The FIPA-OS agent platform: Open Source for Open Standards

Stefan Poslad[*], Phil Buckle[+], Rob Hadingham[+]

[*]Imperial College of Science, Technology and Medicine, Exhibition Road, London, SW7 2BZ, UK. Email: s.poslad@ic.ac.uk

[+]Nortel Networks, London Road, Harlow, Essex, CM17 9NA, UK.
Email: {pbuckle, rgh}@nortelnetworks.com

Abstract

FIPA-OS (FIPA Open Source) is an open agent platform originating from Nortel Networks. The platform supports communication between multiple agents using an agent communication language which conforms to the FIPA (Foundation for Intelligent Physical Agents) agent standards. A key focus of the platform is that it supports openness. This is naturally supported by the agent paradigm itself and by the design of the platform itself whose parts have loose coupling such that extensions and innovations to support agent communication can occur in several key areas. The openness is further emphasized in that the platform software is distributed and managed under an open-source licensing scheme. FIPA-OS is being deployed in several domains including virtual private network provisioning, distributed meeting scheduling and a virtual home environment. It has been demonstrated to interoperate with other heterogeneous FIPA compliant platforms and is in use in numerous institutions around the world.

1. Introduction

In Multi-Agent Systems (MAS), heterogeneous distributed services are represented as autonomous software agents which interact using an Agent Communication Language or ACL based on speech acts (Searle, 1969). There are several benefits to this particular approach: the ACL level of supporting computational interaction can be much "richer" and is at a higher level of abstraction (knowledge sharing level) than the classic representation of APIs in distributed object technology. This interaction can remove some of the common causes of communication errors in traditional systems which rely heavily on exact syntactical matching of service requests to service provider interfaces. Agents are designed as components, which are specifically designed to deal with unanticipated requests and they can spontaneously recruit the help of third parties when they need to. Agents are designed to operate autonomously, they can communicate using indirect asynchronous service requests rather than by direct, synchronous service invocation.

However, the wide spread deployment of MAS needs to address several issues. Firstly, many of these multi-agent systems are islands of functionality – agents on different types of platform don't easily interoperate and many systems still do not support agents. Secondly the development, maintenance and management of distributed software services based on the MAS paradigm introduces new complexity and this acts as a barrier to entry to users and developers who wish to assess and use this new technology. Thirdly, the paradigm itself is still maturing, there is a back-

log of issues still to be addressed in addition to interoperability problems which include management and security issues, tool support and simplifying use and customization by the end-user.

In order to address some of these issues, Nortel Networks (http://www.nortelnetworks.com/fipa-os) has released an in-house developed agent platform, which has been deployed in several application domains and which supports the FIPA agent standards as open source.

The rest of this paper is organized as follows: Section 1 continues with a closer examination of the FIPA and other standards to support the social interaction within and between multi-agent systems. It also discusses the benefits of designing agent systems that emphasize openness. Section 2 describes the design and main features of the FIPA-OS open agent platform, which is the first FIPA compliant agent framework available as open source. Section 3 describes how and where FIPA-OS is being used. The paper finishes with an outline of work in progress and conclusions.

1.1. FIPA and other agent standards

In the context of FIPA, an agent[1] is an encapsulated software entity with its own state, behavior, thread of control, and an ability to interact and communicate with other entities - including people, other agents, and legacy systems. This definition puts an agent in the same family as objects, functions, processes, and daemons but it is also distinct in that it is at a much higher-level of abstraction. The agent interaction paradigm differs from the traditional client-server approach: agents can interact on a peer-to-peer level, mediating, collaborating, and co-operating to achieve their goals.

A common (but by no means necessary) attribute of an agent is an ability to migrate seamlessly from one platform to another whilst retaining state information, a mobile agent. One use of mobility is in the deployment and upgrade of an agent.

