Extending the JADE Framework for Semantic Peer–To–Peer Service Based Applications

Agostino Poggi  
DII, Università degli Studi di Parma, Italy  
Michele Tomaiuolo  
DII, Università degli Studi di Parma, Italy

ABSTRACT
One of the main challenges of multi-agent systems is to become the main means to support legacy systems interoperability and to make the realization of scalable distributed systems easy. In the last years, however, two technologies, peer-to-peer and service-oriented, have made an impressive progress and seem to have good chances of competing with multi-agent systems for the realization of scalable and interoperable systems. Conversely, neither of these two technologies is able to provide by themselves the autonomy and social and proactive capabilities of agents and thus the development of flexible adaptive distributed systems may be difficult. This chapter shows how JADE, one of the most known and used software framework for the development of multi-agent systems, has been extended with these technologies both to support the realization of multi-agent systems and to facilitate the interoperability with peer-to-peer and service-oriented systems.

INTRODUCTION
One of the main challenges of multi-agent systems is to become the main means to support legacy systems interoperability and to make easy the realization of scalable distributed systems (Genesereth, 1997; FIPA, 2000). However, in the last years, two technologies, peer-to-peer and service-oriented technologies, had an impressive progress and seem to have good chances to compete with multi-agent system for the realization of scalable and interoperable systems. The problem of both these two technologies is that they cannot provide by themselves the autonomy, social and proactive capabilities of agents and so the realization of flexible adaptive distributed systems may be difficult.

Therefore, an integration of multi-agent systems with such two technologies seems to be the most suitable solution for the realization of scalable and interoperable distributed applications. In fact, in these last years a lot of works have been presented for the integration of multi-agent systems with one of or both the two technologies (see, for example, Willmott et al., 2004; Greenwood et al., 2005; Huhns et al., 2005; Buford & Burg, 2006).

This chapter copes with the problem of the integration of multi-agent systems with peer-to-peer and service-oriented technologies and, in particular, presents how JADE, one of the most known and used software framework for the development of multi-agent systems (Bellifemine et
al., 2001), has been extended with these two technologies to both support the realization of multi-agent systems and to make easy the interoperability with peer-to-peer and service-oriented systems. The next section describes the main features of the JADE agent development software framework. Sections three and four respectively describe the peer-to-peer and the service-oriented extensions of the JADE software framework. Finally, section five summarizes the contributions of our work and points to future lines of work.

**JADE**

**JADE** (Java Agent DEvelopment framework) is a software framework designed to aid the development of agent applications in compliance with the FIPA specifications for interoperable intelligent multi-agent systems (Bellifemine, 2001; Bellifemine, 2008). The purpose of JADE is to simplify development while ensuring standard compliance through a comprehensive set of system services and agents. JADE is an active open source project, and the framework together with documentation and examples can be downloaded from JADE Home Page (JADE, 2009).

JADE is based on a peer-to-peer communication architecture. The intelligence, the initiative, the information, the resources and the control can be fully distributed across mobile terminals as well as computers connected to the fixed network. The environment evolves dynamically together with peers – that in JADE are called agents – that appear and disappear in the system according to the needs and the requirements of the application domain. Communication between the peers, regardless of whether they are running in the wireless or wired network, is completely symmetric with each peer being able to play both initiator and responder roles.

JADE is fully developed in Java and is based on the following driving principles:

- **Interoperability**: JADE is compliant with the FIPA specifications (FIPA, 2000). 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 (edition, configuration and profile). More in details, the JADE run-time provides the same APIs both for the J2EE, J2SE and J2ME environment. In theory, application developers could decide the Java run-time environment at deploy-time.

- **Ease of 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 a computational overhead.

JADE includes: i) the libraries (i.e., the Java classes) required to develop the application specific agents, ii) the implementation of the two management agents that a FIPA compliant agent platform must provide, i.e., the AMS (Agent Management System) agent and the DF (Directory Facilitator) agent, and iii) 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 it provides a homogeneous layer that hides to agents (and to application developers) the complexity and the diversity of the underlying tires (hardware, operating systems, types of network, JVM, etc.). Figure 1 draws the architecture of a JADE multi-agent system deployed on a set of heterogeneous computing nodes.
Figure 1. Architecture of a JADE multi-agent system.

