 – 236, Leuven, Belgium, 1999.
6. Edmund H. Durfee. Distributed problem solving and planning. In Gerhard Weiß, editor, *Multiagent Systems: a modern approach to distributed artificial intelligence*, pages 121 – 164. MIT Press, 1999.
7. Foundations for Intelligent Physical Agents FIPA. Fipa standard specifications, 2002. `http://www.fipa.org/repository/standardspecs.html`, Accessed: 02/04/11.
8. Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of reusable object-oriented software*. Addison-Wesley Professional Computing Series. Addision Wesley Longman Inc., Reading,, 1994.
9. Luca Gardelli, Mirko Viroli, and Andrea Omicini. Design patterns for self-organising systems. In Hans-Dieter Burkhard, Gabriela Lindemann, Rineke Verbrugge, and László Z. Varga, editors, *Proc. of the 5th International Central and Eastern European Conference on Multi-Agent Systems, CEEMAS 2007*, Lecture Notes in Artificial Intelligence, pages 123 – 132, Leipzig, 2007. Springer.

10. Sandra C. Hayden, Christina Carrick, and Qiang Yang. Architectural design patterns for multiagent coordination. In *Proceedings of the 3rd International Conference on Autonomous Agents, AGENTS'99*, 1999.
11. Vincent Hilaire, Olivier Simonin, Abder Koukam, and Jacques Ferber. A formal approach to design and reuse agent and multiagent models. In Michael Luck and James Odell, editors, *Agent-Oriented Software Engineering V; Proc. of the 5th International Workshop, AOSE 2004*, volume 3382, pages 142 – 157, New York, NY, USA,, 2005. Springer.
12. Catholijn M. Jonker, Jan Treur, and Pinar Yolum. A formal reuse-based approach for interactively designing organizations. In Michael Luck and James Odell, editors, *Agent-Oriented Software Engineering V; Proc. of the 5th International Workshop, AOSE 2004*, volume 3382, pages 221 – 237, New York, NY, USA,, 2005. Springer.
13. Lalana Kagal and Tim Finin. Modeling conversation policies using permissions and obligations. In Rogier M. van Eijk, Marc-P. Huget, and Frank Dignum, editors, *AAMAS 2004 Workshop on Agent Communication (AC2004)*, New York, 2004.
14. Manuel Kolp, T. Tung Do, and Stéphane Faulkner. Introspecting agent-oriented design patterns. In S. K. Chang, editor, *Handbook of Software Engineering and Knowledge Engineering,*, volume Vol. 3: Recent Advances, pages 151–176. World Scientific Publishing Co, 2005.
15. Jürgen Lind. Patterns in agent-oriented software engineering. In Fausto Giunchiglia, James Odell, and Gerhard Weiß, editors, *Agent-Oriented Software Engineering III, 3. International Workshop, AOSE 2002*, volume 2585 of *Lecture Notes in Computer Science*, pages 47 – 58, Bologna, Italy, 2003. Springer.
16. Andrea Omicini, Alessandro Ricci, Mirko Viroli, Cristiano Castelfranchi, and Luca Tummolini. Coordination artifacts: Environment-based coordination for intelligent agents. In *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, volume Volume 1, pages 286–293, New York, New York, 2004. IEEE Computer Society.
17. Sylvain Sauvage. Design patterns for multiagent systems design. In *Proceedings of the 3rd International Conference on Artificial Intelligence, MICAI'04*, volume 2972 of *Lecture Notes in Artificial Intelligence (LNAI)*. Springer, 2004.
18. R. Schumann. *Engineering Coordination : A Methodology for the Coordination of Planning Systems*. PhD thesis, Institute of Informatics, Goethe University, 2010. `http://publikationen.ub.uni-frankfurt.de//frontdoor.php?source_opus=8143`, Accessed: 02/04/2011.
19. René Schumann. Engineering coordination in future living environments. In Ralf Dörner and Detlef Krömker, editors, *Proceedings of the ITG / GI Workshop on Self-Integrating Systems for Better Living Environments 2010: SENSYBLE 2010*. Shaker Verlag, 2011. accepted for Postproceedings.
20. Tom De Wolf and Tom Holvoet. Design patterns for decentralised coordination in self-organising emergent systems. In Sven A. Brueckner, Salima Hassas, M·rk Jelasity, and Daniel Yamins, editors, *Engineering Self-Organising Systems: Proceedings of the 4th InternationalWorkshop, ESOA 2006*, volume 4335 of *Lecture Notes in Artificial Intelligence*, pages 28 – 49, Hakodate, Japan, 2006. Springer.

Understanding Agent Oriented Software Engineering Methodologies

Jorge J. Gomez-Sanz, Ruben Fuentes-Fernández, Juan Pavón

GRASIA Research Group,
Universidad Complutense de Madrid,
Avda. Complutense, 28040 Madrid, Spain
{jjgomez,ruben,jpavon}@fdi.ucm.es

Abstract. This paper introduces software engineering concepts to obtain a new perspective on the work done in agent oriented methodologies. Software engineering relies on a body of knowledge that is not available in the agent research community, yet. A transfer of knowledge from this body can clarify what a methodology is for and what it is necessary to define one. The paper concludes that current agent oriented methodologies are still to evolve. This evolution will be possible due to the general adoption of meta-modeling techniques; the interest in covering more development phases; and the growing number of development examples available to the community.

1 Introduction

There has been little discussion of what an agent oriented methodology is. This surprises when considering there is no standard definition. This may have caused an abuse of the term leading to an increasing number of research works claiming to be a methodology. Researchers working in evaluation frameworks [SS03] have raised this issue in the past, though this has not conducted to a self-criticism of the agent oriented software engineering community. As a result, some papers claim to have a methodology just by introducing a modeling language or enumerating a few development activities.

We will start with a simple hypothesis: agent oriented methodologies are the result of a transfer of knowledge from software engineering. As agent researchers, we are the experts to tell if a methodology X does capture the essential of the agent concept. Nevertheless, to assess the capability of a methodology for effectively assisting developers, the experts to ask should be software engineers. Trying to look for answers in this direction, this paper proposes to revisit software engineering basics and draw some conclusions on the current state and directions of agent oriented methodologies.

To do so, the paper first introduces basic concepts of software engineering in section 2. Then, section 3 reformulates the software engineering definitions including, when necessary, the agent orientation. With this definition in mind, some aspects of current methodologies are regarded and their current status assessed. Continuing with the ideas extracted from the software engineering review, some perspectives about the future of methodologies are declared in section 4.

2 Software Engineering and Methodologies

Software engineering is defined by the IEEE glossary [IEE90] as follows:

> (1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software. (2) The study of approaches as in (1).

The term *software engineering* appears in 1968 and was the name of a conference [NR68] sponsored by the North Atlantic Treaty Organization (NATO), where scientists and professionals acknowledged several problems affecting the software industry. These problems crystallized into the *software crisis*, i.e., the inability to conclude development projects before the deadlines, within the initial budget, and satisfying the client demands. Reasons like the complexity of the software were expected to be handled by advances in programming languages and good programming practices, a position defended by Dijkstra [Dij72], but there were other issues that were clearly human like the coordination of working teams and the organization of the development activities, a position illustrated in the known book from Brooks [Bro95]. Both were right, and as a result, the training of a software engineer today is a multidisciplinary one. This is illustrated in the curricula recommendations for Software Engineering promoted by the ACM and IEEE that can be found for undergraduate `http://sites.computer.org/ccse` and master level `http://www.gswe2009.org` courses.

Knowing software engineering today, means, at least, being familiar with the Software Engineering Book of Knowledge (SWEBOK) [ABD+04]. This is an initiative promoted, among others, by the IEEE computer society and part of the software industry. It pursues collecting relevant works, even ISO/IEEE standards, addressing all the issues a developer may find while developing software. SWEBOK introduces eleven *Knowledge Areas*, each one focusing in a concrete subject like software design, software testing, or software quality.

In what respect to the topic of this paper, agent oriented methodologies, SWEBOK has a *Knowledge Area* named *Software Engineering Tools and Methods*. Tools and methods appear together because they are highly coupled. Quoting the SWEBOK:

> Tools are often designed to support particular software engineering methods, reducing any administrative load associated with applying the method manually. Like software engineering methods, they are intended to make software engineering more systematic, and they vary in scope from supporting individual tasks to encompassing the complete life cycle.

This does not mean that a method is unfeasible without tools, but, it makes the engineering tasks more challenging and inefficient. About the methods, SWEBOK says:

> Software engineering methods impose structure on the software engineering activity with the goal of making the activity systematic and

> ultimately more likely to be successful. Methods usually provide a notation and vocabulary, procedures for performing identifiable tasks, and guidelines for checking both the process and the product. They vary widely in scope, from a single life cycle phase to the complete life cycle. The emphasis in this Knowledge Area is on software engineering methods encompassing multiple life cycle phases, since phase-specific methods are covered by other Knowledge Areas.

A method, according to the SWEBOK, can be either heuristic, formal, or prototyping. Heuristic methods can be either structured (e.g. a top-down approach where functions are successively refined), data oriented (e.g. being guided by the data structures the problem needs), or object oriented (the system is viewed as a collection of objects instead of data structures or functions). Formal methods are based on mathematics and focus mainly to specification languages (model-oriented, property-oriented, or behavior-oriented), specification refinement (how to bring it closer to the implementation), and verification of properties. Prototyping methods distinguish between the prototyping style (e.g. if the prototype is going to be thrown away or if it will evolve towards the final system), prototyping target (e.g. if it is going to prototype the user interface or the system architecture), and the prototyping evaluation techniques (referring to the way the results of the prototyping activity are going to be used).

A method concerns as well the *software life cycle* and the *software life cycle processes*. The *software life cycle* is *the period of time that begins when a software product is conceived and ends when the software is no longer available for use* [IEE90]. The second refers to the *process by which user needs are translated into a software product* [IEE90], though the SWEBOK tries to adopt a wider approach. It admits there could be several processes involved in a software life cycle, or a single one, from the perspective of a whole organization performing the development. In any case, the SWEBOK assumes a process usually follows a *software life cycle model*, which provides a high-level definition of the phases that occur during the development, e.g. a spiral model or waterfall. A process can be defined, implemented, measured, and assessed. A *software life cycle processes* or a *software life cycle model* definition can be constructed using different notations (natural language, state-charts, petri nets, and others), though there are standards available as well, like the Software & Systems Process Engineering Metamodel Specification (SPEM) [Obj08] or the Software Engineering Metamodel for Development Methodologies (SEMDM) ISO/IEC 24744 [HSGP08]. It should be noted that SEMDM intends to capture the notion of methodology, conceiving it as *the specification of the process to follow as well as the work products to be generated, plus consideration of the people and tools involved, during a software development effort* [GPHS05]. This is slightly different from the definitions given in the IEEE Glossary and the SWEBOK, as introduced here, though it should be still possible to use SEMDM to define a process.

This is just a rough approach to the problem of engineering software systems, but gives an idea of how challenging is to define a development method.

3 Agent Oriented Methodologies from a Software Engineering perspective

Using as starting point the review of work in software engineering from section 2, some conclusions can be drawn to AOSE. From the categorization of methods in section 2, an agent oriented method could be labeled as an *heuristic method* where the system is viewed as a collection of agents. This would make sense in most cases, though the integration with formal methods and/or prototyping techniques, may complicate the issue. In any case, this method should *aim for making the activity systematic and ultimately more likely to be successful*. The method will provide *with notation and vocabulary, procedures for performing identifiable tasks, and guidelines for checking both the process and the product*. The scope of the method will be *from a single life cycle phase to the complete life cycle*. This scope allows to talk about *agent oriented analysis*, which would focus on how to determine which agents are necessary to provide the system functionality; or *agent oriented implementation*, which would address the way a system specification, regardless of its agent orientation, can be implemented using agent oriented solutions, e.g. an agent oriented programming language or an agent oriented framework. The concrete phase or process is defined separately, and can be an ad-hoc process or be derived from some *software life cycle model*. Clearly, the process or the affected phase will be biased by the use of agent related concepts, since the expected products of the engineering activities necessarily need to refer to those concepts. Finally, there should be support tools that assist in the execution of the different activities addressed by the method.

The value of these conclusions is that they are derived from what an actual software development demands, according to the software engineering practices. For the researcher, it serves as a guideline of the elements to be included in a methodology and helps to focus the contribution in specific phases of the development. For the developer, knowing in advance what is covered by the methodology will permit to be more efficient and applying the methodology in the context it was intended to.

It should not be deduced that what has been published as agent oriented methodologies is wrong. Revisiting the survey from Iglesias et al. [IGCG98], it can be read that a methodology is something whose role is *to assist in all the phases of the life cycle of an agent-based application, including its management* and that extends some known existing development solutions for X. Extension was not trivial and required altering existing solutions in different ways, trying to be coherent with whatever was made with X. These extensions, or recycled methodologies as referred in Müller [M̈97], had to be validated in some way. These extensions were focused in adapting the different existing products, and, in some cases, the involved activities. Assuming that the original methodologies included the above-mentioned elements, which is true in most cases, and that their agent oriented extension were coherent with the inclusion of new elements, it should be assumed that the resulting methodology did provide the requested elements.

It is harder to ensure this when the candidate for agent oriented methodology is built from the scratch. Two big groups of proposals can be found. In one, there is a methodology that first tries to build a notation and vocabulary to address some specific development phases. In the other, notation and vocabulary are embedded in a tool that permits to move from the analysis and design to the implementation.

In the first group, there are works that were introduced as methodologies at that time, but they should be more accurately introduced as phase specific methods. This is the case, in our opinion, of GAIA [WJK00, ZJW03], highly referenced and widely regarded as an agent oriented methodology. It provides a notation and vocabulary, but it is focused on the analysis and design activities. Also, GAIA is not making the MAS construction *systematic and disciplined*, since it only addresses a part of the life cycle. Hence, it makes more sense to talk about the GAIA agent oriented analysis method and the GAIA agent oriented design method, since those are the *software life cycle phases* which are being structured. Similarly, MESSAGE/UML [CCG+02] was introduced as an agent oriented software engineering methodology covering analysis and design. Again, it should have been more accurate to regard this as agent oriented oriented design and analysis method.

In the second group, works are built around a powerful tool support covering an important part of the life cycle. This was the case of ZEUS [NNLC99] methodology and MaSE [WD00]. It has been objected that ZEUS methodology was not such, since it was about the use of a tool, but, from the perspective of this paper, it is rather the opposite. ZEUS was a great tool that allowed a developer go from requirements gathering to system deployment rather efficiently, though it presented some bugs. MaSE and its agentool allowed the execution of similar tasks but with a better documentation of the process to follow, and enabling formal verification of some properties of the defined protocols. Both works have evolved little to the extent ZEUS was discontinued and MaSE had to build new tools to deal with new concepts, the agenttool III which is a plugin for the Eclipse platform.

Time has passed and these two groups have merged. Today, it is expected from candidate methodologies to have tool support similar to the one provided by ZEUS or agenttool, and regard more of the software life cycle than the ancestors. As a result, the number of methodological proposals addressing more parts of the software life cycle has increased. AOSE methodologies such as INGENIAS, PASSI, ADELFE, or Prometheus, have grown up including more standard development phases, concretely, requirements gathering, implementation, and testing. As more phases are incorporated, the closer to a competitive industrial grade methodology it is.

Therefore studying *software life cycle processes* is relevant to understand the scope of the method and apply it correctly. Nevertheless, this part is usually lightly acknowledged in the AOSE community with some exceptions [SCG08]. Frequently, activities tend to be too simplified, to the extent that they are introduced as some items in a short numbered list. The connection with the process

itself is missing or non-relevant in the context of the research contribution. In the other extreme, there are methodologies that either propose new *software life cycle processes*, such as PASSI, ADELFE, or Prometheus, or stick to an existing *software life cycle model*, like INGENIAS. Understanding the importance of processes is something the IEEE FIPA Design Process Documentation and Fragmentation Working Group (DPDF WG) `http://www.fipa.org/subgroups/DPDF-WG.html` is contributing enormously. There are complete specifications of processes for PASSI, SODA, GORMAS, and INGENIAS `http://www.pa.icar.cnr.it/cossentino/fipa-dpdf-wg/docs.htm`. For more information about the role of processes in AOSE, it is recommended to read the survey from Massimo et al. [CGMO11].

There has been an important progress in what refers to how AOSE methodologies address the notation and vocabulary. Formal specifications are reviewed in the survey written by El Fallah-Seghrouchni et al. [EFSGSS11]. The formal representation methods addressed in the survey can be labeled either as first-order logic, temporal logics, process algebras, or automata based. Solutions found in these categories are all tool supported, with a special focus in the verification of properties. Despite the interest in formal notations, AOSE seems to be more fond of semi-formal approaches such as visual modeling languages specified through meta-models. It could be said it all started with the work of Ferber [FG98] on the use of UML to specify a meta-model for MAS based in the concept of organization and role. This contribution had no specific tool support, so it trusted developers used existing UML vendor tools in the way the notation demanded. MESSAGE/UML [CCG+02] was the next step. It was the first to actually apply the meta-modeling approach as we know it today, using Graph-Object-Property-Relationship-Role (GOPRR) to express the syntax of the modeling language and a meta-modeling tool, MetaEdit+, to provide custom editors. This path has been followed by most methodologies today, as Argente et al. [ABFF+11] show in a survey on current agent modeling languages, though using different meta-modeling languages. One meta-modeling approach attracting most of the attention is the Eclipse Modeling Framework (EMF). As a consequence, known methodologies like Tropos, MaSE (which now is o-MaSE) and Prometheus, are migrating towards this platform, with TAO4ME [BDM+06], in the case of Tropos, Agentool III [SCDS09], in the case of o-MaSE, PDT [STP10], in the case of Prometheus. INGENIAS has used this pure meta-modeling approach since the beginning [PGS03], publishing its meta-model as an XML file in its sourceforge distribution site `http://ingenias.svn.sourceforge.net/viewvc/ingenias/trunk/metamodel/`. As a contribution to the community, INGENIAS team recently released to the public the meta-editor framework that was used internally to maintain INGENIAS. Its name is INGENME, which can be downloaded from `http://ingenme.sf.net`, and provides similar functionality to the Eclipse Modeling Framework, but more user friendly.

Progress in the construction of modeling languages have been encompassed with advances in techniques for implementing specifications. A review of them can be found in Nunes et al. [NCdL+11]. From the review it is outstanding

the growing number of automatic translations of specifications into code. Most of them reuse facilities from the Eclipse platform, like the already mentioned TAO4ME, Agentool III or PDT. This approach is the dominant now, with some exceptions where implementation has to be done manually. There are precedents in this kind of manual transition from specification to code. It can be done in a disciplined way, as suggested by Kendall [Ken99], where developers have the assistance of guidelines like design patterns.

4 Perspectives for Agent Oriented Methodologies

None of the existing methodologies can be considered perfect. They need to be used again and again; and this will surely make current methodologies evolve. This evolution will be probably directed towards addressing new development activities and increment the scope of the methodology; a general improvement in the tool support; and an increase of the number of developments that avail the possibilities of each methodology.

