Agent-oriented software technologies: flaws and remedies

Jörg P. Müller, Bernhard Bauer

Angaben zur Veröffentlichung / Publication details:

Müller, Jörg P., and Bernhard Bauer. 2003. "Agent-oriented software technologies: flaws and remedies." *Lecture Notes in Computer Science* 2585: 210–27. https://doi.org/10.1007/3-540-36540-0_17.

Agent-Oriented Software Technologies: Flaws and Remedies

Jörg P. Müller and Bernhard Bauer

Siemens AG, Corporate Technology, Intelligent Autonomous Systems
Otto-Hahn-Ring 6, D-81730 Munich, Germany
{joerg.p.mueller,bernhard.bauer}@siemens.com
www.ct.siemens.com

Abstract. Agent-Oriented Software Technologies, i.e., the engineering of agent systems, agent languages, development tools, and methodologies, are an active research area. However, the practical influence of AOST on what we call mainstream software technologies is very small. As of today, trends in software technologies are not made by agent researchers or companies, but rather by Microsoft and Sun Microsystems. In this position paper, we investigate basic questions: Why are agent-oriented software technologies currently not fully exploiting their potential? Is there another "CORBA syndrome" lurking behind the next corner? And what can we do to better position agent software technologies in the market, and to increase their practical impact?
We are convinced that the most severe problems in today's agent-oriented software technologies and in the way we market them are due to a few basic flaws. In this paper, we will try to identify and discuss these flaws. However, it is also our firm belief that agent-oriented software technologies have a huge potential, and that there are remedies that can be applied to cure the flaws. We shall also identify some of these potential remedies and formulate them as recommendations.

1 Introduction

A (software) agent is a computer system, situated in some environment, that is capable of flexible autonomous actions in order to achieve its design objectives [28]. Since the emergence of multiagent systems and intelligent agents as a research topic in the late 1980s, research on agent technologies has steadily grown, and developed into various strands of research. One of these research strands which emerged towards the end of the 1990s, is called agent-oriented software engineering (AOSE). Like the term *agent* itself, the scope of AOSE is fuzzy and subsumes a variety of approaches to investigate how existing software engineering approaches could be improved by introducing agent concepts, but also what software engineering concepts are required to support the design and development of (multi-)agent systems. What AOSE is about is well illustrated by a quote from the call for papers of the International Workshop on Agent-Oriented Software Engineering [6]:

Just as we can understand many systems as being composed of essentially passive objects, which have state, and upon which we can perform operations, so we can understand many others as being made up of interacting, semi-autonomous agents.

Agent-oriented software engineering comprises a number of issues, including core aspects of software engineering (requirements engineering, analysis and design, development, testing and integration, lifecycle models), but also accompanying technologies such as specific programming languages for agents and agent systems, Integrated Development Environments (IDEs) and tools, as well as methodologies for agent development. Throughout this paper, we shall use the term *Agent-Oriented Software Technologies (AOST)* to denote the union of these topics, including core software engineering, languages, tools, and methodologies.

It appears that Agent-Oriented Software Technologies are an active research area. However, it is also undeniable that today the practical influence of AOST on what we call mainstream software technologies (MSST) is very small. As of today, trends in software technologies are not made by agent researchers, but rather by the ones of Sun Microsystems and Microsoft. Figure 1 provides a strongly simplifying illustration of the current status of AOST. Agent researchers and providers of agent-oriented software technologies are competing with Mainstream Software Technologies for a scarce and difficult to obtain resource: the attention of professional software architects and application developers. A simple comparison, e.g., of the number of downloads of one of the most successful agent platforms, the Java Agent Development Framework (JADE, [26]), with those achieved by Sun's J2SE leaves no doubt in who is ruling this market today: JADE has achieved some ten thousand downloads (mostly by universities and research institutes), while Java has a world-wide circulation with millions of downloads, in particular by industrial companies.

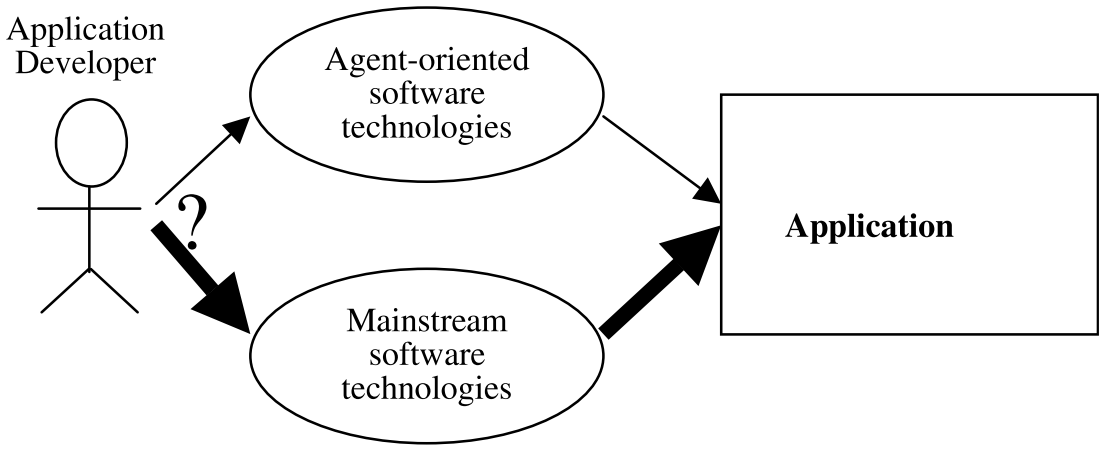

Fig. 1. The software technologies market

This situation may remind some of those agent researchers who have been in the field for a while to the situation in the mid-nineties, which we call the *CORBA syndrome*: at that time, mainstream software technologies such as CORBA started offering software infrastructures that promised software interoperability and dynamic discovery of services (e.g., Trading Services), leaving some agent researchers wondering and an-

gry that (at least from their perspective) some of the good ideas originating from agent technology were taken up from mainstream software technologies without giving the (multi-)agent community credit.