JADE is extremely versatile and therefore it both fits the constraints of environments with limited resources and has already been integrated into complex architectures such as .NET or J2EE (BlueJade, 2003) where JADE becomes a service to execute multi-party proactive applications. The JADE run-time memory footprint, in a MIDP1.0 environment, is around 100 KB, but can be further reduced until 50 KB using the ROMizing technique (Bergenti et al., 2001), i.e., compiling JADE together with the JVM. The limited memory footprint allows installing JADE on all mobile phones provided that they are Java-enabled. Analyses and benchmarks of scalability and performance of the JADE Message Transport System are reported by different works (Chmiel et al., 2005; Zimmermann et al., 2006).

As above 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.

This is possible thanks to a module called LEAP (Lightweight Extensive Agent Platform) that allows optimizing all communication mechanisms when dealing with devices with limited resources and connected through wireless networks (Bergenti et al., 2001). By activating this module, a JADE container is “split” (as depicted in Figure 2) 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 instantiatiating and maintaining the back-end. 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 some 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.
- 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 to 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 the optimization of the usage of the wireless link.

![Diagram](image)

**Figure 2. JADE architecture for the mobile environment.**

From the functional point of view, **JADE** provides the basic services necessary to distributed peer-to-peer applications in the fixed and mobile environment. JADE allows agents to dynamically discover other agents and to communicate with them according to the peer-to-peer paradigm. From the application point of view, each agent is identified by a unique name and it provides a set of services; it can register and modify its services and/or search for agents providing some specific services, it can control its life cycle and, finally, it communicates with all other peers.

**Agents** communicate through the exchange of asynchronous messages, i.e., agents just sends messages to other agents without waiting for an answer, which is a communication model almost universally accepted for distributed and loosely-coupled communications, i.e., between heterogeneous entities that do not know anything about each other. Agents are identified by a name that does not statically include the actual distributed object reference of the agent and therefore there is no temporal dependency between communicating agents. The sender and the receiver could not be available at the same time. The receiver may not even exist (or not yet exist) or it could be not directly known by the sender that can specify a property (e.g. “all agents interested in football”) as an intentional description of destination agents. Because agents identify each other by their name, hot changes of their object reference are transparent to applications.
Despite this type of communication, security is preserved, because JADE provides, for applications that require it, proper mechanisms to authenticate and verify rights and capabilities assigned to agents. When needed, an application can verify the identity of the sender of a message and prevent any not allowed action. For instance, an agent may be allowed to receive messages from the agent representing its boss, but not to send messages to it. All messages exchanged between agents are transported within an envelope that includes only the information required by the transport layer (see Figure 3). This allows, among other things, to encrypt the content of a message separately from the envelope and to guarantee reachability with no loss of security.

The authentication of agents and the verification of their rights can be done by an optional plug-in of JADE (Poggi et al., 2005). This plug-in leverages on the security means provided by Java and extends them to allow the definition of precise protection domains on the basis of authorization certificates. These certificates, attached to requests and messages, list a set of granted permissions and are signed by trusted authorities according to customizable policies, possibly in completely decentralized way. The authorization certificates owned by the agents can also be used to delegate access rights to other agents, to allow them to complete the requested tasks or to achieve delegated goals.

The structure of a message complies with the ACL (Agent Communication Language) model defined by FIPA (FIPA, 2000) and includes fields, such as variables indicating the context a message refers to and the amount of time that is waited before an answer is received, aimed at supporting complex interactions and multiple parallel conversations.

To further 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), programmers can get rid of the burden of dealing with synchronization issues, timeouts, error conditions and, in general, all those aspects that are not strictly related to the application domain.

To facilitate the creation and handling of message contents, JADE provides support for automatically converting back and forth between formats suitable for content exchange, including XML and RDF, and the formats suitable for content manipulation (i.e., Java objects). This facility is integrated with some well known ontology-creation tools, e.g. Protégé, allowing programmers to graphically create the ontology agents should use to validate and provide semantics to messages.

**Figure 3. Encapsulation of ACL messages.**

The authentication of agents and the verification of their rights can be done by an optional plug-in of JADE (Poggi et al., 2005). This plug-in leverages on the security means provided by Java and extends them to allow the definition of precise protection domains on the basis of authorization certificates. These certificates, attached to requests and messages, list a set of granted permissions and are signed by trusted authorities according to customizable policies, possibly in completely decentralized way. The authorization certificates owned by the agents can also be used to delegate access rights to other agents, to allow them to complete the requested tasks or to achieve delegated goals.

