Future of Software Engineering and Multiagent Systems

FOSE-MAS 2008

Danny Weyns (Edt.)

WHITESTEIN
Technologies

AAMAS 2008 FOSE Panel

Opportunities to Support Wide-spread Adoption of Software Agents

Dr. Monique Calisti, Whitestein Technologies AG

Whitestein Technologies AG | Pestalozzistrasse 24 | CH-8032 Zürich
Tel +41 44-256-5000 | Fax +41 44-256-5001 | http://www.whitestein.com

Copyright © 2008 Whitestein Technologies AG
All rights reserved.

Introduction

Whitestein Technologies is a leading innovator in the area of software agent technologies and autonomic computing. Our offering includes various agent-based products, solutions, and services for selected applications and industries, as well as a comprehensive middleware for the development and operation of autonomic, self-managing, and self-organizing systems and networks.

Several years of success with real-world deployments have shown that the advanced capabilities of software agents have matured enough to enable the design and implementation of a new generation of enterprise solutions, which are able to optimize their processes and to flexibly adapt in real-time to changing and unforeseen run-time conditions and requirements.

This, however, can only be realized by:

- Providing solid agent methodologies, platforms, tools and products for enterprise-grade development and deployment of agent technology.
- Facilitating the understanding of agent technology in close combination with specific application-driven business requirements.
- Taking into account dynamics of markets and complementary technologies.

We are happy to contribute to the FOSE-MAS session and present our perspective on its topics by answering the main questions the organizers put forward. We hope that our answers will be the ground for further discussion in Portugal.

Questionnaire

What are the main aspects that hamper progress in software engineering and MAS?

Lack of substantial efforts and focus in:

- Programming languages
- Libraries (meaning availability of deployable software)

Copyright © 2008 Whitestein Technologies AG
Strictly confidential. All rights reserved.

- Inverted precedence between techniques and methodologies: software techniques should come first (meaning earlier research and maturity level) than methodologies.

Why is state-of-the-art in MAS research and engineering insufficiently reflected in state-of-the-practice in complex distributed systems?

Essentially for two main reasons:

The "state-of-the-practice" as such does not typically need any academic/scientific blessing: it just happens. Particularly, in the modern information exchange landscape (open-source movements, community-centric development, etc.) a "normal user" will not bother too much with the labels or underlying conceptual frameworks. She will just make the best use of what is readily available to suit her needs. Given the above-mentioned bias of the agent-MAS research community away from delivering concrete and usable software incarnations of their ideas, the agent concepts end up being severely under-represented.

Secondly, the diversity of people background and agendas within what has become the "agent community", which is a strength in other instances, ends up hampering the possibility to focus and synergize efforts towards a more restricted set of goals. This, however, would typically be a pre-requisite to the effective delivery of practical, directly applicable results, particularly software deliverables.

What is the future for agent-oriented methodologies?

From a research point of view agent-oriented methodologies can keep on refining conceptual and procedural aspects concerning agent-oriented software engineering. On the other hand, we see little hope from an adoption point of view as long as not enough attention is devoted to the delivery of concrete languages, libraries and tools that can be used to build applications, with no pre-requisite knowledge of agent oriented methodologies. Only after that people will start feeling the need of methodological support to bring order in what they are already doing.

What are the strong and weak points of state-of-the-art agent programming languages?

In our opinion, there simply isn't yet enough work about agent programming languages from a software engineering point of view. This means that most of the scientific work one can find when looking for keywords such as "agent-oriented programming" does not really deal with programming language design issues and even less with the specification, realization and assessment of actual prototype programming languages.

Copyright © 2008 Whitestein Technologies AG
Strictly confidential. All rights reserved.

What makes software engineering of MAS different from mainstream software engineering?

Firstly, and luckily, the differences are increasingly being reduced. The fact that both the problem definition and the principled solution architectures of a mainstream system of today are much more similar to a MAS than it was 10 years ago is a sign that the agent community successfully foresaw the evolution of modern software engineering.

However, the full MAS conceptual framework seems to still have more than that in stock. Some of the ever-standing challenges of software engineering (reducing the gap between business users and system architects, enhancing the unforeseen reuse of software components, building homeostatic self-management into software systems) will be, in our opinion, better addressed by relying on suitable concrete applications of MAS ideas.

What are the key research challenges for software engineering and MAS?

Research challenges should not be mandated or defined too narrowly, of course. Here, we only suggest a few directions.

In MAS, one can have different kind of entities:

- Autonomous, reactive software components, typically the agents.
- Non-autonomous, reactive software components, e.g., services, artifacts.
- Passive, representational software components, e.g., objects, data.

Which category should be typed, and how? Should there be the same type system for all these three categories? How would the type system, e.g., for agents look like? Why?

What is it that we have to do to promote industrial adoption of AOSE? Can we do that, and how?

Producing more tangible deliverables (software applications, languages, libraries, tools that are concrete and usable incarnations of agent ideas) is a general pre-requisite. Contrary to popular belief, industry representatives are not hostile towards specific terms (such as "agent") when a concrete solution is provided. Of course, if the sales pitch is totally lacking concrete technological grasp, buzzwords are all that's left. But good ideas that solve relevant problems are always welcome, and names become less relevant.

Another key factor is the availability of technology transfer mechanisms, institutions and ultimately, people: the current situation can be improved. A series of University-level measures can be set up (or improved) to produce graduates that, without any specific research profile, simply have the MAS ideas and techniques as part of their professional bag. At the PhD level, doctors should be able to properly introduce theoretical MAS research concepts in their work environment if/when they apply. What

Copyright © 2008 Whitestein Technologies AG
Strictly confidential. All rights reserved.

seems to be missing the most is a (probably PhD-grade) category of people who could play the role of technology evangelist and technical leader in an innovative industrial setting. More PhD positions should be geared towards this more practical kind of consulting stance, resulting in people who can understand the theory and the research state-of-the-art, but also comprehend the actual business and technical environment they will be operating in.

What actions are required to advance research in software engineering and MAS?

