

# Open Source, Standards and Scaleable Agencies

Stefan Poslad  
Imperial College of Science,  
Technology and Medicine, Exhibition  
Road, London, SW7 2BZ, UK.  
+44 (0)171 594 6319  
s.poslad@ic.ac.uk

Phil Buckle  
Nortel Networks,  
London Road, Harlow,  
Essex, CM17 9NA, UK.  
+44 (0)1279 402637

Rob Hadingham  
Nortel Networks,  
London Road, Harlow,  
Essex, CM17 9NA, UK.  
+44 (0)1279 402775

pbuckle@nortelnetworks.com

rgh@nortelnetworks.com

## ABSTRACT

Numerous agencies and agent systems are being developed or portrayed as vehicles to deliver novel types of e-commerce services to users. However service agents in one agency are probably unable to interoperate or co-operate with agents from other vendors' agencies. Clearly, standardization in this area would help to create a more ubiquitous market for agent-based services. We analyze the relevant standards for agents, highlighting the FIPA (Foundation for Intelligent Physical Agents) agent specification standards. Standard specifications ought to be grounded within a practical framework, which provides a reference implementation, enabling a multitude of developers to build their own implementations. We describe an open agent platform called FIPA-OS (FIPA Open Source), originating from Nortel Networks, which provides such a framework to promote the uptake of FIPA specifications by agent developers. FIPA-OS was the first agent platform to be released as open source and is being deployed in several application domains including virtual private network provisioning, distributed meeting scheduling and virtual home environments. It has been demonstrated to interoperate with other heterogeneous FIPA compliant platforms and is in use in numerous institutes around the world.

## Keywords

Software Agents, FIPA, standards, scalability, open-source, multi-agent platform

## 1. INTRODUCTION

Multi-agent (MA) systems are a powerful, relatively new paradigm for accessing and integrating heterogeneous distributed services. Here, service providers, services users and intermediaries are represented as autonomous software agents, which interact using an agent communication language, based on speech acts [1] and shared vocabularies called ontologies [2]. The use of the agent communication language enables agents to

interact using semantically rich information.

In a heterogeneous world, different vendors are bound to design, develop and realize this vision of agents in different ways - at least in the initial research phase. Following the research phase, different vendors seek to differentiate themselves from their competition by emphasizing the novelty of their design and applications. This concurrent distributed development leads to many types of multi-agent systems that are islands of functionality - agents on different types of platform are unable to interoperate. Agents from different vendors are likely to use different types of messages and message formats and the meaning and interpretation of the content is likely to differ. For example, an agency that provides public transport agent services may not be able to interact with accommodation service agents in another agency. The driving force for interoperability is partly the customer who strives for simplicity and universality when accessing multiple services, and partly producers who often need to act in unison to obtain a critical mass for a sustainable customer-base.

Early adopters, who produce new technology and services, tend to be wary where there is no commonly agreed standard for interoperability and of a standard that lacks the support of a large consortium of companies. The standardization process helps shift the emphasis from the development of the infrastructure to the use of the infrastructure.

However, there is inherent high complexity, cost and risk in developing a good standard that is practical, and in developing systems that adhere to the standard. These act as a barrier to entry to users and developers who wish to assess and use this new technology. Clearly, reference implementations are useful here and there seems to be (at least) two main approaches to develop these. Essentially, reference implementations can be developed under an open license or a closed license. In order to address some of these issues of developing standard agent infrastructures, Nortel Networks were first to release an in-house developed agent platform, under a full open source license that supports the FIPA agent standards.

The rest of this paper is organized as follows: Section 2 continues with a discussion of issues in standardizing agent systems. In particular, there is a close look at the FIPA agent specifications for multi-agent systems. Section 3 examines the issues in implementing the FIPA agent specifications with particular to the first 'reference implementation' of the FIPA specifications released as open-source. Section 4 presents conclusions.

