

Towards Seamless Agent Middleware

Andrea Omicini

DEIS, Università degli Studi di Bologna
Via Venezia 52
47023 Cesena, FC, Italy
andrea.omicini@unibo.it

Giovanni Rimassa

Whitestein Technologies AG
Gotthardstrasse 50,
8002 Zurich, Switzerland
gri@whitestein.com

Abstract

This paper describes the current evolution of agent middleware, supporting and promoting an autonomous agent component model. We first assess the current state of agent technology, then we make a case for increased integration between agents and their environment, composed by entities with different models. We notice that this integration trend emerged more and more in the last few years, but there are still important issues to be tackled and solved. Examples drawn from our experience with concrete agent infrastructures such as JADE are used to ground the discussion.

1. Introduction

With the rise of Internet-centric software development, almost every software package developed today has to be distributed or at least needs to be provided with some basic networking capabilities like Web registration or automatic update. However, distributed systems programming is not an easy task, so the notion of *middleware* became more and more popular in the past ten years. The very word *middleware* suggests something belonging to the middle: but, middle between what? The traditionally accepted definition refers to middleware as sitting in the middle between the operating system and its applications. This layered view results in emphasizing *vertical* interfaces from middleware to the OS and from middleware to applications.

However, being a kind of distributed infrastructure, middleware strongly benefits from the so-called *network effect*; that is, the value of owning a fixed part of the infrastructure strongly increases with the overall infrastructure size. Therefore, *interoperability* became a key issue for just about any proposed middleware solutions. This driving force, together with the constraint to preserve existing systems as much as possible while introducing the latest technologies in newly developed software, contributes to define the *Enterprise Application Integration (EAI)* task, and to put it forward as one of the main challenges that any middleware solution must face.

When tackling EAI issues, the focus is shifted towards *horizontal* interfaces, connecting different applications or different middleware platforms. The result of a naïve

approach often exhibits a very low *conceptual integrity* (the property of being understandable and explainable through a coherent and possibly limited set of concepts). This hampers attempts at analyzing the combined applications as a whole and ultimately yields a hard to maintain system.

Adopting a middleware technology is supposed to ease both new application development and legacy systems integration. To achieve integration while limiting conceptual drift, every middleware solution tries to define a *model* and cast it on the heterogeneous software landscape where it has to operate. This practice is meant to inject conceptual integrity into older systems and make them melt with the newly added components and, despite being a customary approach, it has been explicitly recognized relatively recently by the OMG in its *Model Driven Architecture (MDA)* specification [11]. Beyond addressing the *EAI* issue, another crucial motivation for middleware adoption is leveraging infrastructure development effort: reusing a single middleware across many applications saves development and testing costs. While the authors are well aware of this important aspect, they will not deal with it in this paper, because they feel that it is less relevant to the notion of seamless agent middleware.

2. Mainstreaming Agent Technology

The research areas that can be somehow connected with the word *agent* span several domains from distributed artificial intelligence to software engineering, and agent researchers did a lot of work to tackle the heterogeneous software integration issue. The approach of *multi-agent systems* sees the whole system as composed by autonomous interacting agents, and introduces a higher description level using social notions to capture interactive system behavior [8]. Looking at distributed, heterogeneous software systems from an organizational perspective, multi-agent systems research was able to take inspiration from such diverse fields as management theory, economics, and theory of natural language.

A major aim of multi-agent systems is to enable software integration on a deeper level, namely shifting the integration process from *syntactic interoperability* to *semantic interoperability*. This means producing a model

and an infrastructure where independently developed components can interact by making assumptions on each other that are perceived by human users as if the components could actually understand one another. From an EAI perspective, imposing a deeper model on existing systems requires greater analysis effort, but has the benefit that the resulting system can be reasoned about more extensively without breaking its unifying abstractions.

Technologies, like most human endeavors, follow a life cycle from their first conception to their eventual abandon, and they all come to a point where either they are widely adopted and last until they become obsolete, or they fail and are forgotten without even being tried for real. When a technology reaches this critical branch, several factors can affect whether it will either succeed or fail. The complete scenario has at least three dimensions:

- *Programming paradigm.* Adopting a new technology changes the concepts, the techniques and the programming styles used to create software and to reason about it.
- *Development Process.* Adopting a new technology affects the actual software development activities and how they are related.
- *Economical Environment.* Adopting a new technology can advantage some stakeholders more than others, and a market situation can turn out to be more or less favorable to a new technology.

