JAAMAS manuscript No.
(will be inserted by the editor)

The First Twenty Years of Agent-Based Software Development with JADE

Federico Bergenti · Giovanni Caire · Stefania Monica · Agostino Poggi

Received: September 9, 2018 / Accepted: January 27, 2020

Abstract A recent survey provides convincing evidence that JADE is among the most widely used tools to develop agent-based software systems. It finds application in industrial settings and to support research, and it has been used to introduce students to software agents in various universities. This paper offers a perspective on the current state of JADE by first presenting a chronicle of the relevant events that contributed to make JADE what it is today. Then, this paper enumerates some of the abstractions that JADE helped to identify and that are now commonly adopted in the community of researchers and practitioners interested in software agents and agent-based software development. Such abstractions have been successfully applied to construct relevant software systems, and among them, this paper reports on a mission-critical system that uses the abstractions that JADE contributed to identify to serve millions of users every day. Finally, this paper discusses an outlook on the near future of JADE by sketching a recent project that could contribute to provide a new perspective on the use of JADE.

Keywords JADE · Agent-Oriented Software Engineering · Software Agents

F. Bergenti, S. Monica
Dipartimento di Scienze Matematiche, Fisiche e Informatiche
Università degli Studi di Parma
Parco Area delle Scienze 53/A, 43124 Parma, Italy
Tel.: +39-0521-{906929,906913}
E-mail: {federico.bergenti,stefania.monica}@unipr.it

G. Caire
TIM – Innovation
Via Reiss Romoli 274, 10148 Torino, Italy
Tel.: +39-011-2286107
E-mail: giovanni.caire@telecomitalia.it

A. Poggi
Dipartimento di Ingegneria e Architettura
Università degli Studi di Parma
Parco Area delle Scienze 181/A, 43124 Parma, Italy
Tel.: +39-0521-905728
E-mail: agostino.poggi@unipr.it

1 Introduction

The *Java Agent DEvelopment framework* (*JADE*) (e.g., [6]) consists of the well-known framework to build software agents together with some tools needed to effectively operate agent-based software systems. JADE has been offered to the open-source community of Java developers through its official Web site [44] for more than twenty years. A recent survey [45] provides convincing evidence that JADE is still among the most widely used tools to develop agent-based software systems in various contexts. The results of the survey clearly show that JADE has been intensively used in the academia to support research on software agents and to introduce students to *Agent-Oriented Software Engineering* (*AOSE*) [14]. In addition, JADE has been successfully used in industrial projects all over the world, as documented in another recent survey [52] that focuses on the practical applications of software agents and agent-based software development.

Besides its applications to the relevant contexts mentioned above, JADE contributed to shape the understanding of software agents and agent-based software development that the research community shares today. For example, JADE contributed, together with other platforms (e.g., [29,32]), to the idea, which is now commonly adopted to design agent-based software systems, that an agent platform is an ensemble of connected agent containers spread across a network.

This paper is intended to account for the role of JADE in the process of understanding software agents, agent-based software development, and their peculiar abstractions. The abstractions that JADE contributed to introduce emerged through a slow community process that is tightly connected with the historical development of the JADE project. This is the reason why, before discussing the abstractions that JADE contributed to identify, this paper provides in Section 2 a brief chronicle of the most relevant steps that led to the framework that is used today. It is by far not a complete chronicle, and it is definitely biased toward emphasizing the steps that contributed to evolve the framework, rather than toward enumerating its relevant uses. The description of how JADE contributed to significant projects is left for papers written by researchers and practitioners that actually worked on such projects. After the chronicle of relevant developments, this paper briefly summarizes, in Section 3, the abstractions that JADE contributed to identify and that are now core elements of the abstract view of software agents and agent-based software development that the framework advocates. Note that such an abstract view is accepted today by a significant research community also because it showed to be effective to use software agents in relevant applications. One of such applications is in the context of network and service management, and it is briefly described in Section 4. Such an application contributed to provide practical evidence that software agents, and JADE agents in particular, can be effectively used to deploy mission-critical systems designed to serve millions of users every day. Finally, the twenty years that elapsed from the initial prototypes to the current version of JADE witness significant changes in software engineering and related technologies, which need to be taken into serious consideration to ensure that JADE would have a prosper future. Such changes, and a possible way to take them into account, are briefly discussed in Section 5, where one of the latest projects that use JADE is succinctly described. Such a project is intended to offer a new way to use the framework, which is proposed to effectively relocate software agents in the software development landscape of today.

2 A Short Chronicle of the History of JADE

The two decades of the Java Agent Development Environment, almost immediately renamed to *Java Agent DEvelopment framework* (*JADE*), started in 1998 to support a relevant initiative of the time called *Foundation for Intelligent Physical Agents* (*FIPA*) [54]. FIPA was established in 1996 as an international non-profit association of companies, and since then it counted more than 60 members from 20 countries [60], which included companies, research centers, and universities, to produce interoperability specifications for software agents and agent-based software systems. Since its establishment, FIPA played a crucial role in the development of interoperability specifications for software agents, and it promoted a number of initiatives that contributed to the development and uptake of agent technologies. Interested readers should consult the official Web site of the foundation [35] to access current specifications and to review its interesting history [60].

The establishment of FIPA was promoted by its first president, Leonardo Chiariglione, who had already been considered a distinguished researcher at the time for his leading role in the *Moving Picture Experts Group* (*MPEG*) [51]. His prominent role in FIPA contributed to promote the foundation to important international players, and to consolidate the role of FIPA in the research community. He and colleagues envisioned that software agents could play a relevant role in the provision of next-generation multimedia services, and they also quickly understood that a set of accepted specifications to support interoperability among technologies from different vendors was necessary to promote the construction of open and dynamic services. FIPA initially received relevant attention, and it significantly contributed to attract fundings on an international scale.

The interoperability specifications that FIPA started to deliver in 1997 needed a reference implementation to ensure that technical decisions were, at least, not contradictory. In addition, several competing implementations of FIPA specifications quickly appeared [46], and periodic interoperability checks were considered necessary. At the time, Fabio Bellifemine was working for Telecom Italia [71], which was one of the five funding members of FIPA, and he was asked to contribute to some of the specifications that FIPA was about to deliver. He quickly became an active contributor to FIPA specifications, and he helped to spread the idea, also shared by other researchers (e.g., [61]), that FIPA necessitated a reference implementation to validate specifications. The scarcity of internal development effort, and the solid competences on software agents that the research group lead by Agostino Poggi at the University of Parma had already shown, contributed to make Telecom Italia decide to jointly develop an implementation of FIPA specifications with researchers working in Parma, namely, Giovanni Rimassa, Federico Bergenti, and others. The result of such a joint venture, which also received public funding (e.g., [1,58]), is the core of JADE. Even if JADE is now a worldwide community project, researchers working for Telecom Italia (now TIM) and for the University of Parma are still active in the project.

It is worth noting that, concurrently with the establishment of FIPA, also the *Object Management Group* (*OMG*) [55] launched an initiative to support the standardization of mobile agents and related technologies. The *Mobile Agent System Interoperability Facility* (*MASIF*) [47] was delivered by OMG to support interoperability among technologies for mobile agents, and the specifications delivered by FIPA and by OMG were quickly requested to become compatible (e.g., [43]).

