How to Get Multi-Agent Systems Accepted in Industry?

Danny Weyns*, Alexander Helleboogh, Tom Holvoet

DistriNet Labs, Department of Computer Science,
Katholieke Universiteit Leuven, Belgium
E-mail: danny.weyns@cs.kuleuven.be
*Corresponding author

Abstract:

With many researchers in the multi-agent system community, we share the opinion that too much of the quality and relevant research in the area of multi-agent systems is under-represented in the development of complex distributed systems in practice today. In our experience, a Babylonic mismatch is a crucial factor in this fact – research on multi-agent systems profiles itself as an isolated community and as such may create artificial thresholds to convince mainstream software developers of its merits.

We argue that integrating concepts and techniques from agent-based software engineering within mainstream software engineering provides opportunities to amplify industrial adoption of multi-agent systems. To ground this position, we discuss multi-agent system engineering from the perspective of the software engineering area we are most familiar with: software architecture.

Keywords: multi-agent systems, MAS, software engineering, software architecture

Reference to this paper should be made as follows: D. Weyns, A. Helleboogh, and T. Holvoet (2008) 'How to Get Multi-Agent Systems Accepted in Industry?', Special section on Future of software engineering and multi-agent systems, International Journal of Agent-Oriented Software Engineering (IJAOSE).

Biographical Notes: Danny Weyns is a post-doctoral researcher at the Katholieke Universiteit of Leuven. He obtained a PhD in computer science in 2006 for research on the connection between multi-agent systems and software architecture. Danny's main research interests are in software architecture and middleware for self-managing systems and decentralized systems.

Alexander Helleboogh is a post-doctoral researcher at the Katholieke Universiteit of Leuven. He obtained a PhD in computer science in 2007 for research on simulating distributed control applications in dynamic environments. Alexander's main research interest is in software architecture, in particular documenting software architecture and software product lines.

Tom Holvoet is professor at the Computer Science Department of the Katholieke Universiteit of Leuven were he leads a research group on multi-agent systems and software architecture. Tom obtained a PhD in computer science in 1997 for work in open concurrent software development. Since then he is active in research on multi-agent systems, software architecture, middleware, and large-scale simulation.

1 INTRODUCTION

In the early 1990s, multi-agent systems (MAS) were put forward as a promising paradigm for modeling and engineering complex distributed systems. Agents provide engineers with a higher level of abstraction enabling a natural step in software engineering next to the object-oriented paradigm. Over the years, MAS research has developed a wide body of knowledge on foundations and engineering principles for designing and developing complex distributed systems. Despite the enormous research efforts and a number of successful industrial applications, the state-of-the-art in MAS research and engineering is insufficiently reflected in state-of-the-practice in complex distributed systems. The fact that the potential of MAS has not been fully utilized yet in industrial practice is confirmed by several MAS researchers; for a recent example, see [13].

From our experience, a Babylonic mismatch is a crucial factor in this fact – research in MAS profiles itself as an isolated community and as such may create artificial thresholds to convince mainstream software developers of its merits. A poignant example of the isolation of the MAS research community is the lack of any reference to results of MAS research in the collection of invited papers of the future of software engineering track presented at the International Conference on Software Engineering 2007 [4].

In this paper, we reflect on how to enhance industrial penetration of MAS. We argue that grounding agent-oriented software engineering in mainstream software engineering provides opportunities to amplify industrial adoption of MAS. To underpin this position, we discuss MAS engineering from the perspective of the software engineering area we are most familiar with: software architecture.

The remainder of the paper is structured as follows. In the next section, we start with a brief introduction of the role of software architecture in mainstream software engineering. Section 3 looks at the position of software architecture in MAS. In section 4, we illustrate how integration of MAS engineering with mainstream software architecture provides opportunities to amplify industrial acceptance of MAS. Finally, we draw conclusions in section 5.