*LEAVE BLANK THE LAST 3.81 cm (1.5")  
OF THE LEFT COLUMN ON THE FIRST PAGE  
FOR THE COPYRIGHT NOTICE*

## 2. AGENT STANDARDS

Before discussing particular, de facto, agent interoperability standards, it is worth thinking about the more general issues in developing, disseminating and implementing agent standard specifications. These issues can be summarized as drivers, timing, viability, dependencies and competition.

The main driving force for de facto standards, from multiple-vendor forums such as OMG and FIPA, is to seed the market and attain a critical mass of customers, applications and products in the medium and long-term. The seeding of the market is enhanced when the standards are publicly available - they are easier to be taken up by non-members. In standardizing, companies potentially give up a competitive edge in the short-term goal but regain it in the medium to long term by producing products that add value to and enhance the standard in a much larger market.

Standards need to be developed at the right time. The development of new products and markets seems to follow a double peak of activity with a dip in between [3]. The optimum time to standardize is towards the end of the first (research) peak or in the dip between the peaks, before the second (development) peak occurs. Standardizing too early before the research is finished can lead to immature, bad standards whereas standardizing too late, once companies have made significant investments can mean the standards are ignored.

A standard needs to be viable. Both FIPA and KQML seek to establish or fix a generic core, for example, the set of speech acts in the agent communication language. There are pros and cons to this. Generic patterns do exist at the right level of abstraction and that the existence of a generic core can ease interoperability and interaction. A common core may well be a compromise and this may only be realizable after extensive experimentation but this is only natural - robust standards take time to mature. The argument against is that it can be difficult to define a common denominator, e.g., communication between heterogeneous parties in a wide range of heterogeneous domains, is very diverse. This argument has also been applied against the standardization of core facilitator agents such as directory services and management service agents. Certainly within the FIPA and KQML communities, there is a steady stream of requests to add or enhance speech acts, add or enhance interaction protocols and add or enhance the representation, syntax and semantics of the speech acts. Other alternatives to the core model approach are to define model specifications that are extensible or to define an abstract model that can be used to define more specific instances and extensions of the model. FIPA uses a combination of these techniques.

Standard specifications rarely consist of complete vertical slices of infrastructure, from provider via some infrastructure, to the user. Often specifications are dependent on other horizontal layers, e.g. they may use an existing software infrastructure. Key issues here are the scope of the specification, how much

leverage to make from existing technology and how to link to an existing infrastructure. The scope of the MA specifications generally includes the interpretation and handling of ACL messages, the use of facilitator agents, and the use of but not the actual specification of: 'message transport' protocols, agent life-cycle management and message persistence schema, to underpin agent communication. There is some controversy as to whether or not the use of externally specified services such as a message transport, directory (facilitator) services ought to be modeled as full agent services, proxy agent services or non-agent services accessed via some internal interfaces (see below).

Standards are not developed in a vacuum. There are often several overlapping standards bodies. If standards are not developed quickly enough for the market in one standards body, they are in danger of superseded by work in another body. Often standards seek to 'piggy-back' on existing popular standards. For example, in 1997, CORBA and OMG's Internet Inter-ORB Protocol (IIOP) showed great promise and FIPA specified it as its mandatory transport protocol. The next section examines the main de facto agent standards.

### 2.1 Overview of agent standards

There are three important agent standardization efforts that define interoperability between agents on different types of agent platform: KQML community, OMG's MASIF and FIPA. There are also other relevant communication standards initiatives that are not primarily agent-centric, e.g., MPEG-21 and XML, these are not considered further here.

Of the previous three: KQML and FIPA both define interaction in terms of an Agent Communication Language (ACL) whereas MASIF defines interaction in terms of Remote Procedure Calls (RPC) or Remote Method Invocation (RMI). In contrast to MASIF, both KQML and FIPA emphasize agency and social interaction between multi-agents as the defining properties for software agents.