We foresee three main action streams:

- Include in the research agenda the aspects, which are more software-centric (design, implementation and testing).
- Try to establish more liaisons with research communities that focus on programming languages, libraries and infrastructures (particularly for distributed systems).
- Try to better promote the agent-driven research programme within institutions and bodies funding and organizing the R&D broad context like the EU Commission in the ICT space.

Copyright © 2008 Whitestein Technologies AG
Strictly confidential. All rights reserved.

Moving Multiagent Systems from Research to Practice

By Scott A. DeLoach, Kansas State University

The state-of-the-art in multiagent research and engineering is insufficiently reflected in state-of-the-practice in complex distributed systems for the basic reason that we have yet to demonstrate, or at least publicize, the significant benefits of using true agent-oriented approaches to solve complex problems. I believe that many practitioners do not see the multiagent approach as being technically superior; for every multiagent system that achieves success, it is possible to envision a non-agent approach that is equally suited for the task. After all, almost all agent systems are programmed in the same programming languages as non-agent systems. What we have failed to demonstrate is that the agent approach can yield technically competitive (or better) solutions with a real benefit, most likely in terms of reduced costs, greater reliability, greater flexibility, or a greater chance of repeatable success. Agent-oriented software engineering lies directly at the heart of this problem. What we need to show is that we can build reliable complex, distributed systems using agent-oriented approaches that are repeatable and sound.

However, there are currently several obstacles that hamper progress towards being able to use multiagent systems and agent-oriented software engineering in mainstream applications. These include

1. the lack of a common understanding of key multiagent concepts,
2. the lack of a set of common notations and models, and
3. the lack of flexible, industrial strength methods and techniques for developing multiagent systems.

The lack of an agreement on the key multiagent concepts and their definitions is the first obstacle to be breached in the battle toward making multiagent systems a mainstream paradigm. For instance, the vast majority of computer science students and practicing professionals would be easily able to define and generally agree upon the basic definitions of the object-oriented notions of objects, classes, generalization, specialization, and aggregation. Yet, at the same time, most experienced multiagent researchers would have a difficult time trying to reach agreement on the commonly used notions of agents, roles, conversations, plans, organizations, or capabilities. The closest thing we have to agreement is on the definition of an *intelligent agent* as a computational system that senses and acts autonomously in a dynamic environment in order to realize a set of goals [6]. Although many researchers and practitioners use the names to represent similar concepts, the real problem lies in the relationships between the concepts.

A second major obstacle I see is the lack of a common notation and models for multiagent concepts. Of course, given that we have not decided on the definition of the concepts and their relationships themselves, finding a common representation may seem like an insignificant problem. However, a lack of a common notation makes it hard for practitioners to investigate different methods and techniques since they have to relearn notation for each different approach. Also, a common notation makes the similarities between approaches and models much easier to spot. In recent work with Lin Padgham and Michael Winikoff, we found that after putting our respective set of models (O-MaSE [5] and Prometheus [4]) into a common notation, the similarities between the two methodologies and the concepts we used was much more readily apparent.

The third obstacle is the lack of strong industry acceptance for any current agent-oriented methodologies. Reasons for this lack of acceptance include the variety of concepts and approaches upon which these methodologies are based along with a lack of tools to support them. However, I believe that one of the major reasons for this lack of acceptance is that the current set of methodologies tends to be inflexible and hard to extend for a variety of applications. One solution is to allow users to customize methodologies to the different types of applications being developed. There have been some suggestions for increasing industrial acceptance. For instance, Odell et al. suggest presenting new techniques as an incremental extension of known and trusted methods [3], Bernon et al. suggest the integration of existing agent-oriented methodologies into one highly defined methodology [1], and Henderson-Sellers suggests the use of method engineering [2].

Based on these observations, I think that agent-oriented software engineering researchers and multiagent systems researchers must address the obstacles address above. We must work on defining a core set of concepts that are well understood and accepted amongst all multiagent practitioners. Essentially, this boils down to defining a core metamodel for multiagent systems. While not all concepts and relationships need be represented, the core concepts and their relationships must be defined. There has been work toward defining a common [1], unfortunately, the proposed metamodels tend to be overly complex and of limited practical use. Once this first step is in place, the next two steps, creating a common notation, and creating industrial strength methods and techniques can be pursued. Finally, having a industrial strength methods and techniques in place will enable the ultimate goal of demonstrating the usefulness of multiagent approaches in the development of sound and repeatable complex, distributed systems.

[1] Bernon C., Cossentino M., Gleizes M., Turci P., and Zambonelli F.: A study of some multi-agent meta-models. In: Odell, J., Giorgini, P., and Müller, J. (eds.) Agent Oriented Software Engineering V. LNCS 3382. Springer-Verlag, Berlin Heidelberg New York (2004) 62–77.

[2] Henderson-Sellers, B., and Giorgini P. (eds.): Agent-Oriented Methodologies, Idea Group Inc., 2005.

[3] Odell J., Parunak V. D., and Bauer B.: Representing Agent Interactions Protocols in UML. In: Ciancarini, P., and Wooldridge, M. (eds.): Agent Oriented Software Engineering. LNCS. 1957. Springer-Verlag, Berlin Heidelberg New York (2001) 121–140.

[4] Padgham, L. and Winikoff, M. *Developing Intelligent Agent Systems: A Practical Guide*. John Wiley and Sons, 2004. ISBN 0-470-86120-7.

[5] Garcia-Ojeda, J.C., DeLoach, S.A., Robby, Oyenan, W.H., and Valenzuela, J. O-MaSE: A Customizable Approach to Developing Multiagent Development Processes. In Luck, M., and Padgham, L. (eds.), *Agent-Oriented Software Engineering VIII: The 8th International Workshop on Agent Oriented Software Engineering* (AOSE 2007), LNCS 4951, 1-15, Springer-Verlag: Berlin.

[6] Russell, Stuart J. & Norvig, Peter (2003), Artificial Intelligence: A Modern Approach (2nd ed.), Upper Saddle River, NJ: Prentice Hall, ISBN 0-13-790395-2

The MAS – SE Gap: Bridging the Divide

Michael Georgeff
Precedence Research and Monash University, Melbourne, Australia

What are the main aspects that hamper progress in software engineering and MAS?

