Available online at www.sciencedirect.com

ScienceDirect

Information and Software Technology 50 (2008) 10–21

INFORMATION AND SOFTWARE TECHNOLOGY

www.elsevier.com/locate/infsof

JADE: A software framework for developing multi-agent applications. Lessons learned

Fabio Bellifemine [a,*], Giovanni Caire [a], Agostino Poggi [b], Giovanni Rimassa [c]

[a] *Telecom Italia, Via G. Reiss Romoli, 274, I-10148, Torino, Italy*
[b] *Dipartimento di Ingegneria dell'Informazione, University of Parma, Parco Area delle Scienze, 181A, I-43100, Parma, Italy*
[c] *Whitestein Technologies (formerly he was at University of Parma), Pestalozzistrasse 24, CH-8032, Zürich, Switzerland*

Available online 22 October 2007

Abstract

Since a number of years agent technology is considered one of the most innovative technologies for the development of distributed software systems. While not yet a mainstream approach in software engineering at large, a lot of work on agent technology has been done, many research results and applications have been presented, and some software products exists which have moved from the research community to the industrial community. One of these is JADE, a software framework that facilitates development of interoperable intelligent multi-agent systems and that is distributed under an Open Source License. JADE is a very mature product, used by a heterogeneous community of users both in research activities and in industrial applications. This paper presents JADE and its technological components together with a discussion of the possible reasons for its success and lessons learned from the somewhat detached perspective possible nine years after its inception.

Keywords: Agent technology; Open source software; Distributed systems

1. Introduction

Since a number of years, researchers have promised that agent technology will give an important contribution to distributed software systems by simplifying their development and by allowing smarter, more flexible and efficient applications. While not yet a mainstream approach in software engineering at large, a lot of work on agent technologies has been done, a lot of research and application results have been presented and some software products have moved from the research community to the industrial community. One of these products is JADE ([1,2,18]), a software framework to facilitate the development of interoperable intelligent multi-agent systems that is used by a heterogeneous community of users as a tool for both supporting research activities and building real applications.

The initial software developments, that eventually created the JADE platform, were started by Telecom Italia (formerly CSELT) together with the University of Parma in July'98, motivated by the need to validate FIPA specifications [14] through a concrete compliant implementation. The project also aimed at realizing a middleware for application development that, as one of the main challenges, put great emphasis on usability and simplicity of its API's such as it was usable not just by its programmer's team but also by other programmers without extensive experience in the FIPA specifications and multi-agent systems theory. In order to increase its diffusion in the communities of FIPA and of agent systems developers, and in order to facilitate its usage also in commercial products, in 2000 JADE went Open Source distributed by Telecom Italia under the LGPL (GNU Lesser General Public License) license [20]. After that, JADE grew both in usage and in offered functionalities

[*] Corresponding author. Tel.: +390112286175; fax: +390112286190.
E-mail addresses: fabioluigi.bellifemine@telecomitalia.it (F. Bellifemine), giovanni.caire@telecomitalia.it (G. Caire), poggi@ce.unipr.it (A. Poggi), gri@whitestein.com (G. Rimassa).

and performances, thanks also to the contributions (e.g. bug reporting and fixing, code extensions, etc.) of the large community of users that, as depicted in Fig. 1, counts today more than 120,000 downloads from the official web site [18]. Fig. 1 shows how the number of downloads from the official JADE web site progressed after each release (the version number is reported close to each plotted point). The latest version of JADE is 3.5, released on 25 June 2007. As for most open source projects, it is very difficult to estimate the real number of users but it can be said that, even after 7 years of activity, the user forum is still very active with an average of ∼4 exchanged e-mails/days.

As a matter of fact, an important topic in multi-agent systems research is the development of programming languages and tools for the realization of such systems; in the last decade, more than a hundred of such tools have been realized. However, most of them were created as short-term research activity and their development and maintenance ended after only two or three years. [7] is a recent survey on multi-agent languages and platforms, while a short list of tools that are still active projects and that, in our opinion, represent related work of JADE, are the followings: Cougaar [16], JACK [24], 3APL [12], and Agent Factory [22].

Cougaar [16] is an agent architecture developed under two DARPA programs in the domain of military logistics planning and execution. It is a Java-based framework for large-scale and flexible distributed agent-based applications. It uses a flexible component model to dynamically load components by inserting component "binder" proxies to mediate interactions with system services. It also uses a multi-tiered interaction model: within agents, components interact via a local publish-subscribe mechanism, while inter-agent communication happens via message passing. Moreover, Cougaar enables both dynamic negotiation of agent relationships via a hierarchical service discovery mechanism and the organization of agents into communities.

JACK [24] is a commercial software development framework for the realization of BDI agent-based systems [21]. Rather than using a logic-based language, it extends the Java language through a set of syntactic constructs to deal with agent plans and agent beliefs. JACK offers a good degree of modularization through the composition of agent plans and the concept of agent teams. It provides also a set of development tools allowing, for example, the editing of agent code through a dedicated editor and a graphical plan editor that allows the construction of a plan via visual components similar to state charts.

3APL [12] is a programming language for implementing agent-based systems via programming constructs to deal with agents' beliefs, goals, basic capabilities (such as belief updates, external actions, or communication actions) and via a set of practical reasoning rules through which agents' goals can be updated or revised. 3APL is integrated within an IDE that helps a programmer to write the programs that implement the individual agents, to execute such programs in either a step-by-step or continuous mode, and to monitor their execution by accessing agent internal state and tracing the messages exchanged by the agents. It is also well integrated with Prolog and Java programming languages.

Agent Factory [22] is also based upon BDI concepts and it also includes an interpreter embedded within a distributed FIPA-compliant Run-Time Environment that allows the deployment of agents on desktops and servers, personal digital assistants, and mobile phones. Agent Factory provides a set of tools to help programmers in the development tasks; in particular, a tool for the construction of agents from a set of UML sequence diagrams and a suite of tools that facilitate the testing and debugging of applications.