The traditional separation between mobile agents and intelligent agents started to fade when the corresponding standards bodies aimed at compatible specifications, which ultimately generated relevant implementations like Grasshopper [5].

FIPA ceased to exist as a standalone foundation in 2005, when the Board of Directors opted for the opportunity to become an IEEE Computer Society Standards Committee. Since then, FIPA is better known as IEEE FIPA, but its mission statement firmly remains to deliver interoperability specifications to support the widespread adoption of software agents and agent-based software technologies.

JADE in its mature form was first described in a scientific journal in a dedicated article [8] published in 2000. The paper emphasizes that software agents can be effectively used as software components (e.g., [9,15]), and that JADE can provide the practical means to do it. Concurrently, other agent platforms started to be presented at scientific fora, and some of such platforms, namely, FIPA-OS [63], Zeus [53], and many others [46], significantly contributed to validate FIPA specifications and to suggest corrections and improvements. Besides the relevance of the mentioned platforms, and of some others (e.g., [26,45]), it is worth noting that JADE played a singular role in the development of FIPA specifications because of two projects co-funded by the European Commission, as follows.

JADE was chosen to experiment the deployment of software agents on mobile devices by three of the most important telephony manufacturers of the time, which were all prominent FIPA members. Motorola [50], Siemens [69], and Broadcom [30] teamed with Telecom Italia, ADAC [2], and the University of Parma to work on a project intended to experiment the deployment of software agents on innovative devices called *Java-enabled telephones*. The project, called *Lightweight and Extensible Agent Platform* (*LEAP*) [21,22], was co-funded by the European Commission to deliver a branch of JADE intended to work under the limitations of Java-enabled telephones, both in terms of available resources and of processing power. The branch was accurately designed to be reintegrated into the principal development line, and right after the end of the project JADE was equipped with a dedicated build procedure to deliver a downsized version of the framework for Java-enabled telephones. Even if Java-enabled telephones are no longer a target for the deployment of software agents, it is worth recalling the LEAP project for, at least, two reasons. First, it contributed to emphasize the role of software agents to support mobile applications and services. Second, the flexible build procedure designed in the scope of the LEAP project was later used to make JADE deployable on Android devices [12]. The porting of JADE to Android devices is no longer a matter of constrained resources or limited processing power, but it regards the support of the peculiarities of smart devices [62] like, for example, the possibility to interface onboard sensors [18,48,49] and the possibility to provide users with sophisticated interfaces for highly interactive applications [11].

The second project co-funded by the European Commission that contributed to spread the use of JADE was called *Agentcities.NET* (e.g., [59]). It was the first effort intended to create an open society of software agents by deploying a worldwide network of FIPA-compliant agent platforms. The objective of the project was to explore the deployment of software agents in an open environment, and to this end, it focused on the application of existing tools and technologies rather than on the development of new ones. By the end of the project, the Agentcities.NET network counted more than 160 registered agent containers [59], which were based on more than ten heterogeneous platforms that included JADE, FIPA-OS, and

Zeus. More than 75% of the active platforms [59] were based on JADE, and this fact marked an important result to quantify the diffusion of JADE among FIPA members. JADE started to be considered the *de facto* reference implementation of FIPA specifications [61], and the compliance with FIPA specifications was often measured in terms of the interoperability with JADE.

The community of researchers and practitioners interested in software agents and agent-based software development is still habitually using JADE, as witnessed by the two recent surveys [45, 52] mentioned in the previous section. JADE is generally considered a commodity to be used freely to support research and experimentation on agents and multi-agent systems. Many authors no longer cite papers on JADE when they acknowledge that they used it for their research, and some of them do not even feel the need to expand the acronym. Independently of the number of missing citations that the habitual use of JADE causes, its relevance for the research on software agents and agent-based software development can be measured in terms of the number of scientific papers that mention it explicitly. Scopus [66] counted on August, 5th 2019 more than 1,100 papers that mention the two words *JADE* and *agent* in their titles, abstracts, or keywords, in the Computer Science subject area from 2000. Actually, a close inspection of the results that Scopus produced shows that, in the last 10 years, more than 60 papers every year describe research that is somehow related with JADE. In addition, it is worth noting that when the query was restricted to consider the titles of the papers only, Scopus counted 140 papers, on the same day and for the same period, independently of the subject area. Such a modified query is relevant because it can be used also with Google Scholar [40] to have a wider coverage of considered sources. Google Scholar counted more than 300 papers that match the modified query, which suggests that the results obtained using Scopus underestimate the actual bibliographic impact of the research that uses JADE. A close inspection of the results obtained with Google Scholar shows that all obtained papers either discuss developments of JADE or present projects implemented with JADE.

3 Major Abstractions Contributed by JADE

This section briefly discusses the principal abstractions that JADE contributed to identify and that are now part of the common knowledge of the research community that works on software agents and agent-based software development. Note that the abstract view of software agents that JADE adopts is closely related to the descriptions of software agents and agent-based software systems found in FIPA specifications, mostly because of the tight link between JADE and FIPA discussed in the previous section. More specifically, a subset of the abstractions that JADE contributed to identify were generalized and included in the FIPA abstract architecture [37], while some of the abstractions in FIPA specifications, like, for example, *Agent Communication Language (ACL)* messages and interaction protocols, are implemented by JADE. In the remaining of this section, only the abstractions that were originally conceived in the design of JADE are discussed.

JADE agents are single-threaded software entities that execute in a runtime environment composed of (*agent*) *platforms* and (*agent*) *containers*, with each platform composed of at least one container. Each container executes on a host of the network, and therefore a platform is naturally distributed across multiple hosts. Each platform is managed by a special container called *main container*, and the other (*peripheral*) *containers* register with it to join the platform. The major duty of containers is to provide agents with runtime environments to enable their executions and interactions. Among the possible types of interactions in a multi-agent system (e.g., [24]), message passing is the form of interaction primarily enabled by containers. Independently of the platforms and the containers that host senders and receivers, agents send messages to other agents by explicitly addressing receivers using *Agent Identifiers* (*AIDs*), which are strings of characters that uniquely identify agents. The design of platforms as set of connected containers roots in JADE and it is borrowed from distributed software components. Distributed software components are normally hosted in runtime environments distributed across heterogeneous networks. Such runtime environments provide the essential functionality needed to manage the lifecycle of components, and they support message passing across networks. JADE containers are applications of such an approach to software agents.