This is a big question and I don't propose a complete answer, but rather restrict my comments to where I think MAS can have an impact on SE. The key to SE is having the right models and levels of abstraction for capturing data and process. We have done a reasonable job of with the data. But we have not moved very far in modeling processes so that they are easy to understand, adaptable, extensible, and capable of handling the complexity of real world applications. In particular, business value depends on the ease with which business processes can be customized to individual customers and business conditions. Different business units must be able to create and change their own services, independently of others. Moreover, today's highly connected business environment requires continuous adaptation and process change. These demands pose severe challenges for service orchestration if the promise of approaches such as SOA—business level adaptability, faster time to market, and lower total cost of ownership—are to be realized. Conventional programming methodologies and business process languages are not up to the task, and it is here that MAS can transform the industry.

Why is state-of-the-art in MAS research and engineering insufficiently reflected in state-of-the-practice in complex distributed systems?

1. We have not done a good job in identifying the value proposition for MAS in a sufficiently large range of application domains to make it compelling;

2. We have done a poor job in translating the way we describe MAS into the framework used in conventional SE and systems development environments, particularly to the people that count (CIOs, CTOs);

3. We have not sufficiently focused on the key ideas, instead selling a whole package of complex concepts and mechanisms that go beyond the needs of mainstream SE; and

4. We have not sufficiently well integrated our methodologies, frameworks and languages into existing – and more importantly emerging – development and run-time infrastructures.

What is the future for agent-oriented methodologies?

AO methodologies have a strong future in designing agent systems for research and prototyping complex distributed systems, but will have no future in mainstream SE unless we do four things:

1) Stop distinguishing agent systems from other types of system (distributed or not). That is, frame and sell the AO methodologies as general system development or SOA methodologies, not as some specialized methodology only suited for MAS.
2) Bring AO methodologies down to the practice level. The theory is important, but real SEs require real, practical methodologies, patterns and rules that can be understood and used by anyone.
3) Ensure that the terminologies and frameworks we use build on conventional methodologies, at the same time introducing new concepts in the way conventional SEs can understand.
4) Get the target level of abstraction right – AO methodologies work well at the level of goal and services composition, but carry too much baggage for low level process design.

What are the strong and weak points of state-of-the-art agent programming languages?

While this depends very much on the particular language (which cover an enormous range), the best of them have the following strengths and weaknesses:

Strengths: Expressive power and flexibility, the concept/semantics of "goals", loose coupling of goals and processes, context sensitivity of processes, integration of event driven and goal driven processing, handling of exceptions, handling of variant processes/extensions, "patterns" or interaction/coordination based on social models

Weaknesses: New languages (always hard to motivate), insufficient perceived value-add to change from conventional approaches, insufficiently powerful/robust/integrated programming development environments to mitigate the risk of adoption, insufficient integration with existing tools and frameworks.

What makes software engineering of MAS different from mainstream software engineering?

Higher, more "natural", more expressive levels of abstraction, context sensitivity of processes, concept/semantics of "goals", handling of exceptions and extensions, loose coupling of service model extended to processes at all levels. Mainstream SE does a much better job than MAS currently does in handling the important practical problems that arise in building real enterprise applications. To get the best of both worlds, we need to do much more to integrate with mainstream SE. Instead of trying to develop our own frameworks, methodologies and languages, an alternative is to introduce the key concepts into mainstream frameworks (e.g., IBM's "Business Services Fabric" uses some of the ideas behind context sensitive goal-directed processing, though not expressed in MAS terms).

What are the key research challenges for software engineering and MAS?

- Bring the key ideas (concepts, methodologies) from software engineering, SOA, and MAS together;

- Plus the usual problems: service composition and orchestration, identifying and handling goal/process interactions (both positive and negative), semantics of goal directed processing, communication (speech) acts and patterns.

What is it that we have to do to promote industrial adoption of AOSE? Can we do that, and how?

Influence the influencers. Get the story out to the businesses, the CIOs of large companies. Who influence them? The Gartner's, Forresters, mainstream CIO publications, and large enterprises that are early adopter of the technology.

Transform the story into something that conventional software engineers understand. Make it an extension or progression of service oriented and event driven architectures. Identify the weak points in SOA and show how MAS solves these problems. (My attempt at this can be found in the June 2006 issue of DM Review: "Service Orchestration: The Next Big Challenge")

Focus on the concepts and methodologies, not new languages. Let the concepts and methodologies drive extensions to BPEL, SOA. Don't try to replace them (yet). Give up on separate standards for agent systems – seeing MAS as different from services oriented systems and needing a special set of standards should be the option of last resort.

Focus on and standardize the key ideas (e.g., goal directed orchestration), preferably by extensions to existing concepts (e.g., loosely coupled services).

If we do build our own tools, languages and frameworks, make sure that they fully integrate with existing programming and development platforms. But this can be a massive expense, and it is likely that only the very large software infrastructure companies such as IBM, MS, Oracle could do so.

Or alternatively, give up on the enterprise level and focus instead on high flexibility, user-driven service composition and orchestration, such as mash-ups and high level composition of services and components at the user end. Build the tools to allow naïve users to do this. Develop "patterns" or "packs" of goal-directed agents that can be re-used (by end users) in multiple applications. Develop specialized "packs" for specialized domains. But make sure the resulting components/services fully integrate/interoperate with the conventional environment and mainstream infrastructure

What actions are required to advance research in software engineering and MAS?

Bring together leadership in SOA and MAS, develop a strongly motivated special interest group of influential researchers and industry in both fields to understand which concepts of MAS can translate to SOA and which MAS concepts add most value to SOA.

How to Get Multi-Agent Systems Accepted in Industry?

Danny Weyns, DistriNet Labs, Katholieke Universiteit Leuven, Belgium

---

We share the sigh with many researchers in the multi-agent system (MAS) community that too much of the quality and relevant research in the area of MAS is under represented in the development of complex distributed systems in practice / in industry today. 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.

