Developing Agent Based Applications with JADE

F. Bergenti, A. Poggi, G. Rimassa, P. Turci and M. Tomaiuolo

JADE (Java Agent Development Framework) is an "open source" FIPA-compliant software environment to build agent systems. JADE offers an agent middleware to implement efficient FIPA2000 compliant multi-agent systems and supports their development through the availability of a predefined programmable agent model, an ontology development support, and a set of management and testing tools. This chapter describes JADE and its use in three international projects to develop applications in the fields of: corporate memory management, integration of fixed and mobile networks, and integration of Web services.

1 Introduction

Agents are one of the most promising information technologies; however, agent-based technologies cannot keep their promises, and will not become widespread, until there are standards to support agent interoperability and adequate environments for the development of agent systems.

In these last years, there is a lot of activity concerning the standardization of agent-based technologies, where in the past KSE (Patil et al., 1992), OMG (Milojicic et al., 1998) and then FIPA (FIPA, 1997) led the activity.

The Foundation for Intelligent Physical Agents (FIPA) is an international non-profit association of companies and organizations sharing the effort to produce specifications for generic agent technologies. FIPA does not promote a technology for just a single application domain but a set of general technologies for different application areas that developers can integrate to make complex systems with a high degree of interoperability. The standardisation work of FIPA is centred on the definition of the Agent Communication Language (ACL), but also specifies the key agents necessary for the management of an agent system and the shared ontology to be used for the interaction between two systems.

In this paper, we present JADE (Java Agent DEvelopment framework), one of most known and used software framework to write agent applications in compliance with the last FIPA specifications for interoperable intelligent multi-agent systems. The next section describes the main features of JADE. Sections three, four and five respectively present three projects, LEAP, CoMMA and Agenticities funded by European Commission. LEAP addressed the need for open infrastructures and services that support dynamic, mobile enterprises and extended JADE to support mobile and wireless applications. CoMMA realized a multi-agent system to help users in the management of an organization corporate memory and in particular to facilitate the creation, dissemination, transmission and reuse of knowledge in the organization. Agentcities created an on-line, distributed testbed to explore and validate the potential of agent technology for future dynamic service environments. Finally the last section summarizes the advantages of using JADE to develop agent applications, its limits and gives a short comparison with other similar software frameworks.

2 JADE

JADE (Java Agent DEvelopment framework) is a software framework to aid the development of agent applications in compliance with the FIPA specifications for interoperable intelligent multi-agent systems. JADE is an "open source" project, and the complete system can be downloaded from JADE Home Page (JADE, 1999; Bellifemine et al, 2001).

The JADE system can be described from two different points of view. On the one hand, JADE is a middleware for FIPA-compliant Multi Agent Systems, supporting application agents whenever they need to exploit some feature covered by the FIPA standard specification (message passing, agent life-cycle management, etc.). On the other hand, JADE is a Java framework for developing FIPA-compliant agent applications, making FIPA standard assets available to the programmer through object oriented abstractions.

2.1 Platform architecture

JADE agent platform architecture tries to offer flexible and efficient messaging, transparently choosing the best transport available and leveraging state-of-the-art distributed object technology embedded within the Java runtime environment. While appearing as a single entity to the outside world, a JADE agent platform is itself a distributed system, since it can be split over several hosts with one of them acting as a front end where AMS and DF agents are placed. A JADE system comprises one or more Agent Containers, each living in a separate Java Virtual Machine and delivering run-time environment support to some JADE agents. The JADE middleware tries to provide efficient and flexible messaging services to user applications. An agent platform must contain a component called Agent Communication Channel (ACC for short), whose task is to transparently provide a Message Transport Service (MTS for short), relying upon one or more FIPA compliant Message Transport Protocols (MTPs), e.g., IIOP, HTTP and WAP protocols.