Before examining MASIF, KQML, and FIPA. it is worth, outlining the difference between the ACL approach that is semantically rich and another approach such as RMI (or XML) that focuses on data syntax and encoding (Figure 1). Consider how a new service agent registers itself with a directory facilitator at the ACL level. It issues a register action. This register action is within the context of a service request rather than an inform or query (performative). The request message is sent within the context of a conversation or pattern of messages, the request is followed by an acknowledge (agree) and then by a inform about the results of the register or a refuse or failure message. In addition, the register action is associated with a service description. At the RMI level, the ACL is modeled (or implemented) in terms of an interface that focuses on syntax and data representation rather than on semantics. The RMI model defines a push type of interaction in terms of a message method, parameter types and return data types and an Object ID to access the method.

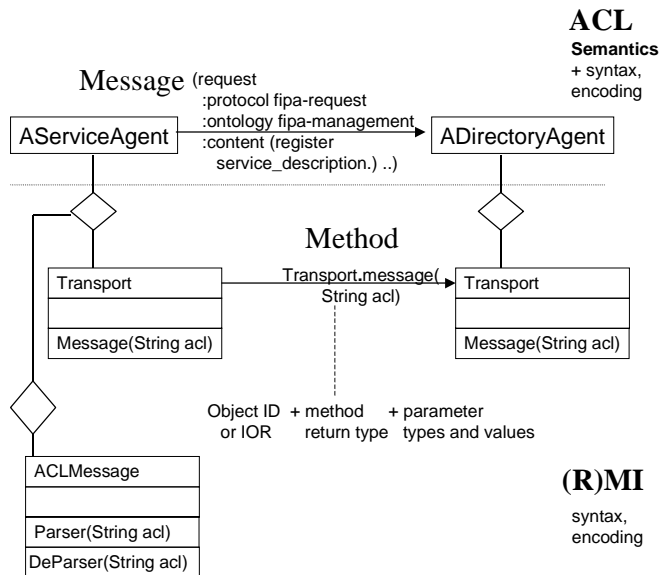


Figure 1. ACL Vs. (R)MI specifications.

## 2.2 OMG MASIF

The OMG or Object Management Group's Mobile Agent System Interoperability Facility, MASIF [4], differs from both KQML and FIPA in that it regards the defining characteristic for an agent as its mobility from one location to another. MASIF does not support or standardize communication between agents on different agent platforms. OMG is exploring how to support other characteristics of software agent than mobile agents. It has issued a 'Request For Information' (RFI) on agents. FIPA has supplied its specifications as input to this request. This is still work in progress at this time.

## 2.3 KQML

KQML or Knowledge Query Meta Language was one of the first initiatives to specify how to support the social interaction characteristic of agents using a protocol based on speech acts. KQML was developed at UMBC by Tim Finin et al [5] and has spread throughout the academic community. KQML however isn't a true de facto standard in the sense that there is no consensus on a single specification (or set of specifications) that it has been ratified by common agreement within an organization or forum of some standing in the community. As a result, variations of KQML exist such as KQML classic, KQML '93 and KQML-Lite, leading to different agent systems that speak different dialects and that are not able to interoperate fully.

## 2.4 FIPA

The Foundation for Intelligent Physical Agents (FIPA), [6] is a non-profit standards organization established in 1996 and registered in Geneva, Switzerland. Its purpose is to promote the development of specifications of generic agent technologies that maximize interoperability within and across agent based applications. Part of its function is to produce a specification for an agent enabling software framework. Contributors are free to produce their own implementations of this software framework as long as its construction and operation complies with the

published FIPA specification. In this way the individual software frameworks are interoperable.

FIPA specifications have been released in three phases:

- 1997: FIPA97v1 specifications (7)
- 1998: FIPA97v2, FIPA98v1 specifications (5)
- 2000: new IETF (Internet Engineering task Force) like process, Pxx00NN specifications (~70)

The first set of specifications are the FIPA97v1 specifications which contained two core specifications: an agent management model (Figure 1) provides the normative framework, agency, within which FIPA Agents exist and operate and a description of the ACL.