A methodology aims to increase the chances of building successfully the system. Consequently, a methodology needs to be applied and assessed. If a software company is contracted to develop a system, the client should be sure that the company is capable of building the correct system before the agreed deadline and without exceeding the budget. A way to rate the capability for success is the Capability and Maturity Model (CMM) [PCCW93] and its evolution the Capability Maturity Model Integration (CMMI) [CMM02]. These models identify a number of key areas that need to be studied, like the technical skills of developers, the degree of adoption of software processes, the existence of evaluation procedures to assess the work done, or the training of personnel within the company, to mention some. Our methodologies are not there, yet. Only a piece of these elements are covered actually by existing methodologies. Nevertheless, knowing more about how methodologies are expected to be applied in the real world serves to advance the future needs. Most likely, management activities will be the next to be considered, after testing is more deeply studied. One of the basic management activities consists in foreseeing how much effort a MAS development will take [GSPG05] so that a realistic development plan can be devised.

Tool support will be improved in all the activities the methodology is involved. A basic improvement will be being capable to enrich a methodology using past experiences. Notations like UML have changed along the years because the application experience of UML told there were things that could be made better. Similarly, the processes applied in the development and the notation proposed by our methodologies should be able to improve. This requires having customizable tools, capable of adapting to the new processes or notations. Thanks to the adoption of meta-modeling techniques by the current methodologies, this kind of tools will be possible with a reduced effort. Changes in the meta-models describing the modeling language or the process will be automatically processed to produce a new tool set adapted to the new meta-models.

Finally, there will be a higher demand of evidences proving the methodology works. These evidences will be complete developments availing the capability of the methodology. There has been intents to develop the same system with Prometheus, O-MaSE, Tropos, and the Multi-Agent Systems Unified Process [LP08]. This can be useful to show the pros and cons of each approach and demonstrate the benefits. Also, some methodologies maintain a list of developments that can be used for testing the methodology and learn what it can do. Tropos maintains a list of papers introducing empirical application of the methodology in `http://www.troposproject.org/node/304`. INGENIAS does the same but provides the code instead in `http://ingenias.svn.sourceforge.net/viewvc/ingenias/trunk/IDK/iaf/tests/`. These developments are actually used as regression tests to verify that code generation capabilities of INGENIAS are not altered, i.e., that the same specification can be transformed in the final system as initially devised.

5 Conclusions

This paper has tried to clarify a pending issue in the AOSE community, which it is the definition of an agent oriented methodology. It was studied using a software engineering perspective based on standards and reference material in the software engineering community. This has permitted to enumerate elements that a methodology should incorporate, and aspects it should address. The work is not finished yet, since other issues in the provided definitions have not been completely explored. For instance, the way a method is expected to describe a procedure and guidelines, and which ones could be meaningful.

Acknowledgements

This paper has been funded by the the project Agent-based Modelling and Simulation of Complex Social Systems (SiCoSSys), supported by Spanish Council for Science and Innovation, with grant TIN2008-06464-C03-01, by the Programa de Creación y Consolidación de Grupos de Investigación UCM-Banco Santander for the group number 921354 (GRASIA group), and by the Project for Innovation and Improvement of Teaching (Proyectos de Innovacion y Mejora de la Calidad Docente) number 127 from the UCM.

References

[ABD+04] Alain Abran, Pierre Bourque, Robert Dupuis, James W. Moore, and Leonard L. Tripp. *Guide to the Software Engineering Body of Knowledge - SWEBOK*. IEEE Press, Piscataway, NJ, USA, 2004 version edition, 2004.

[ABFF+11] Estefanía Argente, Ghassan Beydoun, Rubén Fuentes-Fernández, Brian Henderson-Sellers, and Graham Low. Modelling with agents. In Marie-Pierre Gleizes and Jorge Gomez-Sanz, editors, *Agent-Oriented Software*

Engineering X, volume 6038 of *Lecture Notes in Computer Science*, pages 157–168. Springer Berlin / Heidelberg, 2011.

[BDM+06] Davide Bertolini, Loris Delpero, John Mylopoulos, Aliaksei Novikau, Alessandro Orler, Loris Penserini, Anna Perini, Angelo Susi, and Barbara Tomasi. A tropos model-driven development environment. In Nacer Boudjlida, Dong Cheng, and Nicolas Guelfi, editors, *CAiSE Forum*, volume 231 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2006.

[Bro95] Frederick P. Brooks. *The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition (2nd Edition)*. Addison-Wesley Professional, anniversary edition, August 1995.

[CCG+02] Giovanni Caire, Wim Coulier, Francisco J. Garijo, Jorge Gomez, Juan Pavón, Francisco Leal, Paulo Chainho, Paul E. Kearney, Jamie Stark, Richard Evans, and Philippe Massonet. Agent oriented analysis using message/uml. In *Revised Papers and Invited Contributions from the Second International Workshop on Agent-Oriented Software Engineering II*, AOSE '01, pages 119–135, London, UK, UK, 2002. Springer-Verlag.

[CGMO11] Massimo Cossentino, Marie-Pierre Gleizes, Ambra Molesini, and Andrea Omicini. Processes engineering and aose. In Marie-Pierre Gleizes and Jorge Gomez-Sanz, editors, *Agent-Oriented Software Engineering X*, volume 6038 of *Lecture Notes in Computer Science*, pages 191–212. Springer Berlin / Heidelberg, 2011.

[CMM02] CMMI Product Team. Capability maturity model integration (cmmism), version 1.1. Technical report, Carnegie Mellon, Software Engineering Institute, 2002.

[Dij72] Edsger W. Dijkstra. The humble programmer. *Commun. ACM*, 15(10):859–866, 1972.

[EFSGSS11] Amal El Fallah-Seghrouchni, Jorge Gomez-Sanz, and Munindar Singh. Formal methods in agent-oriented software engineering. In Marie-Pierre Gleizes and Jorge Gomez-Sanz, editors, *Agent-Oriented Software Engineering X*, volume 6038 of *Lecture Notes in Computer Science*, pages 213–228. Springer Berlin / Heidelberg, 2011.

[FG98] Jacques Ferber and Olivier Gutknecht. A meta-model for the analysis and design of organizations in multi-agent systems. In Yves Demazeau, editor, *ICMAS*, pages 128–135. IEEE Computer Society, 1998.

[GPHS05] Cesar Gonzalez-Perez and Brian Henderson-Sellers. Templates and resources in software development methodologies. *Journal of Object Technology*, 4(4):173–190, May 2005.

[GSPG05] Jorge J. Gómez-Sanz, Juan Pavón, and Francisco J. Garijo. Estimating costs for agent oriented software. In Jörg P. Müller and Franco Zambonelli, editors, *AOSE*, volume 3950 of *Lecture Notes in Computer Science*, pages 218–230. Springer, 2005.

[HSGP08] Brian Henderson-Sellers and Cesar Gonzalez-Perez. Standardizing methodology metamodelling and notation: An iso exemplar. In Roland Kaschek, Christian Kop, Claudia Steinberger, and Günther Fliedl, editors, *UNISCON*, volume 5 of *Lecture Notes in Business Information Processing*, pages 1–12. Springer, 2008.

[IEE90] IEEE standard glossary of software engineering terminology. *IEEE Std 610.12-1990*, 1990.

[IGCG98] Carlos Angel Iglesias, Mercedes Garijo, and José Centeno-González. A survey of agent-oriented methodologies. In Jörg P. Müller, Munindar P.

Singh, and Anand S. Rao, editors, *ATAL*, volume 1555 of *Lecture Notes in Computer Science*, pages 317–330. Springer, 1998.

[Ken99] Elizabeth A. Kendall. Role modeling for agent system analysis, design, and implementation. In *ASA/MA*, pages 204–218. IEEE Computer Society, 1999.

[LP08] Michael Luck and Lin Padgham, editors. *Agent-Oriented Software Engineering VIII, 8th International Workshop, AOSE 2007, Honolulu, HI, USA, May 14, 2007, Revised Selected Papers*, volume 4951 of *Lecture Notes in Computer Science*. Springer, 2008.

[M̈97] H. Jürgen Müller. Towards agent systems engineering. *Data Knowl. Eng.*, 23:217–245, September 1997.

[NCdL+11] Ingrid Nunes, Elder Cirilo, Carlos de Lucena, Jan Sudeikat, Christian Hahn, and Jorge Gomez-Sanz. A survey on the implementation of agent oriented specifications. In Marie-Pierre Gleizes and Jorge Gomez-Sanz, editors, *Agent-Oriented Software Engineering X*, volume 6038 of *Lecture Notes in Computer Science*, pages 169–179. Springer Berlin / Heidelberg, 2011.

[NNLC99] Hyacinth S. Nwana, Divine T. Ndumu, Lyndon C. Lee, and Jaron C. Collis. Zeus: A toolkit for building distributed multiagent systems. *Applied Artificial Intelligence*, 13(1-2):129–185, 1999.

[NR68] Peter Naur and Brian Randell, editors. *Software Engineering, report on a conference sponsored by the NATO SCience Committee*. NATO, Science Affairs Division from NATO, 1968.

[Obj08] Object Management Group. Software and systems process engineering metamodel specification (spem), 2008.

[PCCW93] M.C. Paulk, B. Curtis, M.B. Chrissis, and C.V. Weber. Capability maturity model, version 1.1. *Software, IEEE*, 10(4):18 –27, July 1993.

[PGS03] Juan Pavón and Jorge J. Gómez-Sanz. Agent oriented software engineering with ingenias. In Vladimír Marík, Jörg P. Müller, and Michal Pechoucek, editors, *CEEMAS*, volume 2691 of *Lecture Notes in Computer Science*, pages 394–403. Springer, 2003.

[SCDS09] Carles Sierra, Cristiano Castelfranchi, Keith S. Decker, and Jaime Simão Sichman, editors. *8th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009), Budapest, Hungary, May 10-15, 2009, Volume 2*. IFAAMAS, 2009.

[SCG08] Valeria Seidita, Massimo Cossentino, and Salvatore Gaglio. Using and extending the spem specifications to represent agent oriented methodologies. In Michael Luck and Jorge J. Gómez-Sanz, editors, *AOSE*, volume 5386 of *Lecture Notes in Computer Science*, pages 46–59. Springer, 2008.

[SS03] Arnon Sturm and Onn Shehory. A framework for evaluating agent-oriented methodologies. In Paolo Giorgini, Brian Henderson-Sellers, and Michael Winikoff, editors, *AOIS*, volume 3030 of *Lecture Notes in Computer Science*, pages 94–109. Springer, 2003.

[STP10] Hongyuan Sun, John Thangarajah, and Lin Padgham. Eclipse-based prometheus design tool. In Wiebe van der Hoek, Gal A. Kaminka, Yves Lespérance, Michael Luck, and Sandip Sen, editors, *AAMAS*, pages 1769–1770. IFAAMAS, 2010.

[WD00] Mark F. Wood and Scott A. DeLoach. An overview of the multiagent systems engineering methodology. In Paolo Ciancarini and Michael

Wooldridge, editors, *AOSE*, volume 1957 of *Lecture Notes in Computer Science*, pages 207–222. Springer, 2000.

[WJK00] Michael Wooldridge, Nicholas R. Jennings, and David Kinny. The gaia methodology for agent-oriented analysis and design. *Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, 2000.

[ZJW03] Franco Zambonelli, Nicholas R. Jennings, and Michael Wooldridge. Developing multiagent systems: The gaia methodology. *ACM Trans. Softw. Eng. Methodol.*, 12(3):317–370, 2003.

This article was processed using the LaTeX macro package with LLNCS style

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

Benjamin Hirsch[1], Tina Balke[2], and Marco Lützenberger[1]
firstname.lastname@{dai-labor|uni-bayreuth}.de

[1] DAI-Labor, Technische Universität Berlin
Faculty of Electrical Engineering and Computer Science
[2] Chair of Information Systems Management, Universität Bayreuth
Faculty of Law, Business Administration and Economics

Abstract. This paper 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. After analysing this discrepancy by looking at the distribution of application papers at past AAMAS conferences, we analyse relevant stakeholders for agent application papers and provide suggestions for them.
The aim of this paper is twofold. First, we want to provide a basis for discussing the state and status of agent applications in research oriented conferences. Secondly, we provide initial guidelines for authors and reviewers which might further the acceptance of application oriented work in the agent community.

1 The Case

It is generally agreed that agent technology is a topic mainly confined to the realm of research. From key note speeches to the AgentLink roadmap [8, 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. Thus, looking at the IFAAMAS charter [6] 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 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 as well as public awareness for agents, are major aims of the

community. In order to achieve these aims, the AgentLink Roadmap [8, p. 50] points out "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 the other 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.

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

To emphasise the problem, we examined the main track proceedings [4, 5, 10, 14–16] 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 isin 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.

Of course, one can argue that applications are adequately represented by means of industry tracks in theoretical applications, as well as workshops and conferences focusing on the application side of agent technology. However, industry tracks are often only visited by like-minded researchers or industry players.

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

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 those are actually show stoppers for actually *using* the theories. Thus, for industry to consider adopting a new technology, it must not only learn about it in the first place, but in addition learn on how it could help and support the 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 [3, p. 11].
- In order to learn about industry needs and thinking and get in touch with industry partners application papers are a significant step.

That is why this paper tries to address this disconnect with regard to R&D, by focusing both on the agent community's as well industry's point of views. It

thus tries to answer how to move from the mission statements to reality in terms of conference representation.

Therefore, the paper is structured as follows: starting from a short identification of the main stakeholders of agent applications and implementation works, we focus on the interests of the different stakeholders with regard to agent applications and implementations in Section 2. Starting from these stakeholder interests we will then focus on suggesting derived thereof guidelines for both authors and reviewers of agent application and implementation papers and contrast these to the existing AAMAS review criteria in Section 3.

With this paper we intend to stimulate a discussion within the agent community about its wants and needs of the community and possible ways of achieving them in general, and with regard to conference papers in particular.

2 The Stakeholders

Having presented the discrepancy between the IFAAMAS mission statement and the realization of these statements with regard to the representation of application papers in the respective conference, in this section we would like to 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.

Looking at agent application & implementation papers submitted to a conference one 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 shall be looked at in the next sections.

2.1 The Industry view: The Technology Adoption Life-Cycle

To start with, the first interest group's view that will be analysed is the industry. As stated earlier, with regard to scientific papers, the industries 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 [12]. Their original works focused on the adoption (in form of a purchase pattern) of hybrid seed corn by farmers in Iowa and were later generalised to fit the adoption of new ideas and technologies in general [11]. 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, that can be portrait in the innovation context.

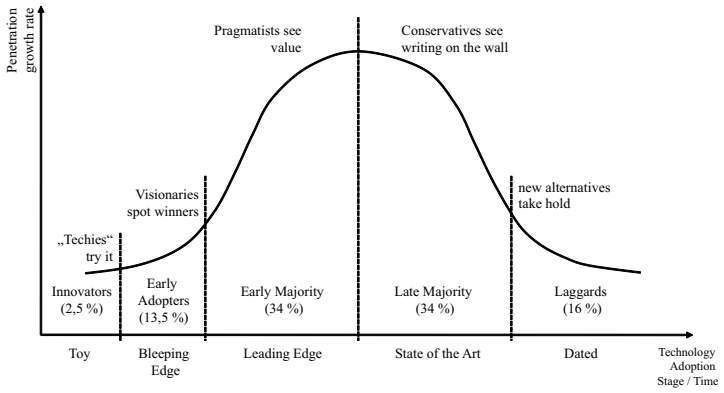


Fig. 2. The Technology Adoption Life Cycle

Figure 2 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 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 [11, 7].

Along the lines of these adopter groups in marketing business theory, six technology life-cycle stages have been identified, that are sketched in figure 2 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, ones 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's 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. Thus, unlike with 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 [9]. But what stimulates industry

to become aware of a new technology? 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.

"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 is 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.

2.2 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 very revenue oriented, academia's goals are somewhat different. Thus – putting 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 partner – 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.

2.3 The Reviewers view:

According to Smith [13], 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. One 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 repayment in their own time. The benefits they have from the task reach from the contribution they make to the research community, to the fact that they might get idea 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 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. The review guidelines given for the past AAMAS conferences are shown in the next section. 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 often demanded to have a general and not only case specific validity results in problems when evaluating r&D work. As pointed on beforehand 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 doesn't necessarily go along with the review process. We identify this discrepancy as one reason for the issues raised in Sec. 1. In order to be able to make specific suggestions and start a discussion on how this discrepancy can be reduced, in the next section we have a closer look at AAMAS review criteria and based on these criteria make suggestions for improvement.

3 Guidelines

In the preceding sections we have established that application oriented papers are a tiny minority. The question that needs to be answered is how this imbalance can be changed. We approach this from two sides, the view of the author and the view of the reviewer.

3.1 For Authors

As opposed to purely theoretical papers, applications of agent technology can often not build on prior work (or previous implementations), nor can they be reproduced easily. The costs of implementing demonstrators or prototypes also precludes the parallel implementation of different versions, for example to provide the basis for comparing different approaches, agent-based or otherwise. Therefore, authors need to ensure that reviewers understand these limitations. From this follows that authors need to make clear that the paper is an application paper, so that the reader adapts his or her expectations accordingly.

When writing application papers, it is easy to end up doing a pure system description. While this can and should be an important part of the paper, it should not be the sole content. This is because the ultimate goal of publications in general is not only to report on work that has been done, but to provide novel insight. A implemented system *might* give such an insight, but as often

as not it does not. As it is usually impractical or even impossible for others to re-implement the system to reproduce the outcome, it is important to discuss design decisions in order to allow the reader to appreciate the made choices.

Following up on the previous point, the paper should allow the reader to learn something (other than that there exists a certain system implementation). Usually, implementations make use not only of agent frameworks but also of theoretical works and approaches. These theoretical aspects should be made clear, discussed, and reflected upon. It is for example our experience that a sizable number of theories that purport to be "practical" are in fact not, be it due to some "hand waving" over implementation aspects that turn out to be not so simple, be it due to scalability issues, or due to extreme simplifications or unrealistic assumptions made. If application papers clearly state the issues and limitations of certain approaches, this feedback can again be used to improve upon the original work. Finally, the authors should try and extrapolate from their implementation.

Below, we give a number of questions that an application oriented paper (and probably others too) should answer:

What do you want to say? The paper should clearly state that an application is described. Application papers should nevertheless also state the initial motivation for the application. What problem was addressed / focused on by the paper?

Who Should Read the Paper, and Why? Related to the previous point, the paper should explicitly state to whom the paper is addressed, e.g. industry, practitioners, domain experts. However, there are no commonly accepted categories.

How did you go about? It might be advantageous to focus partly on the methodology used to develop the application. This might focus the work on areas of the methodology that need refinement. Design decisions need to be discussed properly and in detail.