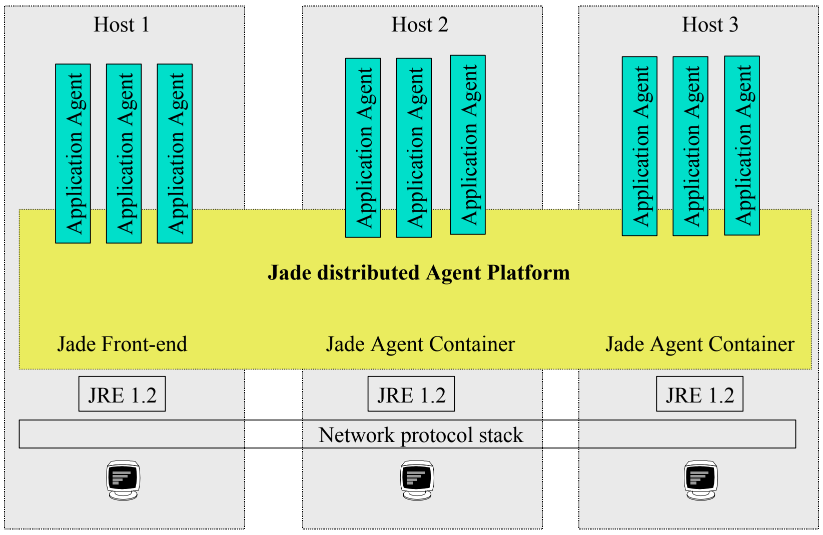

Figure 1. Jade agent platform architecture.

JADE distinguishes between inter-platform messaging (the sender and the receiver agents live on different platforms) and intra-platform messaging (the two interacting agents are within the same platform). While inter-platform messaging has to comply with FIPA specifications, intra-platform message delivery is strictly a JADE issue, so a more convenient proprietary transport can be exploited. JADE usually uses Java RMI for intra-platform communications, but support different transport protocols for intra-platform messaging to allow the distribution of the platform on devices providing different transport protocols.

Since JADE is a distributed agent platform, the ACC component is split in different parts, running on the different agent containers that make up the platform. The major features of JADE ACC are: i) multiple MTPs, deployed as plug-ins on multiple containers; ii) one hop message routing for outgoing and incoming messages, and iii) protocol independent address caching.

The general JADE messaging framework allows to deploy new transport ports during normal platform operation: the JADE administrator can add a new protocol to any agent container, simply logging in the management GUI and providing the Java class that implements the MTP.

An agent platform can now have any number of addresses, scattered around different hosts. Message routing support is needed to manage this rather general topology; the ACC provides a routing service that is guaranteed to require at most one hop. When a message reaches the platform through one of the available external communication ports, the ACC looks up the receiver agent ID to retrieve the agent container where it must dispatch the incoming ACL message. If the agent lives within the same container, the ACC uses an optimized local call; otherwise it relies on Java RMI.

When an agent wants to send a message to another, living on a different platform, it asks its local ACC for delivery service. The ACC reads the address list of the agent ID of the message recipient and tries all the addresses until one of them succeeds; for a specific address, the ACC discovers which MTP has to be used (FIPA addresses are URLs, so they contain a part that identifies the protocol) and checks to see whether that MTP is installed on the current agent container. If so, the locally available MTP is used, otherwise the ACC routes the message to a suitable container using a table that stores the deployment location of each MTP in the agent platform.

The JADE messaging subsystem also has an address caching feature that allows direct communication between agents, without unnecessary table lookups: intra-platform addresses and standard FIPA addresses are cached on each container exactly in the same way: on cache hits, the messaging subsystem does not even need to know whether the receiver is local, intra-platform or inter-platform. The cache is updated according to an optimistic attitude (i.e., if a cached address becomes stale the message delivery operation fails with an exception and the cached item is refreshed) and the cache replacement policy is the usual Least Recently Used one.

The JADE ACC can also be deployed on its own, without a complete agent container. This is meant to enable users to build and deploy agent level gateways and firewalls: a standalone ACC lives within a JVM that can route and filter ACL messages but cannot host FIPA agents.

2.2 Agent architecture

An agent is defined as a collection of behaviours that are scheduled and executed to carry on agent duties. Behaviours represent logical threads of a software agent implementation. According to Active Object design pattern, every JADE agent runs in its own Java thread, satisfying autonomy property; instead, to limit the threads required to run an agent platform, all agent behaviours are executed cooperatively within a single Java thread. So, JADE uses a thread-per-agent execution model with cooperative intra-agent scheduling.

JADE agents schedule their behaviour with a “cooperative scheduling on top of the stack”, in which all behaviours are run from a single stack frame (on top of the stack) and a behaviour runs until it returns from its main function and cannot be preempted by other behaviours (cooperative scheduling).

JADE agent architecture model is an effort to provide fine-grained parallelism on coarser grained hardware. A likewise, stack based execution model is followed by Illinois Concert runtime system (Karamcheti et al., 1996) for parallel object oriented languages.

Concert executes concurrent method calls optimistically on the stack, reverting to real thread spawning only when the method is about to block, saving the context for the current call only when forced to.

