Middleware and Programming Support for Agent Systems

Federico Bergenti, Agostino Poggi,
Giovanni Rimassa and Paola Turci
Dipartimento Ingegneria dell'Informazione
Università degli Studi di Parma
Parco Area delle Scienze, Parma, Italy
email: {bergenti,poggi,rimassa,turci}@ce.unipr.it

Fabio Bellifemine
Telecom Italia Lab
Via G. Reiss Romoli, 274
10148 Torino, Italy
fabio.bellifemine@tilab.com

Abstract

JADE (Java Agent Development Framework) is a software environment to build agent systems for the management of networked information resources in compliance with the FIPA2000 specifications for interoperable intelligent multi-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 paper describes the main features of the JADE system and introduces some of the most important projects based on the JADE software.

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 this period, there is a lot of activity concerning the development of agent systems (see, for example, Bee-gent [Kawamura et al., 1999], MOLE [Baumann et al., 1998] and RETSINA [Sycara et al., 1996]), and the standardization of these systems, where in the past KSE [Patil et al., 1992], OMG [Milojicic et al., 1998] and then FIPA [FIPA, 2000] led the activity.

Even if there is no system accepted as being considerably superior to the others. At this point in time, the FIPA standardisation effort is predominant and more and more systems based on the FIPA specifications are available.

Such environments provide some predefined agent models and tools to ease the development of systems. Moreover, some of them try to inter-operate with other agent systems through a well-known agent communication language (e.g., KQML [Finin and Labrou, 1997]).

However, a shared communication language is not the only element required to support interoperability between different agent systems, common agent services and ontologies are also needed. The standardisation work of FIPA acknowledges this issue and, in addition to an agent communication language, 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), a software framework to write agent applications in compliance with the FIPA2000 specifications for interoperable intelligent multi-agent systems. The next section introduces the FIPA specifications. Section three presents the main features of JADE. Section four describes some projects using JADE. Finally, the last section concludes with a brief discussion about JADE features.

2 FIPA

The Foundation for Intelligent Physical Agents (FIPA) [FIPA, 2000] 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 FIPA standardization process relies on two main assumptions. The first is that the time required to reach a consensus and to complete the standard should be as short as possible, should not impede progress, but should act as a promoter of stronger industrial commitment in agent technology. The second assumption is that only the external behaviour of system components should be specified, leaving implementation details and internal architectures to platform developers. In fact, the internal architecture of JADE is proprietary even if it complies with the interfaces specified by FIPA.

Since 1997, FIPA yearly release a set of specifications, some of them modify the previous ones, some others are new specifications that try to cover some aspects that previous specifications did not cope with. Current FIPA specifications are called FIPA2000. These specifications revised and abstracted an agent platform, improved agent communication, and extend the transport layer to support mobile and wireless applications.

FIPA specifications state the normative rules that allow a society of agents to exist, operate and be managed.

They identify the roles of some key agents necessary for managing the platform, and describe the agent management content language and ontology. Two mandatory roles were identified for an agent platform. The Agent Management System (AMS) is the agent that exerts supervisory control over access to and use of the platform; moreover, it is responsible for maintaining a directory of resident agents and for handling their life cycle. The Directory Facilitator (DF) is the agent that passes on yellow page services to the agent platform. Notice that no restriction is given to the actual technology used for platform implementation: an e-mail based platform, a CORBA based one, a Java multi-threaded application, etc. could all be FIPA compliant implementations.
According to FIPA definition, an agent is the fundamental actor in a domain. It is capable of bringing together a number of service capabilities to form a unified and integrated execution model that can include access to external software, human users and communication facilities.

Of course, the specifications also define the Agent Communication Language (ACL). Agent communication is based on message passing, where agents communicate by sending individual messages to each other. The FIPA ACL is a standard message language and sets out the encoding, semantics and pragmatics of the messages. It does not set out a specific mechanism for the transportation of messages. Since different agents might run on different platforms and use different networking technologies, the messages are encoded in a textual form. It is assumed that the agent has some means of transmitting this textual form.