A JADE agent performs transitions among several *lifecycle states* during its execution. Such transitions are managed by the container that hosts the agent, and each transition can be associated with an event handler that the agent executes just after the transition has actually occurred. For example, as soon as a container starts an agent, the agent enters the initialized state, which normally triggers an event handler associated with the transition. Initialized agents cannot communicate with other agents because they are not yet associated with a valid AID. Right after the initialization, and as soon as a valid AID becomes available, an agent enters the active state, which makes the agent ready to send messages to other agents. In order to process received messages, an agent enters the waiting state, which makes the agent start waiting for external events to occur. When an agent in the waiting state receives a messages, the agent promptly returns to the active state to react to the reception of the message. Finally, agents that are about to terminate their execution enter the deleted state, and their AIDs become invalid. Note that JADE supports other lifecycle states (e.g., [7]) besides the states mentioned above. The identification of the current set of lifecycle states was performed in multiple phases during the design of the framework to support managed deployment of agents across containers and to provide for agent mobility. In particular, the lifecycle states that support agent mobility were designed to couple the runtime model of JADE agents, as discussed below, with the specific form of weak mobility [38] that JADE implements.

The runtime model of JADE agents assumes that each agent engages one or more *behaviours* to perform relevant tasks. The behaviour is the abstraction that the framework introduced to manage the procedures that agents follow to perform tasks. JADE supports several types of behaviours. The most commonly used behaviours are one-shot behaviours, which are executed only once, and cyclic behaviours, which are repeatedly executed until they are explicitly deactivated. Behaviours are provided in terms of actions, and a single behaviour identifies at most one action. The *behaviour action* is the piece of code that the agent executes when the behaviour is activated, which normally occurs in reaction to external events. Note that JADE provides specific support in terms of behaviours for the important events that correspond to the reception of messages. A JADE agent can activate multiple behaviours, which are internally stored in a private queue. The (internal) behaviour scheduler associated with each agent uses such a queue to activate behaviours and to make the agent react to events. The peculiar scheduling of behaviours adopted by JADE is cooperative rather than preemptive, which ensures that the execution of a behaviour is never interrupted in favor of another behaviour. Such a specific form of scheduling was chosen during the early phases of the design of the framework to ensure that a single thread of control is sufficient to execute behaviours. In the view of agent-based software systems promoted by JADE, the interactions among agents are concurrent and distributed, but each agent is a single-threaded (centralized) entity. Actually, the fact that JADE agents are single-threaded entities is rooted, together with the behaviour abstraction, in the runtime model of agents as special forms of actors [3,23] that JADE advocates and contributed to introduce.

In the abstract view of agent-based software systems promoted by FIPA, interactions among agents are grounded on the exchange of ACL messages. Message passing is assumed to be independent of the hosts of the network where agents execute, and the delivery of messages is assumed to be transparent to agents. This is the reason why JADE is in charge of transparently handling the conveyance of messages across containers and platforms. The programmers that use JADE do not need to care about connection, addresses, and hosts because the communication among agents is location-independent, and the underlying communication channels are transparent. The ACL messages that JADE agents exchange comply with FIPA specifications, which ensures interoperability.

Following the approach that FIPA has been advocating since the early versions of its specifications, messages are associated with shared ontologies. FIPA does not mandate which types of ontologies should be used, or how agents should use ontologies to support reasoning. Ontologies for FIPA are just tags attached to messages, and each agent is in charge of using them properly. In order to offer to programmers a set of useful tools to effectively adopt ontologies, JADE restricted the supported ontologies to the so-called *communication ontologies* (e.g., [72]). Actually, JADE introduced a particular type of ontologies that describes conceptualizations in terms of schemas of concepts, actions, predicates, and propositions. Instances of such schemas are automatically serialized by JADE when a message is sent, and automatically deserialized when a message is received. Such a mechanism allows JADE agents to communicate using shared conceptualizations of the application domain. In detail, concepts are entities of ontologies composed of several properties, and they can be defined as extensions of other concepts. Not to be confused with behaviour actions, actions in ontologies are entities that denote the actions that agents can mention in messages. Just like concepts, actions have properties, and they can extend other actions. Predicates are used by agents to state logical expressions in messages. Predicates can extend other predicates, and they can have properties. Finally, propositions are atomic predicates that have no properties. Note that ontologies can be constructed by extending other ontologies, and JADE provides a set of basic ontologies that can be used as building blocks to construct domain-specific ontologies.

The support for ontologies that JADE introduced is rooted in the specific use of ontologies to enable communication. In the view of agent-based software systems that JADE contributed to identify, ontologies are not intended to drive reasoning, and they are only intended to precisely identify the structure of messages. Ontologies are the means used to support interoperability, and each JADE agent is free to choose whether to use them to drive automated reasoning.

4 Network and Service Management with JADE

The previous section briefly described the major abstractions that JADE contributed to the common knowledge of researchers and practitioners interested in software agents and agent-based software development. This section shows how JADE and such abstractions were used to implement a mission-critical system for a large company. The discussed case study is not the unique example of the use of JADE to develop mission-critical systems (e.g., [52]), but it is notable for the large scale of the application domain that it targets.

Telecom Italia, now better known as TIM, is one of the leading operators in the European telecommunications market, with more than 9 million broadband connections in Italy for retail and business clients. It has one of the most penetrating networks in Europe, with over 114 million kilometers of copper lines and over 5.7 million kilometers of optical fibers. Given the enormous business volume that such a network generates, the software systems in charge of ordinary business operations on the network have strict requirements in terms of scalability, robustness, and extensibility. The common opinion shared by researchers and practitioners interested in network and service management is that the traditional approaches are not sufficient when networks are so complex. After more than a decade of daily use, practical evidence suggests that the agent-based software system discussed in this section is a viable alternative to traditional approaches to network and service management. The presentation of the system reported in this section is by far not complete, and it is mainly intended to discuss the role of JADE, and of the abstractions that it contributed to identify, in the implementation of the system. Interested readers should consult a dedicated paper [13] for further details on the roles played by agents in the system, on the hierarchical architecture of the multi-agent system, and on the connections with the underlying network and with other systems. It is worth noting that another system [39] based on an approach to network and service management closely related to the approach used to design the system described in this section was experimented by Telefónica [70].

Traditionally, network and service management is performed by means of two types of dedicated software systems, which are normally called *Operations Support Systems* (*OSSs*) and *Business Support Systems* (*BSSs*) [42]. An OSS is a system that supports the back-office activities that operate a network and that are needed for the provision of services to clients. Complementary, a BSS is a system that supports activities directly related to clients, like billing and order management. A BSS typically interfaces an OSS for some of its activities, such as trouble ticketing and service assurance. In the traditional approaches to network and service management, the relationships between OSSs and BSSs are rigid, and they are often called *Orders Down, Faults Up* (*ODFU*) because orders are passed from BSSs to associated OSSs, and faults take the opposite direction, from OSSs to associated BSSs. The interfaces among OSSs and BSSs are involved in many operations, and the rigidity of the ODFU approach is generally perceived as a serious limitation when the size of the managed network is large.

In order to provide more flexibility and a higher level of robustness, Telecom Italia left the traditional approach based on OSSs and BSSs, and it internally developed a system called *Workflows and ageNTS* (*WANTS*) [13]. WANTS is a network and service management system that uses software agents programmed in terms of hierarchical workflows to ensure improved flexibility and robustness.