The structure of a message complies with the ACL (Agent Communication Language) model defined by FIPA (FIPA, 2000) and includes fields, such as variables indicating the context a message refers to and the amount of time that is waited before an answer is received, aimed at supporting complex interactions and multiple parallel conversations.

To further 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), programmers can get rid of the burden of dealing with synchronization issues, timeouts, error conditions and, in general, all those aspects that are not strictly related to the application domain.

To facilitate the creation and handling of message contents, JADE provides support for automatically converting back and forth between formats suitable for content exchange, including XML and RDF, and the formats suitable for content manipulation (i.e., Java objects). This facility is integrated with some well known ontology-creation tools, e.g. Protégé, allowing programmers to graphically create the ontology agents should use to validate and provide semantics to messages.
To increase scalability and to meet the constraints of environments with limited resources, JADE provides the opportunity of executing multiple parallel tasks within the same Java thread. Several elementary tasks, such as communication, can then be combined to form more complex tasks structured as concurrent finite states machines.

JADE supports mobility of code and of execution state. That is, 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 the stack and the program counter cannot be saved in Java). 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 the debugging and management/monitoring phases of application life cycle. 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, agent life-cycle can be controlled. Figure 4 presents the GUIs of some JADE software development tools.

![Figure 4. JADE software development tools.](image)

**EXTENDING JADE WITH PEER-TO-PEER TECHNOLOGIES**

The traditional, client-server model describes systems where computational resources and data are centralized in few servers, which respond to requests of clients. On the other hand, clients are supposed to have little capabilities and rely on the resources of servers for most of their tasks. The multi-agent model reverses this paradigm and describes systems organized in a peer-to-peer fashion, where each participant potentially has some resources to share and some services to offer to the community of agents. Thus, according to the context, each agent is able to play either the role of client or server.
JADE implements FIPA specifications for multi-agent systems, and so enables the realization of peer-to-peer distributed systems, constituted by smart and loosely coupled agents communicating by means of asynchronous ACL messages (FIPA. 2000).

Nevertheless, JADE does not exploit some important features of modern peer-to-peer networks, in particular:
- The possibility of building a completely distributed, global index of resources and services, without relying on any centralized entity.
- The possibility of building an “overlay network”, hiding differences in lower level technologies and their related communication problems.

Some multi-agent systems, like Agentscape, approached the same issues by developing a dedicated peer-to-peer network layer (Overeinder et al., 2002), our solution is to integrate multi-agent platforms into an already existing and used peer-to-peer environment, i.e., JXTA (JXTA, 2008), thus, benefiting from a well tested system and exposing services to other entities participating in the network.

JXTA technology is a set of open, general-purpose protocols that allow any connected device on the network (from cell phones to laptops and servers) to communicate and collaborate in a peer-to-peer fashion. The project was originally started by Sun Microsystems, but its development was kept open from the very beginning. JXTA comprises six protocols allowing the discovery, organization, monitoring and communication between peers. These protocols are all implemented on the basis of an underlying messaging layer, which binds the JXTA protocols to different network transports.

JXTA peers can form peer groups, which are virtual networks where any peer can seamlessly interact with other peers and resources, whether they are connected directly or through intermediate proxies. JXTA defines a communication language which is much more abstract than any other peer-to-peer protocol, allowing to use the network for a great variety of services and devices. A great advantage of JXTA derives from the use of XML language to represent, through structured documents, named advertisements, the resources available in the network. XML adapts without particular problems to any transport mean and it is already an affirmed standard, with good support in very different environments, to structure generic data in a form easily analyzable by both humans and machines.

**JXTA-ADS**

What usually happens in a multi-agent platform is the cohabitation of multiple agents interacting in a common and cohesive environment, making use of a formal communication language, defined by its own syntax and semantics, in order to complete tasks demanded by users. For making constructive the communication, it is necessary to provide agents with a system allowing them to reciprocally individuate offered services. This happens thanks to the presence of a yellow pages service, provided by the platform, which can be consulted by agents when needed. However this often limits the search inside a single platform. Solutions are possible, which allow the consultation of other yellow pages services, but they necessitate the a priori knowledge of the address of the remote platform where services are hosted or listed.

An alternative solution is represented by a yellow pages service leaning on a peer-to-peer network like JXTA, thanks to which each network device is able to individuate in a dynamic way services and resources of other network devices.