In our opinion, three are the main identifying features of JADE in comparison with the rest of the state of the art:

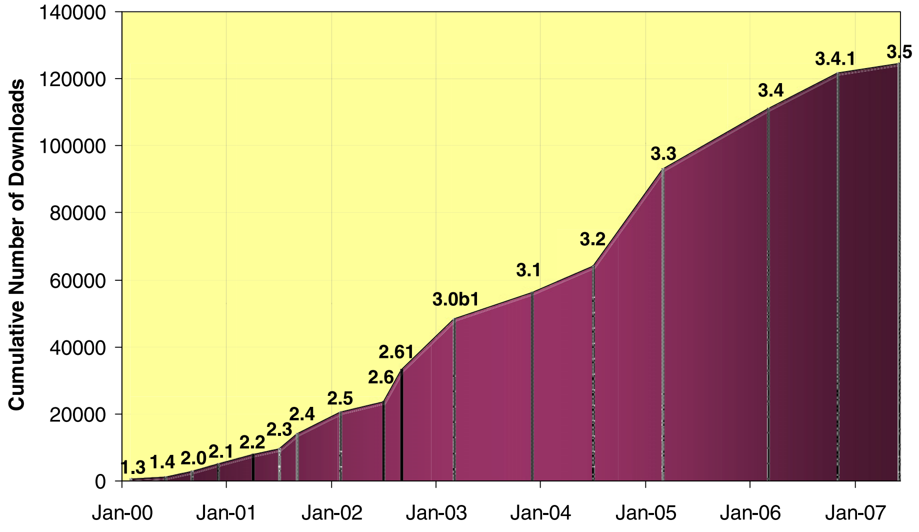

Fig. 1. Cumulative number of downloads after each software release. (the version number is reported close to each plotted point).

– Firstly, JADE is completely based upon the FIPA specifications, so much so that, given also its history, it is often identified with the FIPA specifications itself.
– Then, JADE provides a proper set of functionalities to simplify development of multi-agent systems but it puts very few restrictions on the user code without, in particular, imposing any specific agent architecture. That means users do no need either to understand or use any BDI architecture, architecture which exists in literature in hundreds of flavours and that has not yet gained acceptance in the software engineering community. Users must simply use and write Java code without need to learn any new special construct.
– Finally, JADE can be deployed both on JEE, JSE, and JME devices and its runtime provides a homogeneous set of APIs that is independent of the underlying network and Java technology.

This paper is organized as follows. Section 2 describes the JADE agent architecture and its main features. Section 3 shows how JADE enables agents to adapt to mobile environment characteristics and to devices with limited memory and processing power. Section 4 discusses the integration of agents with other technologies, in particular Web Services and reasoning systems. Section 5 discusses, with an historical perspective, the possible and likely reasons for the widespread usage of JADE; particular care is devoted to those non-technical aspects which might represent lessons learned of general usefulness to other projects. Section 6 gives an overview of an industrial application in the field of network management, and it presents some of the reasons that led to the selection of JADE as development framework for that system. Finally, Section 7 concludes the paper and outlines some future work.

2. JADE software architecture

The JADE architecture is firmly rooted in peer-to-peer ideas and communication patterns [2]. The intelligence, the initiative, the information, the resources and the control can be fully distributed across a group of heterogeneous hosts, including mobile terminals and PDAs, both in the wireless and in the wired network. The environment can evolve dynamically with peers, i.e. agents, which can appear and disappear in the system according to application needs and requirements. Communication between the peers, regardless of whether they are running in the wireless or wired network, is completely symmetric, each peer being able to play both the initiator and the responder role.

The JADE component model organizes applications as structured ensembles of software components, which belong to two categories:

– *Agents* – These are the peers mentioned above, exhibiting autonomy and communicating through asynchronous message passing.
– *Services* – These are non-autonomous components, which can run on a single node or cooperatively on multiple nodes, and whose operations can be triggered by agents.

JADE is fully developed in Java and, from a software engineering standpoint, values highly, among others, the following driving principles:

– *Interoperability* – JADE is compliant with the FIPA specifications [14]. As a consequence, JADE agents can interoperate with other agents, provided that they comply with the same standard.
– *Uniformity and portability* – JADE provides a homogeneous set of APIs that are independent from the underlying network and Java version. More in details, the JADE run-time provides the same APIs both for the JEE, JSE and JME environment. In theory, application developers could decide at deploy-time which Java run-time environment to use.
– *Easy to use* – The complexity of the middleware is hidden behind a simple and intuitive set of APIs.
– *Pay-as-you-go philosophy* – Programmers do not need to use all the features provided by the middleware. Features that are not used do not require programmers to know anything about them, neither they add any computational overhead.

JADE includes both the libraries (i.e. the Java classes) required to develop application agents, and the run-time environment that provides the basic services and that must be active on the device before agents can be executed. Each instance of the JADE run-time is called container (since it "contains" agents). The set of all containers is called platform and provides a homogeneous layer that hides to agents (and to application developers as well) the complexity and the diversity of the underlying tiers (hardware, operating system, type of network, JVM). Fig. 2 draws the architecture of a JADE agent system deployed on a set of heterogeneous computing nodes.

The JADE design choice (in particular not to commit too strongly to specific agent architectures) and its modular architecture allowed JADE to fit the constraints of environments with limited resources. JADE has however also been integrated into complex server-side infrastructures such as .NET ([18], see FAQ section on the Web Site) and JEE [10] where JADE becomes a service to execute multi-party proactive applications. The limited memory footprint allows installing JADE on all mobile phones provided that they are Java-enabled. The JADE run-time memory footprint, in a MIDP1.0 environment, is around 120 KB, but can be further reduced to 50 KB by using the ROMizing technique ([4,5]), i.e. by compiling JADE together with the JVM. Analyses and a benchmark of scalability and performance of the JADE Message Transport System can be found in [9,23].

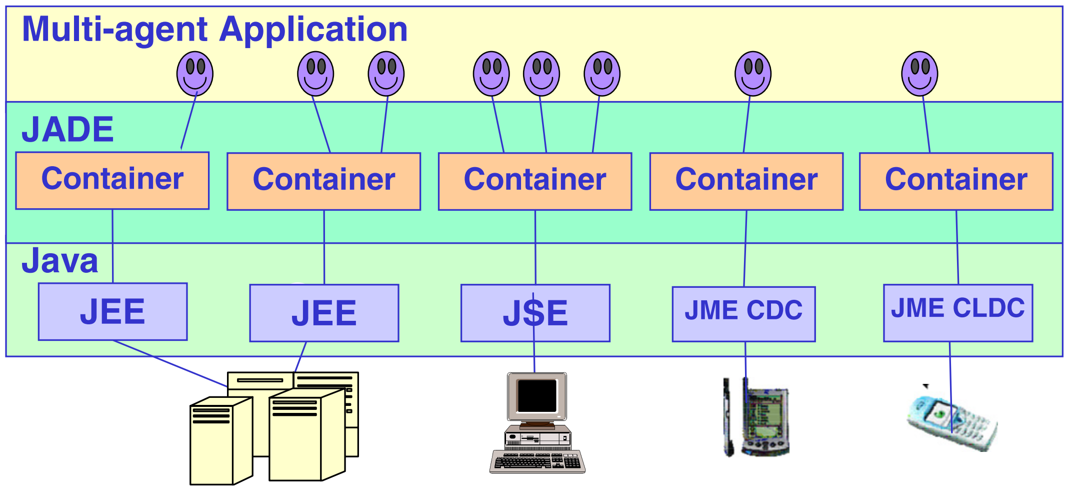

Fig. 2. The architecture of a JADE agent system.