What new did you learn? The paper should clearly state in how far the work has created new knowledge about implementation, theory, or applications thereof. How does the work impact theories that were used?

Why does it matter? What is it that can be taken away from the paper, in terms of relevance to the field? Is a generalisation from the specific application to a general field possible? What are potential limitations of the generalisation?

3.2 For Reviewers

AAMAS reviewer guidelines alert reviewers to look for different aspects when reading an application paper. For example, they state for *technical quality* that

> If the paper describes an application, is there:
> – a clear and compelling motivation for why a multi-agent approach is important?

– a careful description of the design and implementation of the system?
– a thorough evaluation of the system with respect to a clearly-stated set of functional and quality requirements?

For other relevant aspects of the review, such as *originality*, *relevance*, or *significance* there are no review hints specifically for application descriptions.

The author's guidelines in a way also guide the reviewer, in that the reviewer needs to asses in how far the posed questions are answered by the paper. Depending on the background of the reviewer, he should try and asses the impact that the paper can have on his community, but also on the other. This means that someone with a theoretical background should focus not only on the theoretical aspects but also try and assess in how far the paper makes valid points for practitioners.

Additionally, the relevance of the work to the intended audience needs to be taken into account. In the case of AAMAS, it is our belief that practical papers are indeed relevant to the community at large.

We realise that reviewing is very much dependent on one's background, and even papers reviewed by colleagues working in similar fields can differ wildly. It has to be said here that AAMAS gives fairly extensive guidelines to reviewers to try and ensure consistent reviews.

4 Conclusion

It is our belief that application descriptions are an important part of the work being done in the agent community, and that it should therefore play a lager 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 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.

References

1. G. M. Beal, E. M. Rogers, and J. M. Bohlen. Validity of the concept of stages in the adoption process. *Rural Sociology*, 22(2):166–168, 1957.
2. J. M. Bohlen and G. M. Beal. The diffusion process. Special Report 18, Iowa State College, May 1957.
3. CORDIS. Rules for submission of proposals, and the related evaluation, selection and award procedures, August 2008.
4. K. S. Decker, J. S. Sichman, C. Sierra, and C. Castelfranci, editors. *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems, Budapest, Hungary*. International Foundation for Autonomous Agents and Multiagent Systems, May 2009.

5. E. Durfee, M. Yokoo, M. Huhns, and O. Shehory, editors. *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems, Honululu, HI, USA*. Association for Computing Machinery, May 2007.
6. IFAAMAS. Charter for the international foundation for autonomous agents and multiagent systems.
7. T. Levitt. Exploit the product life cycle. *Harvard Business Review*, 43(6):81–94, 1965.
8. M. Luck, P. McBurney, O. Shehory, and S. Willmott. *Agent Technology: Computing as Interaction – A Roadmap for Agent Based Computing*. AgentLink, 2005.
9. V. Marik and D. McFarlane. Industrial adoption of agent-based technologies. *IEEE Intelligent Systems*, 20(1):27–35, 2005. IEEE Educational Activities Department.
10. L. Padgham, D. C. Parkes, J. Müller, and S. Parsons, editors. *Proceedings of the 7th International Conference on Autonomous Agents and Multiagent Systems, Estoril, Portugal*. International Foundation for Autonomous Agents and Multiagent Systems, May 2008.
11. E. M. Rogers. *Diffusion of Innovations*. Free Press, 1962.
12. B. Ryan and N. C. Gross. The diffusion of hybrid seed corn in two iowa communities. *Rural Sociology*, 8:15–24, 1943.
13. A. J. Smith. The task of the referee. *Computer*, 23(4):65–71, April 1990. IEEE Computer Society Press.
14. P. Stone and G. Weiß, editors. *Proceedings of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems, Hakodate, Japan*. Association for Computing Machinery, May 2006.
15. W. van der Hoek, G. A. Kaminka, Y. Lespérance, M. Luck, and S. Sen, editors. *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems, Toronto, Canada*. International Foundation for Autonomous Agents and Multiagent Systems, May 2010.
16. P. Yolum, K. Tumer, P. Stone, and L. Sonenberg, editors. *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems, Taipei, Taiwan*. International Foundation for Autonomous Agents and Multiagent Systems, May 2011. To appear.

Dynamically Adapting BDI Agent Architectures based on High-level User Specifications

Ingrid Nunes[1,2], Simone Barbosa[1], Michael Luck[2], and Carlos Lucena[1]

[1] PUC-Rio, Computer Science Department, LES - Rio de Janeiro, Brazil
{ionunes,simone,lucena}@inf.puc-rio.br
[2] King's College London, Strand, London, WC2R 2LS, United Kingdom
michael.luck@kcl.ac.uk

Abstract. Users are facing an increasing challenge of managing information and being available anytime anywhere, as the web exponentially grows. As a consequence, assisting them in their routine tasks has become a relevant issue to be addressed. In this paper, we introduce a software framework that supports the development of Personal Assistance Software (PAS). It relies on the idea of exposing a high-level user model in order to increase user trust in the task delegation process as well as empowering them to manage it. The framework provides a synchronization mechanism that is responsible for dynamically adapting an underlying BDI agent-based running implementation in order to keep this high-level view of user customizations consistent with it.

Keywords: Personal Assistance Software, Framework, User Agents, User Modeling, Adaptation, Software Architecture.

1 Introduction

Personal Assistance Software (PAS) is a family of systems whose goal is to assist users in their routine tasks. The popularity of these systems, e.g. task managers and trip planners, is increasing as the web grows, because people are increasingly facing the challenge of dealing with huge amounts of information and being constantly accessible through mobile devices. In this context, the development of PAS is strongly related to personalization and recommender systems but brings several challenges. In particular, individual user characteristics must be captured to provide personalized content and recommendations for users; i.e. there is a need to elicit, represent and reason about user preferences. In addition, as the typical scenario used in research on preferences is online stores, two key concerns are that users are generally unwilling to provide information and they cannot be expected simply to blindly trust recommendations. Research on implicit learning [4] (learning without users providing explicit information) and explanation interfaces [12] has been addressing these issues, respectively.

Most current research is concentrated on identifying user preferences with elicitation and learning processes, in order to personalize systems solely in terms of data. In our work, we examine a different scenario: we aim to support the development of PAS able to automate routine tasks, in a way that enables configuration directly by users, by choosing from the *services* it provides and customizing them. Such systems must

thus be able to have their architecture adapted dynamically, which is the main issue addressed in this paper. These adaptations must also take into account preferences in order for agents to be able to act appropriately on behalf of users. Furthermore, the ability to understand what the system knows about users and providing them with a means for controlling the system are key issues in automation. Since users directly benefit from interacting with their personal application, they can tolerate spending more time in providing it with information, in comparison to web applications, in which implicit learning is essential, since these are not personal user systems.

In response, in this paper, we introduce a software framework that provides a reusable infrastructure for the development of PAS. The main characteristic of the framework is the adoption of a high-level user model exposed to users (*transparency*), and which they can manage (*power of control*). Here, user customizations are realized by lower level software components, thus we propose a synchronization mechanism that is responsible for dynamically adapting an underlying BDI agent-based running implementation and keeping this high-level view of user customizations consistent with it.

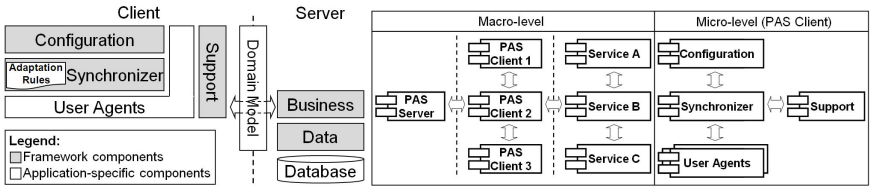
The remainder of this paper is organized as follows. Section 2 describes our PAS framework and our dynamic adaptation mechanism. Section 3 makes a qualitative analysis of our proposal by discussing relevant aspects from it. Section 4 presents related work followed by Section 5, which concludes this paper.

2 A Two-level Framework for Developing PAS

In this section we provide an overview of our framework and detail its two main components: the PAS Domain-specific Model (DSM), which models user customizations in a high-level way, and the synchronization mechanism for dynamic adaptation in belief-desire-intention (BDI) architectures. This architecture provides abstractions and a reasoning mechanism, which are adequate and widely adopted to develop cognitive agents, in particular agents able to automate user tasks. Moreover, this architecture is composed of loosely coupled components and makes an explicit separation between what to do (goals) and how to do it (plans). These two characteristics of the architecture make it very flexible, which facilitates the adaptation process by evolving and changing components with a lower impact in the running system.

2.1 Framework Overview

Our framework is a reusable software infrastructure for developing a family of systems whose goal is to assist users in their routine tasks in a customized way. The main characteristic of our approach is the adoption of two levels of abstractions that capture user customizations: the (end-)user and implementation levels. The first makes user customizations explicit and modularized, as well as understandable by users, so that the current state of the PAS is transparent to users, and empowers them to manage customizations. As users evolve and personalize their PAS over time, and there is an underlying implementation that must be consistent with the user high-level specifications, there is a need to keep both levels synchronized. So changes at the user level drive dynamic adaptations in the underlying implementation, in order for the latter to



(a) The Client-server Architecture. (b) Macro and Micro-level Views.

Fig. 1: PAS architecture.

reach a state consistent with the user-level representation. User *customizations* at the user level are represented in a high-level language, and can be: (i) *configurations*: direct and determinant interventions that users perform in a system, such as adding or removing services and enabling optional features. These configurations can be related to environment restrictions (e.g. a device configuration, or functionalities provided by the system); or (ii) *preferences*: information about user values that influences their decision making, and thus can be used as resources in agent reasoning processes. Preferences typically indicate how a user rates certain options better than others in certain contexts.

The implementation of the PAS can be seen not only as a unique software system but as a set of software assets that can be integrated to form different customized applications for diverse users. Therefore, all optional parts of the PAS must be modularized in the code, so they can be added and removed from the running application instance. Each set of assets that realizes a variable part of the PAS is the implementation-level representation of user customizations.

Our framework supports the implementation of PAS using a client-server model. User data is stored in a centralized database in the server. The server is structured with a *Business* layer that provides services for PAS clients, and a *Data* layer composed of Data Access Objects (DAOs) [1] that access the database. Both can be extended to incorporate application-specific services. PAS clients, in turn, have: (i) a *Configuration* layer, which enables users to manage the model that represents customizations at a high-level; (ii) a *User Agents* layer, which implements application-specific services and functionalities for users; (iii) a *Synchronizer* layer, which executes adaptation rules to keep (i) and (ii) consistent; and (iv) a *Support* layer, which provides core PAS services, such as login. The client-server model is illustrated in Figure 1a, highlighting the layers in which the framework is structured. The *Domain Model* is a layer common to both the client and the server sides of the PAS, and is thus shown between them in Figure 1a.

From a macro-level viewpoint, PAS clients can be seen as autonomous and proactive agents that represent users in a multi-agent system. PAS clients communicate to the PAS server to access stored information and other business services, and can communicate with each other. They also communicate with services available on the web. For instance, if our framework is instantiated for the trip planning domain, services are agents representing hotel and airline companies. A PAS client at the macro-level can be seen as a single agent representing a user, but at the micro-level it is decomposed into

autonomous components (also agents), each of which has different responsibilities. Figure 1b shows both macro and micro-level views of an instance of our framework.

2.2 PAS Domain-specific Model

The PAS DSM has a key role in our approach. It is a meta-model that defines abstractions to model domain-specific concepts of PAS, such as features and preferences. The goal is to use abstractions close to the vocabulary of users. The PAS DSM was previously defined in [10], however we introduce the parts that are relevant for this paper (see [10] for further details), and highlight improvements over the previous version. The PAS DSM consists of two parts: definition models and the user model. Definition models define abstractions of a PAS that characterize a particular PAS application, e.g. domain entities and provided features. They also serve as a basis for instantiating user models. Abstractions of definition models provide both domain entities (e.g. features and ontology concepts) to be referred to in users models, and restrictions used for defining valid user model instances. In addition, a user model can evolve over time. Definition and user models are instantiated in a stepwise fashion. The first is built by developers during the instantiation of a PAS application, i.e. it consists of *design decisions*. The latter is instantiated at runtime by users (possibly on learning from users), so it corresponds to *user decisions*.

Definition Models. There are three definition models, which are detailed as follows.

Ontology Model. The ontology model defines the set of concepts within the application domain and the relationships between those concepts. It is commonly used for knowledge representation, so its detailed description is out of the scope of this paper. We mention elements of the ontology model while describing our models, which we refer to as: (a) *Classes*: concepts of the ontology; (b) *Properties*: slots of concepts. They can store primitive values or references to other concept instances; (c) *Enumeration*: a set of named values (*EnumerationValue*); and (d) *ValueDomain*: a particular kind of Enumeration, being composed of *Values*. Value [8] is a first-class abstraction that we use to model high-level user preferences. It describes preferences not over characteristics of the object but the value it brings.

Feature Model. This model defines the set of features available for users to configure their PAS application. A feature is any characteristic relevant to the user that, depending on the user configuration, can be part of the application. For instance, it might be a functionality or a setting (e.g. the interface language). Our feature model is an extended and adapted version of feature models used in Software Product Lines (SPLs) [5], which is a new software reuse approach that aims at systematically deriving families of applications based on a reusable infrastructure with the intention of achieving reduced costs and time-to-market. A PAS application can be seen as an SPL whose products are applications customized for a particular user. However, a main characteristic of PAS is providing users with functionalities that automate their tasks, and feature models do not explicit capture it. Therefore, we have enriched feature models by distinguishing a particular kind of feature: the *autonomous features*, which provide a functionality of acting on behalf of users for performing a user task.

Each autonomous feature has a set of autonomy degrees associated with it, which means the different degrees of autonomy that the feature makes available to the user.

Possible autonomy degrees are `initiate`, `suggest`, `decide` and `execute`, which were defined based on the taxonomy presented in [9]. The availability of these autonomy degrees does not imply that this automation will be performed; this is determined in the configuration of the user model. An example is a *Flight Planner* feature that may be able to `suggest` and `execute` the process of buying a flight ticket for a user, but not to `initiate` and `decide` about it.

Preferences Definition Model (PDM). Because it is desirable that users are able to express preferences in different ways, it is necessary to have a system that can deal with them. For instance, if an application can deal only with quantitative preferences, preferences expressed in a qualitative way will have no effect on the system behavior if there is no mechanism to translate them to quantitative statements. The PDM specifies restrictions over preferences users can express, i.e. the purpose of this model is to specify *how* users can express preferences and *about which elements* of the ontology model.

Users can provide different kinds of preferences statements in the user model, which were chosen using preference statements collected from different individuals and from existing models to reason about preferences. The aim was to consider the different kinds of preference statements in order to maximize the expressiveness of users. There are five different kinds of statements: *order* (an order relation between two elements), *rating* (users attribute a rate to a target), *minimization* or *maximization* (user preference is to minimize or maximize an element), *reference value* (users indicate one or more preferred values for an element) and *don't care* (a set of elements the user does not care about, they are equally (un)important to the user). A preference target can be associated with a subset of these kinds of preference statements, so that it is possible to express those kinds of preference about the target. There is an exception for rating preferences, whose definition is based on rating domains: rating preferences related to a target can be expressed if that target is associated with a rating domain, and the rating must belong to the defined domain. This domain can be numeric (either continuous or discrete), with specified upper and lower bounds; or an enumeration. Targets can be one of four types, which are those described in the ontology model.

User Model. The user model specifies user customizations for each individual user, and is built on abstractions from the feature model and the PDM of an application. These models are used to constrain the user model instance. As stated before, user customizations can be either configurations or preferences, so the user model is composed of two parts: a configuration and a set of preferences. As the user model is managed by users, the *Configuration* layer of the framework provides a graphical interface to manipulate it. The configuration comprises a set of features that are selected from the feature model, and a set of feature autonomy configurations. The set of selected features must be valid according to the feature model. A feature autonomy configuration stores the autonomy *degree* that a user defines for an autonomous feature, i.e. a feature whose autonomy degree set is not empty. Therefore, the feature autonomy configuration associated with each autonomous feature is a subset of autonomy degrees that it provides.

2.3 Dynamically Adapting PAS Clients

Our approach requires synchronization between the user and implementation levels, because the former represents user customizations at a high-level and the latter must re-

flect these customizations. This is achieved by an adaptation process, which is triggered by changes performed in the user model. Figure 2a shows how the previously presented models are related, and how user models are based on them. The feature model provides the features available for users, and autonomy degrees of autonomous features, and the ontology model provides users with a vocabulary to make preference statements. Based on these two models, users create or *evolve* a user model, and it is then validated according to the feature model, which also contains constraints over selected features, and the PDM. A preference statement is valid if the target has the associated allowed preference (or a rating domain, in case of rating preferences) defined in the PDM, or if there is not definition in the PDM, since the default is that all preference types are allowed.

When the user model changes, the synchronization mechanism causes the implementation-level of the PAS client to be consistent with it. The mechanism consists of knowledge that captures how the implementation level must be adapted according to changes in the user model and a process that uses this knowledge to adapt the implementation level as the user model evolves. This knowledge is composed of four main concepts:

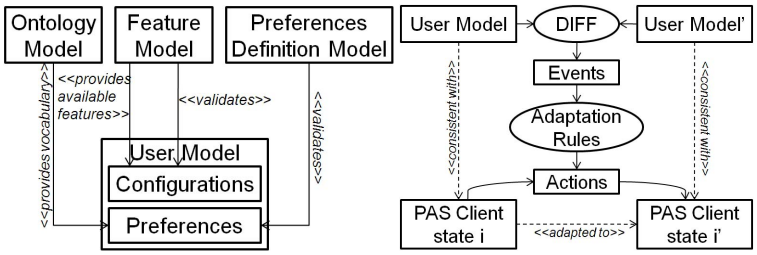
Events correspond to changes that occur in the user model, for instance adding or removing the autonomy degree of a feature.

Event Categories group a set of related events. For example, the events above belong to the autonomy degree category.

Actions are the changes to be performed over software components at the implementation level, e.g. adding or removing agents, beliefs or goals. Here, a *software component* is any software asset that is part of the implemented system, and there are two types of coarse-grained software components: *components* and *agents*. The former provides a reactive behavior, while the latter provides autonomy and pro-activity, has its own thread of execution, and is able to communicate through messages with other agents. As a result, agents are composed of finer-grained software components, namely capabilities, beliefs, goals and plans, required parts of the BDI architecture [13], which we adopt to design and implement agents – this design decision is discussed later.

Rules establish connections between events and actions, and are applied when an event in some category associated with the rule occurs. In this situation, the rule generates the appropriate set of actions to be executed, according to the event(s) that occurred. It is important to highlight that rules are not functions, in the sense that the same set of events does not always generate the same (and unique) set of actions, because generated actions may depend on the previous state of the user model. For instance, let $R1$ be a rule stating that agent $A1$ must be part of the running system if features $F1$ and $F2$ are selected. Then, if the event $Feature(F1, Add)$ occurs, $R1$ generates the action $Agent(A1, Add)$ only if feature $F2$ was previously selected or the event $Feature(F2, Add)$ also occurres.