Technologies inherent to web services are using WSDL as a standard language to publicize all different available resources. In FIPA, a simpler formalism is defined to describe services and
resources exposed by agents and linked to their own domain ontology. JXTA does not establish any constraint on the way to describe and invoke services. JXTA protocols simply provide a generic framework, allowing the use of any mechanism, also WSDL or FIPA service descriptions, to exchange information needed to invoke a service.

Particular peers, called rendezvous peers, are in charge of both indexing resources made available in the network and finding them when requested by other peers. Rendezvous peers can also communicate queries to each other, if they do not possess the right information, thus enabling the discovery of advertisements beyond the local network.

In fact, in JXTA, resources are described by advertisements, which are essentially XML documents collecting metadata of available resources. Advertisements are not stored on some single machine, such as a server, or on a hierarchical infrastructure; they are distributed among rendezvous peers, which implement a distributed algorithm, called shared resource distributed index (SRDI), for the creation and management of the index of resources available in the network. On the basis of some indexed attributes, this mechanism can solve queries made anywhere in the rendezvous network. Basically, the global index is a loosely consistent distributed hash table, where the hash of an indexed attribute is mapped to some peer responsible for storing the actual advertisement.

FIPA has acknowledged the growing importance of the JXTA protocols, and it has released some specifications for the interoperability of FIPA platforms with peer-to-peer networks. In particular, in (FIPA, 2003) a set of new components and protocols are described, to allow the implementation of a DF-like service on a JXTA network (i.e., a yellow pages service that allows to discover on a JXTA network the agents that provide a particular service as is done by a directory facilitator in a traditional JADE multi-agent system). These include:
- Generic Discovery Service: a local DF, taking part in the peer-to-peer network and implementing the agent discovery service specifications to discover agents and services deployed on remote FIPA platforms working together in a peer-to-peer network.
- Agent Peer Group: a child of the JXTA Net Peer Group that must be joined by each distributed discovery service.
- Generic Discovery Advertisement: to handle agent or service descriptions, for example, FIPA df-agent-descriptions.
- Generic Discovery Protocol: to enable the interaction of discovery services on different agent platforms. It’s a request/response protocol to discover advertisements, based on two simple messages, one for queries and one for responses.

The JADE development environment does not provide any support for the deployment of real peer-to-peer systems because it only provides the possibility of federating different agent platforms through a hierarchical organization of the platform directory facilitators on the basis of a priori knowledge of the agent platforms addresses. Therefore, the JADE directory facilitator has been extended to realize a peer-to-peer network of agent platforms thanks to the JXTA technology (JXTA, 2008) and thanks to two preliminary FIPA specifications for the agent discovery service (FIPA, 2003) and for the JXTA discovery middleware (FIPA, 2004).

This way, JADE integrates a JXTA-based ADS (Agent Discovery Service), which has been developed in the respect of relevant FIPA specifications to implement a GDS (Generic Discovery Service). Each JADE platform connects to the Agent Peer Group, as well as to other system-specific peer groups. The GDS is finally used to advertise and discover agent descriptions, wrapped in generic discovery advertisements, in order to implement a DF service, which in the background is spanned over a whole peer group. Figure 5 shows how an ADS agent interacts with
JXTA discovery service for finding agents of other JADE platform connected through a JXTA network.

![Diagram of JXTA discovery service](image)

**Figure 5. Discovery of agents in a JXTA network.**

**JXTA-MTP**

In the course of some large projects based on agent technologies like Agentcities and @lis TechNet (Poggi et al., 2004; Willmott et al., 2006), some recurring problems emerged at the level of connection among remote platforms. The importance of these problems invariably grows with the cardinality and geographical extension of the interconnected infrastructure, and has been acknowledged in other similar large scale environments.

Most peer-to-peer networks specifically address this kind of problems allowing the connection of peers located behind firewalls, Network Address Translators (NATs) and Dynamic Host Configuration Protocol (DHCP) servers, or requiring different and particular protocols like HTTP or WAP. To this end, peer-to-peer networks create an overlay infrastructure above underlying diverse and problematic links in order to realize a more abstract and homogeneous ground and simplify the communications among peers.

