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.