In 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 engineers of its merits. A poignant example of the isolation is the lack of any reference to results from MAS research in the paper collection of the track on the future of software engineering at the International Conference on Software Engineering 2007 [1]. We argue that grounding agent-oriented software engineering in mainstream software engineering can amplify industrial adoption of MAS. Although this may sound as a self-evident claim, the question remains how this can be put into practice.

To underpin our claim, we show how the integration of our MAS expertise in mainstream *software architecture* was crucial for developing an industrial automated transportation system [2]. In this application, we applied a MAS for decentralized control of automatic guided vehicles (AGVs) that transport loads in an industrial environment. The application was developed in a joint R&D project between DistriNet Labs and Egemin, a leading manufacturer of industrial logistic systems.

Dealing with stakeholders' requirements

The general motivation to apply a MAS in the AGV control system were new and future quality requirements, in particular flexibility (deal autonomously with dynamic operating conditions) and openness (deal autonomously with AGVs entering and leaving the system). However, for a complex system such as the AGV control system the stakeholders have various, often conflicting requirements. E.g., performance is a major requirement for customers, configurability is important for deployment engineers, while budget is a prime concern of the project leader. To clarify system requirements before starting architectural design, we organized a four days Quality Attribute Workshop (QAW). A QAW is an established method to identify and prioritize important quality attributes in terms of concrete scenarios. The highest ranked quality scenarios are the main drivers for architectural design. The QAW enabled us (1) to precisely specify the qualities addressed by adopting a MAS, and (2) to determine their importance relative to other qualities. This was important for preventing the industrial partner from overestimating or underestimating agent technology.

Managing complexity

AGV control systems are very complex software systems. The design and implementation of the MAS-based AGV control system needed +8 man-years of effort. The delivered code base consists of about 100,000 lines of C# code. Such complexity can only be managed through abstraction. Software architecture is centered on the idea of reducing complexity through abstraction and separation of concerns. In the AGV control system, software architecture allowed us to manage the complexity of the MAS at different levels of abstraction (intra-agent and inter-agent structures, behavior, and hardware/software allocation).

Integrating MAS with its software environment

In an industrial setting, systems are not built in isolation. When introducing a MAS, it must be integrated with its environment (common frameworks, legacy systems, etc.). In Egemin, .NET is the standard environment and the company uses an in-house developed framework called E'pia that provides common middleware services to support inter-node communication, persistency, security, and logging. Examples of legacy systems with which the MAS needed to be integrated are the warehouse management system that generates the transport tasks and the low-level control software of the AGVs. Software architecture was the key to accommodate the integration of the MAS with its environment. We integrated E'pia as a basic layer that provides the required services to deal with various crucial requirements. With respect to legacy systems, we were able to develop proper mediator components/agents to integrate legacy systems with the MAS.

Architectural design and evaluation

Preceding experiences with developing MAS applications with characteristics and requirements similar as the AGV control system yielded a set of architectural patterns for MAS and a supporting middleware for mobile applications. Initially, we faced the problem how we could exploit these reusable assets and integrate them in the design of the AGV control system. The solution was the Attribute-Driven Design method (ADD). ADD is a well-established method for architectural design that is based on understanding how to achieve quality goals through proven architectural approaches. During the architectural design, we employed the patterns for MAS, together with a number of common architectural patterns, to decompose and structure the system and realize the required functionalities and qualities. To pinpoint the qualities and tradeoffs implied by the decentralized MAS architecture, a disciplined evaluation of the software architecture was necessary. Therefore, we organized a one day ATAM (Architectural Tradeoff Analysis Method). During the ATAM an external evaluation team, together with the main stakeholders, determined the trade-offs and risks with respect to satisfying important quality attribute scenarios, particularly scenarios related to flexibility, openness, performance, and robustness. One important outcome of the ATAM was an improved insight on the tradeoff between flexibility and communication load.

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 company's organization. At Egemin, the existing AGV control system has a centralized server-oriented architecture. The MAS-based approach on the other hand has a decentralized architecture. Switching from a centralized toward a decentralized agent-based architecture is a big step with far reaching effects for a company, not only for the software but for the whole organization. To give one example: in the centralized architecture task assignment to AVGs is based on application-specific rules that are associated with particular locations in the environment. A team of layout engineers is responsible for defining these rules. In the decentralized architecture, however, tasks are assigned by means of a dynamic protocol between AGV agents and transport agents. This protocol must be tuned per project, but this requires other skills. Our experience indicates that the integration of an agent-based approach should be done in a controlled way, step-by-step. Software architecture is the indispensable vehicle for stepwise integration of MAS. It provides the required level of abstraction to reason about, and dealing with gradual integration of MAS.

Conclusion

We have put forward the position that grounding MAS in mainstream software engineering can amplify industrial adoption of MAS. By linking MAS to software architecture, we were able to convince the industrial partner of the benefits of MAS in the AGV control system.

Self-adaptability, scalability, and local autonomy are generally considered as key properties to tackle the growing complexity of software. 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 sharing our know-how and putting it in a broader setting of mainstream software engineering, especially software architecture, the fruits of our research will develop to their full abilities.

Bibliography

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

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

The Future of Agent-Based Software Engineering: Goals and Verification & Validation are Key

Michael Winikoff*
RMIT University
winikoff@gmail.com

Marketing

The key question is this: how to "market" the results of work in our field to other research fields and to practitioners?

There is no easy answer, but a multi-pronged strategy needs to be used which includes:

1. Clearly identifying (and quantifying? [1]) the "value-add" of agent technology, and telling a simpler and easier-to-understand "story". I believe that this story should focus on the words "complexity", "adaptability", and "goals"; and not on the words "agent" or "autonomous".
2. Working against the perception of agents as "esoteric AI" by continuing the excellent work of AgentLink in documenting case studies of agent-based solutions to real problems [5].
3. Providing agent-based solutions in other areas, such as service-oriented computing [3] and autonomic computing, and publishing in those areas' conferences and journals.

Finally, I believe it's important to produce useful tools, not just papers; and that it's essential to "close the loop" by using our tools and techniques "in anger" [6], thus gaining feedback to guide further work, including detecting unrealistic assumptions.