Another common type of agent is the intelligent agent, one that exhibits 'smart' behavior. Such 'smarts' can range from the primitive behavior achieved through following user-defined scripts, to the adaptive behavior of neural networks or other heuristic techniques. In general, intelligent agents are not mobile since, in general, the larger an agent is the less desirable it is to move it; coding artificial intelligence into an agent will undoubtedly make it bigger. There is an exception to this last statement, 'Swarm' intelligence. This is a form of distributed artificial intelligence modeled on ant-like collective intelligence. The ant-like 'agents' collaborate to perform complex tasks, which individually they are unable to solve due to their limited intelligence (e.g. ant-based routing) (Schoonderwoerd et al,1996).

Another prevalent, but optional, attribute of an agent is anthropomorphism or the 'human factor': this can take the form of physical appearance, or human attributes such as goal-directed behavior, trust, beliefs, desires and even emotions.

There are three important agent standardization efforts which are attempting to support interoperability between agents on different types of agent platform: KQML community, OMG’s MASIF and FIPA.

---

[1] The term agent is loaded; it means different things to different people.

Of these 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 the traditional RPC-based paradigm, the ACL as defined by FIPA provides an attempt at a universal message-oriented communication language. The FIPA ACL describes a standard way to package messages, in such a way that it is clear to other compliant agents what the purpose of the communication is. Although there are several hundred verbs in English, which correspond to performatives, the ACL defines what is considered to be the minimal set for agent communication (FIPA ACL consists of 20 or so performatives). This method provides for a flexible approach for communication between software entities exhibiting benefits including:

- dynamic introduction and removal of services;
- customized services can be introduced without a requirement to re-compile the code of the clients at run-time;
- more de-centralized peer-peer realization of software;
- a universal message based language approach providing consistent speech-act based interface throughout software;
- asynchronous message-based interaction between entities.

1.2. KQML

KQML or Knowledge Query Meta Language (Finin et al, 1997) was one of the first initiatives to specify how to support the social interaction characteristic of agents using a protocol based on speech acts and is now also one of the most pervasive ACLs. KQML however isn't a true de facto standard in the sense that there is no consensus in the community on a single specification (or set of specifications) and it has not yet been ratified by common agreement between members of an organization and forum of some standing in the community. As a result, variations of KQML exist and different agent systems which speak different dialects may not be able to interoperate fully.

1.3. OMG MASIF

The Object Management Group (http://www.omg.org) Mobile Agent System Interoperability Facility, MASIF, differs from both KQML and FIPA in that it regards the key characteristic, which defines agents as mobility of the agent from one location to another. In contrast to MASIF, both KQML and FIPA emphasize agency and social interaction between multi-agents as the defining properties for software agents. MASIF does not support or standardize interoperability between non-mobile agents on different agent platforms. Further, MASIF restricts the interoperability of agents to agents developed on CORBA platforms whereas the focus of FIPA is to directly support the interoperability of agents deployed on agent frameworks which can support heterogeneous transports.

OMG is exploring how to support other types of software agent than mobile agents. It has issued a Request For Information on agents. FIPA has supplied its specifications as input to this request. This is still work in progress at this time.

1.4. FIPA

The Foundation for Intelligent Physical Agents, FIPA (http://www.fipa.org), 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 specifications. In this way the individual software frameworks are interoperable.

The FIPA agent standards (O'Brien and Nicol, 1998) aims to bring the commercial world a step closer to true software components, the benefits of this will include increased re-use, together with ease of upgrade. Early adopters of new technology tend to be wary when there is no commonly agreed standard, which lacks the support of a large consortium of companies - this is addressed by FIPA. An agent standard will provide added confidence to potential adopters of this technology. Finally, the standardization process shifts the emphasis from longer-term research issues to the practicalities of realizing commercial agent systems. FIPA allows for focused collaboration (of both industrial and academic organizations) in addressing the key challenges facing commercial agent developers as they turn agent technology into products.