In this paper, we shall investigate some basic questions: Why are agent-oriented software technologies currently not fully exploiting their potential? Is there another CORBA syndrome lurking behind the next corner? And what can we do to better position agent-oriented software technologies in the market, and to increase their practical impact?

We are convinced that the problems in today's agent-oriented software technologies and in the way we market them are due to a few basic flaws. In this paper, we will try to identify and discuss these flaws. However, it is also our firm belief that agent-oriented software technologies have a huge potential, and that there are remedies that can be applied to cure the flaws. We shall also identify some of these potential remedies and formulate them as recommendations.

This paper is structured as follows: In Section 2, we describe the context of agent-oriented software technologies. In Section 3, we investigate the structure of today's' research on AOST. Section 4 provides a critical analysis of the relationship between AOST and mainstream software technologies. In Section 5, we identify what we believe are the essential flaws of AOST with respect to application development. Finally, Section 6 concludes and gives some basic recommendations how the position and impact of AOST can be improved.

2 Setting the Context

Agent technology interconnects various research disciplines [30]; in particular, agents are closely related to and appear in the context of:

- *software engineering*, in particular viewing agent oriented software development as a new way of abstraction;
- *distributed systems*, in particular multi-agent systems are inherently distributed systems; and
- *artificial intelligence*, in particular models for autonomy based on planning and knowledge representation (e.g., belief-desire-intention models)

Thus, and as stated in [28], agent technology implies a new view on understanding and modelling as well as implementing complex, self-organising, and distributed systems. Therefore, as we said in the introduction, we chose the term *agent-oriented software technologies* to subsume agent-oriented software engineering as a whole, including tools, methodologies, platforms, and languages.

We want to clarify this context by using a model originally proposed by Douglas Engelbart and adapted in [37] to agent technology (shown in Figure 2).

The *A-Level* provides the perspective of the application developer, i.e. the end-users of agent technology, who usually are not agent experts and mostly unfamiliar with the latest agent technologies. However, these users are often accustomed to and sometimes highly skilled in state-of-the-art software technologies like J2SE/EE,

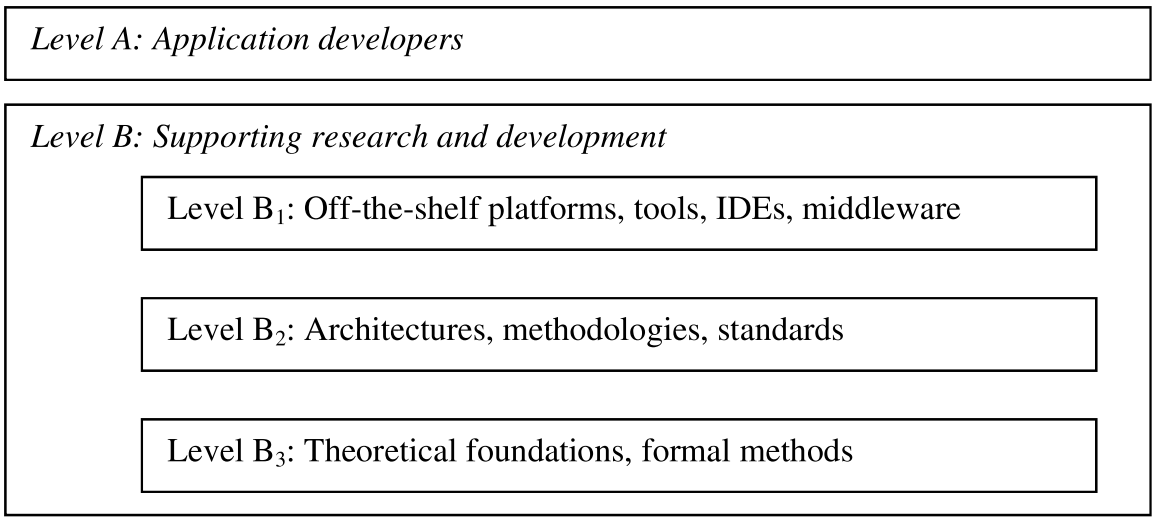

Fig. 2. A layered view of software technologies

WebServices and XML standards[1]. In any case, they recognise and appreciate tools that help them do their job more quickly or more conveniently.

The *B-Level* sums up the support that Software Technologies can provide the *A-Level* users. It is divided into three sub-levels:

The *B1-Level* provides off-the-shelf platforms and components, integrated development environments, test-beds, and other tools (e.g. for debugging or system analysis and design), but also infrastructures and middleware services, applied by the *A-Level* end-users. However in the case of AOST, commercially usable platforms and integrated development environments are still rare (see e.g. [2] for an overview of available platforms and [4] for a wider overview of agent related software); the market for programming languages and development environments is small and developing such tools is therefore unattractive from a business perspective (i.e., unprofitable). Hence, a common business model for agent platform providers (like Tryllian [46] or LivingSystems [34]) is to market applications built using their platforms.

The *B2-Level* sub-services the *B1-Level* with architectures, abstractions, methodologies, and standards. This is another point where agent technology misses out. At the moment, and very much alike in the early phases of object-oriented programming, no unified underlying formal model exists (since there is not even a unified shared notion of an agent); also there is no unified architectural approach to building agent platforms and agent applications. Methodologies exist (see e.g., [6], [30]) but refer to different underlying agent models, are hence not standardised and usually not supported by good-quality tools.