From the functional point of view, JADE provides the basic services for distributed peer-to-peer applications in the wired and mobile environment. JADE allows each agent to dynamically discover other agents and to communicate via the peer-to-peer paradigm. From an application point of view, each agent is identified via a unique name and it provides a set of services. It can register and modify its services and/or search for agents providing a given service; it can also control its life cycle and, in particular, communicate with all other peers.

Agents communicate by asynchronous message exchange, a communication model almost universally accepted for distributed and loosely coupled interactions, i.e. between heterogeneous entities. In order to communicate, an agent just sends a message to a destination. Agents are identified by a name, therefore the send operation does not need the destination object reference and, as a direct consequence, there is no temporal dependency between communicating agents. The sender and the receiver could be available at different times. The receiver may not even exist (or not yet exist) or could not be directly known by the sender that can specify a property (e.g. "all agents interested in football") as a destination. Because agents identify each other only by name, any change at run-time of their object reference is fully transparent to applications.

Despite this type of communication, security can be preserved via suitable mechanisms to authenticate and verify "rights" assigned to agents. When needed, therefore, an application can verify the identity of the sender of a message and prevent actions that a principal is not allowed to perform (for instance an agent may be allowed to receive messages from another agent, but not to send messages to it). All messages exchanged between agents are carried out within an envelope including only the information required by the transport layer. That allows, among others, to encrypt the content of a message separately from the envelope.

The structure of a message complies with the Agent Communication Language (ACL) defined by FIPA [14] and includes fields (such as variables indicating the context a message refers to, and time limits within which an answer has to be received) aimed at supporting complex interactions and multiple parallel conversations. In order to support the implementation of complex conversations, JADE provides a set of skeletons of typical interaction patterns to perform specific tasks, such as negotiations, auctions and task delegation. By using these skeletons (implemented as Java abstract classes and identified as 'interaction protocols'), programmers can get rid of the burden of dealing with synchronization issues, timeouts, error conditions and, in general, all those aspects not strictly related to the application logic. In order to facilitate the creation and handling of message content, JADE provides support for automatically converting back and forth between formats suitable for content exchange, including XML and RDF, and the format suitable for content manipulation (i.e. Java objects). This support is integrated with some ontology creation tools, e.g. Protégé, to enable programmers to graphically create their ontology and validate the messages exchanged by the agents of the system (see chapter 13.1 of [1]).

In order to increase scalability and also to meet the constraints of limited resource environments (such as mobile phones), JADE adopts a thread-per-agent concurrency model, where a single Java thread is assigned to an agent to execute all its tasks as opposite to a thread-per-conversation (or per-task) model. Each agent has an embedded round-robin scheduler of *Behaviour* objects where the *Behaviour* class is the reification of an agent task representing a scheduling and execution unit. Scheduling is cooperative, that is a *Behaviour* is not executed until the previous one yields control back to the scheduler. Structural composition of agent behaviours is implemented by applying the Composite and the Chain of Responsibility patterns [15] to active objects task scheduling: hierarchical trees of *Behaviour* objects can be composed where each intermediate node can freely implement its policy to schedule the elements belonging to its sub-tree. In this way, several elementary tasks can be combined at run-time to form complex structures of task scheduling, including a set of concurrent Finite State Machines.

In the JSE and Personal Java environments, JADE supports mobility of code and execution state (see Chapter 6 of [1]). An agent can stop running on a host, migrate on a different remote host (without the need to have the agent code already installed on that host), and restart its execution from the point it was interrupted (actually, JADE implements a form of not-so-weak mobility because a standard JVM does not enable saving the stack and the program counter). This functionality allows, for example, distributing computational load at runtime by moving agents to less loaded machines without any impact on the application.

The platform also includes a naming service (ensuring each agent has a unique name) and a yellow pages service that can be distributed across multiple hosts. Federation graphs can be created in order to define structured domains of agent services. Another very important feature consists in the availability of a rich suite of graphical tools supporting both debugging and management/monitoring. By means of these tools, it is possible to remotely control agents, even if already deployed and running: agent conversations can be emulated, exchanged messages can be sniffed, tasks can be monitored, and agent life-cycle can be controlled.

3. Specializing JADE agents for the mobile environment

As already mentioned, the JADE run-time can be executed on a wide class of devices ranging from servers to cell phones, for the latter the only requirement being the availability of Java MIDP1.0 (or later versions). In order to properly address the memory and processing power limitations of mobile devices and the characteristics of wireless networks (GPRS in particular) in terms of bandwidth, latency, intermittent connectivity and IP addresses variability, and at the same time in order to be efficient when executed on wired network hosts, JADE can be configured to adapt to the characteristics of the deployment environment. JADE architecture, in fact, is completely modular and, by activating certain modules instead of others, it is possible to meet different requirements in terms of connectivity, memory and processing power.

More in details, a module called LEAP allows optimising all communication mechanisms when dealing with devices with limited resources and connected through wireless networks. By activating this module, a JADE container is ‘‘split’’, as depicted in Fig. 3, into a front-end running in the mobile terminal and a back-end running in the wired network. A suitable architectural element, called mediator, is in charge of instantiating and maintaining the back-ends. To better face high workload situations, it is possible to deploy several mediators, each of them managing a set of back-ends. Each front-end is linked to its corresponding back-end through a permanent bi-directional connection. It is important to note that it makes no difference at all, to application developers, whether an agent is deployed on a normal container or on the front-end of a split container, since both the available functionality and the APIs are exactly the same. This approach has a number of advantages:

– Part of the functionality of a container is delegated to the back-end and, as a consequence, the front-end becomes extremely lightweight in terms of required memory and processing power.