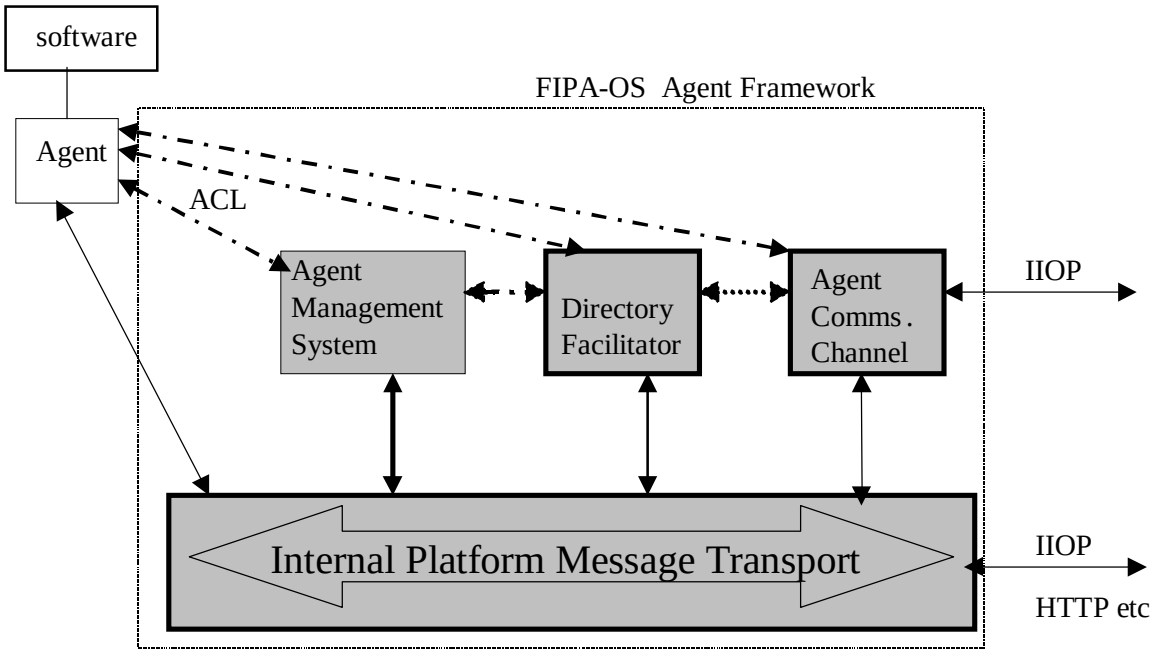

Figure 1: The FIPA 97 agent reference model.

The FIPA 97 agent reference model (Figure 1) currently provides the normative framework within which FIPA Agents exist and operate. Combined with the Agent Life cycle, it establishes the logical and temporal contexts for the creation, operation and retirement of Agents.

The Directory Facilitator (DF), Agent Management System (AMS), Agent Communication Channel (ACC) and Internal Platform Message Transport (IPMT) form what are termed the Agent Platform (AP). The DF provides "yellow pages" services to other agents. The AMS provides white-page services and life-cycle management services for agents and the ACC supports inter-agent communication. The ACC supports interoperability both within and across different platforms. The Internal Platform Message Transport (IPMT) provides a message forwarding service for agents on a particular platform, which must be reliable, orderly (O'Brien and Nicol, 1998).

These are mandatory, normative components of the model. The ACC, AMS and DF are capability sets, they may be performed by one agent or by three different agents – this is left to the agent platform developer.

To be minimally FIPA compliant requires compliance to the Agent Management FIPA specification, the Agent Communication Language FIPA specification and the [non-agent] software-agent integration specification. There are differing degrees of compliance to, and interpretation of, FIPA compliance. For example, if the FIPA platform has no requirement to support access to non-agent services by agents, then adherence to the software-agent integration specification is not required.

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 may hinder multi-agent co-ordination (Charlton et al, 2000).

The FIPA normative 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 (Charlton et al, 2000). Useful information for developers is also given in an informative part of the specifications called the FIPA 97 Developer's Guide.

The FIPA specifications themselves may evolve to solve current shortcomings and to meet future needs. For example, currently, agents are managed via a message-passing interface at the ACL level, i.e., agents are managed by interaction with the three core FIPA platform agents: the DF agent, the AMS agent and the ACC agent. FIPA is currently considering offering one or more of these core services via a non-agent interface such as method invocation interface rather than by encapsulating it as a service whose sole interface is via the specialized service agent. This reduction in message passing during bootstrapping may enhance scalability and ease the paradigm change from standard APIs to message-passing using an ACL. FIPA agent platforms will need to evolve to comply with new standards or replacement standards.