Among its major functionality, WANTS manages a model inventory that stores the executable descriptions of the business processes that can be deployed to agents in terms of workflows. The model inventory also contains the descriptions of the actual network resources, and WANTS automatically synchronizes the model inventory with actual resources, so that changes in the network are immediately propagated to agents to possibly change active workflows. Finally, WANTS adapts to the fluctuations of the network load because it uses real-time traffic observations to promptly react to interesting events by means of adaptive management techniques for resource allocation. At the time of writing, WANTS manages the part of the broadband network called access network. The access network is the part of the network that reaches clients' premises, and it includes the lines to the over 4 million modems used by clients. The lines in the access network generate an average of 300 events per second, and WANTS gathers and monitors them to enable real-time control over the quality of service provided to clients.

WANTS is an agent-based software system entirely implemented using the *Workflows and Agents Development Environment* (*WADE*) [31], which is nothing but JADE enriched with workflow engines embedded in agents together with platform-level functionality related to fault tolerance and scalability. Note that WADE agents are JADE agents whose behaviours can be expressed either by explicitly writing Java code or in terms of hierarchical workflows. Also note that WADE is incremental with respect to JADE because it adds features to support fault tolerance and scalability, but it does not remove or alter other features.

The architecture of the multi-agent system that WANTS implements is based on a set of roles played by agents [13], as schematically shown in Fig. 1 for a simple illustrative network. *Resource Proxy* (*RP*) agents are responsible for creating, maintaining, and managing the images of the resources deployed in the network. The *Agent Applications* (*AAs*) agents implement the so called *Fault, Configuration, Accounting, Performance, and Security* (*FCAPS*) functionality [42]. AAs work together by means of dedicated interaction protocols to implement the cooperative and distributed execution of business processes related to network and service management. In particular, AAs are responsible for local performance monitoring, and they are in charge of sending messages to interested agents about the actual load of the network. AAs communicate with RPs to interface network resources. Finally, *Manager Application* (*MA*) agents manage the distribution of workflows to other agents, and they manage the distribution of the descriptions of the actual network resources. In addition, MAs monitor the state of the network using the information provided by AAs, and they interact with external legacy systems. Note that the agents in the architecture do not require software updates to support new services or new types of network resources because the business processes that they execute are described in terms of hot deployable agent behaviours expressed as WADE workflows.

The decade that have passed since the introduction of WANTS generated interesting methodological and practical outcomes [13]. The successful adoption of WANTS to manage part of one of the most complex networks in Europe provided practical evidence that agent-based software systems are a viable alternative to mainstream technologies for this type of systems, especially for large networks, in which the rigidity of the ODFU approach is generally perceived as a limitation. Actually, WANTS is normally regarded as a best in class system that can compete with alternatives from major software vendors in terms of functionality, maturity,

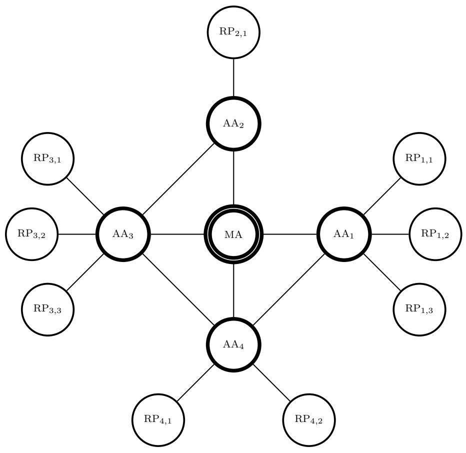

Fig. 1 The high-level view of the multi-agent architecture implemented by WANTS for an illustrative (very simple) system that comprises one MA, four AAs, and nine RPs.

and return on investment. In addition, WANTS is one of the rare cases in which a large service company changed its ordinary *buy strategy* to the more risky and ambitious *make-open-source strategy*. The major reasons for the change in the case of WANTS can be enumerated as follows. First, the network that WANTS manages has many peculiarities caused by its protracted stratification, and solutions designed for networks with less stratification are not sufficient. Second, the network is still considered the most specific part of the core business of the company, and therefore the knowledge of the network must be securely kept within the boundaries of the company.

The major reasons for the successful development of WANTS can be briefly summarized as follows [13]. The developers that worked on WANTS could bring in the project their advanced knowledge of the most recent trends in network and service management. The system was developed using medium-term plans, which took advantage of the fact that WANTS is based on JADE, and it can count on the large community of JADE users. The design of the system could be based on the systematic use of open-source components, and only the components that could list a significant number of successful applications in industrial settings were chosen. Actually, the development effort needed to implement WANTS, measured in terms of the fraction of significant components that were developed specifically for the system, is less than 30%.

Finally, the adopted make-open-source strategy is considered by some of the researchers that worked on the development of WANTS as one of the causes that contributed to the effective implementation of the system. Developers that worked on the project could improve their personal skills on cutting-edge technologies, and this opportunity generated appreciation for WANTS for medium-term career plans. In addition, managers could make sure that the competencies and best practices regarding network and service management were kept inside the boundaries of the company, which was set as a strict requirement for a project intended to deliver the software system in charge of the daily management of one of the most valuable assets of the company.

5 Toward Next Decades: JADE without Java?

The common practice of using JADE for the construction of agent-based software systems ranging from simple didactic examples to mission-critical applications, like the one described in the previous section and many others (e.g., [52]), provides evidence that JADE is a valuable tool with wide applicability. Unfortunately, informal observations of students during artificial intelligence and software engineering classes at the University of Parma in the last ten years suggest that approaching agent-based software development with JADE is often perceived as a difficult task. The reasons for such a perception are multiple, but the following are two of the most relevant. First, the continuous growth of the framework in the last twenty years has been increasing its inherent complexity, and the number of details that programmers are demanded to master has equally grown. The support offered by the community in terms of documentation, examples, and participation in the mailing list has been constantly relevant since the early days of JADE, but the opinion that students currently share is that JADE is a difficult framework to approach if no specific background on software agents is available. Second, the choice of Java as the unique programming language for the construction of JADE agents is now considered inappropriate in many situations, especially when a native support of the abstractions of *Agent-Oriented Programming* (*AOP*) [67,68] would be valuable. Note that a first attempt to detach JADE from Java was tried with WADE and its workflows. WADE workflows are diagrammatic views of JADE behaviours, and they can be used to program agents instead of using Java. Essentially, they provide abstract views of JADE behaviours in terms of flowcharts that programmers can easily produce using a dedicated tool [31] integrated with Eclipse [33]. Unfortunately, the limited expressive power of the diagrammatic notation that WADE uses often forces programmers to embed significant pieces of Java code into workflows, even to make agents perform simple tasks.

In order to seriously address both mentioned issues, which have the ultimate effect of limiting the adoption of JADE for new projects, a new programming language called *Jadescript* [19,20,56,57] has been recently proposed to provide support for the construction of JADE agents without using Java. Jadescript is an AOP language designed to offer to programmers a rich set of agent-oriented abstractions, most of which were originally introduced by JADE, and to reduce the complexity of using such abstractions by means of a dedicated syntax. The main goal addressed with the introduction of Jadescript is to give to programmers the possibility of effectively using agents without major methodological changes with respect to the direct use of JADE. In particular, Jadescript provides support for the major abstractions that JADE provides, and the addition of other abstractions, like the ones provided, for example, by Jadex [28], is left for future works.