Finally, the *B3-Level* comprises theories, formal frameworks, and methods. In this area a considerable amount of research has been done and good results were accomplished, e.g., as documented by the ATAL workshop series [3]. However the transi-

[1] Note, that this strongly depends on the market sector. In some areas of industry, application developers are working on mainframes or developing mostly assembler, COBOL or C/C++ code.

tion from this level to the *B2 level* is not as well-described as it is e.g., for object-oriented software technologies.

3 The Agent Level

Looking at Figure 2 again, it appears that on the way from theories and formal approaches over languages to developed systems, information is often lost in a way that is hard to control and that is – to a degree – unavoidable. A good example are belief, desire, intention (BDI) agents [39], where a considerable gap can be observed - between the modal logics theory of BDI agents and the implementations of this theory, such as dMars [17] based on the Procedural Reasoning System (PRS, [24]). This mismatch is even stronger in the context of Java-based agent platforms, like JADE [26] or FIPA-OS [22], where the underlying BDI theory is maintained (if at all) merely at a folk-psychological level. Similar observations hold for reactive agent architectures (see [49] for an overview) that are being implemented e.g., using rules engines such as JESS [29]. JESS, however, is connected with Java-based platforms without defining formal semantics concerning internal state of the agent and its external behavior. So there is an easily recognizable (and undisputed) gap between theories and implementations of languages (model-theoretical and operational semantics) as well as platforms. This fact alone is not peculiar to agents; similar is true e.g., for object-oriented languages. There is to our knowledge no complete formal semantics for Java and all its packages (although there is active research on this topic, see e.g. [5][23]).

However, since it is argued that the major benefit of agent technology stems from the ability to describe and use the semantics of communication, and from describing complex systems by using concepts such as beliefs, desire and intentions, allegedly the consequence of the gap between theories and platforms is critical for AOST than it is for e.g., object-oriented software technologies. One could argue that at least these parts of agent technology that create the unique selling point of this software technology should be provided in a clean and formally rigid way. This mismatch becomes obvious when looking at the FIPA 97 specification part 2 on Agent Communication Languages[2] [21] there, a formal semantics is given for the library of communicative acts; however, when it comes to the specification of interaction protocols, the same document states ([21], p. 50):

> *A designer of agent systems has the choice to make the agents sufficiently aware of the meanings of the messages, and the goals, beliefs and other mental attitudes the agent possesses, [...]. This, however, places a heavy burden of capability and complexity on the agent implementation, [...]. An alternative, and very pragmatic, view is to pre-specify the protocols, so that a simpler agent implementation can nevertheless engage in meaningful conversation with other agents, simply by carefully following the known protocol.*

In addition the implementation of the communicative acts as well as the interaction protocols are pragmatic (i.e., they do not rigidly follow a formal model) in all cur-

[2] Note, that no major revisions were made on this topic since 1997.

rently available FIPA-compliant agent platforms. Another lapse in the FIPA specification is between the semantics of communicative acts on the one hand and the agent architecture on the other: The basis for the formal semantics of communicative acts is a modal logic model similar to the one used for BDI; in particular the definition of the semantics of the communicative act implies some kind of BDI architecture of the agent. However, FIPA leaves the choice to the platform providers which kind of agent architecture (e.g. BDI, re-active, hybrid) their platform supports. Therefore, the semantics of communicative acts imposes constraints on the agent architecture, which are not enforced by the standard.

4 AOST and Mainstream Software Technologies

As Figure 1 illustrates, the main purpose of agent-oriented and mainstream software technologies is to support application developers in writing applications. Hence it is natural that AOST and MSST have certain challenges and solutions in common, and that they compete in the market for acceptance.

4.1 Infrastructure and Middleware

A major flaw of today's AOST is that the actual usefulness of introducing agent concepts in software engineering (defined e.g., by a cost-benefit ratio) is not sufficiently clear. To one end, there is a strong overlap between the functionality supported by work done in AOSE and that provided by mainstream programming. For instance, considerably-sized agent infrastructure projects over the past ten years spent considerable efforts to enable two computers to exchange character sequences.

This negative impact of this tendency is increased from a different direction, i.e., by the convergence of distributed and object-oriented middleware. Many features that the agent community believed to be agent-specific were simply done quicker, better, and more professionally by the ones of Sun, Microsoft, Rational, or OMG. Examples are the different directory services, like LDAP [33] and UDDI [47], the development of middleware supporting distributed programming like CORBA [12] or Java RMI and last but not least the recent hype of WebServices [48] (with the implementations .net [1] and SunOne [45]). For agent researchers, focusing on the platform development and not building agents to extend existing standards and technologies has the inherent risk of running into another CORBA syndrome, i.e., for agents to stay an experimental science or become a niche market – a fate similar to that of functional or logic programming. This concern holds in particular for proprietary agent specific programming languages like APRIL [7] and Congolog [31]. These new programming languages only have a chance to take off if a large player will push them, like it was the case for Java with the same concepts as Smalltalk, but with a better marketing strategy and the right language at the right time. More likely though, and as it was witnessed in the case of functional and logic programming, the concepts underlying these agent-based programming languages will be taken up by large players in the more distant future.

Clearly, agent technology is (or: should be) more than middleware and infrastructure, but provides additional functionality and a valuable abstraction layer on top of these mainstream technologies.

From the multi-agent system point of view, the abstraction layer is very much compliant with the view taken by the Web Services approach, in that different (web) service provider agents can be used by different requester agents in a distributed environment. The crucial add-on of agent technology to this web services view, the capability of flexible interaction, is discussed in detail later on.