**JXTA** is one of the most used technologies to improve connectivity on a global scale. In fact, JXTA does not suppose a direct connection is available between all couple of peers. Peers can use the Peer Endpoint Protocol to discover available routes for sending a message to a destination peer. Particular peers, called routers, are in charge of responding to such queries providing route information, i.e. a list of gateways connecting the sender to the intended receiver. A gateway acts as a communication relay, where messages can be stored and later collected by their intended recipient, overcoming problems related to limited connectivity.

**JADE**, on the other hand, offers an extensible mechanism for the transport of messages among platforms, in the form of pluggable Message Transport Protocols (MTPs). The default implementations are based on IIOP and HTTP, which are both limited by the requirement of a direct connection between sender and receiver. Therefore, exploiting the extensibility of JADE platforms, a JXTA-MTP implementation has been realized. This MTP allows the exchange of messages between two platforms through JXTA pipes which are dynamically bound to specific
endpoints (typically an IP address and a TCP port). JXTA pipes are advertised on the network in the same way as other services offered by peers, and provide a global scope to peer connectivity.

The JXTA-MTP implementation allows using not only plain JXTA pipes, but also secure ones with encryption and signature mechanisms guaranteeing privacy, integrity and authenticity of exchanged messages.

EXTENDING JADE WITH SERVICE ORIENTED TECHNOLOGIES

Industry is increasingly interested in executing business functions that span multiple applications, thus requiring high-levels of interoperability and a more flexible and adaptive business process management. The most appropriate response to this need seems to be having systems assembled from a loosely coupled collection of web services. This technical area appears to be an interesting environment in which the agent technology can be exploited with significant advantages. As a matter of fact, several researches belonging to the agent community have dealt with the issues concerning the interconnection of agent systems with W3C compliant web services, with the aim of allowing each technology to discover and invoke instances of the other. One evident benefit of this is the central role that agents could play in a service oriented scenario, by efficiently supporting distributed computing and allowing the dynamic composition of web services.

Several works proposed the integration of JADE with service-oriented technologies (see, for example, Martinez & Lespérance, 2004; Nguyen & Kowalczyk, 2005; Soto, 2006). However, there exists a JADE official solution for such an integration that is based on the JADE WSIG add-on (Greenwood & Calisti, 2004) and on an extension of JADE, called WADE (Caire et al., 2008a; Caire et al., 2008b), and supports the use of workflows in the realization of applications integrating agents and web services.

WSIG

JADE supports the invocation of agent services as web services and the capability to realize applications as composition of agents and Web services through a JADE add-on, called WSIG (Web Service Integration Gateway) that is able to automatically expose agent services as web services and to convert SOAP invocations into ACL requests (Greenwood & Calisti, 2004). More in details, JADE agents publish their services through the DF agent of a JADE multi-agent platform. 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 defines the actions that the agent is able to perform. WSIG listens to registrations with the DF agent 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. 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. When an agent needs to invoke a web service, it directly creates the SOAP message and sends it to the provider, e.g., exploited AXIS2 API. In this case, no particular support is provided by WSIG. To date, WSIG supports only simple WSDL descriptions of web services, without taking into account emerging technologies related to the semantic Web.
**WADE**

In general, the modus operandi to carry out web service compositions is similar to that concerning the definition of workflows, such that existing techniques for workflow pattern generation, composition, and management can be partially reused for this purpose (Henoque & Kleiner, 2007). Therefore, even if workflows can be useful for composing agent tasks they can be also an interesting means to support the realization of applications merging the execution of agents and web services (Singh & Huhns, 1999; Chen et al., 2000; Savarimuthu et al., 2004; Vidal et al., 2004; Zao et al., 2007; Trappey et al., 2009).

**WADE** (Workflow and Agent Development Environment) is a software platform, built on top of JADE, for the development of distributed applications based on the agent-oriented paradigm (Caire et al., 2008a). WADE adds to JADE the support to the workflow execution and a few mechanisms to manage the complexity of the distribution, in terms of administration and fault tolerance. WADE adds some additional components to a JADE application:

- A boot daemon on each host on which the application is deployed with the duty of activating the JADE containers of the application on the current host.
- A configuration agent on the main container of the application. This agent is responsible for interacting with the boot daemons and controlling the application life cycle.
- A controller agent for each container in the platform and they are responsible for supervising activities in the local container and for all the fault tolerance mechanisms provided by WADE.
- A set of workflow engine agents able to execute workflows.