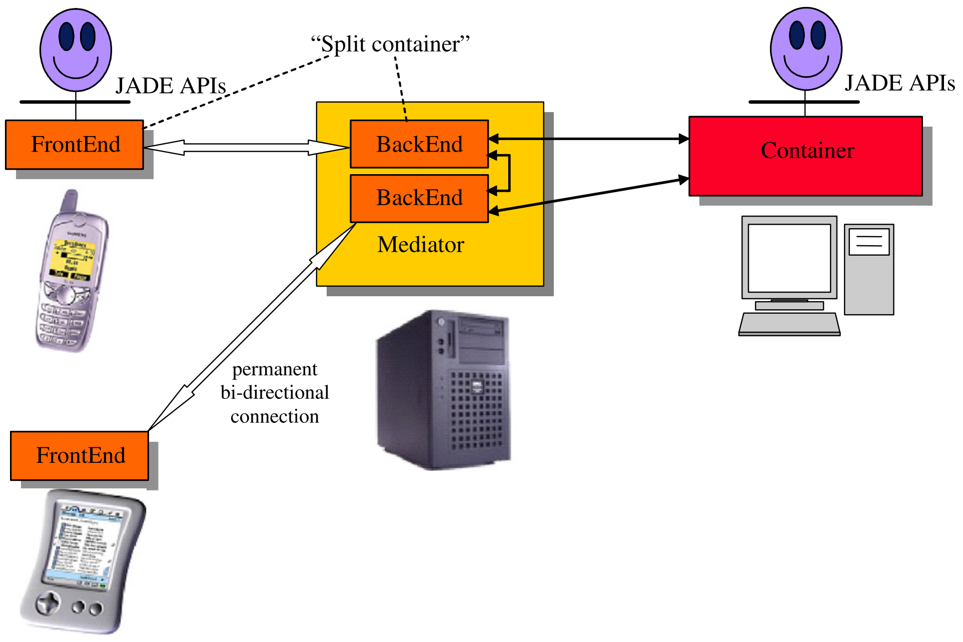

Fig. 3. JADE architecture for the mobile environment.

– The back-end masks to other containers the current IP address dynamically assigned to the wireless device, thereby hiding to the rest of the multi-agent system a possible change of IP address.
– The front-end is able to detect connection losses with the back-end (for instance due to an out of coverage condition) and re-establish the connection as soon as possible.
– Both the front-end and the back-end implement a store-and-forward mechanism: messages that cannot be transmitted due to a temporary disconnection are buffered and delivered as soon as the connection is re-established.
– A lot of information that containers exchange (for instance to retrieve the container where an agent is currently running) is handled only by the back-end. This approach, together with a bit-efficient message encoding between the front-end and the back-end, allows optimising the usage of the wireless link.

4. Integrating JADE agents with other technologies and tools

Nowadays, the success of every software middleware or framework strongly depends on the possibility of easy integration with other off-the-shelf technologies. JADE was designed to satisfy this requirement and, in its history, both the JADE team and third-parties provided different add-ons and extensions to integrate JADE with Web Services, Web servers, .NET applications, rule engines and reasoning tools, etc. In particular, this section focuses on the work done to integrate JADE agents with Web services and to encapsulate "intelligence" into JADE agents. These two are by no means the only aspects where JADE has shown its integration capabilities, but a complete exploration of the varied landscape is outside the scope of this paper and can only be achieved through an extensive literature survey (see [1] and [18]). We decided to select the Web Services integration and semantics-driven agent reasoning add-ons because they are in a sense symbols of the whole crossroad where the multi-agent systems approach happens to thrive, that is at the meeting point of interaction-centric advanced distributed systems and AI-inspired sophisticated information processing.

Nowadays, Web Services are probably the most widespread solution to develop Internet-accessible applications. Therefore, both the possibility of invoking agent services as Web services, and the capability to realize applications as composition of agents and Web services, are very important features for the usability of the JADE software. This functionality is achieved by means of a JADE add-on called *Web Service Integration Gateway* (*WSIG*) (see Chapter 10 of [1]). WSIG is able to automatically expose agent services as Web services and to convert SOAP invocations into ACL requests. More in details, JADE agents publish their services in a registry called Directory Facilitator, as defined by the FIPA Specification [14]. Each registered service is described via a data structure called Service-Description. This structure specifies, among others data, the ontologies that must be used to access the published service and that define the actions that the agent is able to perform. WSIG listens to registrations with the Directory Facilitator and, for each registered agent service, it automatically exposes a Web Service described by a WSDL description whose operations correspond to the actions supported by the registering agent. Fig. 4 depicts the mapping between agent service-description and WSDL. If properly configured, WSIG is also able to publish the exposed Web Service in a UDDI registry in order to simplify integration of a JADE-based system within a SOA environment. At invocation time, WSIG performs the following tasks: (1) it converts incoming SOAP messages into requests of execution of the corresponding actions, (2) it forwards these messages to the proper agents, (3) it handles action results, (4) and, finally, it sends back the responses to clients encoded as SOAP messages.

Pro-activity is claimed to be a distinguishing property of agent systems where, in simple words, each agent has its own goals and shows a behavioural freedom in the way it pursues these goals. Agents can show proactive behaviours only if they have some degree of intelligence that enables reasoning about goals, as well as planning and selecting the most appropriate actions to achieve their goals. However, according to the state of the art, usage of reasoning techniques represents still a small niche in the software programming community as it is well known that: (i) it is not yet proven that reasoning-based systems are adequately robust and effective for large scale deployment, (ii) reasoning about explicit representation of goals and other mental attitudes is costly in terms of processing time and memory, (iii) reasoning techniques increase the software complexity and require more skilled and expensive software programmers. On the other hand, just a few applications and a small subset of the agents of a system would require pro-activity. The design choice of JADE was to keep the agent abstraction simple, without requiring explicit representation of goals and mental attitudes, however different solutions were provided by the JADE user community and integrated as add-ons.

Drools4JADE (D4J) [3] integrates JADE agents with the Drools rule engine [13] and guarantees both the advantages of full rule-based agents (i.e., agents whose behaviour and/or knowledge is expressed by means of rules) and the advantages of rule-enhanced agents (i.e. agents whose behaviour is not normally expressed by means of rules, but that use a rule engine as additional component to perform specific reasoning, learning or knowledge acquisition tasks). In fact, D4J integrates the Drools rule-engine into a JADE agent in the form of a JADE behaviour. D4J implements an API to interact with Drools through ACL messages allowing both remote storing and retrieval of knowledge and the cooperation among different rule-based agents. Moreover, this API allows rules mobility, i.e. a rule-based agent can migrate a rule to another rule-based agent.