Even if the literature counts a number of proposals for various AOP languages (e.g., [27,36,41,64,65] and [4,26]), Jadescript is somehow singular because it attempts to couple the original understanding of AOP (e.g. [67,68]) with the experience gained during the last twenty years of development and use of JADE. In particular, Jadescript capitalizes the lesson learned from JADE by providing programmers with the same abstractions that characterize JADE. Jadescript isolates such abstractions and equips them with a dedicated syntax to ensure that they can be used effectively. Note that the goal that motivates the introduction of Jadescript is perfectly coherent with the original understanding of AOP. Actually, AOP is equally related to the abstractions that programmers use for the construction of agents, and to the concrete syntax, and underlying semantics, that programmers adopt to manage such abstractions. The fact that Jadescript provides a dedicated syntax to easily adopt the principal abstractions that characterize JADE is expected to simplify the correct and effective use of such abstractions.

Jadescript follows the path traced by its predecessor, a domain-specific language for AOP called *JADEL* (e.g., [10,16,17]), to carve the link between JADE and Java. However, Jadescript is significantly different from JADEL because all recollections of object-oriented programming still present in JADEL have been completely removed from Jadescript. The procedural parts of JADEL agents are written using an enriched version of Xtend [25], which is an extensible dialect of Java tightly coupled with the Xtext infrastructure [34]. The choice of adopting a dialect of Java for the procedural parts of the language had the effect of bringing object-oriented abstractions into the core of JADEL, which ultimately contributed to suggest frequent misuses. Jadescript originates from the choice of removing all references to Xtend from JADEL, and to replace them with syntactic constructs explicitly designed for the purpose. The result is that Jadescript shares some characteristics with popular scripting languages, like semantically-relevant indentation and type inference. Note that the current implementation of Jadescript tools, which include a compiler that generates Java classes and a syntax-highlighting editor for Eclipse, still depends on Xtext, but the language is completely independent of it.

The most relevant abstractions that Jadescript borrowed from JADE are presented in the remaining of this section using a simple illustrative example. The chosen example is a variation of the music shop example, which is well-known to JADE programmers because its implementation is analyzed in one of the most popular tutorials on JADE available from the Web site of the project [44]. Interested readers should consult the mentioned tutorial to compare the JADE implementation described there with the Jadescript implementation described in this section. Even a superficial comparison between the two implementations reveals that Jadescript ensures much higher readability, and therefore maintainability, because programmers are provided with dedicated constructs to use the major abstractions that JADE offers.

The considered example implements a multi-agent system that contains a group of seller agents and a disjoint group of buyer agents. Each seller agent manages a music shop that sells two types of items, books and DVDs, to interested buyer agents. The application domain is intentionally minimalistic, and all items are described in terms of a unique title and a price. Each DVD is also associated with a list of tracks, each of which has a unique title and a duration. The considered scenario starts when a buyer agent decides to buy an item, and therefore the seller agent that offers the lowest price for the chosen item must be identified. In order to obtain the lowest price, the buyer agent uses a simple strategy: it sends requests to all seller agents, and selects the seller agent that returns the best offer. In detail, the buyer agent sends call-for-proposal messages to all seller agents to ask them for the price of the desired item. Seller agents quote the requested item, if available at their shops, and they send a propose message back to the buyer agent. If the requested item is not available at the shop managed by a seller agent, the seller agent returns a refuse message. If no seller agent returned a valid offer, the buyer agent drops the intention to buy the item. In all other cases, one of the seller agents that returned the best offer is chosen and the buyer agent completes the transaction.

Listing 1 Declaration of the ontology used to implement the music shop example in Jadescript.

```
1  ontology MusicShop
2    concept item(title as text, price as double)
3    concept track(duration as double)
4    concept book extends item
5    concept DVD(tracks as list of track) extends item
6
7    predicate owns(title as text)
8    predicate quote(buyer as text, good as item)
9
10   action sell(buyer as text, title as text)
```

The described scenario is nothing but a FIPA contract-net protocol [37], but this fact is not used both in the discussed Jadescript implementation and in the original JADE implementation. Note that the discussed Jadescript implementation is not intended to be interoperable with the corresponding JADE implementation. The purpose of the presented Jadescript implementation is to exemplify the use of the language by means of a didactic example that shows some of the features that Jadescript offers to programmers.

As discussed in Section 3, the peculiar type of ontologies that JADE supports is among the most relevant abstractions that JADE contributed to introduce. The adopted type of ontologies is considered adequate to support communication among agents, even if it is not sufficiently expressive to support reasoning in many contexts. The ontologies that JADE contributed to introduce are used by agents to share descriptions of concepts, actions, predicates, and propositions. Jadescript adopts the same type of ontologies, and Listing 1 shows the declaration of the ontology used to implement the music shop example. Note that Listing 1 also shows some of the characteristics of the language. First, the use of semantically-relevant indentation, which makes Jadescript follow a consolidated practice of modern scripting languages. Second, the use of the keyword `as` to statically associate a data type with a name (e.g., line 2). Third, the use of statically typed expressions. Jadescript provides a rich set of primitive data types, which includes `text` and `double` (e.g., line 2), and it also supports structured data types in terms of lists and maps, and of domain-specific concepts, actions, predicates, and propositions declared in ontologies (e.g., line 5).

Behaviours were introduced in the design of JADE to describe how agents operate during their lifecycle, and to specify how agents should react to events. In Jadescript, behaviours are built on top of JADE behaviours, and they are characterized by the peculiar scheduling mechanisms of JADE behaviours. A minimal behaviour is declared in Jadescript by stating its name and its type, which can be `cyclic` or `one-shot`. More complex types of behaviours, as supported by JADE, are planned for future versions of the language. In Jadescript, the declaration of a behaviour can restrict the agents that can adopt the behaviour by means of the construct `for-agent`, and it can link the behaviour to an ontology with the construct `uses-ontology`. The declaration of a behaviour is completed with a list of features to be used to actually implement the behaviour. Such features are properties, functions, procedures, actions, and (event) handlers.

Listing 2 Declaration of a behaviour used by seller agents to implement the music shop example in Jadescript.

```
1  cyclic behaviour WaitForRequests for agent Seller uses
2    ontology MusicShop
3    on message m when performative of m is cfp and
4      content of m is owns do
5      content = content of m as owns
6
7      buyer = sender of m
8
9      catalogue = catalogue of agent
10
11     title = title of content
12
13     if catalogue contains title do
14       good = catalogue[title]
15
16       send propose quote(buyer, good) to buyer
17     else do
18       send refuse sell(buyer, title) to buyer
```

Listing 2 shows a behaviour used to implement the music shop example. The `WaitForRequests` behaviour is scheduled by seller agents to wait for incoming call-for-proposal messages. When one of such messages is received, the behaviour is used to check if the requested item is available at the shop and to reply accordingly. Note that the behaviour uses an event handler to execute a sequence of statements upon the reception of a message. The handler specifies a condition that is used to filter appropriate messages (lines 3 and 4). The condition refers to the properties of the received message expressed using the elements of the referenced ontology, which is declared with the `uses-ontology` construct (lines 1 and 2). The two parts of the condition shown in Listing 2 use the `of` operator to access the elements of a structured value, and the `is` operator to check if the type of the left-hand operand is compatible with the type used as right-hand operand. When a message satisfies the condition, the behaviour extracts the content of the message to react appropriately. The content of the message is extracted using the `=` operator, the `of` operator, and the `as` operator (line 5). Jadescript provides a single statement to declare variables and to perform assignments using the common `=` operator. In Jadescript, the types of variables are inferred by the compiler from mandatory initialization expressions, and there is no need to make them explicit. Note that it is always possible to make data types explicit by means of the `as` operator.