In particular, the workflow engine agents embed a workflow engine able to executed workflows encoded by an extended version of XPDL (XPDL, 2008) that adds the possibility of associating the direct execution of pieces of Java code with the activities of a workflow. In fact, the main challenge in WADE is to bring the workflow approach from the business process level to the level of system internal logics, i.e., its main goal is not to support the high level orchestration of services provided by different systems, but the implementation of the internal behavior of each single system. Moreover, the execution of a workflow can be shared by a set workflow engine agents because each workflow engine agent can delegate the execution of some subflows to some other agents. These agents are selected at runtime and can also run on remote computers; of course, the selection criteria depend on the application and are provided to the configuration agent before the execution of the workflow.

The development of the workflows to be executed by the WADE software platform can be easily done thanks to the use the WOLF graphical development environment (Caire et al., 2008b). WOLF is implemented as an Eclipse plugin and allows the management of the whole life cycle of workflows. In particular, WOLF provides support for: i) graphically editing of workflows, ii) controlling a local or remote WADE-based application and iii) deploying and executing workflows in the controlled application. Figure 6 shows the WOLF workflow editor.

The use of WOLF provide an easy way for realizing workflow whose activities invoke web services. In fact, WOLF allows a developer to browse WSDL files repository and then imports the WSDL files describing the web services to be used in the workflow. The result of the import operation is the generation of a set of classes that will be used during the execution of a workflow for invoking the web services described by the imported WSDL. Unless they have specific customization needs, developers don’t need to care about such classes because WOLF and WADE hide all the details of the web service invocation process. In particular, if the imported WSDL declares some complex types, suitable Java bean classes are also generated.
BEJA

A more adequate solution for realizing a more flexible and effective agent-based business process management with JADE is to use BEJA (BPEL Enhanced JADE Agent). BEJA is a software library, built on the basis of the results of the development and experimentation of two previous systems, GAIN (Negri et al., 2006) and MASE (Poggi et al., 2007), that adds the possibility of using into a JADE application some special agents that couple the execution of WS-BPEL (WS-BPEL, 2007) workflows with the semantic retrieval of web services provided by a semantic UDDI.

The reason to choose WS-BPEL as the language to express workflows is twofold. On the one hand, it is now the prevalent approach for defining workflows based on web services. On the other hand, it allows describing a business process in two different ways: the executable process and the abstract process. While the former is a fully specified business process the latter is the description of a business process interface. The abstract process offers some more flexibility, if compared with the executable process, by allowing workflows to refer to abstract service interfaces rather than concrete instances. This means that when using an abstract process, services can, in principle, be selected dynamically at run-time, depending on current service availability. Actually, looking deeply to the standard specifications it emerges that the degrees of freedom are limited by the fact that service instances must adhere exactly to the syntactic interfaces specified by the workflow designers. We get round this problem by using semantic concept, belonging to a shared ontology, to describe the abstract process and introducing particular kind of agents in charge of carrying out the transformation between semantic concepts and concrete parameters.

Two key components of the BEJA software library are: a software tool that transforms a standard WS-BPEL workflow XML representation into a Java object-oriented structure that can be easily decomposed and manipulated during its distributed execution, and a WS-BPEL engine that, while executes a workflow through the use of the most appropriate Web services, can also delegate the execution of some its sub-workflows to other engines.

BEJA comprises four groups of agents: workflow manager agents, component manager agents, broker agents and service provider agents. Once the business process is expressed in terms
of a WS-BPEL workflow, it is executed by the workflow manager agent. The workflow manager agent, containing a WS-BPEL engine, is responsible for initiating and coordinating the entire execution process. It generates a process instance and controls its enactment, creating the execution context, which will represent the reference context during the execution process. Next, it will identify those parts of the workflow (e.g. scope activities, sub-activities of the flow activities, and so on), which if executed remotely will positively affect the performance of the system, and will delegate their execution to specific component manager agents, which are agents responsible for executing sub-workflows and are therefore coordinated by the workflow manager. The workflow manager and its delegated agents, assisted by a broker agent, find suitable service instances to be invoked and set time-out values for each task in the workflow, so that their expected utility is maximized. The broker agent is an agent responsible for mediating between the manager agents and the service provider agents. Its mediation role and coordination property are essential in a semantic environment where services are described, discovered and invoked semantically. The broker agent could directly exploit a semantic search made available by some semantic UDDI registries, but for a clearer separation of responsibilities, it manages it through some service provider agents, responsible for the interaction with semantic UDDI registries.