These concepts are used in our adaptation process, shown in Figure 2b, implemented as part of the `Synchronizer` agent (Figure 1b). The process uses as input a previous and an updated version of a user model, and a set of adaptation rules. A new user model (initial state) is a configuration with the core features selected and no preference statements. First, the set of all events that caused the user model to be updated is generated. Then, the set of rules that are triggered by at least one of the categories from those events is selected. Next, a set of actions to be executed is constructed from the union



(a) Relationships among models. (b) Adaptation Process.

Fig. 2: Evolving a PAS architecture.

```
interface Event
    Set<EventCategory> getEventCategories()

interface EventCategory

interface AdaptationAction
    void doAction() throws AdaptationException

interface AdaptationRule
    Set<EventCategory> getObservableEventCategories()
    List<AdaptationAction> getAdaptationActions(
        UserModel oldUserModel, UserModel newUserModel,
        Set<Event> events) throws AdaptationException
```

(a) Interfaces of our Mechanism.

```
rule AutonomyDegree (
    eventCategory AutonomyDegree (
        feature = AccommodationPlanner,
        autonomyDegree = DECIDE),
    ONA = {
        action Plan (
            capability = AccommodationService,
            plan = DecideAccommodationPlan) },
    OFFA = {
        action Plan (
            capability = AccommodationService,
            plan = DecideAccommodationUserPlan),
        action Belief (
            capability = AccommodationService,
            plan = DecideAccommodationActionBelief) })
```

(b) Autonomy Degree Adaptation Rule.

Fig. 3: Framework interfaces and adaptation rule.

of each set of actions generated by each selected rule. Finally, each action is executed. The complexity and performance of the algorithm that implements this process is not discussed because it depends of how application-specific rules generate actions.

It can be seen that the process is simple in that it does not specify any concrete event, event category, rule or action. It is developed in this way to address any instance of any one of these concepts. They are built in the framework as interfaces (Figure 3a), therefore one can create concrete classes by implementing such interfaces. Therefore, we have a generic structure to make adaptations. The goal of providing a generic structure is to make it extensible. However, we also provide a set of predefined events, event categories, rules and actions as part of our framework, presented in Table 1. Actions allow manipulation on software components of the BDI architecture. Due to space restrictions we do not describe each of the elements of this set, but give an overview by explaining one of the rules – the $FeatureExpression$ rule. This rule receives as parameter a logic formula expression, in which literals are features. The rule is associated with a set of event categories composed of $Feature(F)$ for each literal F of the formula. Then, the generated set of actions will be the following: if the formula is evaluated to a $false$ value in the previous version of the user model and a $true$ value in its updated version, the set of actions will be the *ONA* set with action operators set to add together with the *OFFA* set with the action operators set to $remove$. If the formula was evaluated to

Events	Rules
AutonomyDegree(F, AD, ET) Feature(F, ET) Preference(P, ET)	AutonomyDegree(ADC, ONA, OFFA) FeatureExpression(LF, ONA, OFFA) OptionalFeature(F, ONA, OFFA) *Preference(PEC)* – Abstract
Event Categories	**Actions**
AutonomyDegree(F, AD) ClassPreference(Class) EntityPreference(Class) EnumPreference(Enum) EnumValuePreference(EnumValue) Feature(F) *Preference* – Abstract PropertyPreference(Property) ValueDomainPreference(VD) ValuePreference(V)	Agent(A, AO) Belief(C, B, AO) BeliefSetValue(C, B, O, AO) BeliefValue(C,B,O) Capability(A, C, AO) Component(Female, Male, AO) ComponentValue(Female, Male) Goal(A, G, AO) Plan(C, P, AO)
Legend: ET: Event Type (Add, Remove); F: Feature; AD: Autonomy Degree, P: Preference; ADC: Autonomy Degree Category; VD: Value Domain; V: Value; ONA: On Action Set; OFFA: Off Action Set; LF: Logic Formula; PEC: Preference Event Category; AO: ActionOperator (Add, Remove, Set); A: Agent; C: Capability; G: Goal, B: Belief; P: Plan; O: Object.	

Table 1: Predefined Set of Events, Event Categories, Rules and Actions.

$true$ and $false$ respectively, the action operators would be inverted. Figure 3b shows the information that is defined in a rule, which is applied when the autonomy degree `DECIDE` of the `AccommodationPlannerFeature` changes.

2.4 Implementation Details

Several agent platforms implement the BDI architecture, e.g. Jason, Jadex, 3APL and Jack. Most of these are based on Java, as in our framework, but agents are implemented in these platforms using a particular language that is later compiled or interpreted by the platform. This prevents us from taking advantage of the Java language features, such as reflection and annotations, that can help with the implementation of our adaptation mechanism, and complicate integration with other frameworks. Due to this limitation of existing BDI agent platforms, we have developed BDI4JADE [11], a BDI layer on top of JADE[3] (Java-based agent platform that provides a robust infrastructure to implement agents, including behavior scheduling, communication and yellow pages service). We have leveraged these features, provided the BDI abstractions, and built a BDI reasoning mechanism for JADE agents. Agents developed with our JADE extension are implemented only in Java, and not by using additional files in XML, for example, as used in Jadex. As BDI4JADE components are extension of Java classes, they can be instantiated by other frameworks and plugged into the running application, as opposed to other agent platforms that instantiate and manage their components.

This was needed for supporting the implementation of our adaption mechanism, which is extensively based on the use of the Spring framework,[4] a Java platform that provides comprehensive infrastructure support for developing Java applications. It is designed to be non-intrusive, so the domain logic code generally has no dependencies on the framework itself. Mostly, we took advantage of the Dependency Injection

[3] http://jade.tilab.com/

[4] http://www.springsource.org/

and Inversion of Control module, which allows declaration of the application software components (*beans*, in the Spring terminology) and dependencies among them. Thus, actions in our synchronization mechanism receive strings (bean identifiers) as parameters, referring to software components of the running PAS to be adapted. These bean declarations can correspond to either a singleton or a prototype instance of the bean. In addition, rules and actions are also declared in the Spring configuration file.

3 Discussion: a Qualitative Analysis of our Approach

As an initial step for the evaluation of our framework, we provide a qualitative evaluation of key aspects of our framework regarding decisions made about architecture and software quality attributes. The framework was instantiated in a simple application in the trip domain, in order to test the infrastructure. As our long term research aims to increase user trust in PAS by exposing user models at a high level, part of our future work will involve a user study to validate this aspect of our approach.

3.1 Advantages of a Two-level Architecture

Our previous work has shown that user preferences and configurations can be seen as a concern that is spread all over PAS [10]. This is an intrinsic characteristic of preferences because they play different roles in reasoning and action [6]. Systems that adapt their behavior according to an evolving specification, in our case the user model, must have an architecture that supports variability and its management. This issue is less evident in systems that are concerned only with content customization, as a single and static architecture is sufficient for providing personalized data, yet the scope of our family of systems is wider than that.

The key advantages of our high-level user model are twofold: (i) it provides a complementary representation that is a global view of user customizations, thus allowing variability management and traceability (captured by rules); and (ii) it provides a means for users to understand their model (transparency) and manage it (power of control). Moreover, our two-level abstraction architecture brings additional benefits: (i) user customizations have an implementation-independent representation; (ii) the vocabulary used in the user model becomes a common language for users to specify configurations and preferences; (iii) the user model modularizes customizations, allowing modular reasoning about them; (iv) the user model can be used in mixed-initiative approaches, in which learning techniques can be used to create initial and updated versions of user models, and users have a chance to change them; and (v) by dynamically adapting PAS, we eliminate unnecessary reasoning (which can be time-consuming) if customizations are represented as control variables that regulate the control flow of the system.

3.2 Benefits of Providing a BDI Agent-based Implementation

Our framework uses software agents at the implementation level of PAS, following the BDI architecture. We made this design decision due to the benefits of adopting an agent-based approach. The most important reason is that the BDI architecture is very

flexible. In this architecture, components are loosely coupled and there is an explicit separation between what to do (goals) and how to do it (plans). Thus, evolving and changing an agent architecture is easier in comparison with objects. The BDI architecture thus facilitates the implementation of user customizations in a modularized way so that components can be added and removed as the user model changes. The adaptation rule presented in Figure 3b helps to illustrate this situation: for making an accommodation reservation (top level goal), the agent has to achieve three subgoals – search and suggest, decide and execute. There are two ways of deciding for an accommodation: (i) asking the user to decide; or (ii) deciding on behalf of the user. Simply by changing the plan that is part of the agent plan library, the agent behavior is changed without impacting the course of actions or other actions to be executed.

In addition, the BDI architecture and other agent approaches are composed of human-inspired components, thus reducing the gap between the user model (problem space) and the solution space. Furthermore, plenty of agent-based artificial intelligence techniques have been proposed to reason about user preferences, and can be leveraged to build personalized user agents.

3.3 Software Quality Attributes

By providing a software framework for developing PAS, we also provide a reusable infrastructure to build a family of systems. In addition to following the two-level approach we are proposing, in order to build a high quality architecture, we made design decisions that take into account software quality attributes, as follows.

Reuse. The primary advantage of a framework is reuse, together with its benefits, e.g. higher quality and reliability in a relatively short development time. Using our framework speeds up the process of building PAS systems due to the infrastructure that is ready-to-use and ready to extend, including models that are common in our target application domain. In addition, as we considered good software engineering practices to develop our framework, such as design patterns, this will be inherited by the framework instances.

Maintainability. User customizations are a cross-cutting concern, because they are spread over different points of the application. In our approach, individual customizations are localized in each part of the system that they are related to: if a behavior of an agent (plans, goals) depends on preferences about *X*, this variability will be encapsulated in that part of the system. At the same time, the high-level user model and rules provide the information needed to trace and manage user customizations as a whole. This modularization of user customizations thus facilitates the maintenance of PAS because software components of the system have high cohesion and are loosely coupled. For the same reason, this structure reduces the impact of evolving the system, such as adding a new user agent with new services for users.

Scalability. PAS typically involves complex algorithms, which require much processing, such as reasoning about preferences. Running this kind of system with a large number of users at the same time, on a single server, thus is not scalable. As a consequence, we adopted a client-server model to distribute this processing of users across different clients, by still allowing users to access the application configuration in different clients, making it possible to build different client versions.

3.4 Limitations

Both our framework and its underlying approach for developing PAS have some limitations. Our synchronization algorithm generates a set of actions that are performed at the implementation level of PAS. Nevertheless, we do not consider *order* in such actions. This is important mainly when we have dependencies among features. Up to now, our studies have not required consideration of action order, but investigating scenarios in which order matters is part of our future work.

In addition, we have not considered the consistency analysis of the adaptation process and user preferences. The correctness of the adaption process is related to the correct definition of rules and actions. It is responsibility of developers to ensure that they are specified in the right way. In addition, users have different forms of expressing preferences, which might be inconsistent. We do not provide mechanisms for detecting such situations.

There are aspects of PAS that are not covered by our approach: learning; security and privacy; and user explanations. Our goal is to extend our framework architecture to accommodate such modules, using this as a reference architecture for PAS. A complete approach for the first two aspects is out of the scope of our research, but we have already taken steps to integrate user explanations into our framework. Even though users can control their user models, there are decisions that agents make on their behalf. Explaining to users the rationale behind decisions is another important factor to increase user trust in PAS.

4 Related Work

Much research has been carried out in the context of PAS. For example, a multi-agent infrastructure for developing personalized web-based systems, Seta2000 [2], provides a reusable recommendation engine that can be customized to different application domains. Huang et al. [7] describe a personalized recommendation system based on multiple-agents, providing an implicit user preference learning approach, and distributing responsibilities of the recommendation process among different agents, such as learning, selection & recommendation and information collection agent. The Cognitive Assistant that Learns and Organizes (CALO) project [3] has also explored different aspects to support a user in dealing with the problems of information and task overload.

However, in such work, personalization in the system is in the form of data, so that architecture adaptations are not investigated, which is the main issue addressed in this paper. None of them address an evolving system, and consequently systems are not tailored to users' needs in the sense of features that the system provides. In particular, [2] and [7] provide a reusable infrastructure for building web-based recommender systems, but they do not provide new solutions in the context of personalized systems: they leverage existing recommendation techniques and provide implemented agent-based solutions.

5 Conclusion

In this paper we have presented a software framework and a dynamic adaptation mechanism, which provide a reusable infrastructure for developing Personal Assistance Software (PAS). The main idea underlying our framework is to provide a two-level view of user customizations: an end-user high-level model and the realization of customizations at the implementation level. The end-user view aims to give power of control over task automation to users. The framework aggregates a dynamic adaptation mechanism that is responsible for keeping these two levels consistent, and comprises: a PAS Domain-specific Model, a synchronization mechanism, graphical interface components to manipulate models, support components that provide core functionalities, models persistence, a BDI layer over JADE, and patterns for implementing agents. We have evaluated our approach with a qualitative analysis, identifying its main benefits and software quality attributes.

Our short term future work includes addressing some current limitations of our approach, which are dealing with the order of actions and user explanations. In addition, recently, we performed a user study in which we collected about 200 preferences specifications that will be used to refine our preferences model.

References

1. Alur, D., Malks, D., Crupi, J.: Core J2EE Patterns: Best Practices and Design Strategies. Prentice Hall PTR, Upper Saddle River, NJ, USA (2001)
2. Ardissono, L., Goy, A., Petrone, G., Segnan, M.: A multi-agent infrastructure for developing personalized web-based systems. ACM Trans. Internet Technol. 5(1), 47–69 (2005)
3. Berry, P.M., Donneau-Golencer, T., Duong, K., Gervasio, M., Peintner, B., Yorke-Smith, N.: Evaluating user-adaptive systems: Lessons from experiences with a personalized meeting scheduling assistant. In: IAAI'09. pp. 40–46 (2009)
4. Claypool, M., Le, P., Wased, M., Brown, D.: Implicit interest indicators. In: Proceedings of the 6th international conference on Intelligent user interfaces. pp. 33–40. IUI '01, ACM, New York, NY, USA (2001)
5. Czarnecki, K., Eisenecker, U.W.: Generative programming: methods, tools, and applications. ACM Press/Addison-Wesley Publishing Co., USA (2000)
6. Doyle, J.: Prospects for preferences. Computational Intelligence 20, 111–136 (2004)
7. Huang, L., Dai, L., Wei, Y., Huang, M.: A personalized recommendation system based on multi-agent. In: WGEC '08. pp. 223–226. IEEE (2008)
8. Keeney, R.L.: Value-focused thinking – A Path to Creative Decisionmaking. Havard University Press (1944)
9. Malinowski, U., Kühme, T., Dieterich, H., Schneider-Hufschmidt, M.: A taxonomy of adaptive user interfaces. In: HCI'92. pp. 391–414. USA (1993)
10. Nunes, I., Barbosa, S., Lucena, C.: An end-user domain-specific model to drive dynamic user agents adaptations. In: SEKE'10. pp. 509–514. USA (2010)
11. Nunes, I., Lucena, C., Luck, M.: BDI4JADE: a BDI layer on top of JADE. In: 9th International Workshop on Programming Multi-Agent Systems (ProMAS 2011). Taipei, Taiwan (2011), to appear.
12. Pu, P., Chen, L.: Trust-inspiring explanation interfaces for recommender systems. Know.-Based Syst. 20, 542–556 (August 2007)
13. Rao, A., Georgeff, M.: BDI-agents: from theory to practice. In: ICMAS'95 (1995)

Socially-aware lightweight coordination infrastructures

Marc Esteva, Juan A. Rodriguez-Aguilar, Josep Lluís Arcos, and Carles Sierra

Artificial Intelligence Research Institute (IIIA-CSIC)
Spanish National Research Council
08193 Bellaterra, Spain
{marc,jar,arcos,sierra}@iiia.csic.es

Abstract. Coordination infrastructures have played a central role in the engineering of multi-agent systems (MAS). Although MAS research has produced a number of notable coordination infrastructures of varying features, these have been mainly conceived to host software agents and to facilitate multi-agent programming. Thus, human agents have been mostly kept out of the picture, hence hindering the use of agent-oriented infrastructures for the coordination of hybrid multi-agent systems. Moreover, current MAS tools supporting coordination heavily rely on a *dedicated* infrastructure. In this work we touch upon these two issues. On the one hand, we analyse the kind of coordination support required by humans in a hybrid multi-agent system. On the other hand, we propose how to achieve coordination with little, lightweight infrastructure.

1 Introduction

Multiagent systems (MAS) are composed by a group of agents that interact within an environment to achieve their common or individual goals. Typically, the achievement of such goals require the effective coordination of participant behaviours. From an engineering point of view, building a MAS entails the problems of designing a distributed concurrent system plus additional difficulties due to the autonomy of their entities. Therefore, the development of infrastructures that provide support to agent coordination play a central role in the engineering of MAS. While, initially, each MAS was designed *ad hoc* and developed its own infrastructure from scratch [1], as the area has evolved, certain tasks have been abstracted and gradually provided by MAS infrastructure as domain independent functionalities or services. The services offered by infrastructures vary from providing communication and yellow pages services, as for instance in FIPA compliant platforms [2], to give support to the execution of agent organisations or institutions [3, 4]. In this later case, the infrastructure supports the execution of the system following the coordination model defined by the regulations imposed by the organisation or institution. Furthermore, having a explicit representation of the coordination model, which is the case in most of these approaches, facilitates its run-time adaptation to continue being effective under changing circumstances.

The purpose of this paper is two-fold: (i) to review the state of the art in coordination infrastructures for MAS, and (ii) to present what we identify as the main challenges to face for the next generation of coordination infrastructures. Regarding the later, we advocate supporting human participation as one of the main challenges. In other words, to support the execution of hybrid systems where participants may be human and software agents. Notice that this systems can be regarded as a kind of open systems, and thus, they can be engineered following a MAS approach. However, developed infrastructures mainly support the participation of software agents, and the human role is limited to acting behind the scenes by customising provided agent templates that participate in the system on humans' behalf. We regard this as a limitation to the spread of agent technologies, since in many applications users are reluctant to delegate their participation in the system to an agent. For instance, in e-commerce applications where the participation in the system may imply spending the user money. We advocate, that supporting the execution of this kind of hybrid systems would increase the use of MAS technologies. In order to support human participation appropriate tools and interfaces have to be developed.

Normally, human organisations offer services that help users to participate in the system and achieve their goals. For instance, these services may provide users with information about the organisation regulations, or the steps to follow to achieve a certain goal. We regard the definition and incorporation of this kind of services, that we call *assistance services*, into coordination infrastructures as another challenge for the area. Notice that this services can provide assistance to both human and software agents.