2.1. Programming Paradigm Dimension

When pushing a new technology, early evangelists like to use (and sometimes abuse) the *paradigm shift* expression, to mean that their new technology is a complete conceptual revolution, a new foundation for software design and development. While this strategy has advantages such as improved media visibility, it can just as well backfire on the new technology that can be perceived as incompatible with the existing systems; moreover, too grand revolutionary claims often end up being seen as hype with no substance whatsoever. Agent technology is new and of course has some differences with respect to current software technologies, but can be seen as a coherent step of the general evolution of programming, after structured and object-oriented stages.

Agent technology tries to extend object technology mainly by enriching the component communication model and raising the abstraction level. Moreover, just like structured programming constructs are still present within object oriented code, so it's likely that not all entities in a software system will be autonomous or even rational, but there will still be a lot of passive components that cannot be modeled and used as agents. This consideration hints at a more general trend in agent middleware, that of having agents that are more and more immersed in larger systems, that will be further analyzed in the next sections of this paper.

2.2. Development Process Dimension

No technology can really hope to succeed without taking into account the reality of software development. The standard answer by researchers to this issue is to try and create development methodologies suitable to be used with the new technology. In our case, this means *agent-oriented methodologies*.

As it can be evinced from an analysis of the majority of proposed methodologies [1,6,21], most of the research efforts has been dedicated to define new metaphors, symbols and diagrammatic notations. At present, most methodologies focus on the early development phases, and the nearer the implementation is, the most similar to object oriented methodologies they become.

However, providing a methodology can be a first step, but is not enough. The actual, day-to-day needs of software developers have to be addressed. Generally, when a company considers the adoption of a new promising software technology, it launches one or more small projects using it, the so-called *pilot projects*. If programmers cannot use the new technology effectively, they will not be able to write good software in the pilot projects they are involved in. Of course, this will most likely lead the company to discard the new technology because it is perceived as not usable.

2.3. Economical Environment Dimension

In 1999, a Master Thesis from Eve M. Phillips at the MIT [17], with the witty title "*If It Works, It's Not AI: A Commercial Look at Artificial Intelligence Startups*", described and analyzed the rise and fall of the major AI startup firms. Academic bias and a lack of understanding of customers' needs and target markets were prominent. An interesting point was the confusion between product-based and service-based business models. Most startups tended to put the focus on their highly innovative *product*. But, software is quite a peculiar product: it has nearly zero marginal costs and almost infinite production capability.

However, exactly because it was so innovative, software from the AI startups was not readily usable by customers, who needed support *service*. This added consultant hiring and training expenses to the marginal costs. Moreover, moving from a product-based business model to a service-based one meant that the number of customers was limited by the number of consultants available and by their training time. This resulted in a major cut to the expectedly huge return of investments.

An important lesson that can be learned from the above story is that innovation has to be handled with caution. Early promoters of new ideas stress the conflict of their proposal with the state of the art rather than the similarities: however, this should not lead them to believe that general rules do not apply to their technology. A lot of errors mentioned in [17] were caused by the "*we are different*" attitude, all too common among

researchers and high tech startup founders. With respect to the product vs. service dilemma, the Open Source community [18] has shown that most software adds value as a service rather than as a product: this statement held for AI software in the eighties, and apparently keeps on holding for any software based on agent technology nowadays.

3. Supporting seamlessly situated agents

Middleware, being the concrete realization of a conceptual model with software artifacts, can help with technology transfer in all the three dimensions described above. In short, the paradigm shift or adaptation is eased by a well-defined set of APIs, middleware can effectively cooperate with tools to shape the development process, and a good quality, freely available middleware can rapidly increase the adoption of a new technology, thereby creating market opportunities for services and applications based on that technology.