1.5. Openness and Open Source

1.5.1. Openness and reconfigurability

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. Minimal openness typically exposes the interfaces 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 a rich message-passing protocol suite 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, a particular message transport could be substituted for or used alongside a different transport. 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.

1.5.2. Open source

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 executables supplied out of the box or build their own executables 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 (http://www.tuxedo.org/~esr/writings/cathedral-bazaar/). Users can directly reduce the time-scale for bug correction by providing their own fixes- this effect can be dramatic within 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 open source check-ins are managed as is the case with the FIPA-OS open source.

2. Related Work

The field of MAS is currently very much an active research area. We relate FIPA-OS to a sub-set of multi-agent systems that have the following characteristics: they focus on communication between multi-agents using an agent communication language; they support the FIPA specifications and they promote openness in terms of extensibility and licensing.

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 agents themselves are naturally co-operative. 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 differently by the platform.

Several non-FIPA platforms such as OAA (Bradshaw et al, 1997), JATLite (Frost and Cutkosky, 1996), and JIAC (Wieczorek and Albayrak, 1998) offer some degree of openness for reconfiguration including replacement of the platform services. For example, JIAC describes several interchangeable components including service broker, message transport, knowledge base and security.

There are now several MAS platforms which report their support for the FIPA agent standards, these include: JADE (Bellifemine et al 1999), Grasshopper (Breugst et al, 1998) and ZEUS (Nwana et al, 1999). Of these only FIPA-OS and ZEUS are freely available under a general public license and are released with source-code under an open-source license.

There are several important differences between FIPA-OS and ZEUS. FIPA-OS was designed at the onset to focus on supporting agent communication using the FIPA agent standards. ZEUS initially supported only KQML as the agent communication language and has since been re-written to support the FIPA ACL. FIPA-OS was designed to operate in a heterogeneous open service environment – it does this by for example by supporting multiple transports such as IIOP using a variety of CORBA APIs, RMI and TCP and by supporting multiple encodings for the content. The interoperability of the publicly released version of FIPA-OS is being evaluated in a heterogeneous FIPA environment in the FACTS project. The current ZEUS platform does support the FIPA ACL specification, it does not however yet support the FIPA agent management specification and its interoperability within a heterogeneous FIPA environment has not yet to been reported. FIPA-OS initially focuses on providing abstractions and interfaces (APIs) for developers who wish to extend, enhance and integrate an agent platform with existing software infrastructures. ZEUS provides much higher level abstractions for developing agents – it hides the API and instead provides an integrated development environment to allow developers to configure the agent platform.

3. User requirements and System Design

3.1. FIPA-OS description

FIPA-OS is designed to support the FIPA agent standards. The FIPA reference model discussed earlier defines the core components of the FIPA-OS distribution: the Directory Facilitator (DF), Agent Management System (AMS), Agent Communication Channel (ACC) and the Internal Platform Message Transport (IPMT). Currently, there is no formal or clear mechanism to determine the compliance of a FIPA architecture implementation. In addition, there are different versions of the architecture available, which are not completely backward compatible. For these reasons, we describe the features of the FIPA specifications incorporated within FIPA-OS in general.