Regulated environments for multiagent systems usually rely on a heavy infrastructure that has to be up and running before the agents can actually interact. As the central aspect of this kind of approaches is the idea that participants must accept the norms, defined at design time, that regulate it. This must happen before any action is performed by an agent. Otherwise, the consequences of the actions might be unknown by agents, and therefore any subsequent sanction would be unfair. We argue that a peer to peer (P2P) approach can provide a more light-weight infrastructure for agent coordination. The essence of P2P computing is that peers when active can be called on duty to give to the other peers knowledge declared as available and to execute services or agents that are also made available to the rest. This view permits community creation, facilitates the sharing of resources, and when created appropriately (i.e. by giving incentives to their usage) permits an explosion in usage. A key component of this approach is a repository of declarative specification of coordination patterns. Hence, agents can reason about them and instantiate the ones needed to coordinate and to achieve their goals. We illustrate this approach by applying it to electronic institutions. The result is a lightweight infrastructure and a click-away from usage.

The rest of the paper is structured as follows. Next, in section 2 we review the state of the art in coordination infrastructures. Thereafter, in section 3 we focus on supporting human participation (3.1), and assistance services (3.2). In

section 4 we present a proposal of a lightweight infrastructure based on a P2P approach, while in section 5 we conclude.

2 State of the art

As above mentioned as MAS research has evolved different coordination infrastructures providing more domain independent services have been developed. Hence, they can be reused in the the deployment of different applications reducing the development time and cost. We regard the FIPA specification for agent platforms as the first important improvement in order to provide coordination infrastructures for MAS [2]. In particular, FIPA proposed that any agent platform has to provide the following services: (i) an agent management service that manages agents executions and permits to find other agents (i.e. white pages service); (ii) and a directory service to register and discover agent services; and (iii) a reliable transport service among agents. Hence, FIPA compliant infrastructures, such as JADE [5], must provide these services to participating agents facilitating agent coordination by offering mechanisms to discover other agents and the services they offer, and a transport service transparent to agent physical locations

Organisational approaches provide a higher level abstraction to define the coordination model among agents. In general all of them include a social structure defining the roles agents may play, and a set of regulations or conventions that structure participants behaviours. Some organizational approaches use FIPA compliant platforms to support their execution, but they provide limited support to organisational concepts. However, specific infrastructures have been developed to support organisational models, as for instance AMELI [4] and S-Moise$^+$ [3], In particular, AMELI supports the execution of any electronic institutions specified using the ISLANDER editor. An institution specification defines the social structure, the dialogical interactions (scenes), agents can engage on, and the norms that establish the consequences of their. In order to support the institution execution AMELI implements the necessary mechanisms to enforce the correct execution of the institution according to its specification. One of its main features is that it accepts the participation of agents in any language and internal architecture, S-Moise$^+$ is an infrastructure to run organisations defined following the Moise model, which as an electronic institution it defines a social structure and the interactions agents may have. Moreover, it incorporates a functional specification that takes system tasks (derived from the organisational goals) and divides them into sub-tasks to be carried out by agents.

While previous infrastructures provide support to an execution of a specific organisational model, CArtAgO [6] is a framework to build infrastructures based on the Agents & Artifacts meta-model(A&A) [7]. This model proposes that apart of agents the environment is populated by a set of objects, called artifacts. Among other features artifacts can be used to coordinate agent activities. For instance, when an interaction protocol is instantiated an artifact can

be created to coordinate the protocol execution. Rather than proposing an organisational model, CArtAgO can be used to build an infrastructure to support it. For instance, in OR4MAS [8] it is used to provide an infrastructure for the Moise model. That is, the system organisation is defined using the Moise model, and CArtAgO is used to build the system infrastructure.

Related to web services, the ALIVE framework was proposed to support the engineering of service-oriented systems [9]. ALIVE proposes three design and execution levels that incorporate coordination and organisation mechanisms as a way of facilitating dynamic capabilities to web services. Specifically, the organisational level is used to dynamically select, compose, and invoke services. Notice that the dynamic selection provides an adaptation of the coordination model depending on current situations.

This later issue, the adaptation of the coordination model to varying situations, is an important topic due to the dynamicity of MAS. Although we regard this as an open research issue for coordination infrastructures, there are some works that have addressed it. In particular, in S-Moise$^+$ there is a special role *Reorg* in charge of reorganising how the task are assigned to agents, while in [10] authors propose an extension to AMELI to endow it with self-adaptation capabilities. Moreover, there are other proposals to endow organisations with self-adaptation capabilities that can be incorporated into MAS infrastructures [11, 12].

3 Coordination support

3.1 Human computer interaction

We advocate that human incorporation is one of the main challenges for coordination infrastructures. In order to do so appropriate tools and interfaces have to be developed to address human requirements, which are different of software agent ones. Hence, they require different ways of participating. For instance, interacting in the system by exchanging messages in an agent communication language is appropriate for software agents but not for human users. Another key aspect is how to represent the relevant information to successfully participate in the system. This is information about the regulations of the coordination model, the system state and the actions performed by other participants. Notice that all this information determine the the valid actions a participant can do, and normally is used in its decision process.

Obviously, web pages or 2D interfaces can be used to facilitate human participation, but we advocate that more immersive environments as *3D virtual worlds* can do a better job in supporting human participation. *3D virtual worlds* is a technology that has emerged in nowadays computing with enormous strength. Humans are social and therefore the concept of virtual worlds is very appealing as they permit a much more immersive environment for their interactions. These are computational immersive environments that emulate real world using 3-dimensional visualisation. Humans participate in those environments represented as graphical embodied characters (avatars) and operate by using simple

Fig. 1. Snapshot of a Virtual Institution execution.

and intuitive control facilities. The immersive environment provides many possibilities for representing the system state, and the regulations defined in the coordination model. For instance, other participants are represented as avatars, and they appearance can be used to display the role they are playing. We argue that 3D Virtual Worlds technology can be successfully used for ÒopeningÓ Multiagent societies to humans

This idea was explored by Bogdanovich [13] who proposed the concept of virtual institution as a combination of electronic institutions and 3D virtual worlds. In this context, electronic institutions are used to define the regulations that structure participants interactions, while users participate in the system by controlling an avatar in an automatically generated representation of the institution in the virtual world. Ongoing activities (interaction protocols) are represented as 3D room in the generated virtual world. To support the execution of this kind of systems, he proposed a run time infrastructure where a middleware causally connects AMELI and the 3D virtual world. This causal connection is performed by transforming user actions in the virtual world to messages understood by AMELI and updating the visualisation whenever the electronic institution state changes. Hence, the run-time infrastructure maintains consistent the institution state and its representation in the virtual world. The idea was further explored in the context of the itchy feet project where a prototype for the tourism domain was developed [14]. Recently, Trescak et al [15] proposed an extension of the infrastructure that among other features supports the connection of several

virtual worlds to the very same institution. Hence, the system allows users to participate from different virtual worlds. The result is an hybrid system which allows the participation of human and software agents.

Figure 1 displays a snapshot of an auction room of a virtual institution execution. We can regard that the room recreates auction rooms in real live, where buyers participants have to sit in one of the room chairs. Avatars representing software agents buyers are represented with blue skin, while the ones with green skin represent software agents controlling the auction execution. In particular, the one on the left is in charge of auctioning goods, while the one on the right is in charge of validating and announcing auction results. The appearance of avatars controlled by human users are decided by each participant. The panel on the wall is used to represent the information of the current auction round. Notice that the created representation permits the user to easily perceive who are the other participants in the auction room, the role they are playing along with information of the current state.

3.2 Assistance services

Participating in a MAS can be difficult due to the complexity of the system. On the one hand, agents must be aware of the coordination model that structure their interactions. Notice, that over time more complex coordination models are proposed, Furthermore, this coordination model can be adapted and change at run-time. On the other hand, the system state, the environment where agents are situated, and participants may dynamically change over time. Notice that both issues have to be taken into account by agents to achieve their goals, and in the decision making process. Hence, we propose that infrastructures should incorporate some services to help participants to successfully participate in the system and to achieve their goals. In other words, this services focus on assisting coordination. Notice that in general human organisations provide services and devote some resources to assist their users. Failing to provide adequate assistance to users may lead to the failure of the organisation. We regard assistance even more important for computational organisations. Notice that assistance services are useful for both software and human agents.

This services can vary from providing agents with the necessary information about the coordination model and system state, to proposing plans to achieve a provided goal. In particular in [16] authors identified the following services: (1) providing agents with information to participate in the MAS, (2) justifying action result, (3) giving advice to agents, and (4) estimating action consequences. The basic one is the information service that provides information about both the current coordination model and system state. Currently, there are some MAS approaches that already provide some of this information. For example, in S-Moise+, the special agent in charge of the organisation (OrgManager) informs participants when they acquire new obligations, Besides, in an Electronic Institution , the special agent in charge of an interaction activity (Scene Manager), informs participants when an agent joins/leaves an interaction protocol. The second one, the justification service, gives agents an explanation about the result

of an action. For instance, it can explain why an action has not been allowed to an agent, or why he has acquired a new obligation. The advice services provides agents with a set of alternate plans (action sequences), to achieve their goals. FInally, the estimation service provides agents information about the consequences that executing an action would have. For instance, about the obligations that an agent would acquire, The service is called estimation because the action is not really performed, so its consequences if later on the agent decides to execute it may be different.

Among these services we regard the advice service as the more complex and interesting one. Notice that the infrastructure may have more information about the system state than an agent, so, it may provide better plans. Furthermore, it simplifies an agent development and reasoning process, as the agent can reason about provided plans to achieve its goals, and do not need to compute them itself. Information can be provided pro-actively by the infrastructure, as for instance, when the coordination model is adapted, or after a request by the agent. Furthermore, for some services participants can decide if they want to have the service active or not. In the case of human users the 3D immersive environment can be used to represent the information provided by each service.

4 Lightweight coordination

The essence of peer-to-peer (P2P) computing is that peers when active can be called on duty to give to the other peers knowledge declared as available and to execute services, or agents, that are also made available to the rest. This view permits community creation, facilitates the sharing of resources, and when created appropriately (i.e. by giving incentives to their usage) permits an explosion in usage. The purpose of this section is to introduce a P2P model for electronic institutions (EIs). As a result, we shall obtain a coordination infrastructure that is light and click-away from usage. Moreover, further interesting features (inherent to P2P systems) are inherited: high degree of decentralization, self-organisation, low barrier to deployment (compared to client-server systems), organic growth, resilience to faults and attacks, and abundance and diversity of resources [17]. Notice that although we focus our discussion on electronic institutions, these have been taken as a case study. Further research should aim at analysing the generality of our proposal.

4.1 A peer-to-peer model

The idea behind the architecture is already present in the concept of governor in electronic institutions. The interactions of each agent within an EI are mediated by a governor, which only accepts as valid the agent's actions that abide by the rules of the institution. This simple idea seems that can now become the base of a kernel for a P2P node in a network. The proposal is based on a P2P node that should be a downloadable component helping in: (i) *knowledge management*: sharing EI related materials; (ii) *search*: searching for components; and (iii)

distributed execution: supporting the enactment of EI executions transparently to a human/agent user.

More concretely, the two sides of the node could be based on the following design principles:

- *Repository of EI specification components.* Each node may publish EI components: ontologies, scenes, norms, etc. The node could either publicise their own components or may make available components found elsewhere in the network. These components could be written in the current XML language used in ISLANDER with some extra annotations to help on provenance or in certificates. The node should incorporate a tool to permit the combination of specifications to constitute EIs. An institution can be seen then as a distributed performative structure where some sub-performative and/or scenes are located in different nodes. This opens the possibility of versioning, or local modifications that are immediately incorporated into those institutions that combine a particular bit being modified. For instance we could imagine that many institutions contain a sub-performative structure developed and maintained by Verisign in order to check for certificates. What institutions are made visible to the community must be a decision of the node manager.
- Repository of institutional agents. Agents playing the basic roles of 'Governor', 'Boss', 'Scene manager', and 'Transition manager' should be made available in many/most/all nodes in the network. These agents must be certified by their creators as they might be provide by different manufacturers with different levels of privacy guarantees, or different efficiency implementations. Creators of EIs might possible propose or give guidelines with respect to what are the preferred institutional agents to use.
- Repository of agents. Agents that can incarnate roles in EIs are also made available and shared through the node. Given a concrete EI agent creators will make publicly available agents that can play certain roles. These agents can be copied in different nodes according to users interest. Nodes should permit users to 'activate' agents by telling the node what agents playing what roles in which EI specification are allowed.
- Searching for components. One of the basic functionalities of the node should be about searching for different types of components: all those mentioned in the previous points plus trust information, ontology mappings, and running institutions.
- Execution support for a P2P infrastructure. A node might decide to enact an EI (e.g. I want to create an auction house following certain rules to sell second hand books). In order to do so, the P2P network should give the user the functionality to: (i) search for a node that incarnates the role of boss of a given specification, (ii) pass the control to the boss, and (iii) wait for the EI to get enacted and interact with it. To have this, the infrastructure (the boss) will need to recruit other institutional agents (scene managers and the like) through the network, publicise the execution to potential external agents (in our example, buyers) and monitor the execution of it. The state of the execution will therefore be distributed in the (hierarchical) organisation of

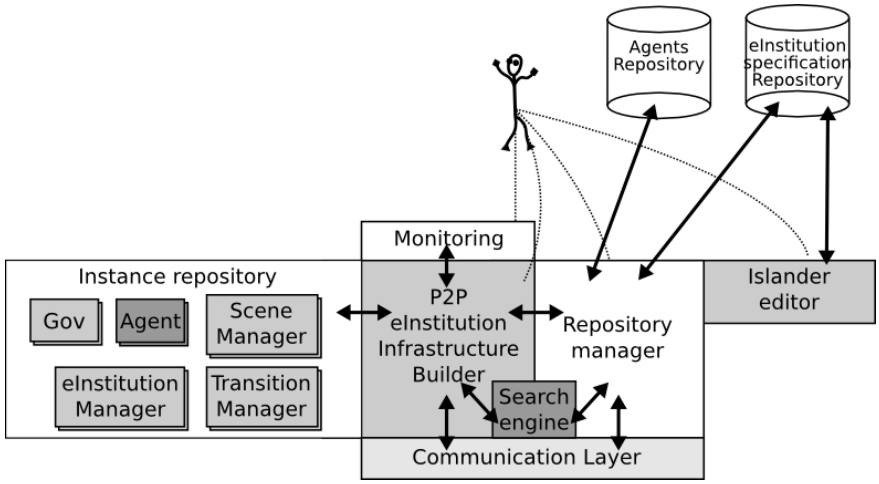


Fig. 2. Peer architecture and components

institutional agents that supports the execution. The failure of a node in the hierarchy would permit the P2P network to try and recover by appointing another agent to cover the role of the 'dead' agent. In some cases part of the state can perhaps be recovered if we keep backward paths in the organization, or if we allow for a more graph oriented organisation to increase robustness. The network should deal with the necessary routing to pass the messages among agents enacting the institution in a transparent way. Furthermore, it should be possible to put requirements for the execution of an EI (e.g. I only permit the participation of IPs that I know and trust).

4.2 A P2P architecture for electronic institutions

A P2P EI will be composed of peer nodes and a non-peer element, the Discovery Service.

P2P Electronic Institution Peer Figure 4.2 shows the architecture and components of a peer in a P2P EI, that we explain next. The communication between peers and the *Discovery Service*, will be through the *Communication Layer* component. This layer will abstract the communication process. For instance, we can think of using *JMS* (http://java.sun.com/products/jms/) through its *ActiveMQ* implementation (http://activemq.apache.org/). EI specifications are stored as XML files in the *EI specification Repository*.

Each peer also contain an *agents repository* storing implementations of Institutional and non-institutional agents. For each agent there must be a description about them: what EI is this agent coded for, the authors, a certificate, the agent version, etc. The agent implementation and this description can be bundled into a kind of *zip* file or *jar* file.

The *Repository manager* is the module in charge to publish into the *Discovery Service*: (i) the shared EI specifications available for download: (ii) the shared agent implementations available for download; (iii) the EI elements that the peer agrees to run for each EI specification: agents, governors, EI managers, scene managers, transition managers; and (iv) If the peer agrees to act as coordinator for an EI. Using the *Repository manager* the user can also search for resources published by other peers and select them to download. The needed search engine is provided by another peer component.

All instances of EI components that are currently participating at any EI are stored in the *Instance repository.* Inside the *Instance repository* we can find: agents, governors, EI managers, scene managers and transition managers. When an element at the *Instance repository* want to communicate with other elements, this communication is made through the *P2P EI Infrastructure Builder*. It knows whether the target is a local element located at the *Instance repository* and the message can be delivered locally, or is located at other peer and need to be sent through the *communication layer*.

Discovery Service (DHT) The Discovery Service provides a lookup service. It can be implemented by a DHT (Distributed hash table). This service is used by peers to inform about the resources that they share nd about the services that they can offer

4.3 A P2P Electronic Institution at work

Launching a P2P EI When a user wants to start a new EI, the XML EI specification and a set of restrictions are passed as parameters to the *P2P EI Infrastructure Builder*. This restrictions can include which peers are allowed to participate, what institutional elements are allowed to be instantiated by each peer, what agents implementations are allowed to participate, etc. Then, the user is asked if the *P2P EI* must be coordinated by his peer or if another coordinator is preferred. If another peer is preferred, a search process is launched. The *Search engine* is asked for available coordinators, and the returned list of available coordinators are asked if they agree to coordinate the P2P EI with that set of restrictions. If someone agrees, that peer will continue with the process of launching the P2P EI.

The peer playing as coordinator will publish itself at the *Discovery Service* as the P2P EI manager. Thereafter, by using the search engine, the coordinator asks to the *Discovery Service* about the available peers that agree to run each one of the needed elements. As a result, the search engine returns a list of the available peers, that is filtered by the coordinator to satisfy the restrictions. With the list of available peers, the recruiting process can be started. The recruiting process can be done in two ways: by hand (using the user interface) or automatically. Moreover, the coordinator can contain a reputation system to help to chose the more reliable peers. The recruiting consists in asking to the selected peers if they agree to participate into the P2P EI with the selected role (agent, governor, EI

manager, scene manager or transition manager) and informing about the peer that is coordinating the P2P EI. If enough peers to start the execution of the EI accept, the process continues and the EI is launched. Otherwise, the process is aborted.

The first element to be instantiated is the *EI manager*. So the coordinator sends a message to the corresponding peer to create the *EI manager* and wait for confirmation. Each time that the *EI manager* needs to create an institutional agent, scene manager or transition manager, it will ask to the coordinator to create it, and will wait for confirmation.

Joining a running P2P EI A user can decide to join with an agent a currently running P2P EI. To perform this task, he asks the *Search Engine* about the currently running P2P EIs executing the desired EI. The *Search Engine* returns a list of coordinators that manage running instances of the desired EI. The next step is to ask the coordinator if it's possible to participate. If the peer can participate, the coordinator returns the address of the peer running the *EI Manager*, so the user can request that agent to enter the EI playing the desired role. If accepted the *EI Manager* replays with the address of the *Governor* assigned to the agent. Thereafter, all the interaction with the EI will be done through that *Governor*.