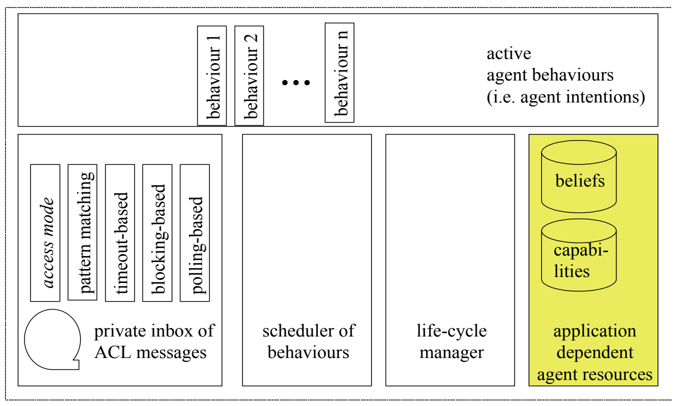

Figure 2. JADE agent architecture.

JADE thread-per-agent model can deal alone with the most common situations involving only agents: this is because every JADE agent owns a single message queue from which ACL messages are retrieved. Having multiple threads but a single mailbox would bring no benefit in message dispatching. On the other hand, when writing agent wrappers for non-agent software, there can be many interesting events from the environment beyond ACL message arrivals. Therefore, application developers are free to choose whatever concurrency model they feel is needed for their particular wrapper agent; ordinary Java threading is still possible from within an agent behaviour.

The developer implementing an agent must extend Agent class and implement agent-specific tasks by writing one or more Behaviour subclasses. User defined agents inherit from their superclass the capability of registering and deregistering with their platform and a basic set of methods (e.g. send and receive ACL messages, use standard interaction protocols, register with several domains). Moreover, user agents inherit from their Agent superclass some methods to manage the agent behaviours.

JADE contains ready made behaviours for the most common tasks in agent programming, such as sending and receiving messages and structuring complex tasks as aggregations of simpler ones. JADE recursive aggregation of behaviour objects resembles the technique used for graphical user interfaces, where every interface widget can be a leaf of a tree whose intermediate nodes are special container widgets, with rendering and children management features. An important distinction, however, exists: JADE behaviours reify execution tasks, so task scheduling and suspension are to be considered, too.

Thinking in terms of software patterns, if Composite is the main structural pattern used for JADE behaviours, on the behavioural side we have Chain of Responsibility: agent scheduling directly affects only top-level nodes of the behaviour tree, but every composite behaviour is responsible for its children scheduling within its time frame.

2.3 Ontology Management

Complex knowledge management domain leads to complex interactions between agents; in order to support this complexity it is necessary to have a good support for content language and ontology. JADE offers a general support for ontologies based on a model of the content language, which is able to describe:

i) object, construct that represents an identifiable entity; this is mainly important to realize a typed knowledge base;

ii) proposition, e.g. the content of an “inform” communicative-act is a predicate (a subtype of proposition);

iii) action, e.g. in the “request” communicative-act, tries to express an activity that can be carried out by an object;

iv) IRE (Identifying Reference Expression) , e.g. in the “query-ref” communicative-act.

This model is composed of:

i) An abstract content language; ontology independent abstract model, that is a generic model of the concepts that any content language must be able to express (e.g. schemas representation, like predicate, action, etc.);

ii) a concrete content language (instance of the abstract content language with possible different logic frameworks, like modal logic, deontic logic, etc.).

The whole ontology support framework is characterized by four hot spots: it is possible to have different content language instances (e.g., deontic logic content language, modal logic content language, etc.), different content language concrete syntaxes (e.g. RDFS, SL-expression), different ontologies, and different ontology concrete representations (RDFS, SL-expression).

2.4 Management and Testing Tools

JADE also offers some tools to manage the running agent platform and to monitor and debug agent societies. All these tools are implemented as FIPA agents themselves, and they require no special support to perform their tasks, they simply rely on JADE AMS. The general management console for a JADE agent platform is called RMA (Remote Management Agent). The RMA acquires the information about the platform and executes the GUI commands to modify the status of the platform (creating new agents, shutting down peripheral containers, etc.) through the AMS. On the one hand, the RMA asks the AMS to be notified about the changes of state of platform agents, on the other hand, it transmits to the AMS the requests for creation, deletion, suspension and restart received by the user. The Directory Facilitator agent also has a GUI of its own, with which the DF can be administered, adding or removing agents and configuring their advertised services.

The three graphical tools with which JADE users can debug their agents are the Dummy Agent, the Sniffer Agent and the Debugger Agent.