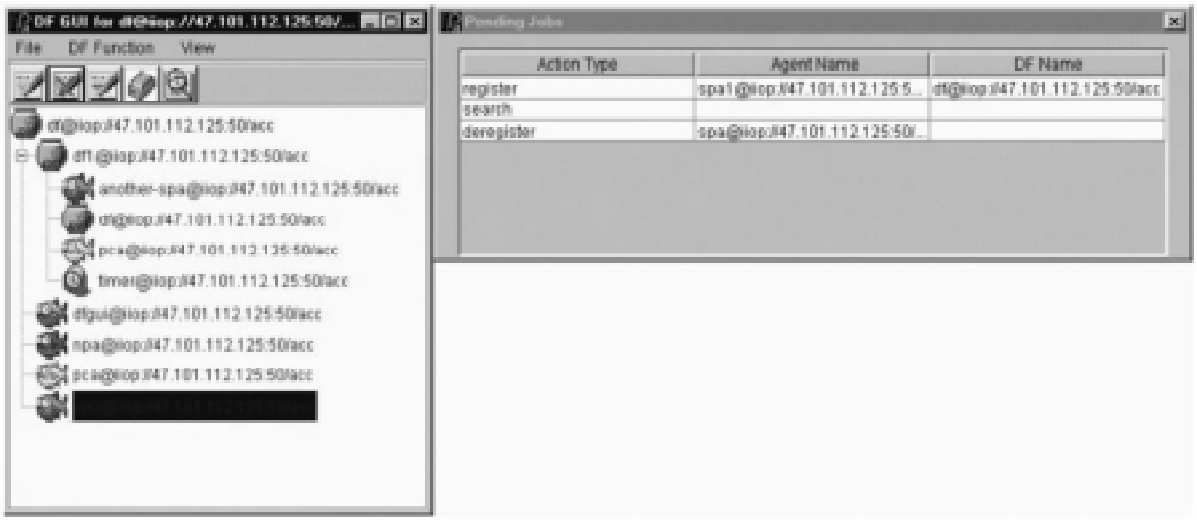

Figure 2: FIPA-OS visualization tool showing the agent services registered with the DF agent.

In addition to the mandatory components of the FIPA Reference Model, the FIPA-OS distribution includes support for:

- Different types of Agent Shells 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) and multiple encodings.
- Abstract interfaces and software design patterns
- Diagnostics and visualization tools (Figure 2).

The FIPA-OS distribution contains class files, Java source code and documentation. It also includes simple test agents to access the agent platform services and some visualization software.

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, lower or mid layers can be replaced, modified or deleted.

3.2. Agent Shell

There are a variety of ways in which new agents can be added to FIPA-OS platforms. 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. Agents built using the first shell have built-in support for message transport, message retrieval and message buffering. Another more advanced shell 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 use 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 compatible transport.

3.3. Multi-tiered ACL Communication

Understanding an ACL message requires processing the message with regard to its temporal position within a particular interaction sequence. This involves understanding the type of communication called a communication act, (as specified in the message: this 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 (Finin et al, 1997). FIPA-OS supports ACL communication using four sub-layers of components: conversation, ACL message, content (syntax) and ontology (content semantics). Not all of these components need to be used by each agent and different combinations of different types of each these components can be supported at each layer. This flexibility is needed because in a heterogeneous world, different agents may encode and transport the informaation differently.

The conversation or message interaction layer is described in the next section. FIPA-OS supports both ASCII string and XML encoding of the ACL message using the appropriate decoder and parser. There are several supported encodings for the ACL message content including, FIPA SL0 and FIPA SL1 and the proposed FIPA-RDF specification for encoding the content in XML

The current version (1.03) of the platform supports IIOP as the baseline transport protocols as specified by the FIPA agent architecture – this may be used as both the IPTM protocol and to support communication by the ACC agent. There are different transports APIs to interface different CORBA products. The name and directory services supported by the AMS and DF respectively are implemented using the CORBA name services. To support cross-platform access, agents are accessed using CORBA IOR (Interoperability Object References) object references which are stored as HTML-encoded strings on a Web server. In addition, Java RMI may also be used as an IPTM protocol.