Agent-Oriented Software Engineering (AOSE)

I agree with Scott DeLoach that reducing differences and moving towards standardisation are important (although I believe that this alone is not sufficient).

I disagree with Mike Georgeff that we still need to "*Bring AO methodologies down to the practice level*": I believe strongly that this has already been achieved by recent methodologies such as Prometheus.

A key challenge is how to design more dynamic agent systems, including those where the structure of the system changes at runtime, and those that exhibit emergent behaviour. Current methodologies are mostly still limited to the design of relatively static and predictable agent systems. Another key area for future work is the "non-classical" phases of the software life-cycle: debugging, testing and software maintenance and evolution.

*This "type 2" position statement was written while on sabbatical at Otago University, Dunedin. I would like to thank Stephen Cranefield for comments on a draft.

Key Research Issues

Goals: I believe that we will (eventually) look back and view the contribution of our work as being "goal oriented programming/design", rather than "autonomous agents". Goals, achieved persistently and flexibly, are what give agents their adaptability, and, if our agents are adaptable, then it arguably makes sense to investigate building on this to form a robust and adaptable society of agents. However, there is more research to be done: how to make the design process more goal-oriented? what goal types are useful? how to deal with interactions between goals? how to use flexible and robust goal achievement by single agents to enable multiple agents to achieve goals in a robust and flexible way? how does this compare with conventional approaches for choreography/orchestration? how does this compare with agent-based approaches (e.g. norms, protocols, social commitments, teamwork)? how to design goal-based agents in systems which exhibit emergent behaviour? how to ascribe goals to non-goal-based (and possibly emergent) behaviours?

Validation & Verification: A recurring issue in practice [5, 4] is how to obtain confidence that an agent system will behave appropriately in a range of situations. Conventional testing is less effective for flexible adaptable software, since there are many more possible behaviours to be tested. An obvious answer to this problem is the use of some form of formal methods [2]. However, much more research is needed to make this practical. One key challenge is that typically we don't want to just check well-known properties such as liveness or safety, but also various forms of domain-specific correctness. For instance, what does it mean for a manufacturing management and optimisation system [5, section 3] to be functioning correctly? A second key challenge is dealing with systems that are dynamic, including those whose structure may change. One possible approach is to use an assume-guarantee style reasoning [2], another is to defer (some) reasoning and checking to run-time.

References

[1] Steve S. Benfield, Jim Hendrickson, and Daniel Galanti. **Making a strong business case for multiagent technology**. In Peter Stone and Gerhard Weiss, editors, *Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 10–15. ACM Press, 2006.

[2] Matthew B. Dwyer, John Hatcliff, Corina Pasareanu, Robby, and Willem Visser. **Formal software analysis: Emerging trends in software model checking**. In *International Conference on Software Engineering: Future of Software Engineering*, pages 120–136, May 2007.

[3] Michael N. Huhns, Munindar P. Singh, Mark Burstein, Keith Decker, Edmund Durfee, Tim Finin, Les Gasser, Hrishikesh Goradia, Nick Jennings, Kiran Lakkaraju, Hideyaki Nakashima, H. Van Dyke Parunak, Jeffrey S. Rosenschein, Alicia Ruvinsky, Gita Sukthankar, Samarth Swarup, Katia Sycara, Milind Tambe, Tom Wagner, and Laura Zavala. **Research directions for service-oriented multiagent systems**. *IEEE Internet Computing*, 9(6):65–70, November-December 2005.

[4] M. Luck and P. McBurney **Computing as interaction: agent and agreement technologies**. In: V. Marik (Editor): Proceedings of the 2008 IEEE International Conference on Distributed Human-Machine Systems. Athens, Greece, March 2008.

[5] S. Munroe, T. Miller, R.A. Belecheanu, M. Pechoucek, P. McBurney, and M. Luck. **Crossing the agent technology chasm: Experiences and challenges in commercial applications of agents**. *Knowledge Engineering Review*, 21(4):345–392, 2006.

[6] Philip Wadler. **Functional programming: An angry half-dozen**. *SIGPLAN Notices*, 33(2):25–30, February 1998.

Industry traction for MAS technology: Would a rose by any other name smell as sweet?

(Position paper submitted to the AAMAS-2008 session on the "Future of software engineering and multi-agent systems")

Aditya Ghose, Decision Systems Lab, University of Wollongong, NSW 2522 Australia (aditya@uow.edu.au)

The multi-agent systems (MAS) community faces a crisis that many are unwilling to acknowledge. We are all acutely aware of how MAS concepts and technologies seem to have failed to gain significant industry traction (a few notable exceptions aside). Other research communities (e.g., the service-oriented computing community) have gained far greater industry acceptance for their outputs, in far shorter time. But the crisis stems from an even greater threat: the co-opting of agent/MAS concepts in other research communities, particularly the service–oriented computing community. The connections between agents and services run deep, and are to some degree recognized and addressed in the literature. Like agents, services can be viewed as autonomous, reactive components. Think of a dynamic service broker – itself a service - that re-computes service compositions on the fly (in response to changing service requirements and a dynamic operating context), in a manner akin to reactive agent programming, from a library of available services, just as one would compute composite agent plans from a plan library. Thus agent planning can manifest itself as service composition, agent negotiation as SLA (service level agreement) negotiation, and so on. I did an informal analysis of the papers presented at a recent major conference on service-oriented computing, and concluded that at least 60% of these had agent technology underpinnings in some form or the other. In other words, even as the MAS community worries about marginalization by industry, our research outcomes *are* finding useful and significant industry application, but under the banner of services.

It might be argued that this is not necessarily a bad thing. We might ask, for example, whether the nomenclature of our research (e.g., agents vs. services) should matter as long as we get to explore the really interesting and important questions. I submit that the nomenclature does indeed matter. We know from the sociology of research that different research cultures exist within different research communities. The MAS community offers a research culture that encourages the exploration of different questions, and in a different style, to those that the services community encourages. So should we worry about ending re-labelling our work as services research in our quest for industry relevance? Yes, we should. The label matters – it can influence research culture, style and content.