For all processed messages, the `WaitForRequests` behaviour uses the list of items available at the shop, called `catalogue` in the example (line 9), to respond appropriately. The `catalogue` is a property of the agent that is currently executing the behaviour, whose type is specified using the `for-agent` construct (line 1). Note that the keyword `agent` (line 9) can be used in a behaviour to refer to the agent that is currently executing the behaviour. Besides mentioned primitive and structured data types, behaviour types and agent types are provided by Jadescript for the manipulation of behaviours and agents, respectively. The `catalogue` is a map that contains values that represent the items available at the shop, which are described using the elements of the ontology referenced by the behaviour. Listing 2 shows

Listing 3 Declaration of the buyer agents used to implement the music shop example in Jadescript.

```
1  agent Buyer uses ontology MusicShop
2    on create with arguments as list of text do
3      title = arguments[0]
4
5      create shops as list of text
6
7      skip = true
8
9      for seller in arguments do
10       if skip do
11         skip = false
12       else do
13         add seller to shops
14
15     activate behaviour SendCFPs with shops = shops,
16       title = title
17
18     activate behaviour WaitForReplies
```

some examples of the features that Jadescript offers to manipulate lists and maps. Ordinary square brackets are available to access the elements of lists and maps (line 14). When applied to lists, they require a nonnegative integer to be used as index. When applied to maps, they require a value compatible with the type of the keys to access the corresponding value in the map. Note that square brackets can also be used at the left-hand side of the `=` operator to modify lists and maps. In addition, the `contains` operator (line 13) is provided to check if the collection used as left-hand operand contains the value used as right-hand operand. The `contains` operator works on the elements of lists and on the keys of maps.

Jadescript provides the classic `if-else` statement to express conditional sequences of statements (lines 13–18) that can be nested in multiple branches. In the example shown in Listing 2, the `if-else` statement is used to choose between two `send` statements intended to send appropriate replies (lines 16 and 18). The `send` statement is used in Jadescript to create a message and send it. Only the performative, the list of receivers, the content, and, implicitly, the ontology can be specified in the `send` statement.

Finally, Jadescript allows declaring agents, which are essentially JADE agents, in terms of the following features: properties, procedures, functions, and (event) handlers. Listing 3 shows the Jadescript implementation of the buyer agents of the music shop example. The actions that buyer agents perform to implement the example are declared in the referenced behaviours `SendCFPs` (line 15) and `WaitForReplies` (line 18). The names of such behaviours are evocative of their roles in the example. The shown `on-create` handler (lines 2–18) has a parameter called `arguments` that is associated with the command line arguments passed to the agent upon creation. The first of such arguments is the title of the item that the user intends to buy (line 3). The remaining arguments are the AIDs of seller agents, which are collected in the `shops` list. Jadescript provides the `create` statement to create instances of structured data types (line 5) and, if needed, a list of named arguments can be provided after the optional keyword `with`.

Jadescript supports iteration over lists and maps with the `for-in-do` statement (lines 9–13). At each iteration, an element from the collection that follows the keyword `in` is extracted and, before the body of the statement is executed, the element is assigned to the variable whose name is declared after the keyword `for`. A set of statements is provided by Jadescript to work on collections, and, in particular, the `add-to` statement can be used to modify lists (line 13). Finally, Jadescript provides the `activate-behaviour` statement to instantiate and activate behaviours (lines 15 and 18). Such a statement can be used in the scope of the actions of agents and behaviours to instantiate a behaviour and to activate it. Some behaviours need a set of arguments to be properly initialized, and such arguments can be provided by means of the optional keyword `with` (line 15 and 16).

6 Conclusions

This paper presented a perspective on the first twenty years of agent-based software development with JADE. The introductory chronicle on the history of JADE was meant to highlight the major steps of the development of the framework and to clarify the relationships between JADE and other projects and initiatives. JADE was considered the *de facto* reference implementation of FIPA specifications, which were intended to support interoperability and to bring the concept of open multi-agent system in the landscape of the research on software agents and agent-based software development. As such, JADE had some influence on the applied research on software agent and agent-based software development, and it is worth noting that, after more than twenty years, scientific papers still mention the use of JADE for prototyping and to support experiments.

JADE is important also because it contributed a set of abstractions to the research on software agents and agent-based software development that are now part of the common knowledge of the community. For example, the design of agent platforms in terms of connected containers, which JADE contributed to promote, is now part of the common nomenclature of agent-based software development, and terms like main container, peripheral container, and behaviour are known to researchers, practitioners, and students.

The two decades of JADE witness its successful adoption in relevant application domains. Among the various examples documented in the literature, the use of JADE to support the management of a large telecommunication network can be considered as a notable success, and it can also be considered as a relevant success for the research on software agents and agent-based software development. The successful adoption of JADE in such a high-profile domain witnesses that the technology delivered by the research on software agents and agent-based software development is mature, and that it is ready for mainstream adoption in synergy with other stable technologies.

In order to relocate JADE in the contemporary landscape of software technologies, a new way to deal with its characteristic abstractions was recently proposed. We claim that the introduction of a dedicated language, characterized by expressivity and readability, is the first step to provide programmers with a high-level way to construct software agents and agent-based software systems, and we believe that such an higher level of abstraction is presumably needed to ensure that JADE would be ready for the challenges of the next decades.

Acknowledgements Authors would like to gratefully thank the large and lively community of researchers, practitioners, students, and enthusiasts that contributed to the development and the application of JADE during its first twenty years. Authors would like to fondly and admiringly dedicate this paper to the memory of Fabio Bellifemine for his prominent contributions to the area of software agents and agent-based software development, and for his propulsive role in the conception and in the launch of JADE.