From the point of view of an individual service / agent, there is an additional abstraction process: describing the behavior of an agent based on goals, tasks and autonomous and rational decision-making, based on techniques such as planning, decision theory, game theory, and machine learning. This abstraction layer should be taken up by the agent community and has to be integrated in or build on top of mainstream technologies to be part of them in the future. We are aware that underlying a strict formal model to an agent is likely to put constraints on the freedom of the developer. For instance, many developers find the threaded behavior model underlying JADE [26] an awkward restriction rather than a helpful tool. However, if and when we argue that these concepts are the unique selling point for agent technology, we need to tackle these issues and find a way to offer agent developers access to advanced agent architectures and concepts without limiting their choice where to use them and where not. A possible starting point to overcome the coding effort for JADE behaviors is the approach presented by Martin L. Griss et al. in this volume, where JADE behaviors are generated from hierarchical state diagrams.

4.2 Benefits of Agent Technologies

We strongly believe that to be successful in the software technologies market, agent-oriented software technologies will need to focus on the essentials of agency and to clearly market themselves based on this added value. In our view, agent technology has an opportunity to prove its unique selling point in four major directions:

Expressing and Exploiting Semantics

The usage of semantics, where agent scientist have done respectable work, in contrast to pure syntax, as it is the standard today, combined with mainstream approaches from the SemanticWeb initiative, like DAML-OIL [14] or DAML-S [15], will bring additional benefit to the customers, thus resulting in increased funding for research on this topic, focusing on two aspects in particular:

- *Ontologies* and in particular the definition, maintenance and usage of large ontologies is a hot topic. Starting points could be standards from the W3C consortium like RDF(S) [40] and DAML-OIL which are already applied in the context of agent platforms (for instance, the JADE agent platform supports the import of ontologies defined using the ontology tool Protégé [38]).
- Self-organization, self-description and self-configuration of software based on semantic information of the agent itself, its environment and the services available

in its surroundings are other key functions that agents can provide to main stream technologies. This insight is the main background behind what IBM calls *autonomic computing* [25]. First steps in this direction are undertaken by DAML-S allowing a semantic description of the interfaces of software components.

Enabling Flexible Interaction

Flexible interaction within multi-agent systems is mainly based on communicative acts and interaction protocols. They are the basis for negotiation and the development of specific negotiation strategies, but also for the usage of market mechanisms for dynamic resource scheduling. In this context an open issue is the interworking of loosely coupled agents.

- *Communicative acts* or *speech acts* and the *interaction protocols* that are created on top of them have to be defined with a pragmatic semantics for advanced communication between distributed services or agents. A possible starting point could be WSCL [50]. WSCL allows us to define the abstract interfaces of web services, i.e., the business level conversations or public processes supported by a web service. In its current version, it defines a number of "primitive" speech acts called *interactions*, namely *Send* (the service sends out an outbound document); *Receive* (the service receives an inbound document); *SendReceive* (the service sends out an outbound document and then expects to receive an inbound document in reply); *ReceiveSend* (the service receives an inbound document and then sends out an outbound document); and *Empty*.
- Based on the interaction protocols negotiations can be established, e.g. in electronic market places or for optimization purposes. Specification and adaptation of negotiation strategies are fields that have long been studied in the agent domain and that can be usefully propagated to electronic market places or supply chain management. The necessity of distributed problem solving and the relationship and benefit compared with central solutions have to be outlined and detailed. One large benefit of market-based resource management and optimization is that it does not require global knowledge but can work with reasonable results even under the assumption of local, incomplete and possibly inconsistent knowledge. This setting seems to accurately describe the problem of optimizing virtual organizations. However, given what has been said before, it should be clear that research activities in this area should build on generic standards and business standards like BizTalk [10], ebXML [19], or RosettaNet [42].
- The ability of components designed by different designers to find a common way to communicate meaningfully is hard to achieve. Two aspects have to be considered, namely the formal description of the systems to allow an automatic interworking and combination of the components (e.g. using DAML-S) and the specification of system to allow other developers to use existing components, comparable with a semantically grounded version of UDDI.
- The ability of ensuring global properties of loosely coupled systems is a key property of multi-agent systems when it comes to understanding and designing decentralized, self-organized systems that share a common environment and where

certain global constraints and global welfare functions need to be enforced. Market mechanisms design is a powerful tool to analyze, design, and build these types of systems.

Expressing, Implementing, and Controlling Autonomy

Autonomy is one of the key features of agents, which need to be filled with life. Doing this includes a number of issues:

- The ability to sense, reason, decide and act in a dynamic environment, e.g. for delegation of routine tasks. This can be achieved by pragmatic knowledge representation, i.e. not developing an agent for arbitrary environments but for specific tasks and services. Using planning and goal-oriented reasoning to allow agents to react to unforeseeable events to some extent constitutes a value-added functionality with comprehensible benefit. Generic components based on standards for e.g. knowledge representation should be easily integratable into existing mainstream software such as libraries for constraint satisfaction.
- The ability to adapt the behavior of a system to its situation. When a situation changes, or the environment of an agent undergoes longer-term changes, the agent should be able to reflect upon its behavior, about causal relationships between its behavior and its environment, and about changes necessary to adapt its behavior to the new settings.
- The ability to make rational decision about actions and courses of action, by assigning utilities to actions or world states, enables an agent to perform well, but also to reflect and recognize opportunities for adaptation or optimization.

Supporting Individualization

We characterize individualization as customization of e.g. services and goods to the needs of a person. Individualization takes into account the specific context or situation of a user and can be applied at different levels.