While the above applies to middleware in general, the authors believe that agent-oriented middleware has some more peculiar features of its own. In the case of agent technology, the pivotal concept of *situated agent* brings about some interesting consequences when considered in the context of heterogeneous software integration. In complex applications, where a high degree of system integration is needed, a trend towards *seamlessly situated agents* can be observed; that is, a multi-agent system can be embedded in an application and work together with other sub-systems adopting different component models. In principle, this is in contrast with the *wrapper agent* approach, where every non-agent software entity or subsystem has to be hidden behind an agent. The authors observed the trend towards seamlessness during their active experience with agent technology in the past few years, and think this is the goal that agent middleware has to pursue in order to effectively achieve the mainstreaming of agent technology.

From the perspective of a software agent, the outcome is that most interactions with non-agent software are modelled as interactions with the physical environment [16]. So, the balance between social and physical environment critically depends on the amount of agent wrapping performed on non-agent components. In a highly wrapped solution, such as the one specified in the FIPA Agent-Software Integration document [5], nearly all interactions are likely to occur through ACL. If agents, instead, directly interact with most non-agent software entities, ACL usage will be used for interactions *about* those entities and not *with* the entities.

Language is about representation, so while it is not possible to physically act on a language-level entity, we often have the need to talk about physical domain entities. This means that the actions and events that describe the physical situatedness of a software agent must also have representations at the language level. The most common way to achieve this is by combining ACL

and domain ontologies – this is the approach followed by FIPA specifications, too. The greater balance between the social and the physical environment of an agent, spurred by the trend towards seamlessly situated agents, puts forth the role played by the dimension of *interaction*.

There, in fact, complexity is no longer simply bound to the granularity of the individual system's components, but rather to the many different ways in which components relates to and interact with each others, and with the overall system, too. Managing and governing the dimension of interaction is exactly the goal of *coordination*, which has emerged as a fundamental research areas in many several fields – from software engineering to artificial intelligence, from social theory to economics, from organizational theory to biology. The authors perceive the coordination issue to be the main challenge to address and solve in order to actually obtain seamlessly situated MASs; the complex spectrum that spans from purely linguistic interaction to lower level physical-like perception must be covered and effectively exploited to really connect agents with their social and physical environment.

In the agent field, coordination typically takes two different acceptations, according to the different perspectives adopted on the MAS: a *subjective* view, where the system is seen from the agent's viewpoint, and an *objective* view, where the system is seen from the designer's viewpoint [14]. These distinct views have originated two distinct flows of research, producing quite heterogeneous results that only recently have been recognized as complementary and reconciled [19]. While the notion of subjective coordination calls for agent-like coordinating entities, capable of understanding and deliberating their own course of action, objective coordination by its very nature straightforwardly address the programming paradigm issue. In fact, given that their meta-models typically abstract away from coordinated entities [2], (objective) coordination models can be defined and specified independently of the nature of components – being them processes, objects, agents, whatever – and the corresponding middleware can be exploited to enable and govern the interaction between components of any sort. Also, in the context of MAS, media provided by a coordination middleware typically play a twofold role: constitute the core for agent societies, by enforcing interaction rules and social norms (as in the case of electronic institutions [10]), and mediate / govern the agent's interaction with the environment, where services and resources of any sort can in principle be found. From the agent's viewpoint, then, coordination middleware represent and factorize the environment, and enable agents to deal with resource and service heterogeneity.

Even more, to *enable* interoperability between heterogeneous components is not the same story as to *promote* it. However, the very notion of coordination medium as an artifact, as induced by interpreting coordination as promoted by Activity Theory [19], allows

for a multiplicity of different visions of the medium itself, and of coordination in general. A component (an agent, but also a bean, an active object, a process, etc.) is enabled to interact at several different levels of awareness and power, and may use, see, inspect, reason about, and possibly modify coordination media depending on its nature and capabilities.

Agent coordination middleware impacts on the development process dimension, too. New AOSE methodologies are more and more focusing on the interaction aspects of MAS [12,22], by promoting social metaphors to first-class entities in the engineering process. By endorsing powerful and expressing (objective) coordination models, then, agent coordination middleware provide engineers with effective design abstractions, the coordination media.

Finally, as far as the economical environment dimension is concerned, the recent notion of *coordination as a service* [20] seems to walk along the right direction. No longer are coordination technologies provided as a whole package, nor should their choice necessarily precede or influence any other design commitment. Instead, agent coordination middleware are to be integrated within both new and already running projects or systems, adding the option of mediated interaction. Agents or components of any sort can then choose to exploit the power of coordination media. Agent metaphors like society and environment can then add their value to systems (even non-MAS ones) by organizing component interaction around (design and run-time) abstractions like coordination media.