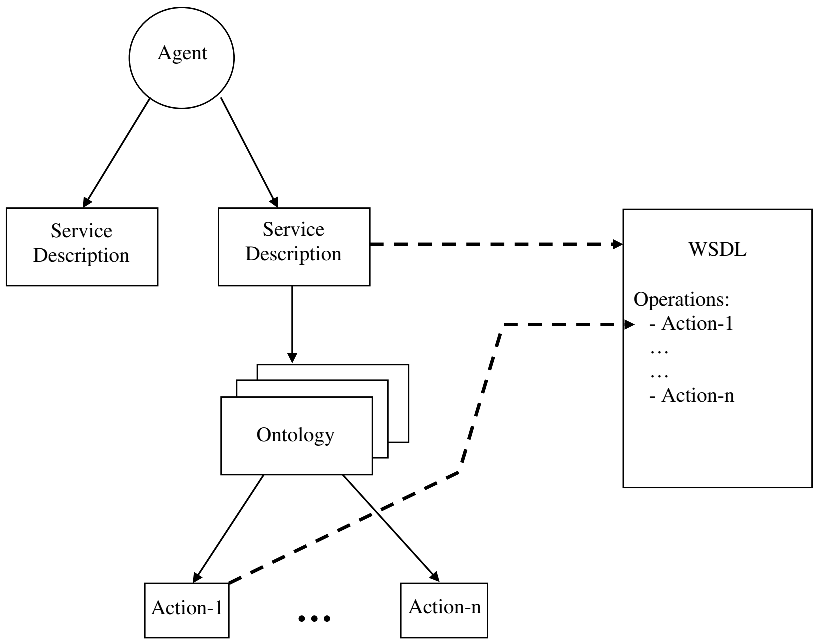

Fig. 4. Mapping between DF Service-Description and WSDL.

BDI [21] is a well-known model for cognitive intelligent agents where agents are specified in terms of three explicit mental attitudes: beliefs, desires, and intentions. The *JADE Semantics add-on* (JSA) (see Chapter 12 of [1]), developed by France Telecom, implements on top of JADE the BDI-like agent abstraction used for the formal specification of the FIPA ACL semantics. Agents can infer deductions from perceived events and they can accordingly modify agent's beliefs and behaviours; message receiving is just another type of event and message sending is just another type of action whose respective meaning is internally represented as FIPA-SL logical formulae and automatically interpreted by the JSA add-on according to the FIPA ACL formal semantics. For instance, without any additional line of code, a JSA agent is able to answer any query about a fact it was previously been informed of. With reference to Fig. 5, agent programming in JSA requires the following three steps: (i) implementing application-specific interpretation capabilities which infer deductions from events, (ii) implementing the application-specific agent beliefs which are facts to be asserted into the agent belief base, (iii) implementing the know-how of the agent which is composed of the ontological actions it is able to perform and their post-conditions and feasibility pre-conditions. The JSA add-on provides to programmers several points of extension, specialization, and customization in order to fit different software design requirements, such as specialization of the interpretation algorithm and of the belief handling mechanism.

5. Lessons learned

Software agent technology is still a niche sector of the software programming science, however, in this sector, JADE can arguably be considered the most popular software agent platform available today. Several companies have used and continue to use it in the context of very different application domains, including network

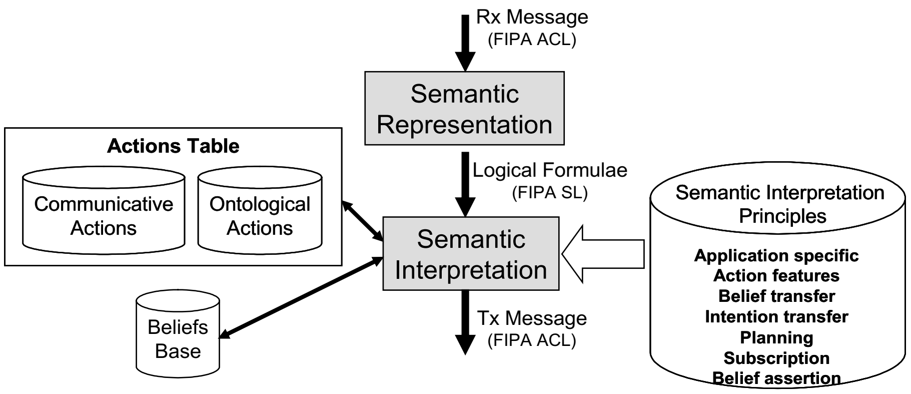

Fig. 5. Schematic dataflow of the JSA interpretation behaviour.

management, supply chain management, rescue management, fleet management, health care, auctions, tourism, etc. (see, for example, [8,17,19,25]). For instance, British Telecommunication's Intelligent Systems Research Centre uses JADE as the core platform of mPower [19], a multi-agent system that is used by BT engineers to support cooperation between mobile workers and team-based job management, i.e. teams rather than individuals receive job assignments and team members must cooperatively manage the assigned jobs. Several universities use JADE, exploiting its Open Source availability and its technical excellence, both to study and teach the strength of the software agent technology and to extend its functionalities with new add-ons (see, for example, Chapters 11 and 13 of [1]).

This section discusses, looking back, possible reasons for the diffusion of JADE, in particular those non-technical aspects which might represent useful lessons learned of general applicability, with the possibility to benefit other future projects.

5.1. Standard compliance and timing

JADE complies with the FIPA specifications [14] that enable end-to-end interoperability between agents of different agent platforms. All applications where inter-organization communication is needed can benefit from interoperability, including e.g. machine-to-machine and holonic manufacturing. As a matter of fact, the JADE project started with the goal of validating the FIPA specifications even if its functionalities soon went much far beyond the standard, even because of the slower evolution of the latter. Surely with some degree of happenstance, the project started at the right timing, and JADE and FIPA were able to realize in early 2000 a symbiosis where JADE validated FIPA specifications, provided new requirements, and often even offered concrete solutions to specification problems; FIPA, on the other hand, provided a sound theoretical foundation for the JADE technology and contributed to the spread of it. A technology becomes a standard after a consensus process between all participating organizations, often the standard status provides the energy for market acceptance of a new technology; as such, standard compliance represented a risk reduction factor and a facilitator to create a 'market of early adopters' of the technology.

5.2. Openness and license

JADE is distributed in Open Source under the LGPL license [20]. Looking back, the choice of this license contributed to JADE widespread acceptance because it assured all the basic rights and duties [20] to facilitate the usage of the software, included usage in commercial products. Unlike the GPL, the LGPL license does not put any restriction on *software that uses* JADE (as opposite to *software based upon* it), and allows proprietary software to be merged with JADE. This was again a risk reduction factor for users and contributed to the adoption of JADE in R&D projects, as well as in commercial projects, which had the chance to start using JADE even if the licensing strategy of the software they were going to produce was not yet defined. Of course, openness had to be properly managed in order to avoid anarchy and avoid appearance of multiple branches of the same software and different variations of the same API: a lot of effort was spent to promptly incorporate user feedbacks and integrate proposed functionalities and add-ons.