- *User context:* The needs of a person are situation-aware, i.e. in different situations the user may have different needs, e.g. in a desert a glass of water may seem more appealing to a person than a large amount of money. Based on personal profiles and preferences and adaptation of them, situation awareness can depend on factors, like geographical context (e. g., positioning), social context (am I talking to my friend? my colleague? my boss?), users preferences and roles (e.g. at home/work), process context (e.g. workflows), temporal context (e.g. night / day), physical environment (e.g. raining), organizational context (e.g. chief / assistant), and emotional context (e.g. hungry, happy).
- *Levels of individualization*: Individualization can occur at different levels: At the *service level* (e.g., preferred hotel categories or travel type (like private or business trip) can be taken into consideration to support the customer with the optimal service or information in the current situation). As a special case of service level individualization, the *content* provided through a service can be prepared to match the user's need. The *user interface level* deals with individualization issues, e.g.

depending on the end-user device, preferred user's look and feel, or physical context (e.g., level of noise). The individual handling of tasks and workflows needs easily customizable *processes*. In addition, *infrastructure level* personalization is of interest for several reasons, e.g. paying for a high quality of service may be relevant for commercial use, while cheaper communication channels may be preferred for private usage.

4.3 Bridging the Gap between AOST and MSST – A Proposal

As already stated there is not only a gap between AOST and MSST, but also a large overlap between the two areas. The agent community deals more and more with topics of MSST, like infrastructures, software engineering methodologies for distributed systems, and vice versa, e.g. directory services (e.g. UDDI), some kind of interaction protocols (e.g. WSCL). Thus effort is invested twice.

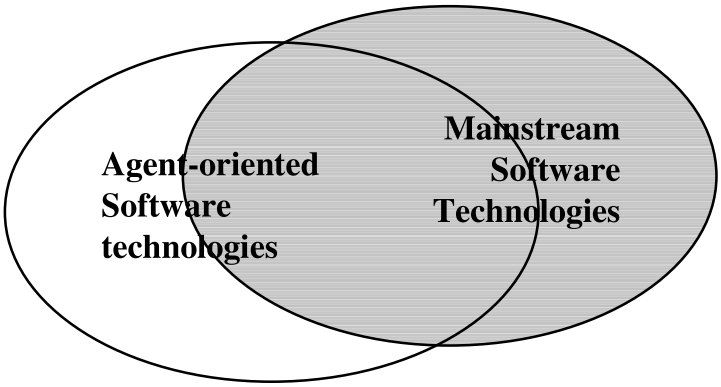

Fig. 3. Overlap between AOST and mainstream software technologies

Therefore we also believe that in order to exploit its potential, AOST needs to establish precise interfaces to mainstream software technologies and build on important standards (like middleware, business, and ontology standards). This will enable agent research not to lose much energy in developing basic technologies and standards, thus re-inventing wheels, but to focus on its unique selling points, i.e., semantics, flexible interaction, and autonomy.

We do believe that these well-defined interfaces need to be created in three directions, as shown in Figure 4: infrastructure and components, system description and specification, knowledge representation and ontologies. Fortunately, for each of these directions, promising standards are available upon which advances in agent technology can build: WebServices, UML, and ontologies. Let us have a closer look at these three aspects:

- *Infrastructure and components*: Based on the WebService infrastructure with directory services and communication channels, new functions could be added to build scalable and smart WebServices based on flexible interaction and autonomy. Based on standards, interaction protocols could be defined and smart components, like planning and learning can be added to present a real add-value.

- *System description and specification*: People in industry are familiar using object oriented analysis and design tools and are not familiar with the latest agent technology. Therefore a starting point is the usage of UML for the specification of agent-based systems, see e.g. [8] for several papers on this topic[3]. However the results are not yet fully convincing and some unification and revisions are necessary, but a migration path is pointed out.
- *Knowledge representation and ontologies*: Looking at current practice in software development the internal object oriented structure, containing to some extend semantic information, are serialized to flat XML structures without semantic information for communication between different systems. This loss of semantics can be overcome with the upcoming SemanticWeb standards, like RDF(S) and DAML-OIL. Using knowledge representation, modeling and reasoning mechanisms can also result in overcoming the bottleneck of standardization of domain specific ontologies, which are time consuming and not fitting the goals of all participants.

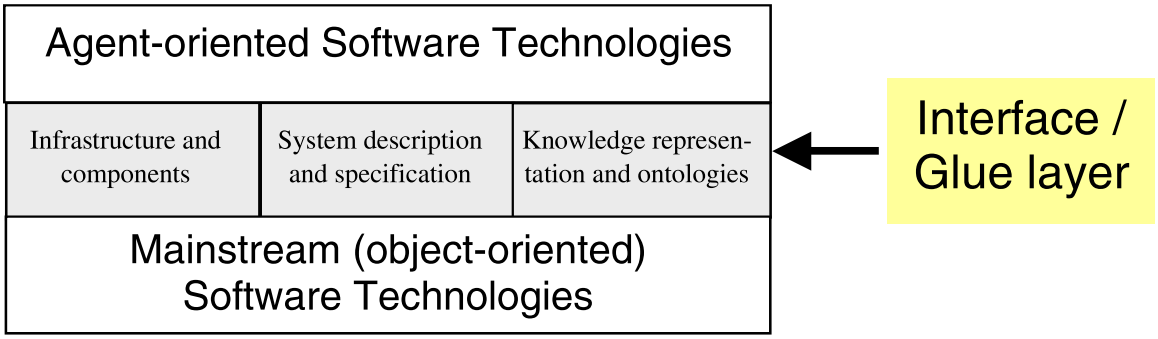

Fig. 4. Providing clear interfaces between AOST and MMST

Note that we do not argue that agent technology should flow into mainstream technologies. But the results, already commercially usable, have to be brought to market by adding it to mainstream technologies. Further research has to be done in variety of directions, see e.g. the AgentLink agent technology roadmap for details [41]. Let us highlight just one aspect that has to be studied in more details: we are still very bad in managing decentralization. More research is necessary in studying interrelationships and dependencies in and between decentralized, loosely coupled systems. While there are some areas which are well-investigated, e.g., market-based interactions (e.g., Sandholm[43]), or coordination (most prominently covered by the work of Lesser [32] and Decker [16]), the link from these theoretical results to practical software engineering is not yet very well explored.