4. The JADE case

Thus far, the three main dimensions of technology transfer (conceptual paradigm, development process and economic milieu) have been presented. Moreover, it has been stated that there is a trend towards seamlessly situated MASs, balancing the social and physical environments of agents and putting forward the aspect of interaction and coordination.

In order to ground our analysis and complete the considerations of the previous sections, we take as an example the *JADE* agent platform. The *JADE* case looks well suited to the task, because *JADE* is an implementation of the FIPA standard, and thus adopts the FIPA communication model that is ACL-centric and emphasizes subjective coordination strategies like negotiation or competition. This section will show how *JADE* itself is part of the ongoing trend towards seamless agent middleware, both at the API and at the application level. Lastly, some specific issues arising in trying to support seamlessly situated agents are described, along with the present and possible future solutions.

4.1. The evolution of JADE

JADE (Java Agent Development framework) is an Open Source software framework to aid the development of agent applications in compliance with the FIPA specifications for multi-agent systems [7].

The *JADE* project started in 1998 and went Open Source with its 1.3 version in February 2000. *JADE* is distributed under the LGPL license, which allows both for an Open Source infrastructure base and for commercial added value services and applications. Along the years, *JADE* has become more and more popular and it is presently used worldwide by a lot of academic institutions and industrial research centers. With the integration of LEAP project results [9], now *JADE* can span the whole spectrum from enterprise servers to lightweight PDAs and cell phones, thus providing the first end-to-end solution for FIPA-compliant multi-agent systems. *JADE* can serve as a significant example to show how the trend towards seamless agent middleware has been followed in practice.

At the API level, *JADE* was originally conceived as a runtime environment that could be shared by many agents, while safeguarding their basic autonomy. So, in the early releases of the framework the only exported API was the one made available to the `Agent` class and its subclasses (that is, application specific software agents). This API was well suited to deal with the software abstractions corresponding to the main FIPA standard assets (such as the FIPA ACL, the standard ontologies and interaction protocols).

However, there was no explicit support to connect agents with their external software environment (e.g. a GUI, a storage service, a computation service); software integration had to be achieved through the regular Java API. Moreover, early versions of *JADE* required the agent platform to be launched as one or more standalone processes containing the agents needed by the application at hand. This approach put agents' autonomy forward: at system startup time a set of software agents is activated, and it is up to them to create or connect to any further software components they deem necessary. Unfortunately, safeguarding agent autonomy with a *MAS-as-application* approach can complicate the task of dealing with existing software systems; a *MAS-as-subsystem* solution would be more suited to that task.

Later versions of *JADE* introduced the *In-Process Interface*, which is exactly an implementation of the *MAS-as-subsystem* idea. This feature allows an external application to start a *JADE* runtime through a local Java API; several agent containers can be created within the application's Java Virtual Machine. In-Process and ordinary (Out-of-Process) containers can be connected freely to form distributed platforms sharing JVMs with external applications. The In-Process Interface greatly eased the legacy integration and agent deployment features of *JADE*, allowing applications to be made of a mixed set of software components, only some of which are agents.

The very same path towards seamlessness can be observed by looking at how JADE was used by two different applied research projects of comparable size and complexity, but issued in two years' distance. The first project, named CoMMA [4], was run within the European Commission IST program from February 2000 to January 2002, and the whole CoMMA system is a multi-agent information system. JADE alone serves as distributed infrastructure and all communication among subsystems happens through FIPA ACL. Software packages used for e.g. learning algorithms and document parsing are enclosed within suitable agents. Starting the CoMMA system just requires launching a deployment-dependent number of JADE agent containers on suitable hosts; this is a pristine example of the *MAS-as-application* option.

The IST Collaborator project [3] was started in November 2001 and successfully ended in October 2003. The aim of the project is realizing a software environment that supports and enhances the activity of working teams. The project blends together features from web portals, multimedia, application sharing and workgroup. Such a diverse requirement set has prompted for a design using different component models for different subsystems: customizable web portal technology for user presentation, Enterprise Java Beans for persistent data storage, and others.