5.3. User support and documentation

It is a common pitfall to distribute software in Open Source without properly considering the need to support users and to provide them adequate documentation. Since its beginning, great care and relevant effort was put to answer to questions and requests of support from users. Furthermore, the code was distributed together with a programming guide and an administrator guide, a number of examples and tutorial, and the complete javadoc of all user APIs. Looking back, availability of good documentation and prompt user support should be considered since the beginning of an Open Source Project.

5.4. Dissemination strategy

Often the decision of a project to go Open Source represents *the* dissemination strategy. According to our experience, the decision of licensing open source does not represent per se a valid dissemination strategy; possibly, it does not represent at all a dissemination strategy that, instead, must be properly defined to support such a licensing decision through an adequate analysis of the target user base. In most cases, open source projects are the output of research projects and, as such, at their beginning, they target a niche market of early innovators that like to *play* with cool software based on new concepts; JADE targeted early adopters of the FIPA standard and the main dissemination channels were the standard organization itself and, under budget restrictions, scientific events, mainly through papers at conferences. It may look somewhat counterintuitive but, in our opinion, in the early stage of an innovative project, the emerging user community is better seeded with coolness and novelty rather than comprehensiveness and robustness; these two qualities are however needed to pass to the second phase of dissemination towards those companies that wish to leverage the technology to add value to their products. Notice that, even if this second phase is the phase of major concern on getting advantages and revenues, industry adoption of a novel technology is strongly facilitated by the presence of competitors that share the same goal of increasing the level of awareness of the technology. Considered that the main business of Telecom Italia is based on services rather then sale of software products, we decided to exploit the free open source nature of JADE in order to get contributions to improve and strengthen the software that we used, and we continue to use, as a middleware to develop Telecom Italia proprietary value added services. In this phase of dissemination we targeted mainly R&D projects, such as those supported by the European Commission; the JADE Web site [18] reports a partial list of EU projects that decided to use JADE and a list of main contributors.

5.5. Extensibility

The stereotypical JADE user is a software programmer with medium–high skills and strong attitude towards innovation, considered that he/she is using a niche technology. He/she is surely someone who likes to 'play' with software, likes to look inside the code and likes to have the possibility of extending it. A nice-to-have feature for this kind of user was surely the presence in JADE of well defined mechanisms for code extension in form of published APIs, such as the jade.mtp API which enables programmers to plug new message transport protocols. These extension points, on one hand made workable for multiple contributors to enhance the platform without creating conflicts, and on the other hand made the software more appealing to skilled programmers who could enjoy themselves by integrating and adding to JADE the results of their research. If extensibility was a strength of JADE, it must be said that a weakness of the project was instead the limits of the tools for collaborative software development: a lack of tools administrator and the presence of some company's policies for network restrictions (e.g. firewall) made collaborative development sometimes difficult as several processes (e.g. account management) had to be done manually instead of automatically. Considering the above described user stereotype, this was surely perceived as a weakness.

5.6. Java language

If today the use of the Java language can be considered quite a common choice, in 1998, when JADE development started, that choice required some considerations as an easier choice would have been the C language. Looking back, the choice of the Java language surely contributed to the spread acceptance of JADE as the majority of multi-agent systems software developers embraced and still continues to use this programming language.

5.7. Quick learning curve and perceived value

Users do not like dedicating too much time to learn using new software; when the software is free, they tend to dedicate even fewer time as they do not even have the financial motivation of protecting their purchase investment. JADE API proved easy enough to learn and use. It provides a lot of functionalities and was designed with orthogonality and separation of concerns in mind [2]: each set of API serves a specific purpose and is independent of other functionalities, therefore, programmers need to understand only those parts of the API that implement their required functionalities. Programming the "hello world" agent typically requires less than one hour and in this time the programmer can already experience some value from using the framework. JADE has been designed to simplify the management of communication and message transport by making transparent to the developer the management of the different communication layers used to send a message from an agent to another agent, and so allowing her/him to concentrate on the logic of the application. Of course, the effect of this feature is to simplify and speed up application development with respect to the time necessary to develop the same application by using only Java standard packages. In particular, when developing distributed applications for mobile terminals, JADE API and ready-to-use features allow strongly reducing the application development time and costs: some estimations have been given that indicate reduction of development time up to 30%.

5.8. Versatility

Several R&D projects, that analysed all available platforms before selecting JADE, considered versatility and JME availability one of the major strengths of JADE. As a matter of fact, JADE provides a homogeneous set of APIs that are independent from the underlying network and Java technology (JEE, JSE or JME). It was appreciated that application developers can reuse the same application code both for a PC, a PDA or a Java phone, and they can postpone this choice as late as possible, ideally until deployment time. JADE was JME-ready already at the beginning of year 2003 thanks to the involvement of some major mobile manufacturers like Motorola and Siemens within the framework of the LEAP European Project [4,5].

5.9. Backward compatibility

Regarding this aspect, two lessons were learned: few users like open source projects with frequent version releases, and all users hate introduction of backward incompatibilities. Seventeen release versions of JADE have been released in its 7-year-long life: at the beginning quite often, then every 6 months, now with a target of 1 new release per year. Great care was given to preserve backward compatibility, even when significant jumps in functionality were granted. Only three times the major version number had to be increased (from JADE 1.x to JADE 3.x) to notify users the introduction of backward incompatibilities which needed modifications to their software. These three times were though still remembered by many users even if guidelines were provided to upgrade user software. How backward compatibility can be preserved through release versions? The only suggestion we can provide is to carefully identify, reduce in number and in content, and have great care in designing, all public APIs, so that implementations as well as subsequent possible extensions do not impact the API itself. Remind that "Public APIs are forever and you have just one chance to get it right" [6]!