5 Conclusions

Coordination infrastructures play a central role for the engineering of multiagent systems. Over years researchers have developed infrastructures providing more domain independent services and capable of enacting more complex coordination models. In the first part of the paper we have revised the state of the art in this field.

Later on, we have shift our attention to the next challenges to be faced by the next generation of infrastructures. The first one is to support human participation. In other words, infrastructures should incorporate the necessary tools and interfaces to "open" MAS to humans and therefore, be socially-aware. We regard 3D Virtual Worlds as an appropriate technology for this task. Another important issue is the development of assistance services to help participants to achieve their goals.

In the final part we have presented a lightweight infrastructure for agent coordination. This infrastructure is based on a P2P approach and then, inherits some of the advantages of these systems as a high degree of decentralization, self-organisation, low barrier to deployment (compared to client-server systems), organic growth, resilience to faults and attacks, and abundance and diversity of resources. We have exemplified this idea by presenting a P2P model for electronic institutions.

Acknowledgments
Work funded by projects EVE(TIN2009-14702-C02-01), AT (CSD2007-0022),

and the Generalitat of Catalunya grant 2009-SGR-1434. Marc Eseva enjoys a Ramon y Cajal contract from the Spanish Government.

References

1. N. R. Jennings, K. Sycara, M. Wooldridge, A roadmap of agent research and development, Autonomous Agents and Multi-agent Systems 1 (1998) 275–306.
2. Foundation for Intelligent Physical Agents, FIPA Abstract Architecture Specification, http://www.fipa.org/specs/fipa00001/ (2002).
3. J. F. Hübner, J. S. Sichman, O. Boissier, S-MOISE$^+$: A middleware for developing organised multi-agent systems, in: Coordination, Organizations, Institutions, and Norms in Multi-Agent Systems, Vol. 3913 of LNCS, Springer, 2005, pp. 64–78.
4. M. Esteva, B. Rosell, J. A. Rodriguez-Aguilar, J. L. Arcos, Ameli: An agent-based middleware for electronic institutions, International Joint Conference on Autonomous Agents and Multiagent Systems 1 (2004) 236–243.
5. F. Bellifemine, A. Poggi, G. Rimassa, Developing multi-agent systems with JADE, Intelligent Agents VII Agent Theories Architectures and Languages LNCS 1986 (2001) 42–47.
6. A. Ricci, M. Viroli, A. Omicini, CArtAgO:A framework for prototyping artifact-based environments in MAS, in: Environments for Multi-Agent Systems III, Vol. 4389 of LNCS, 2006, pp. 67–86.
7. A. Omicini, A. Ricci, M. Viroli, Artifacts in the A&A meta-model for multi-agent systems, Autonomous Agents and Multi-Agent Systems 17 (3) (2008) 432–456.
8. R. Kitio, O. Boissier, J. F. Hübner, A. Ricci, Organisational artifacts and agents for open MAS organisations, in: Proceedings of COIN at AAMAS'08, Vol. 4870, 2008, pp. 171–186.
9. J. Vazquez-Salceda, W. W. Vasconcelos, J. Padget, F. Dignum, S. Clarke, M. Palau Roig, Alive: An agent-based framework for dynamic and robust service-oriented applications, in: Proc. of 9th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2010), 2010, pp. 1637–1638.
10. J. A. R.-A. Josep Ll. Arcos, B. Rosell, Engineering Autonomic Electronic Institutions, Engineering Environment-Mediated Multi-Agent Systems (2008) 76–87.
11. C. Zhang, S. Abdallah, V. Lesser, Integrating Organizational Control into Multi-Agent Learning, in: Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems, 2009, pp. 757–764.
12. J. Campos, M. Esteva, M. Lopez-Sanchez, J. Morales, M. Salamo, Organisational adaptation of multi-agent systems in a peer-to-peer scenario, Computing 91 (2011) 169–215.
13. A. Bogdanovych, Virtual institutions, Ph.D. thesis, University of Technology, Sydney, Australia (2007).
14. I. Seidel, Engineering 3d virtual world applications design, realization and evaluation of a 3d e-tourism environment, Ph.D. thesis, Technischen Universitat Wien Fakultat fur Informatik (2010).
15. T. Trescak, M. Esteva, I. Rodriguez, Vixee an innovative communication infrastructure for virtual institutions, in: Proceedings of The 10th International Conference on Autonomous Agents and Multiagent Systems, 2011, p. to appear.
16. J. Campos, M. Lopez-Sanchez, M. Esteva, Coordination support in mas, in: 8th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 09), Budapest, Hungary, 2009, pp. 1301–1302.
17. R. Rodrigues, P. Druschel, Peer-to-peer systems, Communications of the ACM 53 (10) (2010) 72–82.

Augmenting Android with Agents for increased Reuse of Functionality in Mobile Applications

Christopher Frantz, Mariusz Nowostawski, Martin Purvis

Department of Information Science, University of Otago, New Zealand

Abstract. The increasingly adopted mobile application platform Android has introduced a new application development approach, based on loosely coupled application components. This allows for a flexible composition of applications as well as invocation from third-party applications. The rather coarse-grained application components themselves prohibit more fine-grained decomposition.
To enable this and also foster the reuse of more elementary fine-grained functionality, we suggest the extension of Android application components with the concept of micro-agents.The organisational aspects of the micro-agent model further introduces means to structure functionality in a systematic manner.
In this work we provide a brief overview of Android and its functionality principles. Then we introduce 'Micro-agents on Android' (MOA), a lightweight Android-based agent-oriented software engineering approach. We demonstrate its potential for direct integration with Android and discuss details of how MOA provides a multi-agent extension to the services offered natively by Android. We use a short example to demonstrate the functionality reuse across applications and describe further features which characterize it as a lightweight event-based middleware for Android applications as well as desktop systems. We also provide a performance evaluation to demonstrate that micro-agents interact in a more efficient manner than Android services, making them more suitable for fine-grained decomposition. We finally contrast this approach to existing work on building agent-based systems with Android.
Our approach is an example showing how existing technology can benefit from utilizing the modelling advantages of agent-based technology.

Keywords: multi-agent systems, mobile applications, agent organisation, functionality reuse, micro-agents, android

1 Introduction

With the advent of smartphones, the continuous trend towards ubiquitous computing has reached the mainstream of users. Smartphones combine the abundance of available sensors (e.g. GPS, accelerometer, gyroscope) with the Internet. The perceived 'smartness' of those devices and their applications rather derives from the combination of those different information sources than particularly intelligent features. Operating systems for smartphones cater for these application

characteristics and support notions of loose coupling as well as aspects such as intentionality. One system of this kind is Android [1], whose infrastructure has a fair degree of similarity with multi-agent systems.

Still, beyond well-specified mechanisms on how to compose applications from *application components*, Android does not provide distinct mechanisms to organise lower functionality levels. As a result, particular components combine a wide range of functionality in an application-dependent manner which limits the reusability of functionality subsets. Independent from this Android's architecture itself prohibits the use of functionality beyond the application component level across different applications.

To lever the potential for better application reuse and organisation, while respecting the processing constraints of mobile systems, we suggest the integration of the computationally cheap notion of micro-agents with Android. We first introduce the Android application development principles, followed by the description of our micro-agent concept and its implementation.

The concept of 'Micro-agents on Android' shows how those two technologies can be interlinked, and Android applications seamlessly be backed by a fine-grained cross-application micro-agent organisation.

2 Android and the concept of Micro-agents

2.1 Android architecture and application components

Android [4], developed by Google and released as an open source software platform, is increasingly adopted by smartphone manufacturers. Beyond a Linux-based kernel and the device-specific hardware drivers, it offers a comprehensive software stack of libraries and the Dalvik Virtual Machine, which operates similar to the well-known Java Virtual Machine (JVM). Related library functionality is controlled via so-called managers (e.g. LocationManager for all location-related functionality) where useful. Atop of this applications access both the various managers and library functionality using Java syntax.

The interesting aspect from an architectural point of view is the way applications are composed. Android caters for a concurrent and loosely coupled layout of applications by providing following application components:

Activities are designed for rather short-running functionality with direct user interaction. Multiple activities can be combined to provide more comprehensive functionality such as wizards.

Services in contrast are designed to be long-running in the background.

Broadcast Receivers are instantiated upon registered (system or application) events and execute a particular behaviour and are destroyed upon execution.

Content Providers serve as an abstraction layer for system-wide access to particular persistent storage locations.

All those components (with the exception of content providers) are connected via so-called intents which represent abstract request specifications, have a unified structure, and allow asynchronous messaging between the aforementioned

application components. Intents either allow explicit addressing of target components (by class name) or implicit addressing by matching intent characteristics (such as action (e.g. VIEW to open a viewer application), handled data type, or further component-related attributes (categories) (e.g. PREFERENCE indicating component is a 'Preferences' panel)) against application characteristics which are registered by individual application components (as so-called *intent filters*). Further content of intents can be arbitrarily defined by the application developer and is attached as 'extras' in a hash-map data structure.

The composition of the particular application components and their intent-based interaction are the building blocks for any Android-based application. This also includes core applications such as the caller application, thus giving the application developer the power to access a wide range of system-level functionality. Further information on those architectural principles of Android can be found under [4].

2.2 The micro-agent concept

To provide a context for the suggested augmentation of Android with agent-based technology, we introduce our understanding of the notion of micro-agents and describe their surrounding principles. We define micro-agents as goal-directed autonomously acting entities without assuming a specific internal architecture type (e.g. BDI). Their external architecture supports multiple levels of abstraction for organisational modeling – motivating a strong degree of decomposition which demands for fast message-passing mechanisms. Those aspects can, in a similar sense, be found in many agent systems. However, micro-agents represent an approach to allow a consistent 'modeling in agents' while avoiding the switch to other programming paradigms (such as object-orientation) for more fine-grained functionality as far as possible/useful.

The interpretation of micro-agents given here has not so far clarified the organisational aspects surrounding their use. The metamodel for a micro-agent organisation of our desktop platform μ^2 [3] is based on the KEA model [8] and displayed in Figure 1.

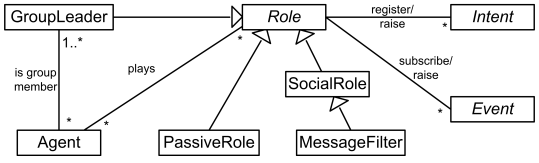


Fig. 1. Core relationships in μ^2

This metamodel recognizes agents and roles as the first-order entities. Each agent plays one or more roles which are specializations of the three first-level specializations identified in the metamodel: *Social Roles*, *Passive Roles*, *Group*

Leaders. Social Roles represent the most expressive role type, making use of asynchronous message passing and an explicit message container. Passive roles only support blocking communication, which makes them useful for very fine-grained functionality, as the interaction hardly involves any performance penalty compared to a direct method call while still retaining the advantages of loose coupling between individual agents.

The particular strength of the lightweight micro-agents is their organisation in groups. By playing the 'group leader role' agents can themselves start a group that further agents can be registered with. The group leader has two functions: It controls its group's members, or respectively dispatches control commands from its group leader, but can also compose its functionality by combining more fine-grained functionality from its group members. The latter agents can lead groups themselves in order to compose their functionality from further sub-agents. As a result of this cascading structure, a multi-level agent organisation emerges (as schematically shown in Figure 2). However, group leaders do not necessarily compose their functionality from sub-agents but can also simply organize sub-agents to structure the agent organisation by functionality aspects. The hierarchy, however, does not restrict the communication of sub-agents; sub-agents can communicate with agents outside their group, allowing the use of their functionality across the whole agent organisation. For consistency purposes agents which are not assigned to a particular group are members of the SystemOwner group to enforce a consistent control structure.

Mechanisms to allow the automated binding of functionality – the key to the composition of comprehensive applications – are *Intents*. Roles which can satisfy requested intents (e.g. sending SMS) register those as *applicable intents*. Any agent can then raise a request which is automatically delivered to the satisfying role (intent-based dynamic binding).

Intents, derived from the mental concept of intentions, are in fact abstract execution requests and are implemented as Java objects with a freely defined property/operation set. Both requester and requestee need to 'know' the semantics of the intent; for other agents this is not relevant. However, as part of the control mechanism, group leaders can restrict/prohibit the adoption of applicable intents by group members at runtime, if those intents are incompatible with the functionality managed in the according group.

The final element of the metamodel mentioned here are *Events*. Each role implementation can subscribe to particular events (such as a notification of the initialization of a new agent or a connected platform). Their implementation is realized by extending an abstract class (which enforces the specification of an event source) with arbitrary class structure – similar to the specification of intents.

The combination of those features in our micro-agent concept enables a fairly direct and clear interpretation of the key characteristics of Agent-Oriented Software Engineering (AOSE) [7]: *Decomposition* describes means of breaking up coarse-grained functionality into more fine-grained elements, *Abstraction* refers to the necessity to limit the scope of a developer at a given time in order

to limit the overall complexity for a given task. *Organisation*, finally, is the structural specification of an agent society resulting from the application of the aforementioned characteristics. The notion of levels and groups as means to specify them allows an effective decomposition while providing an arbitrarily fine-grained structure of functionality elements, both in a horizontal manner – structured by functionality groups – and vertical manner – breaking it up to an atomic level. Abstraction is realized by focusing the developer's view on a single level or multiple adjacent levels of this agent organisation at a given time. The application of those principles with this metamodel are visualized in Figure 2.

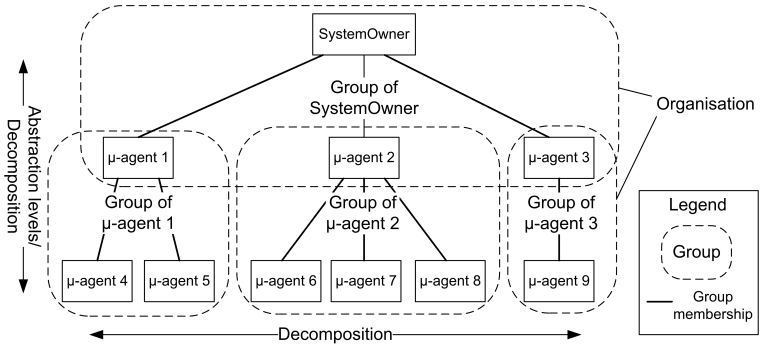


Fig. 2. Representation of AOSE characteristics Decomposition, Abstraction and Organisation in μ^2

Agent models without an organisational perspective limit the possibility to destructure functionality in an explicit vertical manner, but rather concentrate on the decomposition in functionality groups. Further, this organisational model allows the distinct application of abstraction levels by simply suppressing lower or higher levels of the agent organisation where appropriate.

Assigning the according applicable intents to micro-agents (respectively roles) in this hierarchy allows the definition of an explicit structured functionality repository that allows the flexible use by any other agent on the platform.

Reviewing the strictly agent-based modelling principles of this metamodel and the potentially strong degree of decomposition – enabled by a large number of interactions – , it should be reemphasized that high performance of the messaging mechanisms for micro-agents is imperative.

Beyond the mentioned conceptual elements the implementation of the micro-agent platform is distributed, allowing the use of intent-based dynamic binding and raising of events across connected platform instances.

2.3 Limitations of Android application components

Looking at the characteristics of both Android and the introduced micro-agent concept, loose coupling and concurrent communication are core principles of

both. *Services* in Android loosely reflect the notion of *agents*, as they are rather long-running and operate in the background. *Activities* in contrast mediate interaction between service and user and represent visible actions of a service, i.e. *agent operations*. In our micro-agent concept agent operations are not explicitly modelled. *Broadcast receivers* represent an equivalent to an *event subscription mechanism* which, similar to multi-agent systems, integrates agents with events in their surrounding. However, the similarities mentioned here reside on the infrastructural level; services exhibit no motivational autonomy but are purely reactive and additionally do not support long-running conversations in a meaningful manner.

The mentioned application components are a powerful means to structure applications by frontend and backend functionality, in the shape of different runtime containers. However, Android does not provide further mechanisms to allow a structured decomposition of functionality maintained in rather long-running services which – especially in the case of more complex applications – are holding the application's core functionality. Although one possible approach to achieve this is the use of numerous services, the performance of intent-based interaction (which is elaborated at a later point) is prohibitive for fine-grained functionality. Moreover, the decomposition into further services cannot be modelled explicitly in an organisational structure, which limits the reusability of fine-grained functionality across different applications.

To support the principal idea of composing Android applications from multiple loosely coupled entities, we suggest the integration of an organisation-centric micro-agent layer. This will allow effective modeling of agent-based applications on Android systems, provide organisational modeling facilities to legacy Android applications, and foster the reuse of functionality across different applications.

3 Micro-agents on Android

3.1 Design aspects

The similarities between Android and micro-agents suggest an integrated approach which facilitates the support of Android applications with agents to encourage reuse of functionality, offering a lightweight explicit organisational scheme, and enabling the modeling of agent-based applications. Micro-agents themselves can react to external events and access Android functionality which allows them to act in a real environment.

The integration of Micro-agents with Android, constituting 'Micro-agents on Android', is established by linking a particular micro-agent with a dedicated Android service. This makes the interaction virtually seamless for either side; agents make use of the functionality offered by the interfacing agent, while Android application components interact with the interfacing service in the same manner as with other components. Figure 3 shows this linked agent/service entity which represents the core of MOA that will be explained in the following.

In order to link interactions, the different intent concepts of Android and the micro-agent concept are dynamically converted. This approach has limitations,

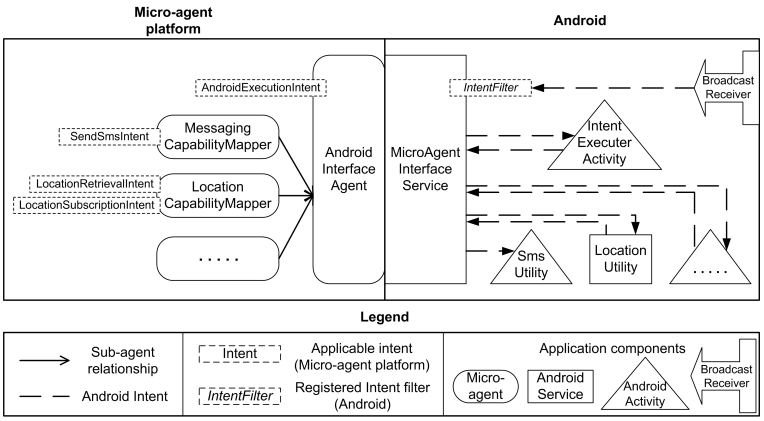


Fig. 3. Architectural schema of MOA

as not all Android capabilities can be directly accessed via intents but demand for additional code, especially when dealing with the 'managers' in the Android concept (e.g. TelephonyManager). Depending on this, Android functionality can thus either be directly invoked (e.g. requesting the user to pick one of the existing contacts) or needs to be mediated with additional functionality.