Agents have a well-defined role within the Collaborator system, and are by no means the only distributed infrastructure used; they are used as personal assistants for each user and as controllers to manage the collaborative session and to assist the human meeting coordinator in administrative tasks. JADE does not wrap every software subsystem with an agent; rather, now JADE is a subsystem itself, enclosed within application boundaries. The embedding features of the newer versions are extensively leveraged to achieve a seamless mix of agents and other kinds of software components.

4.2. Issues with seamless agent situatedness

The main issue that must be faced and solved to effectively support seamless embedding of software agents within heterogeneous software systems is reconciling agent autonomy and tight coupling with its environment.

To investigate this matter we are to focus on the agent/environment boundary. In the case of JADE implementation, this means to examine the `Agent` class and its possible incoming and outgoing method calls. `Agent` is arguably the most important class in JADE, for its central role in both the API and the runtime system. JADE takes several measures to ensure that instances of the `Agent` class are granted the autonomy level needed to comply at least with the weak definition of agency.

These techniques allow JADE to protect agent autonomy in usual situations. When JADE agents run within another application via the In-Process Interface, additional challenges are set. The JADE approach is to allow stricter control over in-process agents (restraining

their autonomy) *only with their consent*, i.e. if they were especially designed for this purpose. A fully autonomous JADE agent cannot be limited in any way by the mere fact it is running within an external application, but agents can be created with a tighter coupling with the application they run within, and can even be controlled by it.

Lastly, JADE satisfactorily handles the central issue of language-level modeling of the physical environment, within the standard framework of FIPA ACL and ontologies. The conceptual analysis of interaction we performed in the previous section suggests a noteworthy application: exploiting the *coordination as a service* [20] idea and the physical/linguistic level integration present in FIPA ACL, it would be possible to insert a coordination abstraction such as a TuCSoN tuple centre [13] within a FIPA platform such as JADE [16]. Such a composite infrastructure finds its inspiration in the Activity Theory: co-operation tasks could be directly performed using the coordination medium, but agents would be aware of the ongoing processes [15]. Moving from the objective coordination laws to their subjective representation with ACL and ontologies, the chasm from co-ordination to co-operation can be crossed at any time during normal operation of the MAS. We believe that such an integrated infrastructure could greatly enhance the effectiveness of agent technology in complex applications with heterogeneous component models, augmenting the conceptual paradigm factor in technology transfer.

5. Conclusion and future work

This paper analyzed the present state of middleware for multi-agent systems, pointing out a trend towards the absorption of MAS within larger systems made of agents as well as of parts with a different component model. This is mostly due to the fact that agent technology is generally used to tackle the most complex distributed systems. Nearly all those systems include older, pre-existing systems as subparts, or have to interoperate with a legacy environment. We believe that this trend is bound to continue and further increase as agent technology is employed in more and more real complex application domains. Among the different aspects addressed by agent middleware, a prominent one is interaction. We highlighted the link between interaction and coordination issues and middleware, showing how paradigm, process and economics are addressed. The JADE platform served as a concrete example.

We are currently focusing on the seamless integration issue both at the model and at the infrastructure level: at the model level, we are investigating the interaction dimension to strike a balance between subjective and objective coordination. At the infrastructure level, we are planning to embody abstractions from TuCSoN coordination infrastructure within JADE FIPA-compliant assets in order to enable flexible transitions between co-

operation and co-ordination activities in cooperative multi-agent systems.

6. Acknowledgement

Research under the auspices of: MIPAF (the Italian Ministry of Agricultural and Forestry Policies), project "SIPEAA", paper no. 25; MIUR (the Italian Ministry of Education, University and Research), COFIN 2003 project "Fiducia e diritto nella società dell'informazione", paper no. 5; and EC (the European Community), FP6 project "AgentLink III".