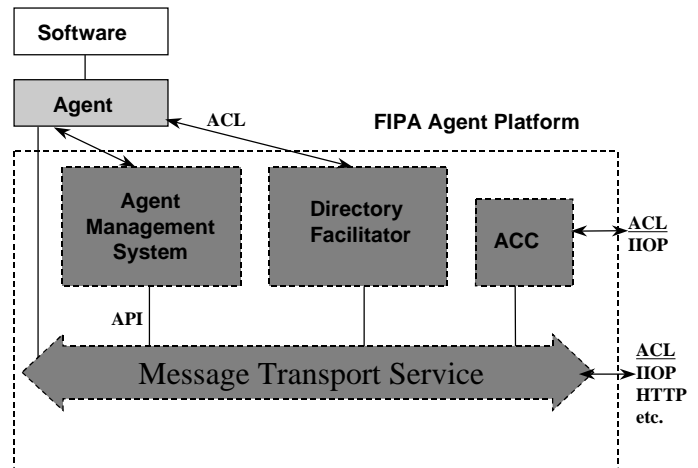


Figure 2. The FIPA 97 agent reference model

In 1998, these two core specifications or agent management and agent communication were revised and three new ones were added. In 1999, the process for producing specifications was revised from yearly cycles to be more IETF-like. Standards start off as preliminary, when mature enough they are deemed to be suitable for experiment (experimental) and after a suitable experiment phase become a full standard. In the preliminary and experiment phase, time-outs are in place - this means that if they don't progress within a suitable time frame, they become obsolete by default. These FIPA specifications obsolete the 1997 and 1998 specifications. There are over 70 of these: there are of two types individual components and profiles that specify how groups of components are inter-linked. So what was termed the FIPA97 agent reference model is now split into several specifications: a management profile, a transport profile and several management and transport component specifications.

One important trend in the FIPA standards is away from the specification of single external interfaces to multiple external interfaces. For example, FIPA in early versions of its specification defined a single so called base-line "transport protocol" - OMG's Internet Inter-ORB Protocol (IOP). This in essence meant most FIPA agent platforms ran on top of CORBA. There was a growing realization that one transport protocol was not suitable for all domains, for example, an interface has now to be defined to a WAP (Wireless Application Protocol) transport and more may follow. The characteristics and limitations of the base protocol. Similarly, FIPA97 specified a single ASCII string encoding for the ACL

message but FIPA now specified multiple encodings such as Unicode and other text language encodings, XML and binary (bit-efficient) encodings).

### 3. IMPLEMENTING THE FIPA SPECIFICATIONS

The FIPA standards in some areas introduce conceptual problems for designers and implementers. For example, the FIPA ACL (Agent Communication Language) focuses on an internal agent mental agency of beliefs, desires and intentions and closure is not enforced (agents are not compelled to answer) - these hinder multi-agent co-ordination.

The FIPA specifications are also not intended to be a complete blueprint or specification for building multi-agent system. For example, FIPA standards do not prescribe how to describe existential aspects of how agents in a discrete world, nor do they define error handling although some aspects of error reporting are covered. Some of the practical issues in using the agent standards are discussed in [7]. Useful information for developers is also given in a FIPA output document, the FIPA Developer's Guide [FIPA].

There are several different processes for deriving working implementations from the standard specifications: these differ in the way an API and its implementation are developed, fixed and disseminated. At one end of the spectrum a closed, group of people develop and fix and control the API and release an implementation under commercial license. At the other end of the spectrum, the API and implementation (even the source-code) can be developed and released to the public for experimentation and to provide the input. We highlight the advantages of the latter (open source license) approach to implementing the FIPA standard.

#### 3.1 Openness, scalability and Open Source

Let's first of all distinguish between openness and open source.