There are things, though, that we can learn from the success story of service-oriented computing in terms of gaining and retaining industry relevance:

- *Offer a simpler value proposition.* The services community offers a very simple value proposition: it is easier to model, design and deploy systems from distributed collections of components packaged as services (some of the recent discourse on service modelling refers to use of anthropomorphic constructs, further blurring the distinctions between our communities). The

MAS value proposition is far more complex. The notion of "agentification", and the questions explored by much of the agent-oriented software engineering (AOSE) sub-community has a similar feel, but the MAS community has a myriad other technology offerings. These offering often come with the baggage of legacy industry (mis)perceptions regarding "heavy" AI techniques that underpin a lot of MAS research.

- *Offer an incremental value proposition.* The MAS value proposition, in some ways, calls for radical changes to the state of industry practice. The service-oriented computing value proposition required far more incremental changes. Industry prefers incremental change to radical change.
- *Define a core agenda, while admitting a diversity of subsidiary themes.* The core research agenda for the services community is driven by software engineering concerns: service specification, discovery, composition and deployment. The AOSE research agenda has a similar feel, but the broader MAS research agenda is far more diffuse.

So what can we do now? Two strategies deserve our attention:

- *Mediate the deployment MAS research in industry through AOSE paradigms:* We have made comments that may appear critical of the MAS community: its complex value proposition for industry, its requirement for radical changes to the state of industry practice, and its diffuse research agenda. These should not be read as critiques. The breadth, depth and complexity of the ideas explored by the MAS community are its strengths and underpin the rich intellectual outputs that the community generates. AOSE research, by definition, addresses industry relevant concerns, and can form the packaging required to make the broader outputs of the MAS community more acceptable for industry.
- *Pro-actively engage the service-oriented computing community (both research and industry):* This reinforces a point made by Michael Georgeff in his panel submission. We need to highlight the substantial intersection between the services and MAS research agendas. We need to make explicit the agent technology antecedents of several key services concepts. We must use the connections with services concepts as yet another avenue for industry deployment of our research.

In the meantime, we must disagree with Shakespeare. He argues for the primacy of substance over form, of content over packaging when he says through the voice of Juliet: "that which we call a rose, by any other name would smell as sweet". In our quest for industry relevance, form and packaging assume unexpected importance and impose on us additional obligations in interpreting our research results for industry.

FOSE-MAS 2008 – Type 2 Position Statement

SWOT-analysis and its implications

Paul Valckenaers, Paul Verstraete and Bart Saint Germain
K.U.Leuven, Belgium

Strengths, **W**eaknesses, **O**pportunities and **T**hreats for the application of multi-agent systems relative to its mainstream alternatives shape the search space for FOSE-MAS.

The most relevant strength of mainstream software engineering and technologies is that they, ceteris paribus, are a user's first choice and enjoy critical user mass. MAS need a compelling reason, in the perception of their customers, to be selected. The most relevant weakness of multi-agent systems is that they are not a mainstream technology and lack the associated maturity and massive support. Inherent qualities of MAS are unconvincing to prospective users whenever they perceive mainstream technologies to answer their needs. Therefore, ***FOSE-MAS must target applications with decisive value to customers for which mainstream IT is unable to offer a satisfactory solution***.

The most relevant weakness of mainstream IT is generally referred to as rigidity: severe limits to its ability to adapt, self-organize, auto-configure, etc. in dynamic and sophisticated environments. Conversely, mainstream IT is able to provide solutions when:

- *The application environment may be adapted to the technical requirements/limitations of the software*. Enterprise Resource Planning systems (ERP) belong to this category: business and administrative processes are re-engineered to fit the ERP software limitations. Note that ERP consultants have a 'rule of dumb' stating that ERP is unsuited for the core business.
- *The application environment is sufficiently stable/simple/important for a customized solution.* Again, rigidity is tolerable because adaptation is not required and/or is achieved by brute force (big budgets, supervision by skilled personnel).

This leaves to FOSE-MAS a target comprising core business related applications that exhibit variability and heterogeneity and in which accounting for the application specifics is vital. Fortunately, this coincides with a generally accepted strength of MAS technology. A large application area with these properties concerns industry, transport, energy, etc. in which the application bottle-neck is the decisive element. In contrast, applications that only manipulate information often are too malleable to qualify.

An opportunity for MAS is its highly-regarded status in software technology. Conversely, a serious threat to MAS is its overly academic image associated with prolonged conceptual discussions, tolerance for combinatorial explosions and application domain ignorance.

More relevant for the research agenda are customers accepting the limitations of mainstream IT because they ignore/disbelieve the ability of MAS to deliver a competitive edge. In this respect, ***MAS-SE needs to deliver confidence-building solutions***:

- *Practical convincing MAS applications*
- *Incremental development with intermediate applicable results but also promising a decisive competitive edge in the end*
- *Speedy application development*
- *Integration with non-MAS systems*
- *Integration and harmonization with mainstream software engineering*

In the above, coping with a dynamic complex environment is crucial. Here, it is insufficient to have MAS technology that is able to handle such environments in principle only. **MAS-SE technology needs to CAPTURE DOMAIN KNOWLEDGE IN SOFTWARE ARTIFACTS – components, frameworks, architectures – to avoid that every application needs to build this from scratch and learn application-related lessons repeatedly**.

AGREEMENT TECHNOLOGIES
Towards a new programming paradigm for agent-oriented technologies

Juan A. Rodríguez-Aguilar, IIIA-CSIC, Spain

Nowadays, most current transactions and interactions at business level, but also at leisure level, are mediated by computers and computer networks. From email to virtual worlds, the way people work and enjoy their free time has changed dramatically in less than a generation time. However, the biggest impact of this pervasive use has been on the way applications are thought and developed. These applications require components to which more and more complex tasks can be delegated, components that show higher levels of intelligence, components that are capable of sophisticated ways of interacting, as they are massively distributed, sometimes embedded in all sort of appliances and sensors. This is precisely the scenario we believe agent-oriented technology can contribute to.

