Evaluating the FIPA Standards and Its Role in Achieving Cooperation in Multi-agent Systems

P. Charlton R. Cattoni [+] A. Potrich[+] and E. Mamdani
Imperial College of Science Technology and Medicine,
Department of Electrical and Electronic Engineering, Exhibition Road, London, UK
E-mail: {p.charlton,e.mamdani}@ic.ac.uk

[+]ITC-irst, Centro per la Ricerca Scientifica e Tecnologica, SSI Division
via Sommarive 18, 38050 Povo (TN), Italy
E-mail: {cattoni,potrich}@itc.it}

Abstract

The paper focuses on the analysis and evaluation of certain aspects of the current specification standards provided by FIPA (Foundation of Intelligent Physical Agents). The work reported here is based on the development of a multi-agent application, an audio video entertainment broadcasting (AVEB) system. This has resulted in determining advantages and limitations of using the FIPA standards to build complex Multi-agent Software Systems. The development and testing of the AVEB application are part of an EU project called FACTS (acts AC317).

A main result of using FIPA has been the identification of the usefulness and power of its protocols. The reason for the importance of the protocols in developing multi-agent systems (MAS) is it provides a degree of expressing cooperation within MAS architecture. As the protocols stand currently they are not sufficient to capture a complete explicit model of the cooperative requirements in multi-agent systems. However, they provide a basis from which to start. We examine this feature of FIPA further in order to evaluate its role as a bridge between the mental agency and social agency requirements in the development of cooperation in mult-agent systems.

1. Introduction

Multi-agent systems (MAS) represent one of the most promising technological paradigms for the development of distributed, open and intelligent software systems. Moreover agent technology is beginning to be used to produce real solutions to real business problems. However, typically agent based systems have been implemented with ad-hoc solutions (communication languages, protocols, and so on) to meet the specific requirements of the application. In this respect FIPA (Foundation of Intelligent Physical Agents [1]) represents an answer to the need for standards governing structure of and communication between heterogeneous agent communities. Running since December 1995, FIPA aims at developing such standards and acting as a focal point for the required research work. To date, FIPA has published a set of official reference documents, but it is only recently that these have been checked through the practical application of the proposed standards. In this respect the primary objective of the FACTS (FIPA Agent Communication Technologies and Services) EU project is to validate the work of FIPA, by building real and complex agent based applications. One of them is an audio video entertainment broadcasting (AVEB) system.

The aim of this paper is to present an analysis and evaluation of certain aspects of the current FIPA standards on the base of the experience gained in developing the AVEB system. The paper is organised as follows: section 2 introduces the main features of the MAS design requirements, communication and FIPA standards while section 3 outlines the AVEB application. Section 4 presents pros and cons of developing MAS with the FIPA standards. Section 5 focuses on the internal of a particular AVEB agent – the User Profile Agent – to emphasise the main concepts by means of appropriate examples. Section 6 reviews the FIPA communication language semantics and protocols in order to achieve cooperation in MAS. Finally some conclusions are discussed in section 7.

2. Multi-Agent System Design and Development

2.1 Architectures, communication and co-ordination.

Many MAS systems place the emphasis on a rich communication language to communicate high level concepts about information to distributed reasoning processes. We start from the communication language requirements in a MAS system to illustrate the common software components that are often involved in supporting the communication part.

The communication part in agent systems is used to some extent to co-ordinate and share information and services. It is the main way an agent externalises its requests or solutions to the rest of the community. Being able to communicate in a rich manner offers a potential infrastructure of openness, autonomy, robustness, scaleability and flexibility. The trade off in order to reach these potential benefits is the complexity in maintaining coherence of distributed information through co-ordination. The main cost is how to co-ordinate these different services without placing too many restrictions on the communication language and the internal reasoning of the agent. There will inevitably be some restriction due to the modelling of complex application domains. The co-ordination feature of a MAS architecture is often distributed over four aspects of the communication language:

1. Protocols handled by the outer language (the content language is classed as an inner language). In knowledge query manipulation language (KQML [14]) this is partly supported by a performative and FIPA ACL [3] is supported by communicative act, FIPA protocol and linked to the content via an action definition.
2. Content part contains the expressions of encoded information and service details in a sharable way (via the language parameter and ontology model) for other agents to interpret syntactically.
3. Language used to express the syntax of the content expressions, e.g., Prolog
4. Ontology model which provides the explicit model of an application domain to allow an agent to apply an intended meaning to the content to be shared.