##### 3.1.1 Openness and scalability

A distributed system is considered open if it is extensible – there is a range of degrees and there exist different models and designs for openness. Openness is linked to reconfigurability. If interfaces to the system are explicitly defined, and parts of the system are loosely-coupled then parts can be exchanged and enhanced. Dynamic reconfiguration is a key characteristic of scaleable systems. Minimal openness typically exposes the interface(s) at the highest level of abstraction, for example, the agent platform service API can consist of an agent life-cycle management API, a directory service API and a message transport system API. In an agent platform, communication facilitators coupled with the use of rich message-passing protocols leads to natural support for an open service architecture. Service provider agents and service consumer agents can be dynamically bound and unbound using the facilitator.

Different service domains can require a range of communication support such as more or less negotiation about the protocol characteristics, more or less throughput and more or less security. Openness at lower levels of abstraction in the platform enables services to be dynamically replaced or enhanced. For example, message transport A could be substituted with or used alongside transport B. Depending on the software language and

the types of interaction, service links between parts could be statically modified before the session starts or dynamically modified during a session.

Most agent platforms including FIPA and non-FIPA platforms naturally offer openness at the agent level in that although the platform itself may be fixed or closed, service and user agents can be dynamically added to the platform and can inter-operate.

To support this requires a common means of representing [encoding], understanding [ontology] and exchanging [protocol] service information. FIPA platforms defines a standard base protocol based on speech acts several standard ontologies: a core one for registering and querying de-registering services [part of the FIPA management ontology] and various domain specific ones. No specific service encoding is mandatory. Non-FIPA platforms require the use of platform specific service ontology, encoding, and protocol combinations.

Many agent platforms support some boot-strapping process which includes agent synthesis for resident agents. Often an agent shell or agent factory API is defined, the shell contains hooks into the platform to use lower-level services such as a transport service. To synthesis an agent, it is not strictly necessary to use the platform API. Providing a standard protocol such as a TCP/IP one is understood by the platform for the exchange of messages, the non resident agent can be synthesized externally and registered with the platform. Of course, resident and non-resident agents may be managed by the platform differently.

##### 3.1.2 Open source and scalability

Open source supplies the user with the source-code in addition to the traditional executable version of commercial software. The source code is free and the license does not restrict users from modifying and distributing it. Users can use the executable or build their own executable, perhaps for a proprietary platform, by using standard build tools such as compilers.

An open source implementation reduces barriers to adoption of the FIPA standards, allowing agent application developers to construct applications using FIPA technology. Minimizing cost is particularly important to encourage take up in education establishments and small medium enterprises.

It improves software quality through third party development - more eyes, less bugs [8]. Users can directly reduce the time-scale for bug correction by providing their own fixes- this effect can be dramatic with a large user base.

Open source is a powerful mechanism for engaging users while the market for FIPA agent technology is developing – it can be regarded as a form of Rapid Application Development in which new user requirements and actual extensions can be incrementally extracted during the development process. Open Source encourages research and innovation. There are different models of how modifications can be checked into the main code base ranging from total control by the platform originator to a fairly unlimited check-in by any user.

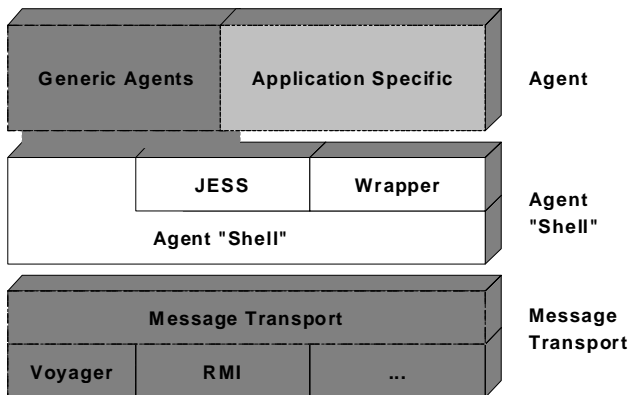
There is also a down-side: open source has the potential to allow chaotic development in any part in any direction and for extensions to cause side-effects. This is less of an issue if the opens source check-ins are managed.