References

1. van Aart, C., Caire, G., Pels, R., Bergenti, F.: Creating and using ontologies in agent communication. In: S. Cranefield, T. Finin, S. Willmott (eds.) Proceedings of the 2002 Workshop on Ontologies in Agent Systems (OAS 2002), *CEUR Workshop Proceedings*, vol. 66. RWTH Aachen (2002)
2. ADAC Web site: `www.adac.de`
3. Agha, G., Mason, I., Smith, S., Talcott, C.: A foundation for actor computation. Journal of Functional Programming **7**(1), 1–72 (1997)
4. Bădică, C., Budimac, Z., Burkhard, H.D., Ivanovic, M.: Software agents: Languages, tools, platforms. Computer Science and Information Systems **8**(2), 255–298 (2011)
5. Bäumer, C., Magedanz, T.: Grasshopper–A mobile agent platform for active telecommunication. In: Proceedings of the 3rd International Workshop on Intelligent Agents for Telecommunication Applications (IATA 1999), pp. 19–32. Springer-Verlag (1999)
6. Bellifemine, F., Bergenti, F., Caire, G., Poggi, A.: JADE–A Java agent development framework. In: R.H. Bordini, M. Dastani, J. Dix, A. El Fallah Seghrouchni (eds.) Multi-Agent Programming: Languages, Platforms and Applications, pp. 125–147. Springer International Publishing (2005)
7. Bellifemine, F., Caire, G., Greenwood, D.: Developing multi-agent systems with JADE. Wiley Series in Agent Technology. John Wiley & Sons (2007)
8. Bellifemine, F., Poggi, A., Rimassa, G.: Developing multi-agent systems with a FIPA-compliant agent framework. Software: Practice and Experience **31**, 103–128 (2001)
9. Bergenti, F.: A discussion of two major benefits of using agents in software development. In: P. Petta, R. Tolksdorf, F. Zambonelli (eds.) Engineering Societies in the Agents World III, *Lecture Notes in Artificial Intelligence*, vol. 2577, pp. 1–12. Springer International Publishing (2003)
10. Bergenti, F.: An introduction to the JADEL programming language. In: Proceeding of the 26th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2014), pp. 974–978. IEEE (2014)
11. Bergenti, F., Caire, G., Gotta, D.: An overview of the AMUSE social gaming platform. In: Proceedings of the 14th Workshop "From Objects to Agents" (WOA 2013), *CEUR Workshop Proceedings*, vol. 1099. RWTH Aachen (2013)
12. Bergenti, F., Caire, G., Gotta, D.: Agents on the move: JADE for Android devices. In: Proceedings of the 15th Workshop "From Objects to Agents" (WOA 2014), *CEUR Workshop Proceedings*, vol. 1260. RWTH Aachen (2014)
13. Bergenti, F., Caire, G., Gotta, D.: Large-scale network and service management with WANTS. In: P. Leitão, S. Karnouskos (eds.) Industrial Agents: Emerging Applications of Software Agents in Industry, pp. 231–246. Elsevier (2015)
14. Bergenti, F., Gleizes, M.P., Zambonelli, F. (eds.): Methodologies and Software Engineering for Agent Systems: The Agent-Oriented Software Engineering Handbook, *Multiagent Systems, Artificial Societies, and Simulated Organizations*, vol. 11. Springer International Publishing (2004)
15. Bergenti, F., Huhns, M.N.: On the use of agents as components of software systems. In: F. Bergenti, M.P. Gleizes, F. Zambonelli (eds.) Methodologies and Software Engineering for Agent Systems: The Agent-Oriented Software Engineering Handbook, *Multiagent Systems, Artificial Societies, and Simulated Organizations*, vol. 11, pp. 19–31. Springer International Publishing (2004)
16. Bergenti, F., Iotti, E., Monica, S., Poggi, A.: Interaction protocols in the JADEL programming language. In: Proceeding of the 6th International Workshop "Programming Based on Actors, Agents, and Decentralized Control" (AGERE 2016) at ACM SIGPLAN Conference "Systems, Programming, Languages and Applications: Software for Humanity" (SPLASH 2016), pp. 11–20. ACM (2016)

17. Bergenti, F., Iotti, E., Monica, S., Poggi, A.: Agent-oriented model-driven development for JADE with the JADEL programming language. Computer Languages, Systems & Structures **50**, 142–158 (2017)
18. Bergenti, F., Monica, S.: Location-aware social gaming with AMUSE. In: Y. Demazeau, T. Ito, J. Bajo, M.J. Escalona (eds.) Advances in Practical Applications of Scalable Multi-agent Systems, pp. 36–47. Springer International Publishing (2016)
19. Bergenti, F., Monica, S., Petrosino, G.: A scripting language for practical agent-oriented programming. In: Proceeding of the 8th International Workshop "Programming Based on Actors, Agents, and Decentralized Control" (AGERE 2018) at ACM SIGPLAN Conference "Systems, Programming, Languages and Applications: Software for Humanity" (SPLASH 2018), pp. 62–71. ACM (2018)
20. Bergenti, F., Petrosino, G.: Overview of a scripting language for JADE-based multi-agent systems. In: Proceedings of the 19th Workshop "From Objects to Agents" (WOA 2018), *CEUR Workshop Proceedings*, vol. 2215, pp. 57–62. RWTH Aachen (2018)
21. Bergenti, F., Poggi, A.: Ubiquitous information agents. International Journal of Cooperative Information Systems **11**(34), 231–244 (2002)
22. Bergenti, F., Poggi, A., Burg, B., Caire, G.: Deploying FIPA-compliant systems on handheld devices. IEEE Internet Computing **5**(4), 20–25 (2001)
23. Bergenti, F., Poggi, A., Tomaiuolo, M.: An actor based software framework for scalable applications. In: Proceedings of the 7th International Conference on Internet and Distributed Computing Systems (IDCS 2014), *Lecture Notes in Computer Science*, vol. 8729, pp. 26–35 (2014)
24. Bergenti, F., Ricci, A.: Three approaches to the coordination of multiagent systems. In: Proceedings of the 17th ACM Symposium on Applied Computing, pp. 367–373. ACM (2002)
25. Bettini, L.: Implementing Domain-Specific Languages with Xtext and Xtend. Packt Publishing (2013)
26. Bordini, R.H., Braubach, L., Dastani, M., El Fallah Seghrouchni, A., Gomez-Sanz, J.J., Leite, J., O'Hare, G., Pokahr, A., Ricci, A.: A survey of programming languages and platforms for multi-agent systems. Informatica **30**(1) (2006)
27. Bordini, R.H., Hübner, J.F., Wooldridge, M.: Programming multi-agent systems in AgentSpeak using Jason. John Wiley & Sons (2007)
28. Braubach, L., Pokahr, A.: Developing distributed systems with active components and Jadex. Scalable Computing: Practice and Experience **13**(2) (2012)
29. Breugst, M., Magedanz, T.: Mobile agents-enabling technology for active intelligent network implementation. IEEE Network **12**(3), 53–60 (1998)
30. Broadcom Web site: `www.broadcom.com`
31. Caire, G., Gotta, D., Banzi, M.: WADE: A software platform to develop mission critical applications exploiting agents and workflows. In: Proceeding of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008), pp. 29–36. IFAAMAS (2008)
32. Collis, J.C., Ndumu, D.T., Nwana, H.S., Lee, L.C.: The ZEUS agent building tool-kit. BT Technology Journal **16**(3), 60–68 (1998)
33. Eclipse Web site: `www.eclipse.org`
34. Eysholdt, M., Behrens, H.: Xtext: Implement your language faster than the quick and dirty way. In: Proceedings of the ACM International Conference "Object-Oriented Programming, Systems, Languages, and Applications" (OOPSLA 2010), pp. 307–309. ACM (2010)
35. FIPA Web site: `www.fipa.org`
36. Fisher, M.: MetateM: The story so far. In: R.H. Bordini, M. Dastani, A. El Fallah Seghrouchni (eds.) Proceedings of the 3rd International Conference on Programming Multi-Agent Systems (ProMAS 2005), pp. 3–22. Springer-Verlag (2006)
37. Foundation for Intelligent Physical Agents: FIPA specifications (2002). Available online at `www.fipa.org`
38. Fuggetta, A., Picco, G.P., Vigna, G.: Understanding code mobility. IEEE Transactions on software engineering **24**, 342–361 (1998)
39. García-Gómez, S., González-Ordás, J., García-Algarra, F.J., Toribio-Sardón, R., Sedano-Frade, A., Buisán-García, F.: KOWLAN: A multi agent system for Bayesian diagnosis in telecommunication networks. In: Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology (WI&IAT 2009), vol. 3, pp. 195–198. IEEE (2009)