2 Software Architecture in Mainstream Software Engineering

During the last decade, software architecture has been the subject of increasing interest in software engineering research and practice. Software architecture is centered on the idea of managing complexity through abstraction. Software architecture is a corner stone for ensuring that systems achieve their quality and functional goals and ultimately serve an organization's or community business and/or mission goals. Bass, Clements, and Kazman define software architecture as "the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them" [2]. Software elements provide the functionality of the system, while the required quality attributes (performance, adaptability, usability, modifiability, etc.) are primarily achieved through the structures of the software architecture. Software architecture constitutes a model for how a system is structured and works. This model is

transferable to other systems with similar requirements and can promote large-scale reuse of design. Besides technical value, today, software architecture is also considered as a key vehicle for communication among stakeholders. Software architecture provides a basis for creating mutual understanding and consensus about the software system [8].

3 Software Architecture in MAS

A MAS is in essence a system that is structured as a set of autonomous agents that are able to flexibly adapt their behavior to changing operating conditions. Individual agents have only limited knowledge and control over the system as a whole. To achieve the overall system functionalities and qualities, agents interact and coordinate their behavior. MAS are known for quality attributes such as adaptability, robustness, and scalability, making MAS particularly interesting to deal with the demanding challenges of complex distributed software applications.

MAS are characterized by specific structures. A MAS can be structured as an organization of selfish agents that play different roles in the organization pursuing their own interests. Such agents typically possess cognitive capabilities, and interact via a high-level communication language and specific interaction protocols. Another approach, inspired by biological systems, is a MAS that is structured as a large number of very simple agents that interact via manipulating marks in the environment in order to achieve a common goal [14]. Since specific MAS structures imbue a software system with certain qualities, while making certain trade-offs, a primary focus of MAS engineering is about the way a system is structured.

MAS research has developed a huge body of knowledge on architecture. Various agent architectures have been developed that describe the internal structures of individual agents. Large bodies of work exists on interaction protocols, roles and organizations, team and coalition formation, etc. Numerous dedicated methodologies have been developed that provide support for engineers to apply this knowledge in practice.

A number of researchers have defined specific architectural patterns for MAS. We give some examples. Castro et al. [6] propose a set of architectural patterns which adopt concepts from organization management theory; examples are joint venture, hierarchical contracting, and bidding. The PROSA reference architecture [5] is built around three types of basic agents: resource agent, product agent, and order agent. PROSA targets coordination and control application, with manufacturing systems as the main domain. Garcia et al. [9] observe that several agent concerns such as autonomy, learning, and mobility crosscut each other and the basic functionality of the agent. The authors state that existing approaches that apply well-known patterns to structure agent architectures, such as the layered architecture proposed by Kendall [11], fail to cleanly separate the various concerns. This results in architectures that are difficult to understand, reuse, and maintain. To cope with the problem of crosscutting concerns, the authors propose an aspect-oriented approach to structure agent architectures.

4 How Can Software Architecture Amplify Industrial Acceptance of MAS?

In spite of the extensive body of knowledge on architecture in MAS, the potential of MAS has not yet been fully utilized in practice. We illustrate how the integration of MAS

engineering in mainstream software architecture provides opportunities to amplify industrial adoption of MAS. Successively, we zoom in on requirements, architectural design, documentation of software architecture, architecture evaluation, integration of the software with its environment, and impact of MAS on a company's organization.