The Dummy Agent is a simple, yet very useful, tool for inspecting message exchanges among agents. The Dummy Agent facilitates validation of an agent message exchange pattern before its integration into a Multi Agent System and facilitates interactive testing of an agent. The graphic interface provides support to edit, compose and send ACL messages to agents, to receive and view messages from agents, and, eventually, to save/load messages to/from disk.

The Sniffer Agent makes it possible to track messages exchanged in a JADE agent platform. When the user decides to sniff a single agent or a group of agents, every message directed to or coming from that agent or group is tracked and displayed in the sniffer window, using a notation similar to UML Sequence Diagrams. The user, who can also save and load every message track for later analysis, can examine every ACL message.

The Debugger Agent makes to trace the internal execution of each agent. In particular, it allows viewing the input message queue, the agent life cycle state and the behaviors running.

3 LEAP

The LEAP project developed a platform, called LEAP, to allow the seamless deployment of agents on all Java-enabled, connected devices ranging from cellular phones to enterprise servers (LEAP, 2000). In order to meet its goal, the LEAP project decided to start the development of its platform from JADE. LEAP is a new kernel for JADE that allows running legacy JADE agents on handy devices without any modification, provided that the device offers sufficient resources and processing power. When running on a device with no severe resource constraints, LEAP provides the same functionality as the original kernel of JADE. Summarizing, the following are the most important characteristics of LEAP:

1. It runs seamlessly on desktop PCs and on handy devices with limited resources, such as Palm Pilots and Java-enabled phones;

2. It adapts its functionality to the available resources in terms of memory, processing power, screen, etc.;

3. It guarantees connectivity to handy devices via wireless networks, like TCP/IP over GSM and GPRS, and IEEE 802.11 wireless LANs.

Figure 3 shows some pictures of LEAP running on different devices, with different connectivity.

The design of LEAP makes it sufficiently lightweight to execute on a handy device, but also sufficiently flexible and open to be a first-class choice for enterprise servers. LEAP can be deployed according to a set of profiles that identify the functionality available on each particular device. The basic profile supports only the functionality that FIPA requires and it suits the smallest device that we address, i.e., a cellular phone. The full-featured profile provides the functionality of an agent platform designed to run on desktop computers and it copes well with any device with sufficient memory and processing power. The choice of implementing LEAP as a lightweight and extensible kernel for JADE allows using the services that JADE offers across all profiles.

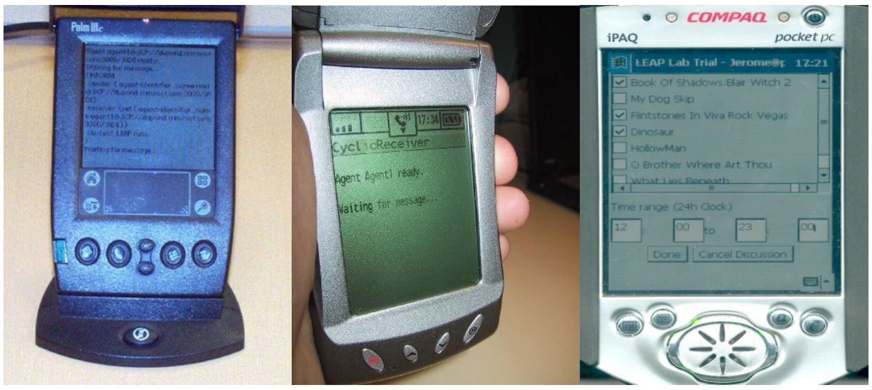

Figure 3. Three devices running LEAP.

In its current implementation, the basic profile provides the subset of the APIs of JADE that do not deal with the behaviour abstraction, and it does not integrate any run-time tool, such as the RMA or the Sniffer. Such limitations save memory and allow running LEAP on the current implementation of the KVM for Palm Pilots, which reserves only 200Kbytes of heap memory for Java applications. On the contrary, the full-featured profile integrates all tools that JADE provides and, from the developer point of view, it offers the same functionality and the same APIs of JADE. All profiles are instantiations of the FIPA abstract architecture and agents running on platforms configured with different profiles can interoperate. LEAP allows different locations of the same platform to be deployed according to different profiles.

The implementation of LEAP is basically different from that of JADE because the latter was not implemented taking into account the limitations of handy devices. The introduction of the concept of profile required a substantial redesign of the internals of JADE, but we succeeded in it without changing the APIs and agents developed for JADE can still run on LEAP. As a consequence, the community of JADE users can run existing applications on handy devices without any modification, provided that the device offers sufficient resources. Reasonably, such applications will need to spread the computation effort across the fixed infrastructure and the handy device. As applications are already decomposed into agents, the load balancing of the tasks is as simple as allocating agents to network nodes on the fixed and on the mobile network. These deployment issues are transparent to the developer as LEAP implements location transparency.