40. Google Scholar Web site: `scholar.google.com`
41. Hindriks, K.V., Dix, J.: Goal: A multi-agent programming language applied to an exploration game. In: O. Shehory, A. Sturm (eds.) Agent-Oriented Software Engineering, pp. 235–258. Springer International Publishing (2014)
42. International Telecommunication Union: M.3400 TMN management functions (1997). Available online at `www.itu.int`
43. Islam, N., Mallah, G.A., Shaikh, Z.A.: FIPA and MASIF standards: A comparative study and strategies for integration. In: Proceedings of the 2010 National Software Engineering Conference (NSEC 2010), pp. 7:1–7:6. ACM (2010)
44. JADE Web site: `jade.tilab.com`
45. Kravari, K., Bassiliades, N.: A survey of agent platforms. Journal of Artificial Societies and Social Simulation **18**(1), 11 (2015)
46. Mangina, E.: Review of software products for multi-agent systems (2002). Available online at `www.agentlink.org`
47. Milojicic, D., Breugst, M., Busse, I., Campbell, J., Covaci, S., Friedman, B., Kosaka, K., Lange, D., Ono, K., Oshima, M., Tham, C., Virdhagriswaran, S., White, J.: MASIF: The OMG mobile agent system interoperability facility. Personal Technologies **2**(2), 117–129 (1998)
48. Monica, S., Bergenti, F.: Location-aware JADE agents in indoor scenarios. In: Proceedings of the 16th Workshop "From Objects to Agents" (WOA 2015), *CEUR Workshop Proceedings*, vol. 1382, pp. 103–108. RWTH Aachen (2015)
49. Monica, S., Bergenti, F.: A comparison of accurate indoor localization of static targets via WiFi and UWB ranging. In: Trends in Practical Applications of Scalable Multi-Agent Systems, *Advances in Intelligent Systems and Computing*, vol. 473, pp. 111–123. Springer International Publishing (2016)
50. Motorola Web site: `www.motorola.com`
51. MPEG Web site: `mpeg.chiariglione.org`
52. Müller, J.P., Fischer, K.: Application impact of multi-agent systems and technologies: A survey. In: O. Shehory, A. Sturm (eds.) Agent-Oriented Software Engineering, pp. 27–53. Springer International Publishing (2014)
53. Nwana, H.S., Ndumu, D.T., Lee, L.C., Collis, J.C.: ZEUS: A toolkit for building distributed multiagent systems. Applied Artificial Intelligence **13**, 129–185 (1999)
54. O'Brien, P.D., Nicol, R.: FIPA: Towards a standard for software agents. BT Technology Journal **16**(3), 51–59 (1998)
55. OMG Web site: `www.omg.org`
56. Petrosino, G., Bergenti, F.: An introduction to the major features of a scripting language for JADE agents. In: Proceedings of the 17th Conference of the Italian Association for Artificial Intelligence (AI*IA 2018), *Lecture Notes in Artificial Intelligence*, vol. 11298, pp. 3–14. Springer International Publishing (2018)
57. Petrosino, G., Bergenti, F.: Extending message handlers with pattern matching in the Jadescript programming language. In: Proceedings of the 20th Workshop "From Objects to Agents" (WOA 2019), *CEUR Workshop Proceedings*, vol. 2404, pp. 113–118. RWTH Aachen (2019)
58. Pitt, J., Bellifemine, F.: A protocol-based semantics for FIPA'97 ACL and its implementation in JADE. In: Proceedings of the 6th Congress of the Italian Association for Artificial Intelligence (AI*IA 1999). Italian Association for Artificial Intelligence (1999)
59. Poggi, A., Tomaiuolo, M., Turci, P.: A testbed for agent mediated service composition. In: Proceedings of the 4th International Symposium "From Agent Theory to Agent Implementation" (AT2AI-4), vol. 2, pp. 529–534. Austrian Society for Cybernetics Studies (2004)
60. Poslad, S.: History of FIPA (2005). Available online at `www.fipa.org`
61. Poslad, S.: Specifying protocols for multi-agent systems interaction. ACM Transactions on Autonomous and Adaptive Systems **2**(4) (2007)
62. Poslad, S.: Ubiquitous Computing: Smart Devices, Environments and Interactions. John Wiley & Sons (2009)
63. Poslad, S., Buckle, P., Hadingham, R.: Open source, standards and scaleable agencies. In: Proceedings of the Workshop "Infrastructure for Scalable Multi-Agent Systems", *Lecture Notes in Computer Science*, vol. 1887, pp. 296–303. Springer-Verlag (2001)
64. Ricci, A., Santi, A.: Designing a general-purpose programming language based on agent-oriented abstractions: The simpAL project. In: Proceeding of the 1st International Workshop "Programming Based on Actors, Agents, and Decentralized Control" (AGERE 2011)

at ACM SIGPLAN Conference "Systems, Programming, Languages and Applications: Software for Humanity" (SPLASH 2011), pp. 159–170. ACM (2011)
65. Rodriguez, S., Gaud, N., Galland, S.: SARL: A general-purpose agent-oriented programming language. In: Proceedings of the 2014 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology (*WI&IAT 2014*), pp. 103–110. IEEE (2014)
66. Scopus Web site: `www.scopus.com`
67. Shoham, Y.: AGENT-0: A simple agent language and its interpreter. In: Proceedings of the 9th National Conference on Artificial Intelligence (AAAI 1991), vol. 91, pp. 704–709. AAAI (1991)
68. Shoham, Y.: An overview of agent-oriented programming. In: J.M. Bradshaw (ed.) Software Agents, pp. 271–290. MIT Press (1997)
69. Siemens Web site: `www.siemens.de`
70. Telefónica Web site: `www.telefonica.com`
71. TIM Web site: `www.tim.it`
72. Tomaiuolo, M., Turci, P., Bergenti, F., Poggi, A.: An ontology support for semantic aware agents. In: Proceedings of the 7th International Workshop "Agent-Oriented Information Systems III" (AOIS 2005), *Lecture Notes in Artificial Intelligence*, vol. 3529, pp. 140–153. Springer International Publishing (2006)