The ACL provides the bases for the specification of interaction protocols, common patterns of conversation between agents aimed at specifying high-level tasks, such as delegating a task, negotiating conditions, and some forms of auctions.

FIPA supports common forms of inter-agent conversations through the specification of *interaction protocols*, which are patterns of messages exchanged by two or more agents. Such protocols range from simple query and request protocols, to more complex ones, such as the well-known contract net negotiation protocol and English and Dutch auctions.

The remaining parts of the FIPA specifications deal with other aspects, in particular with agent-software integration, agent mobility, agent security, ontology service, and human-agent communication; however, these specifications are either not complete or have a limited experimentation.

3 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].

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. The two following subsections will present JADE from the two standpoints, trying to highlight the major design choices followed by the JADE development team.

3.1 Middleware

JADE communication 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 runtime environment support to some JADE agents. The JADE middleware tries to provide efficient and flexible messaging services to user applications.

The previous implementation of JADE [Bellifemine et al., 1999] has been modified according to the new FIPA 2000 specifications: 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* (*MTP*s). Currently, FIPA 2000 has defined MTPs for IIOP, HTTP and WAP protocols, so interoperability can be achieved exploiting any among the three protocols above.

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 uses Java RMI for intra-platform communications, but a new kernel is being developed in the scope of the LEAP project [LEAP, 2000], which will support different transport protocols for intra-platform messaging.

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:

- Multiple MTPs, deployed as plug-ins on multiple containers.
- One hop message routing for outgoing and incoming messages.
- 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.

3.2 Agent Model

FIPA specifications state nothing about agent internals, but when JADE was designed and built they had to be addressed. A major design issue is the execution model for an agent platform, both affecting performance and imposing specific programming styles on agent developers. As will be shown in the following, JADE solution stems from the balancing of forces from ordinary software engineering guidelines and theoretical agent properties.

A distinguishing property of a software agent is its *autonomy*; an agent is not limited to react to external stimuli, but it's also able to start new communicative acts of its own. A software agent, besides being autonomous, is said to be *social*, because it can interact with other agents in order to pursue its goals or can even develop an overall strategy together with its peers.

FIPA standard bases its *Agent Communication Language* on *speech-act theory* [Searle, 1970] and uses a mental model to build a formal semantic for the *performatives* agents exchange. This approach is quite different from the one followed by distributed objects and rooted in *Design by Contract*; a fundamental difference is that invocations can either succeed or fail but a *request* speech act can be refused if the receiver is unwilling to perform the requested action.

Trying to map the aforementioned agent properties into design decisions, we can say:

- Agents are autonomous then are active objects.
- Agents are social then intra-agent concurrency is needed.
- Agent messages are speech acts then asynchronous messaging must be used.
- Agents can say "no" then peer-to-peer communication model is needed.

The autonomy property requires each agent to be an *active object* with at least a Java thread, to proactively start new conversations, make plans and pursue goals. The need for sociality has the outcome of allowing an agent to engage in many conversations simultaneously, dealing with a significant amount of concurrency.

The third requirement suggests asynchronous message passing as a way to exchange information between two independent agents that also has the benefit of producing more reusable interactions. Similarly, the last requirement stresses that in a Multi Agent System the sender and the receiver are equals (as opposed to client/server systems where the receiver is supposed to obey the sender). An autonomous agent should also be allowed to ignore a received message as long as he wishes; this advocates using a *pull consumer*-messaging model, where incoming messages are buffered until their receiver decides to read them.

The above considerations help in deciding how many threads of control are needed in an agent implementation; the autonomy requirement forces each agent to have at least a thread, and the sociality requirement pushes towards many threads per agent. Unfortunately, current operating systems limit the maximum number of threads that can be run effectively on a system. JADE execution model tries to limit the number of threads and has its roots in actor languages.

The *Behaviour* abstraction models agent tasks: a collection of behaviours 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 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.