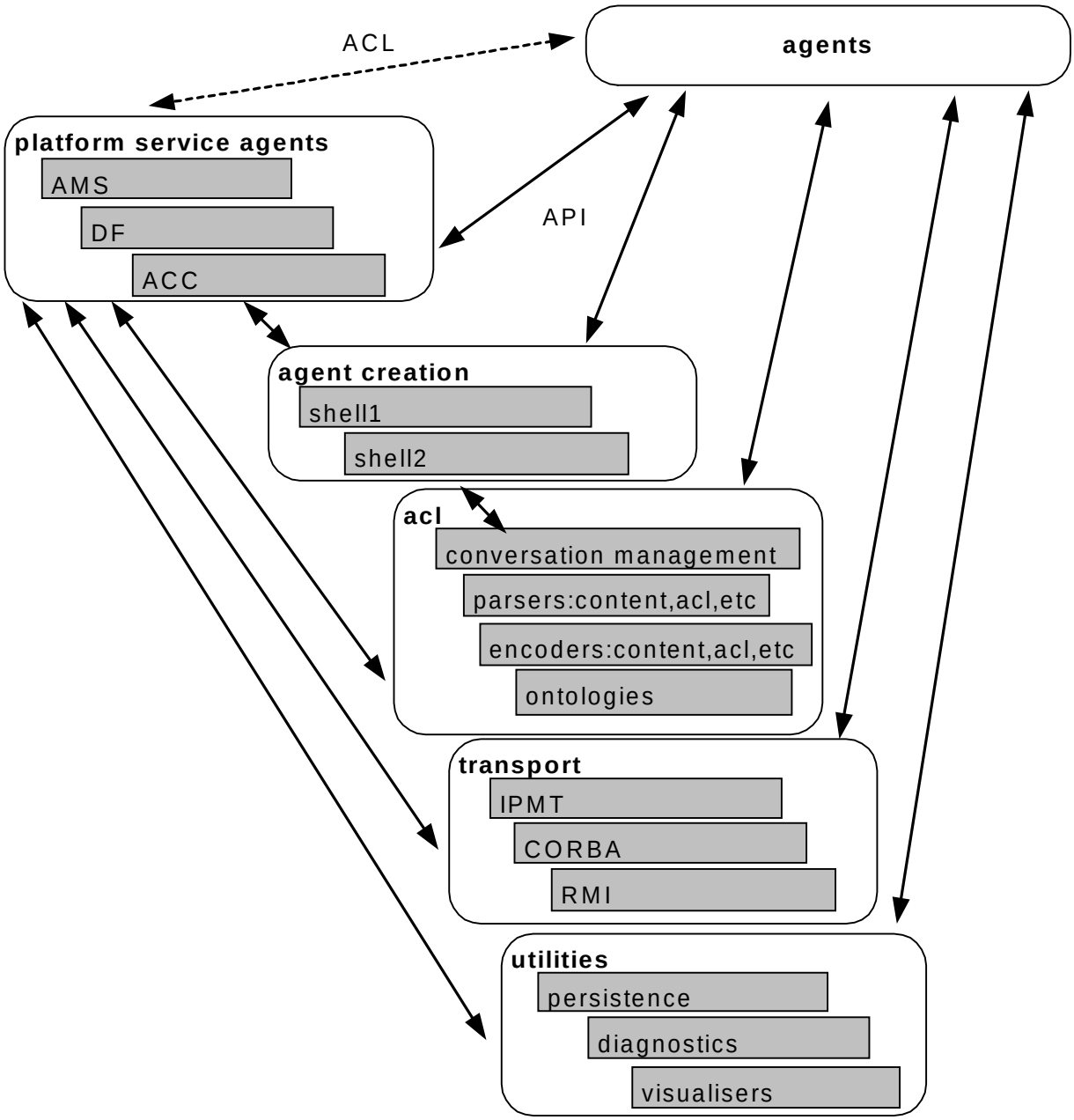

Figure 3: a schematic diagram of the FIPA-OS architecture.

3.4. Conversation management

Messages are usually sent as part of an interaction protocol, for example, the base FIPA multi-agent interaction protocol called FIPA-request supports client-server request-reply type dialogs. Unlike typical RPC invocation, requests are indirect. Agent A can request another agent B to

perform some action on its behalf. Agent B normally sends back an agree message followed by an inform (or refuse or failure) message as the reply to agent A.

Without conversation or dialog management, messages are sent and received independently of any context within any interaction pattern or conversation they occur and it is more difficult to detect whether failures such as an absent reply or inappropriate reply occurred.