This solution allows the realization of more fault-tolerant applications. In fact, during the execution of a workflow, a selected web service could be no longer available due to the expiration of a timeout, a failure of a resource or other unpredictable problems. Therefore, the workflow and component managers can rely on the broker agent in order to achieve their goal by replacing such a web service. In the same way as in the engaging phase, the broker agent is responsible for the dynamic provision of web services, which match the business process needs, mediating between the manager agents and one or more service provider agents and enriching the service provision with advanced features. In particular, the broker is responsible for setting the semantic matching constraints and delegating the semantic search to one or more service provider agents, which finally play the role of middle agents during the invocation phase. In their role of mediator the service provider agents perform a two-step translation, i.e., from the business process representation of a service to its ontological internal representation and from there to the concrete Web service representation (vice versa when they forward the result of the Web service invocation).

In case of failure, the workflow manager agent or possibly a component manager agent will ask a broker agent for help. On the basis of its knowledge of a shared ontology and the functionalities required, the broker will send a semantic query to one or more service provider agents, which in turn will send the query to all potentially interested semantic UDDI registries. The UDDI registries, exploiting a matching algorithm, will as a result give a list of suitable services associated with a degree of matching, which gives the information about how much a particular web service and the received query are semantically compatible. They perform service I/O based profile matching, exploiting ontological concepts as values of service input and output parameters. The algorithm used is similar to the one used in the OWLS-MX matchmaker (Klusch et al., 2006), which, at present, is one of the most effective semantic web service matchmakers which have been developed in the last years. At this point, the service provider agent, applying an appropriate strategy (e.g., it selects the first Web service fulfilling the requirements), will decide which web service to use in order to satisfy the broker request. Finally the broker will inform the workflow manager that a web service with the required characteristics is available.

In view of the fact that there may be mismatches between the syntactic description of the selected web service and its real invocation by the workflow manager (i.e., the abstract service,
described in the workflow, and the concrete web service can have different syntactic interface even though they provide the same service) the manager agent will refer to the service provider agent for the web service invocation given that it is aware of both the abstract and the concrete description of web services and it is able to manage the transformation of the data coming from the workflow manager to the web service and vice-versa.

CONCLUSIONS
This chapter has dealt with the issue of enhancing the multi-agent systems role in the realization of scalable and interoperable systems by exploiting peer-to-peer and service-oriented technologies as key components for their realization. In particular, the chapter has shown how JADE, one of the best known and most used software framework for the development of multi-agent systems, has been extended with these two technologies both to support the realization of multi-agent systems and to facilitate the interoperability with peer-to-peer and service-oriented systems.

This integration shows interesting advantages. On the one hand, the exploitation of the peer-to-peer technology gives a great impulse towards the scalability and interoperability of different agent platforms and agent-based applications. On the other hand, the openness towards the web service standards gives agents the possibility to interoperate with a consolidated industrial reality and one of the most accepted mechanisms used for integration of distributed systems.

Some other works cope with the extension of the JADE software framework with peer-to-peer and service-oriented technologies (see, for example, Yan et al., 2005; Liu et al., 2006). However, the solution presented in this chapter provides a more complete peer-to-peer support and above all provides a sophisticated support for services composition, i.e., BEJA, that uses a standard language for workflow definition and allows the realization of more reliable applications. In fact, it use WS-BPEL as language for expressing workflows and allows a semantic and dynamic discovery of Web services. The use of WS-BPEL as the language to express workflows is important because it is the prevalent approach for defining workflows based on web services. The semantic and dynamic discovery of web services allows a flexible provision of the services and then the realization of more reliable applications.

Of course, further work can done to improve the JADE peer-to-peer and service composition supports described in this chapter. In particular, current work is coping with an important limit of the integration between BEJA and JADE. In fact, BEJA uses a different (and more complex) ontology management support from the one used in a JADE application for supporting the interaction among the agents. It is because in a JADE application, ontologies are used only to define a knowledge model shared by the agents, but not for reasoning about them. Therefore, current work is dedicated to the development of an ontological management support that should replace both the JADE and BEJA ontological supports. It will simplify the realization of JADE / BEJA applications, but also will extend the capabilities of all the JADE agents by providing them the possibility of reasoning about ontologies.

REFERENCES