LEAP is not only meant for handy devices, it is also intended to support enterprise servers and we cannot constraint its functionality when running on devices with no limitations on resources. To achieve this, we decided to go for the worst case, i.e., J2ME CLDC (Sun Microsystems, 2000), and we implemented an extensible architecture over a layer of adaptation. Such a layer is capable of matching the classes available on J2ME CLDC with the ones available on the others Java 2 editions. This allows using the classes that express the maximum functionality where available and others with restricted functionality otherwise, without changing the source code of the agents.

LEAP is naturally distributed as it is composed of a number of locations that provide the run-time resources that agents need to execute. Mimicking the nomenclature of JADE, we say that locations contain agents and we call them agent containers, or simply containers. We impose no restrictions on the way containers can be deployed across the nodes of the network, but the best way of deploying the platform is having one container running in one Java virtual machine for every node. It is worth noting that splitting the platform into several containers distributed on different devices is not mandatory. According to the context and to the requirements of a given application, it is possible to concentrate the whole platform into a single container or to distribute it across the nodes of a network. For example, if the application consists simply of a personal assistant supporting the user in managing the information on her palmtop, probably the best deployment choice is to have a single-container platform running on the palmtop. On the contrary, given an application where each user in a team is assisted by an agent running on her mobile phone and all such agents interoperate, the choice of a distributed platform composed of one main container and several peripheral containers on the users' phones can be the best solution.

The nodes of the network can be of any kind, from cellular phones to enterprise servers, and they can access indifferently any transport mechanism for which a proper handler in the platform is available. At the moment, only handlers based on TCP/IP are available. Such handlers have been designed to support TCP/IP both over wired and wireless connections, such as GSM, GPRS and IEEE 802.11, exploiting the possibility of using full-duplex connections.

The choice of spreading the platform across the network poses some problems for agents running on other platforms to communicate with agents running on LEAP. Following the design of JADE, we solved this problem introducing a privileged container, called main container, to allow agents on other platforms to see the platform as a whole. The main container is unique and it acts as a front-end for the platform. It maintains platform-wide information and provides platform-wide services. This container must be always reachable by all active containers in the platform. A LEAP platform is composed of a single main container and a set of peripheral containers, allowing a high modularity by running lightweight peripheral containers on handy devices. The main container provides

FIPA mandatory services, i.e., the AMS and the DF, and it is mandatory to preserve FIPA compliancy. The amount of resources needed by the AMS and by the DF in their current implementation suggests that the main container should run on a full-featured computer.

4 CoMMA

CoMMA (Corporate Memory Management through Agents) is a FIPA compliant multi-agent system for the management of a corporate memory, implemented by using JADE (Gandon *et al.* 2002). It is the result of an international project funded by the European Commission (CoMMA, 2000). The project lasted two years, ending in January 2002, and it spanned two iterations of a complete development life cycle, including two field trials, carried out by end user project partners. The system was completely implemented and used in different companies to offer a helping service for enhancing the insertion of new employees and as a support system for technology monitoring.

The innovative aspect of the system is the integration of several emerging technologies that were generally used separately in the former information retrieval and management systems. These technologies are: agent technology, knowledge modeling, XML technology, information retrieval techniques and machine learning techniques.

The multi-agent approach, relying on loosely-coupled software components, is naturally prone to facilitate integration of different and heterogeneous technologies in one system. The CoMMA developers decided, therefore, to use agents for wrapping information repositories defining the corporate memory, for the retrieval of information, for enhancing scaling, flexibility and extensibility of the corporate memory and to adapt the system interface to the users. One of the points that makes CoMMA system different from the majority of the former multi-agent information systems is that agents are not only used for the retrieval of information, but also for the insertion of new information in the corporate memory.

The use of JADE increases system modularity and flexibility. The separation between the software platform infrastructure managing agent life-cycle, distribution and communication and the software implementing agent tasks decouples modifications in these two parts. The behavior based agent model offered by JADE allows to separate the software code realizing the different tasks of the agents; therefore, the modification of a task or the introduction of new tasks usually do not cause the modification of other parts of agent code

4.1 CoMMA Architecture

The system aims at helping users in the management of an organization corporate memory and in particular at facilitating the creation, dissemination, transmission and reuse of knowledge in an organization. The services offered by the CoMMA system are the result of three main tasks: insertion of XML annotations of new or updated documents, search of existing documents, and autonomous document delivery in a push fashion to provide her/him with information about new interesting documents (Figure 4 shows a schematic view of the CoMMA multi-agent system).