As an aid to conversation management, FIPA provides two hooks in the ACL itself but the semantics of those hooks are yet to be defined. Firstly, the FIPA ACL defines two fields “reply-with” and “in-reply-to”. The semantics are not defined in whether they are to be used by the sender and receiver respectively and how the values of these two fields are related. Secondly, the FIPA ACL also defines a Conversation-ID field - again the semantics of how this should be used within in a conversation are not defined.

FIPA-OS uses the Conversation-ID field to co-ordinate conversations. The Conversation-ID must be able to unambiguously link the sender and receiver to a particular conversation. To do this, FIPA-OS defines the following semantics. The Conversation-ID is generated by the sender agent and is constructed using the Agent ID and the time the message was created plus a counter to ensure uniqueness. A conversation is defined as an instance of any FIPA interaction protocol. When the conversation is started by the sender, any participating agent must use the same Conversation-ID when sending any further messages within this same conversation instance. There are three objects in the architecture to support conversation. The conversation agent shell starts a new conversation of a FIPA interaction protocol to trigger conversation management. A conversation manager encapsulates and manages all messages created in the sender and receiver agent when a conversation starts. Finally, the message pattern which defines the FIPA interaction protocol is encapsulated in a type of Interaction Protocol object.

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. Developers can design and implement additional types for these plug-and-play components. 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.

3.6. Open source licensing and management

As the platform is open-source, all of its internals are exposed. The community of developers could drive development in multiple revolutionary and evolutionary ways. However, the original architects can constrain this potentially chaotic development through the partitioning of the

system and by highlighting key interfaces (‘hot-spots’) and components which can be replaced. This tends to lead to extensions in particular directions.

The original platforms developers can also control which changes are incorporated into particulr versions of the platform and the time of release of the ‘official’ version at the platform reference web site.

Currently, Nortel Networks gives a commitment to manage development of newer versions of the platform for at least two years. An emailing list or reflector is set up at Nortel Networks and all stake-holders (those users who have registered and downloaded a copy of the platform) will be able to see each other’s requests, view submissions and responses and will be able to examine the decision-making process.

4. Evaluation

Agent technology will only release its full potential to support high levels of multi-agent interactivity, across a diverse range of systems, 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.

The FACTS, FIPA ACTS project (http://www.labs.bt.com/profsoc/facts/) has built demonstrators to prove the viability of this vision by validating the new agent standards emerging from FIPA. FACTS is one of the first European activities to actively apply FIPA, thereby exposing its strengths and weaknesses. Phase 1 of FACTS was completed early 1999 and focussed on the FIPA97 normative specifications. It also drove the development of FIPA98 and FIPA99.

The FACTS validation demonstrators cover three different application areas viz. audio-visual broadcasting and entertainment; service reservation; and electronic commerce. These three areas provide nearly complete coverage of all components of the FIPA specifications. The Service Reservation work-package has developed a demonstrator system for dynamic and competitive service provisioning. This application has been composed of three heterogeneous implementations of FIPA agent platforms and application agents. By testing the interoperability of these divergent implementations, FACTS has identified areas of the standards that are open to misinterpretation, features requiring additions to the standards. It has also demonstrated the practical usability of the standards.

FIPA-OS is being used in several other projects and application domains. These include: a virtual home environment concepts and future wireless services project called CAMELEON (Loryman et al, 1999), personalized retrieval of information found in the retail sector in a project called MAPPA (http://www.sics.se/mappa/) and in a research project on agents infrastructures called CASBAH (Poslad et al, 1999).

5. Future work and conclusions

5.1. Further work

Further work is oriented along three different axes: supporting future FIPA specifications, supporting new user and application requirements and enhancing user support.

New versions of FIPA-OS will be produced to reflect support for newer versions of FIPA agent specifications which have been recently ratified and those currently being drafted. For example, changes to the ACL parser and encoder, changes to the management ontology and changes to the transport service (the ACC service is no longer performed by an agent) will be required. FIPA is also considering removing the mandatory use of an ACL "agentized" interface to the platform directory and management services.