4.1 Dealing with Quality Attributes

Quality requirements are the main drivers to structure a software system. MAS are known for addressing quality attributes such as adaptability, robustness, openness and scalability. A primary concern in the decision to apply a MAS architecture should thus be based on a good understanding of (1) the main quality attributes required by the stakeholders and (2) the quality attributes that can be realized by a MAS architecture. However, for complex systems, stakeholders have various often conflicting requirements. For example, performance is a major requirement for customers, configurability is important for deployment engineers, while reuse is a prime concern of the project leader. Therefore, it is crucial to clarify the main system requirements (and quality attributes in particular) before starting architectural design. A Quality Attribute Workshop (QAW [1]) is one established method for identifying and prioritizing important quality attributes in terms of concrete scenarios. Here is an example of a quality attribute scenario from our own practice [17]:

> *An Automatic Guided Vehicle (AGV) gets broken and blocks a path under normal system operation. Other AGVs have to record this, choose an alternative route—if available—and continue their work.*

Quality attribute scenarios provide a means to transform vaguely formulated qualities such as "the system shall exhibit acceptable adaptability" into concrete expressions that are measurable. Quality attribute scenarios are ordered and the highest ranked scenarios are the main drivers for architectural design. A QAW enables (1) to precisely specify the quality attributes scenarios that should be satisfied, and (2) to determine their importance relative to other quality attribute scenarios.

MAS engineering can benefit from dealing with quality attributes in a disciplined way. Opportunities to improve MAS engineering include (1) rigourously specifying quality attribute scenarios from real-world stakeholders (2) delineating a convincing motivation for applying a MAS architecture by pinpointing real-world quality attributes and quality attribute scenarios, (3) identifying conflicts between quality attributes that are typically associated with MAS and other quality attributes. Clarifying the added value of adopting a multi-agent system on the one hand and determining the tradeoffs implied by the approach on the other hand will allow architects to make well-considered decisions, and prevent industrial partners from overestimating or underestimating agent technology.

4.2 Architectural Design

Architectural design is about moving from system requirements to architectural decisions. Such decisions are based on proven practices, typically by means of architectural patterns. An architectural pattern (or architectural style) is a description of architectural elements and relation types together with a set of constraints on how they may be used [2]. An architectural pattern exhibits known quality attributes. For example, layers is a common architectural pattern that represents a known solution to achieve modifiability and portability, but it may affect performance. Architectural design requires a systematic approach

to develop a software architecture that meets its requirements. Attribute Driven Design [2] (ADD) is a decomposition method that is based on understanding how to achieve quality attributes through proven architectural approaches.

Research on architectural patterns and styles for MAS is crucial to capture MAS expertise. Architectural patterns provide the means to document and mature knowledge and practices with MAS in a form that has proven its value in mainstream software engineering. Documenting patterns for MAS and pinpointing the quality attributes they embody will promote MAS expertise. It will allow software architects to make a well-considered choice and use MAS patterns when the system's desired qualities match quality attributes provided by MAS patterns.

4.3 Architectural Documentation

To be effective, a software architecture must be well-organized and unambiguously communicated to the varied group of stakeholders. It is generally acknowledged that a software architecture should be described by several views that emphasize different aspects of a software architecture [10]. Kruchten introduced four main views of software architecture [12] that emphasize specific architectural aspects that are useful to different stakeholders. Views include the logical view which describes the services the system should offer to the end users; the process view which captures the concurrency and synchronization aspects of the design; the physical view which describes the mapping of the software onto the hardware and reflects its distribution aspects; and the development view which describes the organization of the software and associates the software modules to development teams. A final additional view shows how the elements of the four views work together. Established approaches for documenting software architecture by means of views are described in [15] and [7]. Each view typically includes a primary presentation, an element catalog, a context diagram, a variability guide and a description of the design rationale that explains how important quality attributes are realized.

Architectural views provide a proven vocabulary to document the structures of a complex software system. MAS are complex software systems. Documenting typical MAS concerns such as interaction protocols, roles and organizations, etc. require dedicated notations, probably dedicated views. Integrating the documentation of MAS concerns in the vocabulary of architectural views will improve the accessibility of MAS architectural documentation and its use in practice.