Therefore, there is a need for developing models, frameworks, methods and algorithms for constructing large-scale open distributed computer systems where *autonomy*, *interaction* and *mobility* are the key characteristics. We envision that such technologies can be structured around the concept of agreement among computational agents. We envisage a new programming paradigm that is based on two concepts: (1) a *normative context or agreement environment* [1], that determines the rules of the game, i.e. how the interactions between agents are going to happen, and (2) a *call-by agreement interaction method* that is based on a two step process: first the establishment of an agreement for action between the agents that respects the normative context, and second, the actual program call for the enactment of the action.

We believe that agent programming languages must go beyond the current state of the art, so far focused on the notions of agent architecture and protocol. Even beyond the ideas represented by WADE [2], which attempts at bringing the notion of process into agent development by providing support for the execution of tasks defined according to the workflow metaphor. There is a need for a new programming paradigm that considers the notions of *agreement environment* and *agreement* as first-class citizens. Thus, the new programming paradigm must allow programmers to create agreement environments, their access rules, their composition, and even the adaptation mechanisms that allow them to adapt under changing circumstances. Furthermore, the programming paradigm must allow the dynamic establishment of agreements, the verification of the fulfilment of agreements, and the management of agents that fail to honour their commitments even when agreements are signed. In some sense, we advocate for a new programming paradigm inspired on the way we humans act: we firstly set up constraining environments (via organisations or institutions) wherein social contracts among individuals or companies are dynamically established and honoured.

In order to found a new programming paradigm, we do not depart from scratch. Under the umbrella of agreement technologies we consider the techniques and tools that enable agents to reach and fulfil agreements on the mutual provision of services. For agreement technologies to succeed in building next generation open distributed systems there is a wide number of challenges at sight: the

semantic alignments between the different ontologies employed by agents; the need for negotiation to allow agents to reach agreements; the development of trust and reputation models that allow to cope with agents that fail to honour their commitments even when agreements are signed; formal models and tools for virtual organisations and institutions defining normative contexts, agreement environments, within which to reach agreements; the need for learning models to adapt agreement environments; the adaptability of agents to cope with different agreement environments that may even change over time; and a better understanding of agreement mechanisms by means of game and decision theoretic results.

In our view, agent-oriented software engineering has made (and is doing) so far significant contributions to industry. There is a wealth of agent-oriented methodologies (e.g. AUML), programming languages and agent-based platforms (e.g. JACK, Agentis, Whitestein's Living Systems), and actual-world applications (e.g. the wide range of business solutions developed by Whitestein). Nonetheless, we still believe that there is something missing. We envision that future methodologies, programming languages and tools for MAS-oriented development must grow around three fundamental notions, namely agreement environment, agreement, and agent. Thus, agents are situated in some agreement environment that constrains the dynamic enactment and fulfilment of agreements as reached by agents. Therefore, from a programmer's point of view we shall need to specify and enact environments, to specify and enact agents, and a machinery to process agreements as agents interact.

Notice that this approach is not far from the way we humans daily operate. Thus, we daily enact contracts in the framework of some regulatory body that shape our future interactions.

References

[1] Special issue on environment for multi-agent systems. Autonomous Agents and Multi-agent Systems. Volume 14, number 1. February 2007.
[2] Workflows and Agents Development Environment. http://jade.tilab.com/wade/

Position Statement for FOSE-MAS at AAMAS 2008

Towards an Agent-Oriented Paradigm

Jorge J. Gomez-Sanz*, Fabien Michel†, Eric Platon‡, Alessandro Ricci§

In this position statement, software paradigms are understood as fundamental styles of programming or engineering regarding how solutions to problems are to be formulated in terms of fundamental abstractions. The creation of a new paradigm is mainly motivated by the aim to build better software that can address increasingly complex requirements. One paradigm can be better than another for solving a concrete problem in different ways. For instance, it may be easier to understand, or cheaper to produce/maintain, or more robust.

We argue that a main step essential to advance the research in SE and MAS and, in particular, for a widespread adoption of AOSE in both industrial and academic contexts, concerns the identification and development of an *agent-oriented paradigm*, focussing on the theory and practice of using agents for software development. This paradigm is to be contrasted with mainstream paradigms, such as OO, and its advantages and limitations evidenced.

Agent research elaborates numerous facts and results, but we argue that there is no paradigm yet, well recognised and accepted by SE and computing programming communities, and this hampers progress in SE with agents. A simple evidence for this is given by the very low number of references to AOSE, agent-oriented programming and agent research in general that can be found in mainstream SE and computer programming literature, including papers, surveys, books, textbooks. Conversely, the notion of agent is everywhere in AI and related literature (Russell and Norvig's book is a main example).

When a paradigm is adopted, different developers produce similar solutions for the same problem. Despite the progress in AOSE, this is something that it not happening within our community. So, the question now is why there is no consensus in the use of agents. We think this is the result of several factors:

*Departamento de Sistemas Informáticos y Programación, Universidad Complutense de Madrid, 28040 Madrid, España. Email: jjgomez@sip.ucm.es

†CReSTIC / LERI, IUT de Reims-Châlons-CharlevilleRue des crayères BP 1035, 51 687 Reims CEDEX 2, France. Email: fabien.michel@univ-reims.fr

‡National Institute of Informatics, Honiden Laboratory, 2-1-2 Hitotsubashi, Chiyoda-ku,101-8430 Tokyo, Japan. Email: platon@nii.ac.jp

§DEIS, Alma Mater Studiorum Università di Bologna, Via Venezia 52, 47023 Cesena (FC), Italy. Email: a.ricci@unibo.it

- Too little concern with theory and practice of computer programming and SE, compared to a strong concern with AI. Agents could find a future also outside (Distributed/Classic) AI, if a focus was put on how to use agents to effectively build software systems, in particular complex software systems. There are precedents of a similar evolution in the LISP and Prolog programming languages, which led to the functional and logic paradigms.
- An overly strong emphasis on theory. Even though there is an important effort to change this situation, implementation and deployment is still considered (too) secondary to (pure) theoretical results.
- Not illustrating sufficiently the value of application of agent technology to problems; failure of disseminating evidence for improvements and cost savings obtained. Agent technology needs to be assessed and audited, and to do so, there must exist public benchmarks and code illustrating good practice. These examples should regard real-world problems.
- Weak involvement of industrial players. Differently from what happened for other paradigms (e..g., object and service), the agent community seems not having stimulated the participation and, often, the interest of industrial players in the effort of making the agent paradigm a valuable addition to the current programming and engineering practices.