These tasks are performed through the cooperation among different kinds of agents that can be divided in four sub-societies: document and annotation management; ontology (enterprise and user models) management; user management; agent interconnection and matchmaking.

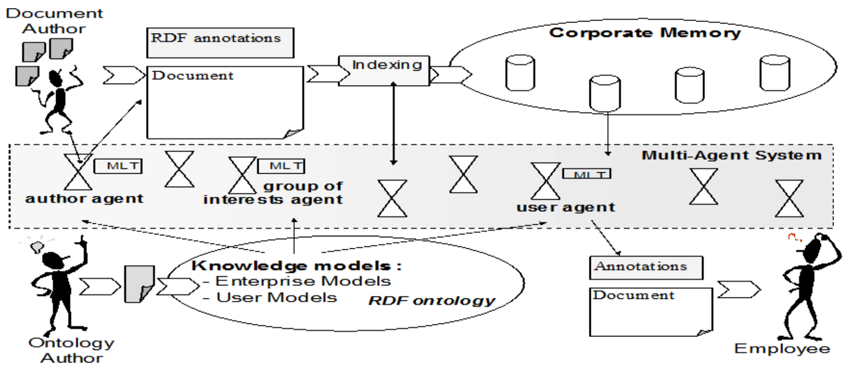

Figure 4. Schematic view of the CoMMA multi-agent system.

The agents belonging to the document dedicated sub-society are concerned with the exploitation of the documents and annotations composing the corporate memory, they will search and retrieve the references matching the query of the user with the help of the onto-logical agents. A hierarchical organization for the document sub-society has been chosen since separates the task of maintaining document repositories from the task of intelligent interface towards the other agents of the system.

The agents belonging to the ontology dedicated sub-society are concerned with the management of the ontological aspects of the information retrieval activity, especially the queries about the hier-archy of concepts and the different views. The ontology repository, composed of RDF schema forms, maintains a set of concepts and their relationships. Documents of the community are annotated us-ing these ontologies and ontologies are used to search documents into the corporate memory and to navigate into it. In particular, the CoMMA ontology describes the documents maintained in the or-ganization corporate memory and the enterprise model describes the structure of the organization ruling, for example, the access to the different type of documents of the corporate memory. A replicated organization for the ontology sub-society has been chosen since ontologies shared by users should be quite stable and most of the queries will need the whole ontology to apply inference algorithms.

The agents belonging to the user dedicated sub-society are concerned with the interface, the monitoring, the assistance and the adaptation to the user. Moreover, they have to maintain the user profile repository and distribute information about user profile to the agents needing it.

Finally the agents belonging to the interconnection dedicated sub-society are in charge of the matchmaking of the other agents based upon their respective needs.

5 Agentcities

Agentcities is a network of FIPA compliant agent platforms that constitute a distributed environment to demonstrate the potential of autonomous agents. It started on the second half of 2001 as a research project funded by the European Commission (Agentcities, 2001).

One of the aims of the project is the development of a network architecture to allow the integration of platforms based on different technologies and models. It provides basic white pages and yellow pages services to allow the dynamic discovery of the hosted agents and the services they offer.

An important outcome is the exploitation of the capability of agent-based applications to adapt to rapidly evolving environments. This is particularly appropriate to dynamic societies where agents act as buyers and sellers negotiating their goods and services, and composing simple services offered by different providers into new compound services.

To allow the integration of different applications and technologies in open environments, high level communication technologies are needed. The project largely relies on semantic languages, ontologies and protocols in compliance with the FIPA standards.

5.1 Network

The Agentcities network grows around a backbone of 14 agent platforms, mostly hosted in Europe. These platforms are deployed as a testbed, hosting the services and the prototype applications developed during the lifetime of the project. The backbone is an important resource for other organizations, even external to the project, that can connect their own agent-based services, making the network really open and continuously evolving.