4.4 Architectural Evaluation

Architectural evaluation is examining a software architecture to determine whether it satisfies system requirements, in particular the quality attributes. An established method for architecture evaluation is the Architecture Tradeoff Analysis Method [8] (ATAM). The general goal of ATAM is to determine the trade-offs and risks with respect to satisfying important quality attribute requirements. To evaluate a software architecture, ATAM focusses on important quality attribute scenarios identified by the stakeholders. ATAM relies on both the architect and the architectural documentation (1) to identify architectural approaches and (2) to assess the way these approaches affect the quality attributes.

The disciplined evaluation of the software architecture of a MAS is hard but invaluable to foster the industrial adoption of MAS. It allows to pinpoint the qualities and tradeoffs

implied by a MAS architecture. In our practice, we used ATAM to evaluate a MAS architecture [3]. The evaluation allowed us to determine the risks with respect to satisfying important quality attributes, such as the tradeoff between adaptability and communication load. An important challenge for the evaluation of MAS architectures is a better understanding of the tradeoffs between the driving quality attributes of MAS and other qualities.

4.5 Integrating MAS with its Software Environment

In an industrial setting, systems are not built in isolation. When introducing a MAS, it must be embedded and integrated with an existing software environment such as legacy systems, frameworks, etc. In MAS engineering, "agentification" is often considered as a general solution for integrating legacy code. However, the integration of concerns such as security, persistency, and transactional behavior often crosscut (parts of) the system. Wrapping falls short when integrating existing infrastructure that supports such concerns.

Such concerns are typically provided as reusable middleware services. An example technology is Enterprise Java Beans that enables software developers to link predefined services ("beans") without having to write much code from scratch. A number of agent-based platforms integrate particular common middleware services. Examples are Retsina [16] that includes basic services for security, performance monitoring, logging, and failure monitoring, and the more recently developed Living Systems of Whitestein Technologies [19] that is integrated with J2EE and provides support for data management with transactions, persistency, client access through Web services, etc. However, in general, integration of MAS with common middleware services remains a significant research challenge.

Since integration of MAS with its software environment is part of any real-world system, such integration is a prerequisite for industrial application of MAS. Given the importance of autonomy and encapsulation of agents' behavior, research is needed to study the integration of crosscutting concerns in MAS. Software architecture can play a key role to reason about and accommodate the integration of the MAS with its environment.

4.6 Impact of MAS on the Company's Organization

From our experience, a crucial issue with respect to industrial adoption of MAS is the impact of MAS on the developing organization. It is generally acknowledged that the software architecture and the structure of the developing organization are interrelated. As a consequence, a dramatic change in the software architecture typically requires corresponding changes in the way people are structured in teams for developing, testing and maintaining the software. Our experience indicates that moving from a traditional client-server architecture to a MAS-based decentralized architecture is a big step with far reaching effects for a company, not only for the software but in particular for the structure of the organization. One approach to manage a change to an agent-based approach in a controlled way is to gradually shift responsibilities from the central server to the autonomous subsystems [18].

Software architecture is the indispensable vehicle that provides the required level of abstraction for the integration of MAS. Studying which organization structures impede or facilitate the adoption of a MAS architecture and investigating suitable adoption strategies is a significant research challenge that is crucial for the industrial adoption of MAS.

5 Conclusions

This paper started from the observation that too much of the quality and relevant research in MAS does not find its way to current practice in complex distributed systems. We have argued that embedding agent-oriented software engineering in mainstream software engineering, in particular software architecture, provides opportunities to amplify industrial adoption of MAS.

Autonomy, adaptability, robustness, and scalability, are generally considered as key properties of complex distributed systems. These are exactly properties that characterize MAS. The body of knowledge developed by the MAS research community is therefore of crucial importance. It is our firm belief that only by integrating the knowledge in a broader setting of software engineering, the fruits of MAS research will find their way to practice.

Acknowledgement

We are grateful to the anonymous reviewers for the critical comments on earlier versions of this article. We thank Michael Winikoff for the valuable feedback. This research is partially funded by the Interuniversity Attraction Poles Programme Belgian State, Belgian Science Policy, and by the Research Fund K.U.Leuven. Danny Weyns is supported by the Foundation for Scientific Research in Flanders (FWO-Vlaanderen).