6. A sample application: the network neutral element manager

Telecom Italia adopted JADE as the reference framework for an important project in the field of network management [11]. The project involves about 30 people for 3 years with the goal of creating a mediation layer, identified as Network Neutral Element Manager (NNEM), to decouple Operations Support Systems (OSS) from the network equipments and functionalities, as depicted in Fig. 6. For telecom operators with a highly variegated multi-vendor network, as Telecom Italia is, this approach brings some important advantages. First of all, the provision of a uniform view over a diversity of technologies (IP, ATM...), network equipments (DSLAM, Router...), management protocols (SNMP, TL1...), vendors and software versions, sensibly reduces the cost of adaptation of OSS chains to changes in the network. For instance, when an equipment of a new vendor is introduced, the logic of interactions with its equipments does no longer need to be replicated in all OSS systems, but it is shared within the mediation layer. As a direct consequence, it is possible to accelerate the rollout of new technologies and services, which is a key success factor in a highly competitive market such as that of telecommunications. Furthermore, having a single system actually interacting with the network makes it possible to keep control over the accesses to network equipments thus avoiding management overload.

The NNEM deployment in Telecom Italia today covers the IP broadband access domain and serves the activation and troubleshooting processes. The plan is to progressively extend it to other domains and processes starting from the new generation network for ultra broadband access that supports rates up to 50 Mb/s. In order to achieve such rates, this new network is based on the Fiber-to-the-building architecture. Fiber cables are extended up to user buildings where proper network elements, called ONU and typically located in the building cellars, act as multiplexer between copper twisted pairs connecting flats and the optical fiber.

Today NNEM manages about 10,000 devices, with the introduction of the new generation network it will have to deal with about 100,000 devices and even more (one per building). As a consequence, scalability is a key requirement for the NNEM. Thanks to its distributed nature, JADE is particularly suited to create applications that run on distributed low cost architectures, such as blade, and therefore it perfectly meets the scalability requirement. Scalability was certainly the main reason that led to the usage of JADE as the basis for the NNEM, but other reasons were identified and had an impact on the final choice, such as:

– capability to serve well both reactive processes, such as activation and troubleshooting, and proactive processes, such as alarm handling;
– capability of executing long and fairly complex tasks such as network element configurations upload (considering response times and network latency this process can take up to one hour);
– capability of automatically exposing agent capabilities as Web Services by means of the WSIG add-on described in Section 4 and therefore simple integration with the above OSS systems.

After two years of intensive developments and testing in the scope of the NNEM project, the following considerations can be drawn.

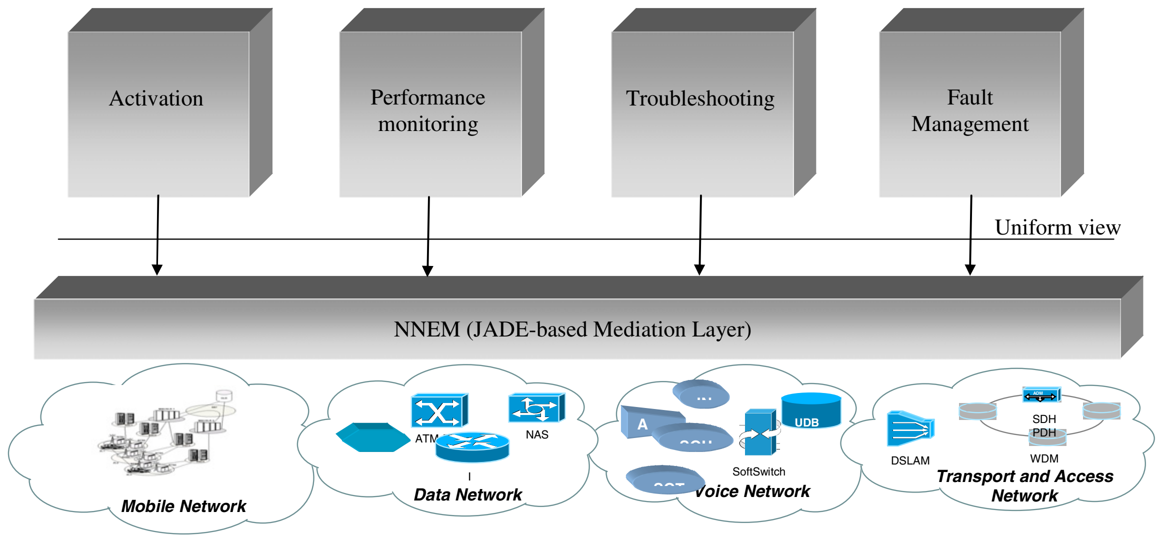

Fig. 6. The network neutral element manager.

6.1. Robustness

In general JADE is a robust software platform that enables the development of commercial-grade applications. In almost one year of deployment in the fields, JADE never stalled the NNEM system.

6.2. Distinguishing features

Thanks to its agent-based nature, JADE is particularly suited for applications that:

1. require the execution of possibly long and fairly complex tasks that can be triggered at any point in time and not just in response to user inputs;
2. involve several interactions between internal components, and especially one-to-many and many-to-many interactions;
3. require components to dynamically appear and disappear in the system and find each other at runtime.

These are certainly important criteria that should be taken into account when deciding whether or not to choose JADE as the basis for the development of a software application. Instead, when dealing with systems that mainly retrieve/store data from/to a data repository following user requests/inputs (as for instance all classical Web applications), usage of state of the art middleware technologies such as JEE application servers is certainly more indicated.

6.3. Component isolation

The agent oriented approach intrinsically facilitates the isolation of functional modules (i.e. agents) that can be implemented separately thus leading to clean design and easier organization of development teams.

6.4. Distribution

Being natively designed with distribution in mind, JADE allows seamlessly spreading components (i.e. agents) across different hosts thus guaranteeing a high degree of scalability. As the NNEM experience highlighted, however, distributed applications are more complex to run and manage. A further level of support is therefore required to hide as much as possible the complexity of the distribution to administrators. For this reason, an additional Platform Configuration and Control module was developed to support configuration, activation, monitoring and problem investigation from a single administration console. Though implemented within the scope of the NNEM project, this module was designed to be generic and domain independent to facilitate its future distribution in Open Source.

7. Conclusions

The paper presented JADE – a software framework to facilitate the development of agent applications – and described its software architecture and its provided functionalities as well as its integration with some technologies, in particular Web Service and reasoning tools and architectures. The paper discussed, looking back, possible reasons for the spread of JADE, in particular those non-technical aspects which might represent lessons learned of general usefulness also to other future projects.

JADE is a mature software development tool, a lot of people are working on its continuous maintenance, improvement and extension, and several commercial and industrial applications built on top of JADE exist in different application domains. The paper gave an overview of the Network Neutral Element Manager, a mission-critical system developed and deployed by Telecom Italia to decouple OSS systems from the network. Today NNEM manages about 10,000 devices, with the introduction of the new generation network it will have to deal with about 100,000 devices and even more (one per building).