There are many potential useful enhancements which could be made to the FIPA-OS platform. Some of these in progress include supporting additional transport protocols, adding hooks for and providing user and management tools and supplying a richer suite of facilitators and sample agents.

To enhance user support, reference agent platforms are being developed to support user-testing, in a multi-platform environment, over the Internet. The first of these is being hosted at Imperial College, London. This will also provide information on designs for high availability and scalability of agent platforms.

5.2. Conclusions

Recently, there has been a 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. Many of the agent toolkits provide support for multi-threading, communication and other basic interfaces such that the agent system designer can concentrate more on the modeling of the particular application domain. Therefore, these toolkits enable software developers to create agent based systems whilst 'removing' some of the more tedious, complex tasks. Of particular interest is the number of commercial templates and toolkits that are available for download on the Web. However, the majority of these toolkits focus on support for mobile agents.

FIPA-OS represents an agent framework, which has been developed for use to construct heterogeneous, multi-agent platforms, agents, and services which adhere to the FIPA agent standards. The platform is open in that many parts of it can be replaced or omitted and multiple types of parts can be used. The platform is freely available as managed open source from http://www.nortelnetworks.com/fipa-os.

Acknowledgements

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.

6. References

1. Bellifemine, F., Rimassa, G., Poggi, A., JADE - A FIPA-compliant Agent Framework. In Proceedings of the 4th International Conference and Exhibition on The Practical Application of Intelligent Agents and Multi-Agents, London, 1999.
2. Bradshaw, J.M., Dutfield, S., Benoit, P., et al. KAoS: towards an industrial strength open agent architecture. In Software agents, Bradshaw, JM (ed.), MIT Press, 1997, pp.375-418.

3. Breugst, M., Hagen, L., Magedanz, T. Impacts of Mobile Agent Technology on Mobile Communications System Evolution. IEEE Personal Communications Magazine,Vol. 5, No. 4, August 1998.
4. 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).
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. Frost, H.R, and Cutkosky, M. R. "Design for Manufacturability via Agent Interaction,". Paper No. 96-DETC/DFM-1302, Proceedings of the 1996 ASME Computers in Engineering Conference, Irvine, CA, August 18-22, 1996, pp. 1-8.
7. Loryman, M., Buckle, P., and Major, B. The Cameleon VAB enabled by a FIPA compliant Agent platform. ACTS workshop (Singapore, September 1999).
8. Martin, D.l., Cheyer, A.J., Moran, D.B. Building distributed software systems with the open agent architecture. In Nwana, H.S., Ndumu, D.T. (Eds.), Proc. PAAM ’98, (London, March 1998), pp. 355-376.
9. Nwana, H., Ndumu, D., Lee, L., and Collis, J..ZEUS: A Tool-Kit for Building Distributed Multi-Agent Systems. In Applied Artifical Intelligence Journal, Vol 13 (1), 1999, pp129-186.
10. Nodine, M.H., and Unruh, A. Facilitating open communication in agent systems: the InfoSleuth Infrastructure’. In Singh, M.P., Rao, A., Wooldridge, M.J. (Eds.), Intelligent Agents IV- Proc. ATAL ’97, 1998, pp.281-296.
11. O'Brien, P. D., and Nicol, R., C. FIPA - towards a standard for software agents. BT Technology Journal, Vol.16, no.3, July, 1998, pp 51
12. 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.
13. Schoonderwoerd, R., Holland, O., Bruten, J. Ant-like agents for load balancing in Telecommunications networks. Proceedings of First International Conference on Autonomous Agents. Practical Applications of Intelligent Agents, London, April 1996.
14. Searle, J. R. Speech Acts, Cambridge University Press Cambridge, UK, 1969.
15. Wieczorek, D and Albayrak, S. Open Scalable Agent Architecture for Telecommunication Applications. In Intelligent Agents for Telecommunication Applications, Albayrak, S., and Garijo, J. (eds.), Springer Verlag, LNAI 1437, 1998, pp. 233-249.