Choosing not to save behaviour execution context means that agent behaviours start from the beginning every time they are scheduled for execution. So, behaviour state that must be retained across multiple executions must be stored into behaviour instance variables. A general rule for transforming an ordinary Java method into a JADE behaviour is:

- *Turn the method body into an object whose class inherits from Behaviour.*
- *Turn method local variables into behaviour instance variables.*
- *Add the behaviour object to agent behaviour list during agent startup.*

The above guidelines apply the *reification technique* [Johnson and Zweig, 1991] to agent methods, according to *Command* design pattern; an agent behaviour object reifies both a method and a separate thread executing it. A new class must be written and instantiated for each agent behaviour, and this can lead to programs harder to understand and maintain. JADE application programmers can compensate for this shortcoming using Java *Anonymous Inner Classes*; this language feature makes the code necessary for defining an agent behaviour only slightly higher than for writing a single Java method.

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.

3.3 Ontology Support

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:

- Object, construct that represents an identifiable entity; this is mainly important to realize a typed knowledge base.
- Proposition, e.g. the content of an “inform” communicative-act is a predicate (a subtype of proposition).
- Action, e.g. in the “request” communicative-act, tries to express an activity that can be carried out by an object.
- IRE (Identifying Reference Expression) , e.g. in the “query-ref” communicative-act.

This model is composed of:

- 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.).
- 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, …), different content language concrete syntaxes (e.g. RDFS, SL-expression), different ontologies, and different ontology concrete representations (RDFS, SL-expression).

3.4 Management and Testing Tools

In addition to a runtime library and an agent programming library, JADE 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 contain-

ers, 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.

4 Applications

Even if JADE is a young project and was designed with criteria more academic than industrial, and even if it has been released under an Open Source License only recently, it has already been used by a number of projects. In particular, it has been used and is used in four projects sponsored by the European Commission: FACTS [FACTS, 1998], LiMe [LiMe, 1999], LEAP [LEAP, 2000], CoMMA [CoMMA, 2000] and Agentcities.RTD [Agentcities, 2001].

FACTS [FACTS, 1998] is a project in the framework of the ACTS programme of the European Commission that used JADE in two application domains. In the first application domain, JADE provides the basis for a new generation TV entertainment system. The user accesses a multi-agent system to help him on the basis of his profile that is captured and refined over time through the collaboration of agents with different capabilities. The second application domain deals with agents in collaboration, and at the same time competing, in order to help the user to purchase a business trip. A Personal Travel Assistant represents the interests of the user and cooperates with a Travel Broker Agent in order to select and recommend the business trip.

LiMe [LiMe, 1999] is a Long Term Research Programme of the European Commission under its `I-cubed' (Intelligent Information Interfaces) programme. The goal of the project was to create a multi-agent system for the enhancement of social interaction within connected communities. JADE has successfully supported the LiMe communicating agents for dynamic user profiling, collective information dissemination and memory management for a 2-day field trial.

LEAP [LEAP, 2000] is a going project in the framework of the IST programme of the European Commission. This project is addressing the need for open infrastructures and services that support dynamic, mobile enterprises and will extend JADE to support mobile and wireless applications. LEAP develops agent-based services supporting three requirements of a mobile enterprise workforce: Knowledge management (anticipating individual knowledge requirements), decentralized work co-ordination (empowering individuals, coordinating and trading jobs), travels management (planning and coordinating individual travel needs).

CoMMA [CoMMA, 2000] is a going project in the framework of the IST programme of the European Commission that is using JADE 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. The main objective of this project is to implement and test a Corporate Memory Management framework based on agent technology. The innovative aspect of the project is to integrate several emerging technologies that were generally used separately until now: agent technology, knowledge modeling, XML technology, information retrieval techniques and machine learning techniques. Integration of these technologies in one system is already a challenge yet another challenge is the definition of the methodology supporting the whole design process. The project intends to implement the system in the context of two scenarios (1) the insertion of *new employees* in the company and (2) the *technology monitoring*.

