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.