7. References

- [1] Caire, G., Chainho, P., Evans, R., Garijo, F., Gomez Sanz, J., Kearney, P., Leal, F., Massonet, P., Pavon, J., Stark, J., Agent Oriented Analysis using MESSAGE/UML, In Agent-Oriented Software Engineering II, Springer Verlag, 2nd International Workshop (AOSE 2001), Montreal, Canada, May, 2001, 101-107.
- [2] Ciancarini, P., Coordination models and languages as software integrators. ACM Computing Surveys, 28(2):300-302 (1996).
- [3] COLLABORATOR Project Home Page. Available at <http://www.ist-collaborator.net>.
- [4] CoMMA Home Page. <http://www.si.fr.atosorigin.com/sophia/comma/Htm/HomePage.htm>
- [5] FIPA Agent Software Integration Specification. Available at <http://www.fipa.org/specs/fipa00079/>
- [6] Giunchiglia F., Mylopoulos J., Perini A., The Tropos Software Development Methodology: Processes, Models and Diagrams, 2002 Autonomous Agents and Multi-Agent Systems (AAMAS 2002), Bologna, Italy, July 2002. ACM.
- [7] The JADE Project Home Page. Available at <http://jade.cse.it>
- [8] Jennings, N.R., Campos, J.R. Towards a Social Level Characterisation of Socially Responsible Agents. IEE Proceedings on Software Engineering, 144(1):11-25, (1997).
- [9] LEAP Home Page. Available at <http://leap.crm-paris.com/>
- [10] Noriega, P. and Sierra, C., Electronic Institutions: Future Trends and Challenges. In Klusch, M., Ossowski, S. and Shehory, O., editors, Cooperative Information Agents VI, Springer-Verlag, 2002.
- [11] The OMG Model Driven Architecture. Available at <http://www.omg.org/mda/>.
- [12] Omicini, A., SODA: Societies and Infrastructures in the Analysis and Design of Agent-based Systems. In Ciancarini, P. and Wooldridge, M., editors, Agent-Oriented Software Engineering, Springer-Verlag, 2001.
- [13] Omicini, A. and Denti, E., From Tuple Spaces to Tuple Centres, Science of Computer Programming 41(3):277-294, November 2001, Elsevier Science B.V.
- [14] Omicini, A. and Ossowski, S., Objective versus Subjective Coordination in the Engineering of Agent Systems. In Klusch, M., Bergamaschi S., Edwards, P. and Petta, P., editors, Intelligent Information Agents: An AgentLink Perspective, Springer-Verlag, 2003.
- [15] Omicini, A., Ossowski, S. and Ricci, A., Coordination Infrastructures in the Engineering of Multiagent Systems. In Bergenti, F., Gleizes, M.-P. and Zambonelli, F., Methodologies and Software Engineering for Agent Systems: The Agent-Oriented Software Engineering Handbook, chapter 14, 273-296, June 2004, Kluwer Academic Publishers.
- [16] Omicini, A., Ricci, A., Viroli, M. and Rimassa, G., Integrating Objective & Subjective Coordination in MultiAgent Systems, 19th ACM Symposium on Applied Computing (SAC 2004), March 14-17, 2004, Nicosia, Cyprus, ACM, 449-455.
- [17] Phillips, E. M., If It Works, It's Not AI: A Commercial Look At Artificial Intelligence Startups. Master Thesis, MIT 1999
- [18] Raymond, E., The Magic Cauldron. Available at <http://www.tuxedo.org/~esr/writings/magic-cauldron/magic-cauldron.html>
- [19] Ricci, A., Omicini, A. and Denti, E., Activity Theory as a Framework for MAS Coordination. In Petta, P., Tolksdorf, R. and Zambonelli, F., editors, Engineering Societies in the Agents World III. Springer-Verlag, 2003.
- [20] Viroli, M. and Omicini, A., Coordination as a Service: Ontological and Formal Foundation. In Brogi, A. and Jacquet, J., editors, FOCLASA 2002 – Foundation of Coordination Languages and Software Architecture. Elsevier Science B. V., 2003.
- [21] Wooldridge, M., Jennings, N. and Kinny, D. The Gaia Methodology for Agent-Oriented Analysis and Design. Journal of Autonomous Agents and Multi-Agent Systems 3(3):285-312 (2000).
- [22] Zambonelli, F., Jennings, N. and Wooldridge, M., Organisational Rules as an Abstraction for the Analysis and Design of Multi-Agent Systems. International Journal of Knowledge Engineering and Software Engineering, 11(3):303-328 (2001).