References

[1] M. Barbacci, R. Ellison, A. Lattanze, J. Stafford, C. Weinstock, and W. Wood. Quality Attribute Workshops. Technical Report CMU/SEI-2003-TR-016, Software Engineering Institute, Carnegie Mellon University, PA, USA, 2003.

[2] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. Addison Wesley Publishing Comp., 2003.

[3] N. Boucké, D. Weyns, K. Schelfthout, and T. Holvoet. Applying the ATAM to an Architecture for Decentralized Contol of a AGV Transportation System. In *2nd International Conference on Quality of Software Architecture, Lecture Notes in Computer Science vol. 4214*. Springer, 2006.

[4] L. Briand and A. Wolf. *International Conference on Software Engineering archive 2007 Future of Software Engineering*. IEEE Computer Society, 2007.

[5] H. V. Brussel, J. Wyns, P. Valckenaers, L. Bongaerts, and P. Peeters. Reference Architecture for Holonic Manufacturing Systems: PROSA. *Jounal of Manufactoring Systems*, 37(3):255–274, 1998.

[6] J. Castro, M. Kolp, and J. Mylopoulos. Towards Requirements-Driven Information Systems Engineering: The Tropos Project. *Informatica Systems*, 27(6), 2002.

[7] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord, and J. Stafford. *Documenting Software Architectures: Views and Beyond*. Addison Wesley Publishing Comp., 2002.

[8] P. Clements, R. Kazman, and M. Klein. *Evaluating Software Architectures: Methods and Case Studies*. Addison Wesley Publishing Comp., 2002.

[9] A. Garcia, U. Kulesza, and C. Lucena. Aspectizing Multi-Agent Systems: From Architecture to Implementation. In *Software Engineering for Multi-Agent Systems III, SELMAS 2004*, Lecture Notes in Computer Science, Vol. 3390. Springer, 2005.

[10] IEEE. Recommended Practice for Architectural Description of Software-Intensive Systems, IEEE Std 1471-2000. *Institute of Electrical and Electronics Engineers*, 2000.

[11] E. Kendall, M. Malkoun, and C. Jiang. Multiagent system design based on object-oriented patterns. *Journal of Object Oriented Programming*, 10(3):41–47, 1997.

[12] P. Kruchten. The 4+1 View Model of Architecture. *IEEE Software*, 12(6):42–50, 1995.

[13] A. Omicini and P. McBurney. Special Issue on Foundations, Advanced Topics and Industrial Perspectives of Multi-Agent. *Autonomous Agents and Multi-Agent Systems*, 17(3), 2008.

[14] H. Parunak. Go to the Ant: Engineering Principles from Natural Agent Systems. *Annals of Operations Research*, 75, 1997.

[15] N. Rozanski and E. Woods. *Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives*. Addison Wesley Publishing Comp., 2005.

[16] K. Sycara, M. Paolucci, M. V. Velsen, and J. Giampapa. The RETSINA MAS Infrastructure. *Autonomous Agents and Multi-Agent Systems*, 7(1-2):29–48, 2003.

[17] D. Weyns and T. Holvoet. Architectural Design of a Situated Multiagent System for Controlling Automatic Guided Vehicles. *Multi-Agent Systems and Software Architecture, Special Issue of the International Journal on Agent-Oriented Software Engineering*, 2(1), 2008.

[18] D. Weyns, T. Holvoet, K. Schelfthout, and J. Wielemans. Decentralized control of automatic guided vehicles: Applying multi-agent systems in practice. In *Development Track OOPSLA, ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*, 2008.

[19] Whitestein. Information Technology Group, Living Systems Technology Platform. http://www.whitestein.com/autonomic-technology-platform/overview. 10/2008.