The dynamic conversion mechanism further needs to handle the particular differences between micro-agent intents and Android intents. Android intents have a fixed implementation (class structure) for dynamically typed content; micro-agent intent implementations are structurally flexible (i.e. their structure is entirely defined by the application developer) and merely need to implement the Intent interface. As a consequence, a micro-agent intent rebuilding the Android intent structure (AndroidExecutionIntent) is attached to the interfacing micro-agent (AndroidInterfaceAgent). This way micro-agents can directly invoke intents in Android. However, as Android demands for the specification of the target component type to be invoked (i.e. Activity or Service), micro-agents need to supply this information as part of the remodelled Android intent.

As Android intents additionally do not allow the specifications of a sender in the case of direct invocations, the use of a mediating IntentExecuterActivity is necessary to cache the sending agent, track the execution result of a particular intent, and return eventual responses to the original requester.

In cases where Android functionality cannot be invoked in a direct manner, the conversion mechanism is additionally augmented with *Utility application components* on the Android side and *Functionality Mapper* agents on the micro-agent side. Those then encapsulate the necessary pre-/post-processing of custom intents and manage the actual functionality. Examples include the subscription to Location proximities which cannot be directly registered via intents but are mediated by the LocationUtility service.

For the point taken here – the use of agent-based technology as an enabler to structure applications and to reuse functionality – the access of agents by appli-

cation components should be emphasized. Application components can address the interfacing service using all available Android mechanisms, thus by means of either explicit intents (using its class name) or particular intent filters, specified by the application developer. Thus the use of MOA does not have any impact on the access by Android application components.

3.2 Application development with MOA

With the use of MOA, application developers can capitalize on implementation effort and offer it – in the shape of micro-agent intents – to other applications running on the same Android device. Following the MOA concept, applications consist of a micro-agent society residing in the backend, and a frontend which is developed using legacy Android application components, such as activities which demands developers to consider both Android and micro-agent concepts but offers unique direct interaction mechanisms between both worlds. A simple example (shown in Figure 4) illustrates the potential of using MOA for application development. In this case two applications make use of the MOA func-

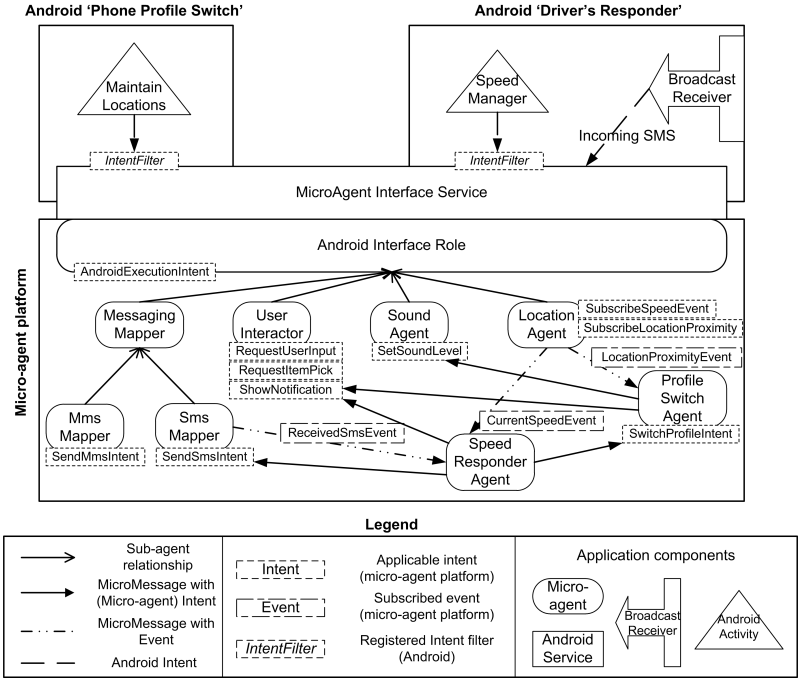


Fig. 4. MOA with multiple applications

tionality set. The 'Phone Profile Switch' application maintains locations with according distinct phone settings. The actual functionality is realized with the ProfileSwitchAgent, which receives Proximity events for maintained locations and adjusts the device settings accordingly (e.g. disables sound and shows notification on device display). Parallel to this another running application can

be the 'Driver's responder' application which automatically replies to incoming text messages if the speed of the phone exceeds a certain threshold. In this case the current speed is delivered as an event and the SpeedResponderAgent reacts on this by switching the phone profile (using the SwitchProfileIntent) and, if driving, automatically responds to incoming messages and eventually shows or suppresses a notification on the display.

Independent from the application those simplistic examples show the reusability potential enabled by micro-agents across different applications.[1] Given the risk of creating interdependencies between applications, the use of agents (in contrast to other modeling paradigms) is useful as they are supposed to handle failed binding requests and find alternatives (here this would be the case if the PhoneProfileApplication is missing). However, the same would be the case of inter-depending legacy Android applications. In order to extend applications by introducing new agents, developers only need to know the internals of relevant intents (e.g. SwitchProfileIntent) in order to use the functionality; the executing agent is automatically resolved upon sending of this intent.

The potential of MOA goes further when considering its distributed operation mode – which allows the integration of agent functionality in a location-independent manner. Micro-agent functionality can be delivered by other mobile devices or devices running the compatible desktop version. This allows extended reuse of functionality, specifically the use of functionality which cannot be provided on the local device (e.g. printing mediated via desktop pc). In turnaround this enables desktop pc's to use functionality of the mobile device (e.g. sending SMS messages).

Development can thus be realized in a consistently agent-based manner involving the provision and implementation of intent functionality as well as events without concern where it is executed. The platforms handle the ad hoc nature of network connections autonomously.

3.3 Performance evaluation

To quantify some benefits of the use of micro-agents on Android, we developed a benchmark measuring the interaction performance for both Android-based services and a version realizing this functionality with micro-agents. It simulates a simple context-aware application, automatically responding to incoming SMS text messages and is shown in Figure 5.

An incoming text message is forwarded to a responding entity (ResponseManager) which coordinates the resolution of the sender's name (via NameResolver), the identification of the priority (PriorityResolver) of the sender, and finally responds to this message (Responder). The functionality is standardized, and in each case a response message is generated to measure the pure interaction performance for both benchmark implementation variants. This scenario has been executed for increasing numbers of rounds to show the scalability of MOA. Each

[1] Please note that the figure does not show the entire agent organisation but a subset relevant for this context.

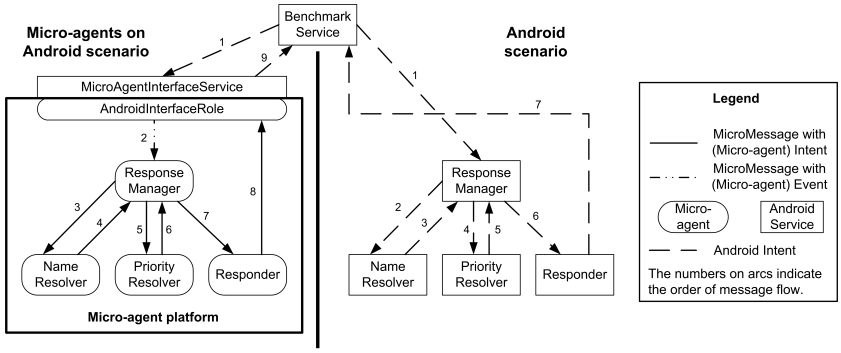


Fig. 5. Benchmark scenario for performance comparison

configuration has been executed ten times, with an initial warm-up run of 5 rounds. Table 1 shows the average durations along with standard deviation and relative performance factor of Android services in comparison to micro-agents.[2]

Rounds	MOA (ms)	σ	native Android (ms)	σ	Factor[a]
5	231	67.62	639	43.57	2.77
10	390	88.93	950	61.07	2.44
25	850	65.12	1875	30.57	2.21
100	3027	68.95	6789	106.77	2.24
250	7387	117.64	16948	735.71	2.29
1000	28404	219.85	70088	379.47	2.47
2500	77451	984.75	201685	1493.96	2.60

[a] Relative performance of Android intents to micro-agents.

Table 1. Selected benchmark results per scenario rounds

Despite the additional two Android intents necessary to realize the MOA variant of the scenario, it still significantly outperforms the purely Android-based interaction. The performance difference is surprising, but we attribute it to the fact that Android's application components are more featureful (potentially allowing IPC, providing a more comprehensive life cycle) and thus demand for a heavier implementation (and processing) than the micro-agents. Micro-agents are directly built on the provided libraries but themselves do not use any of the Android application components for their internals; their purpose is to allow efficient communication between numerous less-featureful entities.

Beyond the qualitative argument for modeling benefits from an explicit agent organisation, this gives a clear indication that a strong decomposition into micro-agents is likely to be feasible without performance loss, and perhaps even results in faster applications.

[2] The benchmark has been run on a HTC Magic smartphone running Android 2.2.1. In both scenarios all entities run in the same process, avoiding computationally expensive Inter-Process Communication (IPC).

4 Related work

This work is not the first targeting the comparatively young Android platform but takes a different approach than existing efforts to run agent-based systems on this platform.

The mobile version of the popular agent platform JADE [5], JADE-LEAP, is available in an Android version, JADE ANDROID [2]. It enables the integration of an Android-based software agent into the comprehensive and mature JADE infrastructure. For distributed use JADE ANDROID relies on a main container (provided by the full JADE version). The number of agents running in one JADE ANDROID instance is currently restricted to one.

Another approach is presented by Agüero et al. [6], who use Android as a basis to implement their Agent Platform Independent Model (APIM), which is derived from the analysis of commonalities in various AOSE methodologies. Their implementation is directly based on the full Android infrastructure (e.g. extension of Services as Agents) and puts a focus on agent internals. Organisational modeling is not of primary concern.

JaCa-Android [9] implements the Agents and Artifacts model on Android. The Agents and Artifact model suggests the use of agents and artifacts as modeling entities to describe application functionality. For agent implementations JaCa-Android relies on the Jason AgentSpeak interpreter and models workspaces via the CArtAgO framework, which allows to build distributed agent applications. Android capabilities (such as SMS message or GPS coordinates) are modelled as artifacts which expose specific attributes and operations and can be handled by agents across different workspaces.

In contrast to existing efforts, the micro-agent approach advocated in this paper offers an agent-based organisational extension to Android's infrastructure which both allows modelling in an agent-based manner while increasing the reuse of application functionality across different applications and platforms. The goal of our implementation is not only to run agents on Android but also to provide an interface for the seamless interoperation of agents with legacy application components on Android devices. Given that the specific application landscape on Android device instances can vary significantly, the potential of micro-agents to formulate Android intents in a proactive manner allows them to treat Android itself as an open system.

5 Conclusion

Android, the mobile application development platform, offers capabilities for a wide range of smart applications and an infrastructure that shows characteristics related to multi-agent systems. Its applications are composed using loosely coupled asynchronously communicating application components.

The degree of loose coupling in Android shows limitations and does not offer an organisational scheme for more fine-grained functionality patterns. We suggest the integration of efficiency-oriented micro-agents with Android services. This enables the comprehensive maintenance of developed functionality and

makes it available for reuse across different applications at an abstraction level convenient for the developer. This offers a low threshold approach to compose required functionality in a consistently agent-oriented manner across a dynamically changing device landscape.

Applications backed with micro-agents are meant to coexist with legacy applications; developers need to consider both Android and micro-agent concepts when modelling applications. Beyond this MOA's unique direct interaction with Android components using Android intents supports the operation of agents in the context of open systems.

Future research will include the extension of the current system towards a more comprehensive agent-based ad hoc middleware, integrating a wider system landscape and features (e.g. web services). Part of this work is also to address the potentially harmful bottleneck of MOA when interacting with numerous legacy application components. The development of applications using this blended approach further needs to be harmonized with existing AOSE methodologies. Micro-agents are mediators for access to low-level functionality on one side and intelligent agent notions on the other side which demands for a clear definition of this position.

Overall the unique approach to interface agent-based modeling principles with legacy technology described here is an example of how agent-oriented software engineering principles can facilitate application development in a cross-paradigmatic manner.

References

1. Android. http://www.android.com/. Accessed on: 25th January 2011.
2. JADE Android Add-on Guide. http://jade.tilab.com/doc/tutorials/JADE_ANDROID_Guide.pdf. Accessed on: 25th January 2011.
3. Micro-agent platform μ^2. http://www.micro-agents.net. Accessed on: 5th March 2011.
4. What is Android? http://developer.android.com/guide/basics/what-is-android.html. Accessed on: 25th January 2011.
5. JADE - Java Agent DEvelopment Framework. http://jade.tilab.com, October 2011. Accessed on: 25th January 2011.
6. J. Agüero, M. Rebollo, C. Carrascosa, and V. Julián. Does Android Dream with Intelligent Agents? In J. Corchado, S. Rodríguez, J. Llinas, and J. Molina, editors, *International Symposium on Distributed Computing and Artificial Intelligence 2008 (DCAI 2008)*, volume 50 of *Advances in Soft Computing*, pages 194–204. Springer Berlin / Heidelberg, 2009.
7. N. R. Jennings and M. Wooldridge. Agent-Oriented Software Engineering. *Artificial Intelligence*, 117:277–296, 2000.
8. M. Nowostawski, M. Purvis, and S. Cranefield. KEA - Multi-Level Agent Architecture. In *Proceedings of the Second International Workshop of Central and Eastern Europe on Multi-Agent Systems (CEEMAS 2001)*, pages 355–362. Department of Computer Science, University of Mining and Metallurgy, Krakow, Poland, 2001.
9. A. Santi, G. Marco, and A. Ricci. JaCa-Android: An Agent-based Platform for Building Smart Mobile Applications. In *In Proceedings of LAnguages, methodologies and Development tools for multi-agent systemS (LADS-2010)*, 2010.

AgentStore — A Pragmatic Approach to Agent Reuse

Axel Hessler, Benjamin Hirsch, Tobias Küster, Sahin Albayrak

DAI-Labor, TU Berlin, Germany

Abstract. In this paper we describe the AgentStore, a mechanism and tool to support reuse by enabling users and developers to share, search and deploy agents. Web- and API-based interactions allow the integration in the common workflow of developers of multi-agent systems. In this work, we set a high value on socialising the agent developer, not the agents.

1 Introduction

In an ideal agent world, the agent is surrounded by many other agents that provide services that can be deliberately selected and used, or considered in plans in a more complex decision and execution process. In the real world, new projects will have an agent-based system as the core of the solution, but there are no agents and no services outside the specified system, or the agents cannot access other systems or services. Inspired by the perplexing simplicity of the installation process of applications to Apple's consumer electronic devices iPod, iPhone and iPad, but also to a huge number of other devices based on the Android platform, we believe that Apple's App Store and Android's Market can be a good pattern to promote reutilisation of agents and agent-based solutions in agent-oriented software engineering. An Apple or Android device is usually delivered bare bone, with only basic applications pre-installed. The user can then go to the App Store or Market and download applications that enhance the basic capabilities with whatever is needed by the user. This can be as simple as a puzzle or as complex as a location-based social network or an augmented reality app.

Taking App Store and Market as prototype, the *AgentStore* creates a place where developers can upload their agents and other developers can find and reuse them off-the-shelf. There are certain implications to the multi-agent infrastructure, development environments and process models, and also the willingness on the part of the developer and project managers to support the pattern, because *re-use* is most often out-of-scope in a single project.

The multi-agent infrastructure requires a number of services that allow to deploy and update agents remotely and request information concerning available runtime, installed libraries and user management. Development tools and build systems must be enabled to provide agents as well as reuse existing agents. Developers and project managers must form the habit of looking up existing

agents and in turn provide the results of their work, so that they will profit from it in the medium term.

For example: At DAI-Labor we have an agent that is capable of sending e-mails to people and provides this capability to other agents as an agent service. While this agent has seen many upgrades of programming language, framework and APIs, the core implementation remained stable with marginal adjustments. Furthermore, this agent has been complemented with other agents with capabilities to post messages to different channels, such as SMS, SIP or Twitter, and extended with abilities to deliberately select the communication channel depending on which contact information is known, what the preferred channel is and what the most likely channel is to reach the addressee.

Now, having finished many projects with focus on multi-agent systems that require this functionality we discovered that we wasted time and effort just implementing this functionality time and again. If there had been a place to store the agent, a process that keeps it up-to-date and the awareness in the head of developers and project managers, this time could have been saved. In the following, we describe the concept of AgentStore in more detail, propose a number of processes around it and inspire the community to "get social". For each aspect of the AgentStore we give the abstract term and then map it to the JIAC agent framework family [3, 7],which is our preferred technology of implementation. But concepts and processes used in this paper are easily adaptable and extendable with other frameworks.

2 Concept

The AgentStore is a set of tools that allows easy access to ready-to-run agents and scripts. Similar to Apple App Store and Android Market for smart devices, functionality can be incorporated in ones own multi-agent systems. After installing and starting a new agent, it registers its services with the service directory (or directory facilitator). The services are immediately available and can be used by other agents. In addition to agents, we have scripts that represent plans or plan elements, which can be deployed to agents that are capable of interpreting these scripts. Once deployed the script is registered as a service as well.

AgentStore targets developers of multi-agent systems. Developers can take one or more of three main roles:

- *User* – select agents and scripts and deploy them to runtime
- *Developer* – develop agents and scripts and upload them to the AgentStore
- *Evaluator* – review, rank and comment on agent and scripts

In Figure 2 we give an overview about the AgentStore design. In brackets you find the technologies we have used to implement the concepts.

As its core, AgentStore is a developer portal with storage space. A web-based user interface allows to access the content and functionality provided by the AgentStore. In particular, the AgentStore offers functions to search and deploy

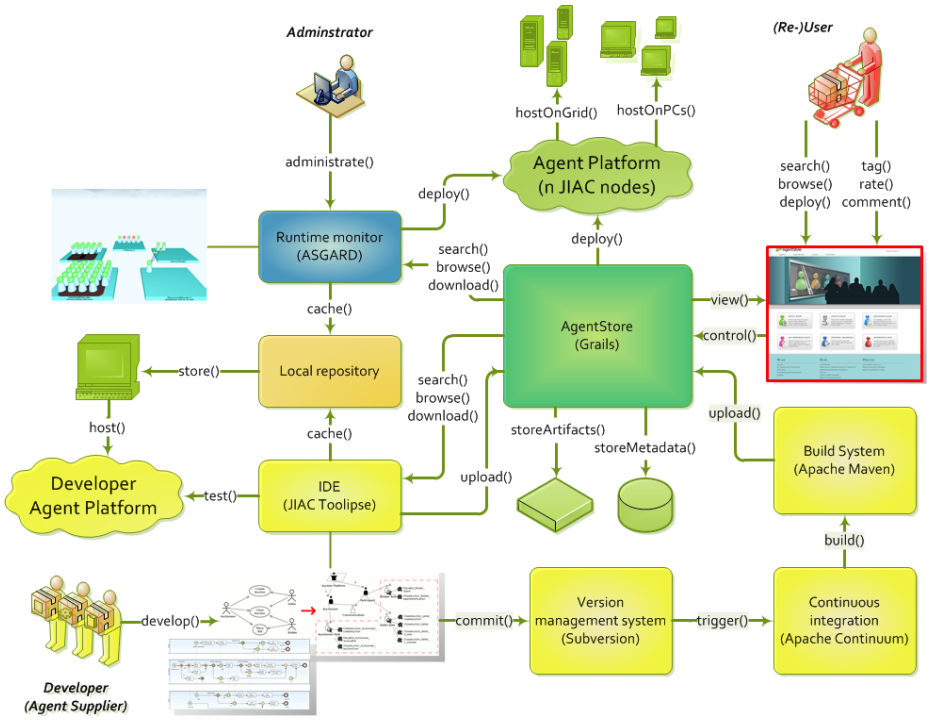


Fig. 1. AgentStore concept overview

agents to agent runtime environments, as well as to upload and modify them (see Section 3).