The actual reasoning about these aspects of a message, which is communicated from an external source (agent), needs to be internalised so the reasoning behaviour of an agent can deal with the intended meaning of the content. The receiving agent must take into account the protocol and its own belief model and current behaviour status of a set of current interactions.

2.2 Common components in MAS.

Most implemented MASs will have a set of software components that provide:

- A communication protocol, like KQML (Knowledge Query Manipulation Language) and/or ACL (agent communication language);
- a syntax for the protocol (e.g. KQML);
- a content language specialised to handle current applications/services, one that is also extensible;
- a general[1] interface (from a formal view this is the base ontological level) which binds the application to the understanding and use of content [15];

In order to support the communication language a MAS will have at least one communication channel (TCP/IP or CORBA are examples of these). Most MASs will support a standard[2] way of registering agents' roles and requesting information about other agents (referred to as a facilitator or broker). Another component that is commonly defined is a dialogue manager which manages some general protocols for an agent to interpret content and communication in context. This should be extensible to permit an agent to specialise dialogues, which are unique to certain communicating agents. Also, most agents in a MAS system will keep track of interactions with other agents and their own internal reasoning state hence a belief database store may also be a general component in the system.

This common set of components found repeatedly in a number of agent architectures provide a set of software abstractions that are now deemed essential for most agent systems. These components start to define the re-usable software patterns in agent oriented development. The agent-oriented approach extends the common set of object patterns but most importantly to make these patterns reusable they have a defined semantics which provides the developer with the necessary constraints. This is key to understanding a fundamental role of agents as a development paradigm shift in software.

2.3 The FIPA Standards.

FIPA's purpose [1] is to promote the development of specifications of generic agent technologies that maximise interoperability within and across agent based applications. FIPA provides normative specifications

[1] General in an architecture sense to permit extensibility and automation

[2] This can only be standard within the architecture being developed as the agent community only has informal standards at this point. However, groups such as FIPA [1] have certain documents and recommendations available.

about all the topics concerning agent technology (for example the agent life-cycle, communication between agents, agent mobility and security). The FIPA standards include also informative specifications about four reference applications, including the one outlined in this paper.

Following a bottom-up approach, FIPA specifications concern in particular:

1. the *Agent Platform* (AP), that is the infrastructure on which the agents perform their operations. The agent reference model and all the issues concerning the agent life-cycle and agent management are addressed [2];
2. the *Agent Communication Language* (ACL), that is the language that agents have to use to encode the messages they exchange. The ACL is based on speech act theory [4]: messages are actions, or *communicative acts*, as they are intended to perform some action by virtue of being sent. The specification consists of a set of message types and the description of their pragmatics, that is the effects on the mental attitudes of the sender and receiver agents [3];
3. the *Content Language*, that is the language used to encode the *content* of a message; the content is the part of a message that represents the domain dependent component of the communication. FIPA does not provide a single mandatory content language, but a set of reference content languages [3];
4. the *Protocols*, that are typical patterns of message exchange. Protocols are defined by means of the communicative acts introduced above and specify how the communication will be carried out by constraining the interactions among agents [3].