Agentcities.RTD [Agentcities, 2001] is a going project in the framework of the IST programme of the European Commission. The project's objectives are to create an on-line, distributed testbed to explore and validate the potential of agent technology for future dynamic service environments. The project aims to produce the following important results: i) an open, stable, scalable and reliable network architecture that allows standards compliant agents to discover each other, communicate and offer services to one another; II) models, methodology and prototype solutions for the integration of business services into the service environment; and III) practical methodologies for the application of agent communication technologies (semantic models, ontology, expression of content and protocols) to service modeling in open heterogeneous environment.

5 Conclusions

In this paper 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.

The development of JADE has not yet terminated. Our intention is, initially, to introduce additional support for Web technology integration and to provide some new tools to ease the development of agent systems such as, for example, a visual tool to compose agent behaviours, and to offer some higher level agent architecture as, for example, BDI architecture.

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

[Agentcities, 2001] Agentcities.RTD Home Page. Available at http://www.agentcities.org/EURTD/.

[Baumann et al., 1998] J. Baumann, F. Hohl, K. Rothermel and M. Straßer. Mole - Concepts of a Mobile Agent System, World Wide Web,1(3):123-137, 1998.

[Bellifemine et al., 1999] F. Bellifemine, A. Poggi and G. Rimassa. JADE - A FIPA-compliant Agent Framework. In Proc. Fourth International Conference on the Practical Application of Intelligent Agent and Multi Agent Technology (PAAM99), pp. 97-108, London, UK, 1999.

[CoMMA, 2000]CoMMA Home Page. 2000. Available at http://www.ii.atos-group.com/sophia/comma/HomePage.htm.

[FACTS, 1998] FACTS Home Page. Available at http://www.labs.bt.com/profsoc/facts/.

[Finin and Labrou, 1997] T. Finin and Y. Labrou. KQML as an agent communication language. In: J.M. Bradshaw (ed.), Software Agents, pp. 291-316. MIT Press, Cambridge, MA, 1997.

[FIPA, 2000] Foundation for Intelligent Physical Agents. Specifications. 2000. Available at http://www.fipa.org.

[Friedman-Hill, 1998] E.J. Friedman-Hill. Java Expert System Shell. 1998. Available at http://herzberg.ca.sandia.gov/jess.

[JADE, 1999] JADE Home Page, 1999. Available at http://jade.cselt.it.

[Johnson and Zweig, 1991] R.E. Johnson and J.M. Zweig. Delegation in C++. The Journal of Object Oriented Programming, 4(7):31-34, 1991.

[Karamcheti et al., 1996] V. Karamcheti, J. Plevyak and A. Chien. Runtime Mechanisms for Efficient Dynamic Multithreading. Journal of Parallel and Distributed Computing, 37:21-40, 1996.

[Kawamura et al., 1999] T. Kawamura, N. Yoshioka, T. Hasegawa, A. Ohsuga and S. Honiden. Bee-gent : Bonding and Encapsulation Enhancement Agent Framework for Development of Distributed Systems. Proceedings of the 6th Asia-Pacific Software Engneering Conference, 1999.

[LEAP, 2000] LEAP Home Page. 200. Available at http://leap.crm-paris.com/.

[LiMe, 1999] LiMe Project Page. Available at http://www.ee.ic.ac.uk/tour/ResearchSections/IntelligentCommunications/lime.html.

[Milojicic et al., 1998] D. Milojicic, M. Breugst, I. Busse, J. Campbell, S. Covaci, B. Friedman, K. Kosaka, D. Lange, K. Ono, M. Oshima, C. Tham, S. Virdhagriswaran, and J. White. 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, 1998.

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

[Searle, 1970] J. Searle. Speech Acts: An Essay in the Philosophy of language. Cambridge Univ. Press, 1970.

[Sycara et al., 1996] K. Sycara, A. Pannu, M. Williamson and D. Zeng. Distributed Intelligent Agents. IEEE Expert, 11(6):36-46. 1996.