Additionally, AgentStore provides APIs for requesting AgentStore items via software clients such as build system, IDE or runtime management tools (see Section 4).

AgentStore can access the multi-agent environment. It can find agent nodes that are candidates for deployment of agents. In JIAC V, agent nodes are interconnected Java virtual machines that provide infrastructure services and control the agent life cycle, and together form the agent platform. The services are made available using the Java Management Extension (JMX) and allow the AgentStore to deploy agents remotely.

For deploying scripts, AgentStore is capable of finding agents that can interpret scripts. The script deployment process is partially language-dependent. The JIAC V agent description language (JADL++) [2] requires an agent capable of interpreting JADL scripts.

2.1 Metamodel

The metamodel is an instance of the project object model (POM) of the Apache Maven Project[1]. Deployable agents are unambiguously identified by *groupId*, *artifactId* and *versionNumber*. A human readable name and description is given. Especially the dependency management is used to also collect transitive dependencies. The obtained list of dependencies is matched during deployment with resources that are available on the target node. Any build system can be used here to build executables and package agents, as long as it provides needed agent parts and required metadata.

The only extensions we have made to this model are identifiers for the target agent framework and version. This allow us to filter agents based on the available platform nodes, and to use appropriate deployment tools.

Developers upload packaged agents. They also provide a semantic description of the agent's functionality and configurable parameters (filled with reasonable defaults). Agents can be labelled using pre-defined categories describing the intended application area. Additionally, agents can be tagged by users using custom key words. Icons can be provided to brand your agents and make them easily identifiable in the AgentStore.

3 Processes and Policies

Upload Developers can upload agents and scripts. The actual code is packaged (as a JAR file in our case). A description of the agent as well as a default configuration have to be provided. Then, the uploader's permissions are checked and, if authorised, the AgentStore will persistently store the given meta data and files. Interested users can subscribe to new uploads and will be informed using their preferred notification channel.

Update Once an agent or script is uploaded, the author can make updates. Each update gets its own version number, and backward compatibility needs to be declared. This is important to ensure that updates do not break systems that rely on specific functions or service calls. Users that have already used earlier versions of this agent are informed about the change and can decide whether to update their agent instance or not.

Search AgentStore provides a number of means to search for agents and scripts. Besides browsing AgentStore entries, agents and scripts can be found by categories or user generated tags.

Deployment Once an agent is selected, the user can adapt the (default) parameters. Then the user can choose where to install the agent from a list of available agent nodes. This list comprises of nodes known to the agent store. Additionally, the user can also provide the URL of a running node — or download a node

[1] `http://maven.apache.org`

from the agent store which then can be executed locally. If nodes provide a load balancing functionality (see [8]), the task of selecting a node to deploy the agent is then delegated to the load balancing agent.

Delete Finally, developers can delete their agents and scripts. Users are notified of the deletion.

In principle, scripts and agents are treated the same. However, if the user downloads a script, an agent with the ability to interpret the script is needed on the selected node. If no such agent is found (or if the user so desires) an agent capable of running the script will be deployed at the same time.

3.1 Evaluator

The evaluator role has been introduced to allow the evaluation of agent and scripts.

Comment We provide room for user reviews of every item in the AgentStore. Users are encouraged to give feedback on the according agent as well as to make feature requests. In a future release of AgentStore it is planned to synchronise comments and issue tracking tools in order to enable handling of user feedback in project management.

Ranking User may rate the agent according to its usefulness in a 5-star ranking. We show a Top-25 list of highest ranked agents at the portal page.

Statistics We also provide usage statistics for users and developers. We count page views and deployments and provide usage and update statistics to support the decision process when looking for useful agents.

3.2 Social networking

When agent developers talk about "social", they think about social capabilities and behaviour of their agents in a multi-agent system, such as speechacts, protocols, joint goals and intentions, or coalition formation strategies. Our aim is to socialise the developer itself by linking AgentStore to social networks.

The AgentStore provides a number of extended functions that deal with the social aspect of developing multi-agent systems. This includes the ability to tag agents, define profiles for automatic search and recommendation, analysis of user behaviour, as well as the ability for users to rank agents and give feedback to the developer. Developers can suggest other agents that are related to the uploaded one or should also be promoted by this developer.

RSS/Atom feeds AgentStore generates a public news feed from events in the store: agents that have been uploaded or updated, latest statistics such as top downloads, and agent ranking.

Promotion Developers can feature their agents in a number of ways. We offer post-to-Twitter and Facebook functionality for both users and developers. For developers this feature can be used to promote the latest agent uploads and updates. For users, this is a fine feature to suggest useful or cool agents to other users. This functionality has potential that has not been explored in depth yet. Twitter posts are automatically inserted into the feeds of people or topics using the "@" and "#" operators to reach certain developers or users. URLs that direct to the AgentStore items are added to the post using an URL shortener, providing additional information about usage statistics in the background. Of course, there is a Twitter agent in the AgentStore, too.

4 Tool Connections

The AgentStore provides an application programming interface (API) for uploading, searching and downloading agents and scripts. In the following, we will introduce a number of development tools for the JIAC multi-agent framework that already make use of this API.

4.1 Agent World Editor (AWE)

The AWE [6] allows to design multi-agent systems (MAS) visually using the concepts of agents, roles and components. The structure of a MAS is designed by drawing agent roles consisting of components (agents beans and scripts), aggregating them to agents and instantiating them on agent nodes. The AWE is capable of looking up agents and scripts in the AgentStore and adding them in the current design scenario. Newly designed agents can be preconfigured and described and then uploaded in order to provide them for reuse.

4.2 Visual Service Design Tool (VSDT)

The VSDT [4] allows analysis and design of workflows using the Business Process Modeling Notation (BPMN). It consists of a full featured BPMN editor, a workflow simulator and a powerful transformation framework that can produce programs from the workflows in a number of programming languages, including JIAC [5]. The VSDT is able to browse agent scripts in the AgentStore and offer them for use as service calls in a workflow. New workflows can be transformed to agent scripts and uploaded to the AgentStore for later reuse.

4.3 JADLEditor

The JADLEditor [1] is a tool for creating and revising agent scripts written in the JADL [2] agent programming language. Besides usual editor functionality such as syntax highlighting and code completion it allows using the AgentStore by browsing agent scripts for service calls and uploading newly written JADL scripts.

4.4 Agent Monitor (ASGARD)

The ASGARD monitor [9] allows monitoring and control of distributed MAS. Agents can be introspected and their lifecycle state can be changed. One can also monitor inter-agent communication and interaction. ASGARD can browse the AgentStore entries. An authorised person can select agents from the AgentStore and deploy them on agent nodes using drag-and-drop.

4.5 Maven build system support

Apache Maven is a powerful software management and comprehension tool. It is based on the concept of what a project is consisting of (the project object model – POM), and manages the whole software project life cycle. It can publish project information and artifacts in many different ways, and allows sharing project artifacts across many projects. The project model can be extended easily by providing plug-ins that allow to define a custom set of project information and to integrate additional tasks in the process model.

We have built a plug-in that extends the POM with the notion of AgentStore in order to allow uploading and updating agents and scripts during automated builds. The upload/update by Maven is triggered when changes are made in the source codes of agents and scripts and committed/pushed to the version control system.

4.6 Local repository

When working in projects with government and industry partners, we are often faced with the problem that we are not allowed online access to software and services. Also, internet access is not always available, especially during travel. Therefore, all tools share a common local repository where agents and scripts are stored for design and deployment. The local repository follows the same conventions as the AgentStore does, which is an application of the default Maven repository layout.

5 Example

AgentStore is being developed as a senior thesis by Fabian Linges[2] together with members of the DAI-Labor of TU Berlin. All main aspects are up and running and the following example is already possible:

Developer A programs a Twitter agent that is capable of posting status updates to his own account on the known service, just for fun. He uploads the agent to the AgentStore. Developer B works in a traffic telematics project and creates a GPS agent that wraps a GPS tracking device and provides agent services for reading position, speed and distance data (see Figure 5). He uploads

[2] Member of Team Brainbug in the Multi-Agent Programming Contest 2010 (see `http://www.multiagentcontest.org/2010`)

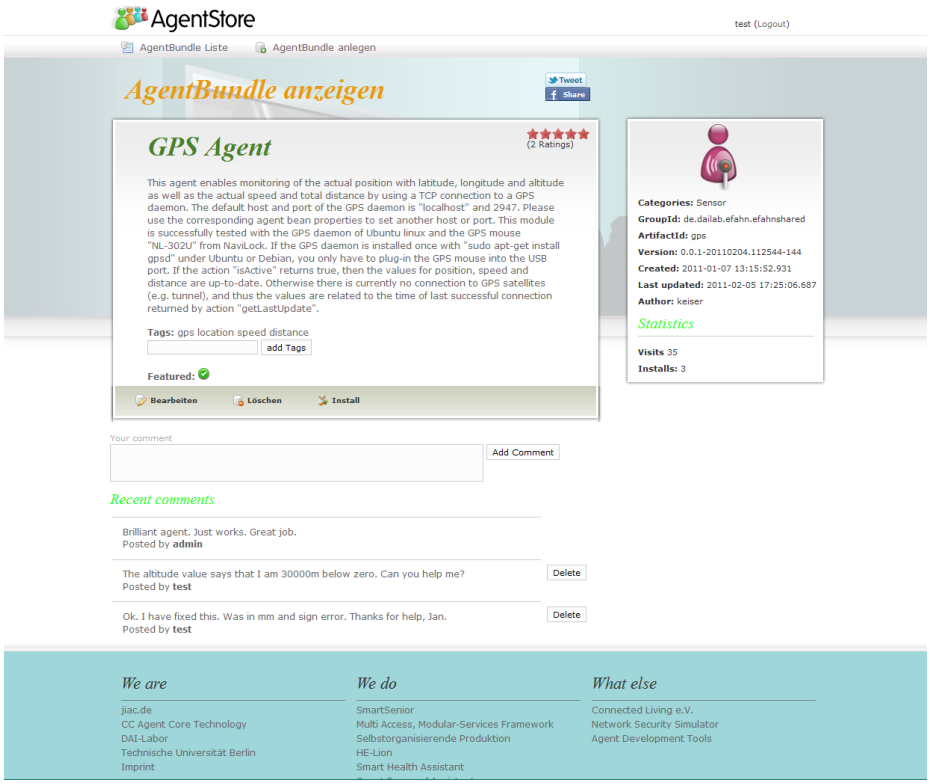


Fig. 2. AgentStore item view - GPS agent

the agent to the AgentStore. Developer C is browsing AgentStore entries and has the brilliant idea to twitter his actual position. No sooner said than done, he first deploys the Twitter agent, configured with his own account credentials. He borrows the GPS tracking device and deploys the GPS agent on his laptop. Then he writes a script that routes the position information to the twittering agent (without coping the details of the GPS or Twitter APIs) and deploys the script in an interpreter agent. And is done. His Twitter feed is now posting the position of his laptop. He uploads the script to the agent store and comments on his success story. Developer D reads the story in her RSS feed and can repeat the success in her own project.

Developer E also wants to use the solution. But he has problems with the GPS agent. He informs B that there is something wrong. B fixes the bug and updates the GPS agent in the AgentStore. All users are informed about the update. Now C and D can decide whether to update their own running agent or not. Now developer A deletes his Twitter agent in the AgentStore because too many tweets are spamming his inbox with acknowledgements. This does not

bother C, D and E because they have copy of the Twitter agent in their local repository. Developer F read about the deletion of the Twitter agent in his RSS reader and compensates the AgentStore entry with his social media agent.

6 Related Work

The AgentStore combines the concepts of an online store with agent oriented programming, and draws inspiration from both of those areas.

While today, most people think of Apple and Apple devices when confronted with a "something" store, the concept has been around much longer than iPhones and friends. However, the App Store and similar Android Market are the most famous instances.

The *Apple App Store*[3] is a program that runs on mobile devices such as the iPhone and iPad and provides access to the iTunes Store, which offers music, videos, and software. The App store focuses on software, and presents the available programs ordered by categories. User can search, browse, buy, and install and give feedback. The user interface is simple and intuitive. The *Android Market*[4] is an analogous program for Android devices, connecting to an application store provided by Google. As opposed to Apple, Android devices can make use of different stores apart from the Android Market. As a final example of mobile stores we want to mention the *Opera Mobile Store*[5] that provides a platform independent web based access to programs for mobile devices running Android, Symbian, and Blackberry devices. Common to these stores is the simplicity with which users can search for and install new functionality for their mobile devices. Often, programs are tightly integrated with the operating system, thereby allowing for genuine extensions of the system.

Somewhat related are web based stores for Perl and LaTeX, called *CPan*[6] and *CTan*[7] respectively. Like the AgentStore, they offer functional extensions that can be browsed by category or searched. However, the software needs to be downloaded and installed manually. CPAN also refers to a command line tool that automatically installs not only the chosen module but also resolves any dependencies, downloading required additional packages without user intervention. *Linux Package Managers* are also quite similar to the Agentstore in that they generally offer some streamlined interface to searching for programs, and support one-click download and installation.

Agentcities [11] was a large deployment of at its height over 100 FIPA compliant platforms, where agents could provide and look for services. Several national and EU-funded projects pushed the deployment of these platforms. Unfortunately, despite its large developer base and industrial backing, agentcities is

[3] `http://store.apple.com/`
[4] `http://market.android.com/`
[5] `http://mobilestore.opera.com`
[6] `http://www.cpan.org/`
[7] `http://www.ctan.org/`

defunct now. However, its core idea is related to the AgentStore, with the distinction that the AgentStore does not provide services but allows for the quick and easy installation of agents on ones own platform. Were agentcities alive it could serve as a target for agent deployments from the AgentStore.

7 Conclusion

In this paper we have described the AgentStore, a mechanism to support reuse by enabling users and developers to quickly and easily share, search and deploy agents and agent scripts. The main focus lies on the usability of the system. Web- as well as API-based interactions allow the integration in the common workflow of developers, thereby fostering reuse without requiring large changes to the normal flow of work. The store is set up such that also other artifacts, such as ontologies, can be stored and retrieved easily [10].

Rather than forcing developers that want to provide functionality via agents to run them on their own hardware, as cloud-based services or agentcities require, the agentstore merely stores the necessary artifacts, and allows users to configure the agent and deploy it on any platform he chooses, cloud-based or local. Thus, the AgentStore is a middle way between static programmer's libraries, requiring lots of manual work to download and reuse components, and directories of running services, which provide easy reuse of services but require permanent availability and potentially vast computing capabilities.

Apple's App Store and Android's Market also incorporate a simple but powerful business model. It is conceivable that this model can be applied to the AgentStore. We want to extend the AgentStore with self-healing mechanisms of our runtime environment, by enabling agent nodes to download agents with required functionality from the AgentStore in order to support additional redundant strategies and to provide substitute services.

References

1. Burkhardt, M., Lützenberger, M., Masuch, N.: Towards Toolipse 2. Tool Support for the Next Generation Agent Framework. Computing and Information Systems Journal 13(3), 21–28 (October 2009), http://cis.paisley.ac.uk/research/journal/vol13.htm
2. Hirsch, B., Konnerth, T., Burkhardt, M., Albayrak, S.: Programming service oriented agents. In: Calisti, M., Dignum, F.P., Kowalczyk, R., Leymann, F., Unland, R. (eds.) Service-Oriented Architecture and (Multi-)Agent Systems Technology. No. 10021 in Dagstuhl Seminar Proceedings, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, Dagstuhl, Germany (2010), http://drops.dagstuhl.de/opus/volltexte/2010/2815
3. Hirsch, B., Konnerth, T., Heßler, A.: Merging Agents and Services — the JIAC Agent Platform. In: Bordini, R.H., Dastani, M., Dix, J., El Fallah Seghrouchni, A. (eds.) Multi-Agent Programming: Languages, Tools and Applications, pp. 159–185. Springer (2009)

4. Küster, T., Heßler, A.: Towards transformations from BPMN to heterogeneous systems. In: Mecella, M., Yang, J. (eds.) BPM2008 Workshop Proceedings (2008)
5. Küster, T., Lützenberger, M., Heßler, A., Hirsch, B.: Integrating process modelling into multi-agent system engineering. In: Huhns, M., Kowalczyk, R., Maamar, Z., Unland, R., Vo, B. (eds.) Proceedings of the 5th Workshop of Service-Oriented Computing: Agents, Semantics, and Engineering (SOCASE) 2010 (2010), to appear
6. Lützenberger, M., Küster, T., Heßler, A., Hirsch, B.: Unifying JIAC agent development with AWE. In: Proceedings of the Seventh German Conference on Multiagent System Technologies, Hamburg, Germany. Springer (2009)
7. Patzlaff, M., Tuguldur, E.O.: MicroJIAC 2.0 - The Agent Framework for Constrained Devices and Beyond. Tech. Rep. TUB-DAI 07/09-01, DAI-Labor, Technische Universität Berlin (Jul 2009), `http://www.dai-labor.de/fileadmin/files/publications/microjiac\_20\_2009\_07\_02.pdf`
8. Thiele, A., Konnerth, T., Kaiser, S., Keiser, J., Hirsch, B.: Applying JIAC V to real world problems — the MAMS case. In: Proceedings of the German conference on Multi-Agent System Technologies. pp. 268 – 277. Springer (2009)
9. Tonn, J., Kaiser, S.: ASGARD - a graphical monitoring tool for distributed agent infrastructures. In: Proceedings of 8th International Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS 2010). Salamanca, Spain (2010)
10. Tudorache, T., Noy, N.F., Nyulas, C., Musen, M.A.: Use cases for the interoperation between an ontology repository and an ontology editor. In: Proceedings of the Workshop on Semantic Repositories for the Web (2010)
11. Wilmott, S., Dale, J., Burg, B., Charlton, P., O'Brien, P.: Agentcities: A Worldwide Open Agent Network. AgentLink News Issue 8 (November 2001)