The FIPA specifications reflect some inspiring principles that have guided the standardisation process. First of all the concept of openness: agents can join (or leave) at run time an agent system without the need of re-compile or re-configure it; moreover an agent willing to communicate with another agent needs to follow the FIPA naming conventions and registration process with the directory facilitator in order to find the location of another agent. A concept strictly related with the previous one is that of interoperability: the FIPA standards tend to specify the minimum set of requirements in order to avoid any commitments with particular hardware, operating system or programming language. Another important principle strongly supported by FIPA is that of explicitness: information and assumptions about the agent system (the role and capabilities of the agents, the way they interact, the meaning of the message content) should be as much explicit as possible. In this respect FIPA provides a number of features:

- by means of the mechanism of the Directory Facilitator (DF) the functionalities offered by the agents of a particular Agent Platform are described explicitly. In this way a new agent looking for a specific service can easily obtain the address of the agent providing it;
- using protocols makes explicit predefined assumptions about the interactions among the involved agents (namely about the flow of the exchanged messages and the type of the communication acts);
- for the understanding of the domain-dependent information FIPA prescribes the use of ontologies. An ontology gives meaning to symbols and expressions within a given domain language. By referring to the same ontology agents ascribe the same meaning to the constants used in the exchanged message.

Finally FIPA claims not to provide specifications about the internal of the agents. However, this claim is partly misleading due to the belief, desire and intention semantics that is used to define the agent communication language. The semantics is based on a mental agency and not on a social agency which is more appropriate for defining explicit cooperative agent behaviour. However, the FIPA-protocols provide a potential part-solution to this problem. We explore this concern further in section 6.

3. An Overview of the AVEB application

The AVEB application is primarily motivated by the consideration that TV programs on offer will soon exceed the monitoring capabilities of the typical user. Besides traditional TV channels, people already receive at home satellite and cable channels, and the time is quickly approaching when a whole category of new entertainment services will become accessible via television - e.g., video on demand. In this perspective, users will be in need of a more sophisticated support in the selection of interesting programs, as well as in the negotiation of the conditions at which programs and services are purchased. In this scenario three main actors may be identified: the providers, who supply the services and the infrastructure; the users, who are interested in getting programs and services at the best possible conditions (e.g., cost, time of delivery), without wasting time zapping through a huge number of channels. Finally are the brokers who facilitate and promote the connection between providers and users.

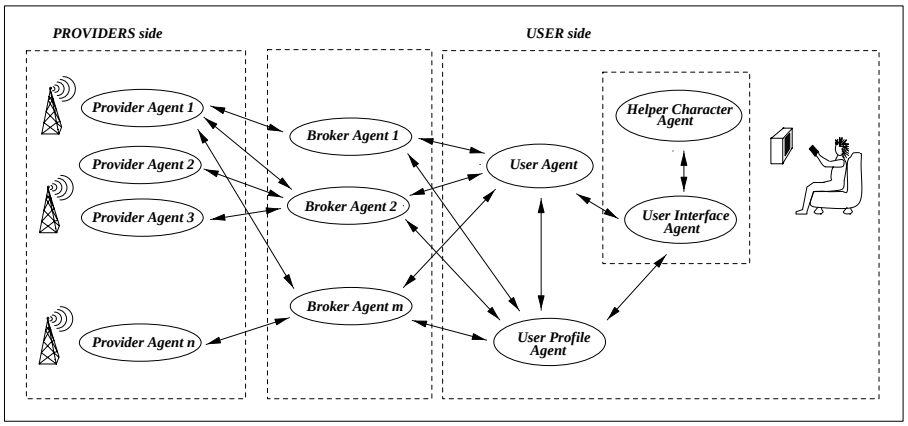

Figure 1: The AVEB architecture: on the right, the user interacts with the system supported by the Helper Agent and the User Interface Agent. Hidden to the user, the User Profile Agent updates the user preferences according to the events registered by the User Interface Agent. In the centre, Broker Agents match requests from the users, with services made available by Provider Agents. Arrows represent the communication flow.