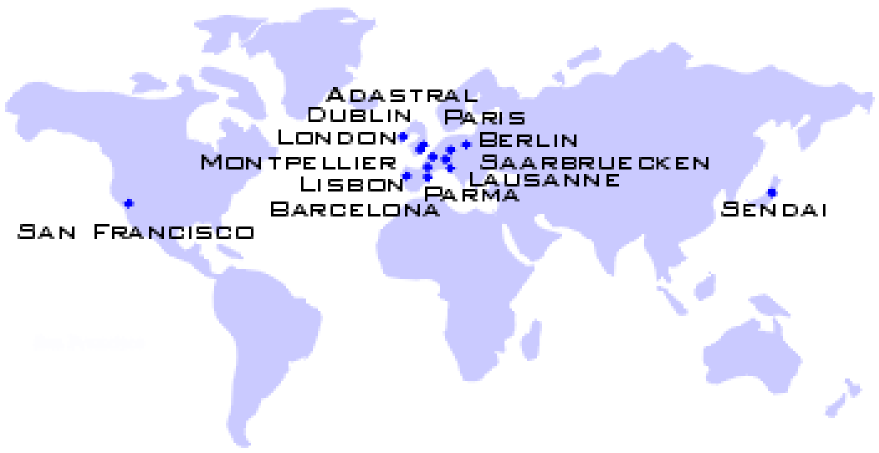

Figure 5. The Agentcities backbone.

Currently, the Agentcities network counts 160 registered platforms, among which 80 have shown activities in the last few weeks. The platforms are based on more than a dozen of heterogeneous technologies, including Zeus (Nwana et al., 1998), FIPA-OS (Poslad et al., 1999), Comtec (Comptec, 1999), AAP (AAP, 1999), Opal (Purvis et al., 2002). More than 2/3 of them are based on JADE and its derived technologies, as LEAP (LEAP, 2000) and BlueJADE (BlueJADE, 2001)

5.2 Service Composition

The main rationale for using agents is their ability to adapt to rapidly evolving environments and yet being able to achieve their goals. In many cases, this can only be accomplished by collaborating with other agents and leveraging on the services provided by cooperating agents. This is particularly true when the desired goal is the creation of a new service to be provided to the community, as this scenario often calls for the composition of a number of simple services that are required to create the desired compound service.

The Event Organizer is an agent-based prototype application showing the results that can be achieved using the services provided by the Agentcities project. It allows a conference chair to organize an event, booking all needed venues and arranging all needed services, and then sell the tickets for the new event.

Using the web interface of the Event Organizer, its user can list a set of needed services, fixing desired constraints on each individual service and among different services. The global goal is then split into sub-goals, assigned to skilled solver agents.

The Event Organizer uses the marketplace infrastructure deployed on the Agentcities network to search for relevant venues. These are matched against cross-service constraints and, if found, a proper solution is proposed to the user as a list of services that allow the arrangement of the event. These services are then negotiated on the marketplace with their providers and a list of contracts is returned to the user. Finally, when the new event is successfully organized, the tickets for it can be sold, once again using the marketplace infrastructure.

The process requires the cooperation of a number of partners. Each of them can exploit the directory services to dynamically discover the location of the others. The Event Organizer directly interacts with a Trade House to search for venues and negotiate selected services. Other agent-based applications, as the Venue Finder and the SME Access, are responsible to offer goods on the Trade House and to negotiate them on behalf of their users. A Banking Service takes care of managing the banking accounts of the involved partners, securing all requests against tampering and eavesdropping. An Auction House is used to create auctions and sell the tickets of the new event.

The interesting part of the game is that these tickets are available for other agent-based applications. In fact an Evening Organizer, that helps its user to arrange an evening out, for example booking a restaurant and buying the tickets for a concert, can discover the new event and bid for some tickets on the Auction House.

6 Conclusions

In this chapter, we presented JADE (Java Agent DEvelopment framework), a software framework to aid the development of agent applications in compliance with the FIPA2000 specifications for interoperable intelligent multi-agent systems.

JADE is written in Java language and comprises various Java packages, giving application programmers both ready-made pieces of functionality and abstract interfaces for custom, application dependent tasks. Java was the chosen programming language because of its many attractive features, which are particularly geared towards object-oriented programming in distributed heterogeneous environments.

Starting from the FIPA assumption that only the external behaviour of system components should be specified, leaving the implementation details and internal architectures to agent developers, we produced a very general but primitive agent model that can serve as a useful basis to implement, for example, reactive or BDI architectures. In addition, the behaviour abstraction of our agent model permits an easy integration of external software. For example, we created a JessBehaviour that makes it possible to use JESS (Friedman-Hill, 1998) as agent reasoning engine. In comparison to the agent development tools introduced in the previous section, JADE offers a more efficient implementation and a more general agent model. Such an agent model is more "primitive" than the agent models offered, for example, by RETSINA (Sycara et al., 1996), however, the overhead given by such sophisticated agent models might not be justified for agents that have to perform some simple tasks. In addition, sophisticated agent models such as BDI and reactive architectures, as previously mentioned, can be implemented on top of our "primitive" agents model.