Devising an agent paradigm means injecting in the land of SE and programming theory/practice a new abstraction layer based on first-class agent concepts for organizing and programming applications and systems, and *making it clearly acknowledged by mainstream SE and computer programming communities and literature*. This implies defining suitable programming languages, and methodologies based on shared and accepted computational models and theories. And these elements need to be based on the experience of the development of systems exhibiting, for instance, adaptability, robustness, scalability, and autonomy. Hence, it will be necessary to gather and organize existing knowledge on the construction of MAS and promote the dissemination of this knowledge.

Acknowledgments

Many people contributed directly or indirectly to this position statement - which actually summarizes the discussion developed in a forum on agent-oriented computing. In particular we would like to thank: Paolo Petta, Michael Luck, Brian Henderson-Sellers, Paolo Giorgini, Yves Demazeaux, James Odell, Juan Pavon, Amal El Fallah Seghrouchni, Peter McBurney, Van H. Dyke Parunak, Alexis Drogoul, Massimo Cossentino.

Domain characteristics lead choice for MAS approach

Virginia Dignum, Frank Dignum
Utrecht University, The Netherlands
{virginia, dignum}@cs.uu.nl

Our main hypothesis is that it is impossible to have one MAS approach that fits all application domains. The lack of uptake of MAS technology stems at least in part from the fact that industry does not get handles on which type of approach is suitable for which type of problem. Therefore it becomes very difficult to assess the possibility of success of this new technology and people rather stick to known technologies.

Current approaches to the design and implementation of agents and multi-agent systems exhibit large differences amongst themselves both at conceptual and engineering levels. The analysis of a problem domain will give rise to different decisions concerning the practical implementation of the following principles of MAS: the decomposition of problems into individual tasks or goals for the agents; deciding on interoperability and communication options for agents; coherence of action and avoidance of conflicts and harmful or useless interaction; and individual possibilities for representation and reasoning about actions, plans and knowledge of others. For example, the decision to use a blackboard structure instead of an ACL for agent communication leads to different types of agent systems (suited for different types of domains).

There is thus a need for a classification that 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.). Of course we would also need a methodology to determine the right class of MAS given a certain application area, such that we could start the design of the MAS from the situation at hand instead of starting from a particular MAS and adjust the situation to fit that type of MAS. So, we are concerned with the relation between system and environment and how to determine which kind of agent approach better suits environment characteristics.

Developers of agent systems have typically a fixed number of system components that can be manipulated, while others are uncontrollable. These components are the agents, their interfaces to the system environment (API), the agent communication channels, and the agent environment. In one extreme, the developer has full control over the agents' internal architecture and interaction while there is no control over any of the agent environment. In the other extreme the agents are unknown and uncontrolled and the developer fully controls the agent environment where interaction occurs. Depending on what is controlled and how those components are specified, different types of agent systems arise.

Hardly any research has been done in the area of decision support to the choice of (MAS) design methodology. We position that such choice should be guided by the characteristics of the domain. Important aspects to consider are the possibility/desirability to control the design of the agents and of the overall organization; the existence of global goals external to the agents; the degree of control over the interactions; requirements related to the control of systemic and agent behavior; and, the need for emergent behavior. We are currently holding a survey to determine the link between domain and approach. Future work is required to further develop these issues, by extensive evaluation of existing systems and starting a discussion between different AOSE research groups in order to understand and make explicit the motivations and differences between approaches.

Concepts, Method Engineering and Future Standardization for AOSE.

Brian Henderson-Sellers, Professor of Information Systems,
Faculty of Information Technology, University of Technology, Sydney, Australia

The challenges for agent-oriented software engineering include, in my opinion:

- The lack of a good example. Most exemplars I see whilst reviewing articles use system designs that I could easily accomplish just as well using an object-oriented development approach. That means that potential users of AOSE are not persuaded to spend their precious time in getting to grips with the new ideas of AOSE.
- The persisting gap between agents in the classical or AI-influenced sense and true AOSE. At agent-oriented conferences like AAMAS, the majority of attendees more readily fit into the first category. To encourage more AOSE, there are moves afoot to create a truly software engineering flavoured agent conference as well as our initiative to create an AOSE focussed journal (*International Journal of Agent-Oriented Software Engineering*).
- The lack of true AO programming languages. Taking a historical parallel, it is often said that the object paradigm, invented in the 1960s, did not take off until languages like C++, Eiffel and Smalltalk were commercially available in the late 1980s. The perception is that current AOPLs are front-ends to Java – ergo, agents are just objects! (Such is the perception of many in both industry and academe)
- The lack of standards. Perhaps this is a little early but in the last few months both ISO and the Object Management Group (OMG) as well as FIPA have been investigating the potential increased output of standards for agents by these respective standards bodies.

I therefore, in broad terms, agree with Scott deLoach's position statement with one exception. I have always believed that it is vital to agree on *concepts* before notation. Concepts represent our shared understanding and we must therefore have an *agreed* ontology or metamodel. Only then can we ask how we might communicate those concepts within our community and to users of our work. This is the notation. Creating a notation requires different skills from creating a metamodel or ontology; for instance, it requires understanding of semiotics and usability, which are irrelevant to building a conceptual basis.

Along these lines, I have recently led an ISO study group assessing the maturity of agent technology for ISO standardization (reporting May 2008). It is likely that one of the recommendations will be to investigate the likely standardization of the conceptual basis through a metamodel – an initiative that is also being pursued within the Object Management Group commencing June 2008. Once such a conceptual base is reasonably well-established, we should then pursue a common notation.

Finally, as you might expect, I endorse Scott's comment that in 2005 I advocated – and still do advocate – situational method engineering as an excellent way to build organizational-specific and/or project-specific methodologies for use by industry. The

approach has been shown to be successful on several object-oriented projects but, to the best of my knowledge, not yet in an agent-oriented software engineering environment. This needs to be done.