Based on these general considerations, a multi-agent system was developed, in which interactions among the human actors are mediated by a community of software agents (see Figure 1). In the system, providers and brokers are represented by their own agents, while the user side is managed by a subsystem composed by four agents: the User Agent (UA), the User Interface Agent (UIA), the Helper Agent (HA) and the User Profile Agent (UPA). The Provider Agent (PA) manages the databases of the programs on offer. The Broker Agent, based on the specific information received from the Provider Agent, tries to match the offer with the demands and interests of the users. For what concerns the user side, the User Interface Agent supplies the user with a graphical interface modelling user details, such as hardware available and currently active subscriptions. Through this interface, the user can interact with the system, register or deregister himself and select programs of interest. In addition to this, the Helper Agent assists the user with human-like interactions which change over time adapting to the specific user. The User Agent manages a set of user models, building relationships between the application and user details. It also acts as a gateway to the Broker Agents for the UIA and HA. The User Profile Agent maintains a dynamic profile of the user preferences based on a Bayesian Belief Network representation [5,6] of the DVB model [7]. The resulting probabilistic representation is used to filter the set of programs on offer, and to proactively search for potentially interesting events.

Presently, a first demonstrator of the system has been developed. For the time being, the services that the user can access are limited to news services supplied by two Provider Agents. The agents have been developed by people belonging to different academic and industrial organisations using different approaches and technologies. The agents live on the FIPA-compliant platform JADE [8] and their interaction complies with FIPA specifications.

4. Pros and Cons of FIPA

This section provides an analysis and evaluation of the advantages (Section 4.1) and drawbacks (Section 4.2) of using the FIPA standards for developing MAS (see table 1 for a summary of FIPA pros and cons)..

4.1 What can one expect from FIPA in developing a MAS?

The FIPA standards provide specifications about a wide variety of topics and issues that cover more than what is typical found in distributed systems literature. These details provided by FIPA are very useful for the conceptual design of a MAS.

First of all the role of specific classes of agents have been identified, and not only for the reference applications like AVEB. In few cases this may be a bit misleading, when the specifications may require further refinements. Nevertheless the reference models and the services that FIPA compliant agents have to supply are general and cover scientifically relevant topics. For example in the AVEB application the role of many agents has been coarsely sketched out by FIPA97 Part 6; moreover FIPA98 Part 8 concerning the human-agent interaction has deeply influenced the design of the User Interface Agent and the User Profile Agent.

FIPA pros	FIPA cons
Defines a set of key components for developing agent systems	Does not define patterns of composition which would support agent development and assist interoperability.
Communicative acts for standard interaction	Semantics is based on mental agency hence may limit interoperability
Protocols for complex message interaction	Not based on any semantics at this stage
Agent communication language syntax	
Content language syntax	SL semantics has been shown to be incomplete [17]
Definition of certain ontology requirements	Better support for ontologies is required if rich autonomous interoperability is to be supported for open services
Agent platform management requirements (DF, Services) assists in distributed development and general software integration	Bootstrap process in the platform is complicated. This may be rationalised for FIPA 2000
	Support for deadlock states, debug tools and general error handling is currently not specified
Flexibility is supported in the agent interaction requirements.	Agents are not compelled to answer hence termination of interaction may be difficult to establish. The use of time preference will create other complexities
Provides general reference models	FIPA assumes an agent driven approach which creates a bootstrap problem for client driven systems
Defines some basic human agent interaction requirements from a protocol perspective	Agent driven approaches does not give a natural way to support proactive behaviour in agents
	Service agent platform dependency
	Limited error handling

Table 1: Summary of FIPA pros and cons

Secondly, FIPA offers a set of application independent specifications that are extremely useful for the design of the agents and their interactions. The predefined communicative acts, which can be considered as atomic message operations, are general and have a rather clear semantics. The FIPA protocols are powerful abstract specifications to structure and guide the flow of communication among agents. Furthermore, the FIPA choice to adopt formal languages to encode the content of messages allows the transmission of complex data structures in a way independent of the specific programming languages used to implement the agents.

Switching from the design aspects of a MAS to those related to the implementation, one of the main advantages in using the FIPA standards is that the realised system is open. This means that an agent running on another system may easily interact at run time with the agents of your system. A distinction has to be made at this point: openness is a feature of an already implemented system, but for the actual implementation phase, one may wonder what are the advantages given by the FIPA standards to agent programmers. Basically the main benefit is that of working within an agent platform. Indeed it hides the low-level details concerning the actual hardware and operating system used, and provides a high-level communication mechanism. In this respect FIPA defines an abstract communication layer comparable with that defined by other standards specialised in message passing [9].