[3] A small success story: parts of UML extension, originally developed for agents in the context of the agent standardization FIPA will be part of the upcoming UML 2.0 specification.

5 Building Applications with AOSE

Application areas for agent technology are wide-spread, like e-/m-business (e.g. marketplaces, SCM), industry (e.g. MES, production), consumer sector (e.g. games, smart mobile services), public services (e.g. human resource market, e-democracy), and telecommunication networks (e.g. network management, QoS provisioning). Looking at today's agent-based applications allows us to learn from a handful of existing, more or less successful examples, as well as from a large number of research prototypes that somehow never manage to get to the state of practical usefulness.

For one, it is still very early days talking about flaws of AOST in terms of application development; for the other we argue that it is a marketing issue rather than a technological issue.

One major flaw here is that agent researchers hardly think about how to bring their solutions together with existing IT landscapes. In the previous sections we pointed out how AOST and MSST can fit together. Migration is crucial for the success of agent technology, and protects companies' hardware and software investments. Don't wait until the killer application will be found. Start small and useful.

A second important issue is that a new technology or solution will only succeed if there is some tangible benefit attached to it. I.e., the first thing to worry about when trying to convince a business to take up agent technology is a business plan. An example of a successful commercially deployed agent-based solution is the manufacturing control system of DaimlerChrysler [11][13]. A precondition to success in this case was not to start with technology, but with the commercial and technical problems in a production line. It was only after these aspects had been analyzed when the necessary technologies were identified; then, a business case was set up showing how a double-digit increase in productivity was within reach using the agent-based solution — a convincing argument for managers.

Bringing agent technology to market requires investigating three different issues:

- *Technical issues*: *Hardware migration* can be a critical part of the investment in introducing agent technology. Looking at the DaimlerChrysler example again, it turned out that all the hardware, and in particular the machines for manufacturing the cylinder heads had to be exchanged to flexible CNC machines. Another point is the *operating system migration* (does the agent system run in the considered environment?). For example, in the manufacturing domain, using a Java-based platform was not an option, putting new restrictions on the agent infrastructure. In particular, the aspects of *performance*, *availability*, and *scalability* are success criteria for industrial application of agent technology. Can the agent platform handle 1000 requests in a minute and run throughout 365 days a year with a reliability of 99.9%? At the moment these are severe problems with agent software. Is the agent platform just „*yet another piece of infrastructure*“ competing with well-established products?
- *Cultural issues*: The problem of *cultural acceptance* of new technologies for people facing these technologies for a first time must not be neglected. Humans are used to think in specific patterns and have their mentalities and mindsets. Thus

new technologies need to change their mental attitudes to work productive in the new area. Moreover some people are in fear of new techniques. In particular with agents performing tasks in an automated fashion and threatening to organize peoples' schedule the lives, acceptance problems almost seem pre-programmed. No need to say that standard usability levels for software tools and solutions are also valid for agent-based software.

- *Economical issues*: Beyond technical and cultural issues the economic issues have to be considered. The cost of migration must not be underestimated, as already stated in the technical issues. Introducing a complete new hardware infrastructure is expensive and has to be calculated well. The benefit of successful migration can be measured in different ways. The direct (short-term) benefits include e.g. productivity and process efficiency increases, but also downtime decrease, throughput increase, etc. Indirect (longer-term) benefits can result e.g., in increased process or product flexibility. Usually management will be easier accessible to business plans based on direct benefits than on indirect benefits.

In summary, the introduction of agent based applications has to cope with technical, social and cultural as well as economical issues. In particular migrations paths need to be provided to deal with these issues, ensuring a smooth transition of existing applications to agent-based applications.

6 Summary and Recommendations

In this paper, we started by identifying and discussing what we believe are some major flaws of agent oriented software technologies — problems that today prevent AOST from being commercially successful. After describing the context of AOST and showing the structure of today's research on AOST, we gave a critical analysis of the relationship between AOST and mainstream software technologies and pointed out how the gap between both can be overcome. We finished our considerations with an investigation of the essential flaws of AOST with respect to application development. We will finish this paper with some basic recommendations on how the position and impact of AOST can be improved.

Recommendation 1:
Play to your strengths

We believe that the concepts of agent technology will play an important role in the development of the next generation solutions and systems over the next few years. In a nutshell, *agent-oriented software technologies is all about building and managing decentralized systems consisting of intelligent components.* We identified a number of core capabilities that agents can provide in comparison with related software technologies:

- Providing enriched, higher level communication (agent communication languages, based on existing transport encoding and underlying networking protocols, co-ordination of tasks, collaboration based on semantics and ontologies)

- Enabling more intelligent service provision and process management e.g. by personalization and integration of different services to value-added services (service wrapping, brokering, matchmaking, negotiation, auctioning, preference modeling, adaptive behavior by learning mechanisms)
- Dealing with the enlarging amount of information and functions (agent mobility, intelligent knowledge/information management, personalization, presentation)
- Allowing self-organizing of systems and processes (autonomous, flexible and pro-active behavior by planning, scheduling, and learning functionality)

As stated several times throughout this paper the strength of agent technology is based on these items and it would be desirable to see research focus more clearly around them. In this context, emerging standards and technologies must be used and applied within the agent community; no wheels must be re-invented. In particular the results and the strengths of AOST have to be marketed to the MSST community. But consider the second recommendation in this context!