JADE is not the only FIPA-compliant software development system: different systems have been realized or are under development (see, for example AAP (AAP, 1999), ASL (Kerr et al., 1998), Beegent (Kawamura et al., 1999), Comtec (Comtec, 1999), FIPA-OS (Poslad, 1999), Opal (Purvis et al., 2002), Zeus (Nwana et al., 1998)). However, JADE seems the most appreciate and used; in fact, in agent platforms network that involved more than 160 nodes more than 2/3 are realized on the top of JADE or on the top of derived platforms, i.e., BlueJADE (BlueJADE, 2001) and LEAP (LEAP, 2000).

Acknowledgments

Thanks to all the people that contributed to development of JADE and to all the partners of the EC projects introduced in the paper. This work is partially supported by TILAB, Torino and by the European Commission through the contracts IST-1999-12217 - CoMMA - Corporate Memory Management through Agents and IST-1999-10211 – LEAP – Lightweight Extensible Agent Platform and IST-2000-28385 - Agentcities.RTD.

References

AAP (1999), AAP Project Home Page", available at http://sf.us.agentcities.net/aap.

Agentcities, (2001), "Agentcities Project Home Page", available at http://www.agentcities.org.

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

BlueJADE, (2001), "BlueJADE Project Home Page", available at https://sourceforge.net/projects/bluejade.

CoMMA, (2000), "CoMMA Project Home Page", available at http://www.ii.atosgroup.com/sophia/comma/HomePage.htm.

Comtec (1999), "Comtec Project Home Page", available at http://ias.comtec.co.jp/ap.

FIPA, (1997), "FIPA Organization Home Page", available at http://www.fipa.org.

Gandon, F., Poggi, A., Rimassa, G., and Turci, P. (2002), "Multi-Agents Corporate Memory Management System*," Applied Artificial Intelligence,* 9-10 (22): pp. 699-720.

Friedman-Hill, E.J., (1998), "Java Expert System Shell, available at http://herzberg.ca.sandia.gov/jess.

JADE, (1999), "JADE Project Home Page", available at http://jade.cselt.it.

Kawamura, T., Yoshioka, N., Hasegawa, T., Ohsuga, A., Honiden, S., (1999), "Bee-gent: Bonding and Encapsulation Enhancement Agent Framework for Development of Distributed Systems", in Proc. of the 6th Asia-Pacific Software Engineering Conference.

Karamcheti, V., Plevyak, J., Chien, A., (1996), "Runtime Mechanisms for Efficient Dynamic Multithreading", Journal of Parallel and Distributed Computing, 37:21-40.

Kerr, D., O'Sullivan, D., Evans, R., Richardson, R., Somers, F., (1998), "Experiences using Intelligent Agent Technologies as a Unifying Approach to Network and Service Management", in Proc. of IS&N 98, Antwerp, Belgium.

LEAP, (2000), "LEAP Project Home Page", available at http://leap.crm-paris.com.

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. (1998), "MASIF - The OMG Mobile Agent System Interoperability Facility", in K. Rothermel and F. Hohl, Eds. Proc. 2nd Int. Workshop Mobile Agents, pp. 50-67, Springer.

Nwana, H.S., Ndumu, D.T., Lee, L.C., (1998), “ZEUS: An advanced Tool-Kit for Engineering Distributed Multi-Agent Systems”, in Proc of PAAM98, pp. 377-391, London, U.K.

Patil, R.S., Fikes, R.E., Patel-Scheneider, P.F., McKay, D., Finin, T., Gruber, T., Neches, R., (1992), “The DARPA knowledge sharing effort: progress report”, in: Proc. Third Conf. on Principles of Knowledge Representation and Reasoning, pp 103-114. Cambridge, MA.

Poslad, S., Buckle, P. and Hadingham, R. (1999), “The FIPA-OS Agent Platform: Open Source for Open Standards”, available at http://fipa-os.sourceforge.net.

Purvis, M., Cranefield, S., Nowostawski, M., Ward, R., Carter, D., and Oliveira, M.A., (2002), "Agentcities Interaction Using the Opal Platform", in Proc. of the Workshop – Agentcities: Research in Large-Scale Open Agents Environments, AAMAS 2002, Bologna, Italy.

Somacher, M., Tomaiuolo, M., Turci, P. (2002), “Goal Delegation in Multiagent System”, Proceedings of the AIIA 2002, Siena, Italy.

Sun Microsystems, (2000), “Java 2 Platform Micro Edition (J2ME) Technology for Creating Mobile Devices”, available at http://java.sun.com.

Sycara, K., Pannu, A., Williamson, M., Zeng, D., (1996), “Distributed Intelligent Agents”, IEEE Expert, 11(6):36-46.