4.2 What can one *not* expect from FIPA in developing a MAS?

In distributed systems characterised by modules (the agents) with a high degree of autonomy and independence, it is difficult to maintain the global consistency. Situations in which a specific interaction between two agents becomes stuck causing a deadlock state for the whole system are rather common, especially during the development. In this respect FIPA does not provide any specifications for useful tools like debuggers or message tracers, so that whether to support such tools or not is left to the specific agent platform employed.

Moreover in FIPA there is a conceptual problem related to the agent interactions. A basic FIPA specification is that an agent is not compelled to answer to a message. This assumption, although introduced for the sake of flexibility, creates the problem of how to establish the termination of an interaction: an agent waiting for an answer does not know whether the peer agent is busy (and will reply later) or it does not want to answer. This has a negative impact on protocols which imply synchronisation between the involved agents and whose termination is subordinate to the reception of all the prescribed communicative acts. To overcome this problem, FIPA allows time specifications to be placed in a message to constraint the reply; but this in turn

introduces new problems when the peer agent needs time to compute the answer.

One of the first development problems, typical of client driven systems, is the bootstrap process. In this respect FIPA makes the implicit assumption that agent interactions are agent driven but are based on a passive approach. The problem is that most FIPA protocols expect an incoming message from somewhere; nevertheless the process has to start from some point of interaction. This passive nature also implies that proactive behaviour in an agent has to be specially handled by the developer.

When implementing AVEB, one of the issues that soon emerged has been that of errors. In complex distributed systems the ability to detect and report clearly the conditions that cause an error is extremely useful for debugging and fundamental (although not sufficient) at run time to avoid that the system goes in a globally inconsistent state by providing explicit error reasoning capabilities. A part from the low-level errors concerning the agent platform (which are transparent to the agents), most of errors typically come from the contents of messages. In this respect FIPA specifications about error detection and reporting are rather weak and preliminar.

Working with a specific agent platform has some drawbacks: first of all the code is portable only if delivered with the platform. This does not depend strictly on the programming language, but on agent platform in itself. FIPA for example does not give specifications for basic commands aimed at exchanging messages (send, receive, poll, check) neither their modalities (blocking, immediate, etc). Other important issues concern the execution of agents, for example multi-threading; complex agents typically organise their computation in a parallel way, assigning to each protocol or interaction a specific and independent thread of execution. Working with an agent platform that supports multi-threading or not, changes radically the way in which the agent (and obviously code) is structured. The reason why FIPA does not provide specifications for such issues is that they concern the internal of agents and therefore are outside the scope of FIPA; however we believe that a standardisation could be both possible and useful.

5. The implementation of an AVEB agent: the User Profile Agent

This section reports on positive and negative aspects of using the FIPA standards for implementing the User Profile Agent. Our discussion is primarily focused on the implementation of the part of the agent making explicit use of FIPA: the interface with the outer community of agents. However, in order to illustrate the application of FIPA standards as having limited constraints on the inside of the agent, the internal functioning of UPA is also outlined. An example of interaction between UPA and other agents is also described.

Main purpose of UPA is to assist the user in the selection of potentially interesting programs from a large set of TV programs on offer. The knowledge necessary to accomplish this task should primarily come from observation of the user's behaviour; in order to do so, a suitable User Profile will be set up for each user, and made evolving according to the information collected by UIA. UPA should also be able - on request - to: (1) register/deregister a user; (2) modify the properties of an already registered user; (3) update his preferences; (4) filter a list of offers; (5) provide suitable information about the registered users. Moreover, UPA should act autonomously querying Broker Agents for interesting programs. Following FIPA97 Part 6, our User Profile consists of two parts: 1) properties concerning personal information about the user (e.g., name, gender, age, occupation); 2) dynamic information related to the preferences of the user with respect to a specific domain (i.e., AVEB). The set of actions that UPA can execute and their corresponding semantics are largely inspired by FIPA97 Part 8. Though not as general as prescribed by FIPA, UPA covers the basic functionalities of the User Personalization Service.