Recommendation 2:
Prove Economic Benefits or **Sell Solutions not Technologies**

The economic benefits have to be shown to management to keep on attracting funding for technology development. I.e., technology is not the key factor: usually you do not sell technology, but solutions that must have an evident benefit for your customer. Thus the following aspects have to be taken into consideration:

- A good business plan is the key for bringing agent technology to your enterprise, especially in the current general depression in trade and industry. However do not overestimate the usefulness of indirect or deferred savings (e.g., the argument of saving money at a later point in time when introducing new product generations by increased product flexibility) as an argumentation for a corporate decision maker, but show that there is a tangible direct, short-term benefit.
- For several areas ideally fitting agent technology, like mobile commerce, telematics or optimization of processes in virtual enterprises, new business models are necessary for the success of these applications. At the moment the business models for these kinds of applications / solutions are weak and not promising for investors. So when thinking of agents, also think about business models for them.
- Build your business ideas on innovative scenarios and markets and show the applicability and benefit in real-world studies. Do not re-invent Microsoft Exchange when e.g., trying to model personal assistance for appointment scheduling (as one example), but rather analyze the shortcomings of commercial products and find out how the strengths of AOST can best be used (see Recommendation 1).

Recommendation 3:
Provide Migration Paths

In our view a main reason for the lack in industrial take-up for AOST is the absence of migration paths for the implementation of agent-based features, solutions, or prod-

ucts. We cannot hope to establish agent technology radically and from scratch. Rather we need to show industry how they can migrate to agent-based solutions gradually, hence protecting existing investments in hardware, software, and skills. Four important characteristics of industrial software development have to be addressed in this context:

- The scope of industrial software projects is much larger than typical academic research efforts, involving many more people and a higher financial investment across a longer period of time. Thus, collaboration among developers and high quality tools are essential;
- Industrial software development is focused more on preserving existing know-how, setting up development methodologies, and managing the development processes than on tracking the latest agent techniques. Thus, codifying best practice is essential;
- Industrial projects have clear success criteria. Thus, traceability between initial requirements and the final deliverable is essential. In particular the switch to agent technology has to be profitable. (see Recommendation 2).
- Business and technical considerations aside, issues of culture and mentality are key for a successful project. If you do not understand the requirements and specific concerns of users, you are likely to run into problems. This is true in particular if you are developing solutions or products for decision-support or personal assistance. There are plenty of psychological issues involved in providing humans with automated assistance, such as straight fears of being made redundant. While some of these problems are outside of your control, they will affect your work, and they will have a deep effect on whether your work will be a success.

Be aware that migration of agent technology towards applications is the most important factor of the ones we discussed here; very likely, whether you manage migration or not will be ultimately responsible for the success or failure of the solution, product, or system that you build.

Recommendation 4:
Promising Foci of Research

From our view several research areas are of main interest for industry:

- As we stated above (see Recommendation 1), the one main challenge for AOST is dealing with decentralized systems. We still know fairly little about how to deal with decentralization, and basic research on the topic seems in place, i.e. How can these systems be to understood, analyzed, designed, built, tracked and traced, monitored, tested, integrated, migrated, operated, and administered? What is the role of self-organization, self-configuration, and self-management in this context? How can these techniques be instrumentalised e.g., in the way Dorigo et al. [18] instrumentalised the concept of Ant Colony Optimization to solve combinatorial optimization problems?
- The current research activities in the context of the SemanticWeb [9] and in particular, their combination with currently emerging WebServices infrastructure,

should be strengthened. How can the SemanticWeb concepts be applied in real-world application? How can large ontologies be maintained and the interoperability of ontologies be assured? Business ontologies like ebXML or RosettaNet are useful and should be used wherever possible; however, difficult to achieve and practical experience shows that there is a necessity to allow more flexibility in defining data structures and to support their interchange. Closely related with the first bullet is the combination of the semantic web and the web services activities, allowing a flexible finding, matching and interoperation of services on a semantic level. Preliminary results on this topic were presented by Sycara et al. in [36], describing how an ontology-based matchmaker can be implemented on top of the UDDI service directory. A more visionary approach is the autonomic computing initiative of IBM [25].

- A third point where agents can add benefit is in the area of mobile and ubiquitous computing. Context recognition and interpretation in mobile and ad-hoc environments, tracing, tracking, monitoring and analyzing information and supporting the user by routine tasks, are examples of difficult problems that will need to be mastered. Here, the ability to take user or group preferences and profiles into consideration in a situation-aware fashion will be crucial.

Successfully bringing agent technology to market ultimately requires us to produce techniques that reduce the perceived risk inherent in any new technology, by showing the benefit and economic profitability, by presenting the new technology as an incremental extension of known and trusted methods, and by providing explicit engineering tools to support proven methods of technology deployment.

As a final note, we would like to state that the opinions and analysis stated in this paper constitute our subjective view; they are based on our experience of using agent technology in commercial environments. It is the nature of a position paper that readers may agree or disagree, and, by disagreeing, new positions will emerge. In writing this paper, our main hope is to encourage agent researchers think about how to position their work — either by agreeing or by disagreeing with our analyses.