The latest version of JADE is 3.5 and was released on June 2007. Future work is focusing on a platform configuration and control module, as mentioned in Section 6 of the paper, as well as on the development of a workflow-based distributed processing engine. This new engine will enable agents to execute application logics described via workflows, and to dynamically delegate to other agents pieces of workflows (i.e. sub-flows) according to criteria that are evaluated at runtime. Such criteria can be based for instance on the current agent workload (similar to a GRID computing environment) or to specific characteristics of the delegated agents. This processing engine, together with an integrated Service Creation Environment that supports the definition of processes in a user friendly graphical form, will form the core added value of the new version of JADE which is planned to be released at the beginning of 2008.

The reader might argue: if JADE is such a great platform, why haven't it made agent-oriented programming a standard technique within industry? JADE is a middleware that enables fast and reliable implementation of multi-agent distributed systems and which can be integrated with Artificial Intelligence (AI) tools; however, the conviction of the benefits of AI and agents is more consolidated in the scientific community than in the industrial one. The reasons for that are outside the scope of this paper and have been plentifully treated in several other fora. While not yet a mainstream approach in software engineering at large, JADE can arguably be considered the most popular software agent platform available today.

References

[1] F. Bellifemine, G. Caire, D. Greenwood, Developing multi-agent systems with JADE, Wiley Series in Agent Technology. ISBN 978-0-470-05747-6, February 2007.

[2] F. Bellifemine, A. Poggi, G. Rimassa, Developing multi agent systems with a FIPA-compliant agent framework, In Software - Practice & Experience 31 (2001) 103–128.

[3] A. Beneventi, A. Poggi, M. Tomaiuolo, P. Turci, Integrating rule and agent-based programming to realize complex systems, WSEAS Transactions on Information Science and Applications 1 (1) (2004) 422–427.

[4] F. Bergenti, A. Poggi, B. Burg, G. Caire, Deploying FIPA-compliant systems on handheld devices, IEEE Internet Computing 5 (4) (2001) 20–25.

[5] M. Berger et al., Porting distributed agent-middleware to small mobile devices, in: Workshop on Ubiquitous Agents, AAMAS 2002, Bologna, July 2002.

[6] J. Bloch, How to design a good API and why it matters. Available from: <http://lcsd05.cs.tamu.edu/slides/keynote.pdf>.

[7] R. Bordini, L. Braubach, M. Dastani, A. El Fallah Seghrouchni, J. Gomez-Sanz, J. Leite, G. O'Hare, A. Pokahr, A. Ricci, A survey of programming languages and platforms for multi-agent systems, Informatica 30 (2006) 33–44.

[8] M. Calisti, P. Funk, S. Biellman, T. Bugnon, A multi-agent system for organ transplant management, in: A. Moreno, J. Nealon (Eds.), Applications of Software Agent Technology in the HealthCare Domain, Birkhäuser, Basel, 2004, pp. 199–212.

[9] K. Chmiel, M. Gawinecki, P. Kaczmarek, M. Szymczak, M. Paprzycki, Efficiency of JADE agent platform, Scientific Programming 2 (2005) 159–172.

[10] D. Cowan, M. Griss, B. Burg, BlueJADE – a service for managing software agents, HP Technical Report 2001. see also <http://sourceforge.net/projects/bluejade>.

[11] G. Covino, G. Caire, Innovative NGN Management in Telecom Italia leveraging distributed agent technology, Tele Management Forum, IT & Operations Track. Nice, May 2007. Available from <http://www.tmforum.org/browse.aspx?catID=4217>.

[12] M. Dastani, M.B. van Riemsdijk, J.-J.C. Meyer, Programming multi-agent systems in 3APL, in: R.H. Bordini, M. Dastani, J. Dix, A. El Fallah Seghrouchni (Eds.), Multi-Agent Programming: Languages Platforms and Applications, Springer, 2005, pp. 39–67.

[13] Drools Web Site. Available from: <http://labs.jboss.com/jbossrules/docs>.

[14] FIPA Specifications Web Site. Available from: <http://www.fipa.org>.

[15] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns: Elements of Reusable Object Oriented Software, Addison Wesley, 1995.

[16] A. Helsinger, M. Thome, T. Wright, Cougaar: a scalable, distributed multi-agent architecture, in: Proceedings of the IEEE Systems, Man and Cybernetics Conference, Hague, The Netherlands, 2004, pp. 1910–1917.

[17] J. Hodík, P. Bečvář, M. Pěchouček, J. Vokřínek, J. Pospíšil, ExPlanTech and ExtraPlanT: multi-agent technology for production planning, simulation and extra-enterprise collaboration, International Journal of Computer Systems Science and Engineering 20 (2005) 357–367.

[18] JADE Web Site. Available from: <http://jade.tilab.com/>.

[19] H. Lee, P. Mihailescu, J. Shepherdson, Realising team-working in the field: an agent-based approach, IEEE Pervasive Computing, June 2007, pp. 85–92.

[20] LGPL license. Available from: <http://www.opensource.org/licenses/lgpl-license.php>.

[21] A.S. Rao, M. Georgeff, BDI agents: from theory to practice, in: Proceedings of the 1st International Conference on Multi-Agent Systems, San Francisco, CA, 1995, pp. 312–319.

[22] R. Ross, R. Collier, G. O'Hare, AF-APL: bridging principles and practices in agent oriented languages, in: R. Bordini, M. Dastani, J. Dix, A. El Fallah Seghrouchni (Eds.), Programming Multi-Agent Systems, 2nd Interantional Workshop (ProMAS'04), volume 3346 of LNCS, Springer Verlag, 2005, pp. 66–88.

[23] P. Vrba, E. Cortese, F. Quarta, G. Vitaglione, Scalability and performance of the JADE message transport system: analysis of suitability for Holonic manufacturing systems, Exp in Search of Innovation 3 (3) (2003) 52–65.

[24] M. Winikoff, JACK™ intelligent agents: an industrial strength platform, in: R.H. Bordini, M. Dastani, J. Dix, A. El Fallah Seghrouchni (Eds.), Multi-Agent Programming: Languages, Platforms and Applications, Springer, 2005, pp. 175–193.

[25] R. Zimmermann, S. Winkler, F. Bodendorf, Supply chain event management with software agents, in: S. Kirn, O. Herzog, P. Lockemann, O. Spaniol (Eds.), Multiagent Engineering – Theory and Applications in Enterprises, Springer, Berlin-Heidelberg, 2006, pp. 157–175.