FIPA-OS (<http://www.nortelnetworks.com/fipa-os>) was the first FIPA based platform to be released under an open source license [9] and provides rich and flexible support for agent communication. Currently, there is no formal or clear mechanism to determine the compliance of a FIPA architecture implementation.

### 3.2 FIPA-OS architecture overview

The FIPA reference model (Figure 2) illustrates the core components of the FIPA-OS (Figure 3). In addition to the mandatory components of the FIPA Reference Model, the FIPA-OS implementation of FIPA includes support for:

- an Agent Shell for producing agents which can then communicate with each other using the FIPA-OS facilities;
- multi-layered support for agent communication;
- message and conversation management;
- dynamic platform configuration to support multiple IPTMs, multiple types of persistence (enabling integration with legacy persistence software).



**Figure 3. Schematic diagram of the FIPA-OS architecture. Dotted shading indicates the FIPA defined components.**

The FIPA-OS architecture can be envisaged as a non-strict layered model (Figure 3). In a non-strict layered model, entities in non-adjacent layers can access each other directly. The developer is able to extend the architecture not only by appending value-added layers such as specialist service agents or facilitator agents on top but in addition, components lower or mid layers can be replaced, modified or deleted.

### 3.3 Agent Shell

There are a variety of ways in which new agents can be introduced with FIPA-OS.

Agents can be built using agent shells. These are implemented as Java base classes with pre-defined hooks into the platform to use platform services. Currently there are two agent shells. If agents are derived from the base agent class then they have in built support for message transport, message retrieval and transfer and message buffering.

Another more advanced shell implemented in the includes this support plus support for management of ACL message within the context of a FIPA interaction protocol (see below).

Agent developers do not have to develop agents using either of these shells to derive their agents. Instead, agents can be developed independently. Providing these non-resident agents support a transport protocol supported by FIPA-OS, they can use FIPA-OS to interact with other agents on the platform. Non-resident agents can use or invoke the transport API of the platform directly or use their own non-platform transport.

### 3.4 Multi-tiered ACL Communication

Understanding an ACL message requires processing the message with regard to its temporal position within a particular interaction sequence between two or more agents. This involves understanding the type of communication called a communication act, (it is specified in the message and may be a request or statement of fact or query), understanding the structure of the content and finally understanding the semantics of the request.

As ACL communication is so rich, it is often represented as a multi-tiered layer in its own right [5]. FIPA-OS ACL communication uses four main components from the ACL and the transport or "comms" layers: conversation or message interaction, ACL message, content structure and content semantics (ontology). Not all of these components need to be used by each agent and different combinations of different types of each layers can be supported at each layer (see below).

The conversation or message interaction layer is described in the next section. FIPA-OS supports a single ACL message encoding. String encodings are decoded using an ACL string parser. There are several supported encodings for the ACL message content including XML-encoding, FIPA SL0 and FIPA SL1.

To support cross-platform access to name services in different CORBA systems, agents are accessed using CORBA IOR (Interoperability Object References) object references stored as HTML-encoded strings on a Web server.

### 3.5 Configuration

There is built-in support to plug in different types of components at key interfaces or hot-spots. These include: comms or transport, message content encoding and the type of ACL message storage. These are plugged in during platform initialization. The plug-in nature of the platform supports a dynamic component-oriented middleware model and promotes a combination of thin client agents and thin platform services. Agents are not required to implement platform services themselves, they access these services in the platform (supporting thin agents) and platforms do not need to load support for all types of each services at start-up (leading to thinner platforms).