References

1. .net: http://www.microsoft.com/net/
2. Agent Platforms: http://www.agentbuilder.com/AgentTools/commercial.php
3. Agent Theories, Architectures, and Languages (ATAL)- International Workshop Series: http://www.atal.org
4. Agentlink Software Report: http://www.agentlink.org/resources/software-report.html
5. Alves-Foss, J., editor, *Formal Syntax and Semantics of Java*, LNCS 1523, Springer, Berlin, 1999.
6. AOSE 2002: http://www.jamesodell.com/aose2002
7. APRIL: http://sourceforge.net/projects/networkagent/
8. AUML: http://www.auml.org/
9. Berners-Lee, T., Hendler, J., Lassila, O.: *The Semantic Web*. In: Scientific American, Issue 05/01, online available at http://www.sciam.com/2001/0501issue/0501berners-lee.html, 2001.

10. BizTalk: http://www.microsoft.com/biztalk/default.asp
11. Bussmann, S., Schild, K.: *An Agent-based Approach to the Control of Flexible Production Systems*. In: Proc. of the 8th IEEE Int. Conf. on Emergent Technologies and Factory Automation (ETFA 2001), pp.481-488 (Vol.2). Antibes Juan-les-pins, France, 2001.
12. CORBA: http://www.corba.org/
13. DaimlerChrysler Produktion 2000+ Project: http://www.agentlink.org/agents-london/presentations/Daimler Ch.ppt
14. DAML-OIL: http://www.w3.org/TR/daml+oil-reference
15. DAML-S: http://www.daml.org/services/
16. Decker, K.: http://www.cis.udel.edu/~decker/ (Homepage)
17. d'Inverno, M., Kinny, D., Luck, M. and Wooldridge, M.: *A formal specification of dMARS*. In: Intelligent Agents IV: Proceedings of the Fourth International Workshop on Agent Theories, Architectures, and Languages, LNAI 1365, pp. 155-176. Springer, Berlin, 1998.
18. Dorigo, M.: *Ant algorithms solve difficult optimization problems*. in: Advances in Artificial Life, 6th European Conference, ECAL 2001, LNAI 2159, pages 11-22, Springer, Berlin, 2001.
19. ebXML: http://www.ebxml.org/
20. FIPA (Foundation of Physical Intelligent Agents): http://www.fipa.org
21. FIPA 97, part 2: http://www.fipa.org/specs/fipa00003/
22. FIPA-OS: http://fipa-os.sourceforge.net/
23. Formal Semantics for Java: http://www-sop.inria.fr/oasis/java/
24. Georgeff, M.P., Lansky, A.L.: *Procedural Knowledge*. In: Proc. Of the IEEE Special Issue on Knowledge Representation, volume 74, pp. 1383-1398, 1986.
25. IBM's Autonomic Computing Initiative: http://www.ibm.com/research/autonomic
26. JADE Agent Platform: http://sharon.cselt.it/projects/jade/
27. Java RMI: http://java.sun.com/products/jdk/rmi/
28. Jennings, N.R, Sycara, K., Wooldridge, M.J.: *A roadmap of agent research and development*. In: International Journal of Autonomous Agents and Multiagent Systems, 1, pp. 275-306, 1998.
29. Jess: http://herzberg.ca.sandia.gov/jess/
30. Kinny, D., Georgeff, M.P.: *Modelling and Design of Multi-Agent Systems*. In: Müller, J.P., Wooldridge M.W., Jennings, N.R., editors, Intelligent Agents III, pp. 1-20, Springer, Berlin, 1997.
31. Lesperance, Y., Levesque, H.J., Lin, F., Marcu, D., Reiter, R., Scherl, R. B.: *Foundations of a logical approach to agent programming*. In: Intelligent Agents II, pp. 331-346, LNAI 1037, Springer, Berlin, 1996.
32. Lesser, V.: http://dis.cs.umass.edu/lesser.html (Homepage)
33. Lightweight Directory Access Protocol (LDAP): http://www.openldap.org/
34. Living-Systems: http://www.living-systems.com/
35. Müller, J.P.: *The Design of Intelligent Agents*. LNAI 1177. Springer, Berlin, 1996.
36. Paolucci, M., Kawamura, T., Payne, T., Sycara, S.: *Semantic Matching of Web Services Capabilities*. In: Proc. of the 1st International Semantic Web Conference, online available at http://www-2.cs.cmu.edu/~softagents/daml_Mmaker/daml-s_matchmaker.htm, 2002.
37. Petta, P., Müller, J.P.: *Guest Editorial*, Engineering Agent Systems: Best of 'From Agent Theory to Agent Implementation (AT2AI) 3', Applied Artificial Intelligence, 16, 2002, forthcoming.
38. Protégé: http://protege.stanford.edu/index.shtml

39. Rao, A.S., Georgeff, M.P.: *Modeling rational agents within a BDI architecture*. In: R. Fikes and E. Sandewall, editors, Proc. of the 2nd International Conference on Principles of Knowledge Representation and Reasoning, pp. 473-484, Morgan-Kaufmann,1991.
40. RDF(S): http://www.w3.org/RDF/ and http://www.w3.org/TR/rdf-schema/
41. Roadmap AgentLink on Agent Technology to be published on http://www.agentlink.org/
42. RosettaNet: http://www.rosettanet.org/
43. Sandholm, T.: http://www-2.cs.cmu.edu/~sandholm/ (Homepage)
44. SemanticWeb: http://www.semanticweb.org/
45. SunOne: http://wwws.sun.com/software/sunone/
46. Tryllian: http://www.tryllian.com/
47. UDDI: http://www.uddi.org/
48. WebServices: http://www.webservices.org/
49. Wooldridge, M.J.: *An Introduction to Multiagent Systems*. John Wiley&Sons, 2002.
50. WSCL: http://www.w3.org/TR/wscl10/