The types for these plug-and-play components are currently configured in terms of profiles for the platform as a whole, and for each individual agent. These profiles are encoded in XML/RDF and stored in resource files.

As further user requirements arise, additional hot-spots may be identified. Developers can design and implement additional types for these plug-and-play components.

FIPA-OS is being used in several other projects and application domains. These include: the FACTS, FIPA ACTS project [10] for network service reservation, a virtual home environment

called CAMELEON [11] for personalized information retrieval of various kinds of information found in the retail sector (MAPPA project [12]), in research into agents infrastructures (CASBAH project) [13].

#### 4. CONCLUSIONS

Agent technology will only release its full potential with high levels of multi-agent interactivity across a diverse range of systems and for a wide range of applications if it has very high degrees of openness and dynamism - hence the need for standardization and validation of the standardization.

In the last year or so there has been particular proliferation in the development of agent-based frameworks or software toolkits. Building an agent system is time consuming and in some cases can be expedited by using an infrastructure provided by an agent toolkit. Toolkits enable software developers to create agent based systems whilst removing some of the tedious and often complex tasks.

FIPA-OS represents an agent toolkit that has been developed for use in constructing and supporting a world of heterogeneous multi-agent platforms, agents, and services that adhere to the FIPA agent standards. The platform is freely available as managed Open Source and other developers are encouraged to take it, apply it and tune it.

#### 5. ACKNOWLEDGMENTS

The author from Imperial College gratefully acknowledges joint support for the CASBAh project from Nortel Networks and the UK Engineering and Physical Sciences Research Council (EPSRC) under grant GR/L34440. We also acknowledge the considerable input from other Nortel Network developers and other collaborators who have contributed to the development of the FIPA-OS platform. The views expressed here are those of the authors and do not purport to represent the FIPA membership.

#### 6. REFERENCES

[1] Searle, J. R. *Speech Acts*. Cambridge University Press Cambridge, UK, (1969).

- [2] Guarino, N. *Formal Ontology, Conceptual Analysis and Knowledge Representation*. International Journal Of Human-Computer Studies, Number 5/6, 625-640, (1995).
- [3] Tanenbaum, A.S. *Computer networks*. 3rd Edition. Prentice Hall., New jersey, 40-44 (1996).
- [4] Mobile Agent Service Interoperability Service. <http://www.omg.org/>
- [5] Finin, T., Labrou, Y., Mayfield, J. 'KQML as an agent communication language'. In *Software agents*, Bradshaw JM (ed.), MIT Press, 1997. pp.291-316.
- [6] FIPA (Foundation for Intelligent Physical Agents) home page. <http://www.fipa.org>
- [7] Charlton, P., Cattoni, R., Potrich, A., and Mamdani, E. Evaluating the FIPA standards and its role in achieving cooperation in multi-agent systems. "Multi-agent systems, Internet and applications". HICS-33 software technology minitrack, (Mawi, Hawaii, Jan 2000).
- [8] Cathedral and Bazaar. <http://www.tuxedo.org/~esr/writings/cathedral-bazaar/>
- [9] Poslad S. J., Buckle S.J., and Hadingham R. The FIPA-OS agent platform: Open Source for Open Standards. Proceedings of PAAM 2000, Manchester UK, (April 2000).
- [10] FIPA ACTS (FACTS) project home-page. <http://www.labs.bt.com/profsoc/facts/>
- [11] Loryman, M., Buckle, P., and Major, B. The Cameleon VAB enabled by a FIPA compliant Agent platform. ACTS workshop (Singapore, September 1999).
- [12] MAPPA (Multimedia Access through Persistent Personal Agents) project home page. <http://www.sics.se/mappa/>
- [13] Poslad, S., Pitt J., Mamdani, A., Hadingham, R., Buckle, P. Agent-oriented middleware for integrating customer network services. In *Software Agents for Future Communication Systems*, Hayzelden, A., Bigham, J., Eds., Springer-Verlag, 1999.