Two different modalities can be identified in UPA's behaviour: in the first case UPA acts as a server providing the other agents with a set of functionalities; in the second, it behaves in a more proactive fashion: from time to time, UPA can autonomously ask the brokers registered on the platform for a list of programs of interest for the user; collected programs are then filtered by exploiting the information stored in the User Profile. In the following, UPA-server and UPA-proactive will indicate embodiments of these functionalities. Looking more closely at the internal organisation of UPA, a third, independent component can be identified: the BBN-server. This service, which is only accessible by the UPA, manages all the operations related to the user preferences component of the User Profile. The BBN-server is written in the C-language and employs the NETICA API(c) library. Both UPA-server and UPA-proactive communicate with the BBN-server through a socket stream.

UPA is written in the Java language within the JADE environment [8], which supplies the FIPA-compliant platform [2] on which agents live. By providing white and yellow pages services, message routing and life-cycle management, JADE allowed the developers to concentrate on the high level design of the

agent. In the development of the interface layer between UPA and the other agents, the set of communicative acts and protocols established in FIPA proved very useful. The advantage has been twofold: first, the design of the interface reduced to just selecting, from the set of protocols defined by FIPA, those most suited to support the desired agent behaviours. In fact, the whole interface is based exclusively on the FIPA-request and the FIPA-query protocols. Secondly, consistent use of FIPA protocols yielded modular, reusable software. For example, the FIPA-request protocol, which is used to ask UPA to perform an action, has been implemented as a self-contained, generic method. In this way, it is easy to provide the agent with the capability of carrying out unforeseen activities without modifying the interface, or reuse it for a different application. Even though the communication of both UPA-server and UPA-proactive occurs through the same FIPA protocols, the way they are used is different. The UPA-server never initiates the communication; it is always waiting for a message of either *request* or *query-ref* type, to which it responds with one or more messages, depending on the protocol and the situation. Therefore, the problem never occurs of what to do in case the peer agent does not respond. Conversely, UPA-proactive starts FIPA-query and FIPA-request protocols on its own. After sending a *request* or a *query* communicative act, UPA waits for an answer. While waiting, it does not know whether the peer agent will respond or not. To overcome this problem, FIPA introduces the message parameter ":reply-by"; the idea is that if the message does not arrive within the time specified, the agent can terminate the protocol. Since this does not guarantee consistency, cooperative behaviour of agents needs to be enforced. This is a pretty strong assumption, which implies that agents always respond to the incoming messages. Relying on this collaborative framework, UPA-proactive always waits until it receives an answer. The internal organisation of UPA is such, however, that UPA-server can independently go on answering the incoming messages.

The FIPA choice of encoding the content of messages by means of suitable content languages enables the exchange of complex information across the community of agents - independently of the specific programming language used to implement the agents. However, a possible drawback may consist in an increased difficulty of detecting and reporting the errors found in the message. On the other hand, error detection and reporting is crucial to avoid inconsistencies and to permit agents to reason explicitly about potential inconsistencies at run time, as well as to facilitate the developing phase. Unfortunately, FIPA has not as yet specified anything about error management. In our system, this deficiency was overcome by defining a set of error types - some general, and some specific of the application. Error management is reflected in the definition of the AVEB ontology. For the sake of simplicity, we chose to report only the first error found. Reporting all the errors could be useful for debugging purposes (when it is a person to interpret the messages), but may make recovering at run time excessively complicated.

To give an example of how FIPA standards have been employed to implement interactions among agents, we sketch the registration process of a new user by UPA-server. When a new user joins the system, the User Interface Agent (UIA) establishes with UPA (UPA-server) a FIPA-request protocol, sending a *request* message to register the new user. When the message is correct, UPA informs UIA (with an *agree* message) that the new user will be registered; otherwise, UPA answers either with a *not-understood* or with a *refuse* message containing information about the error being detected. To register the new user, UPA creates a new structure with the user property, including a new identifier. Then, it sends the request of registering the new identifier through the socket connecting to the BBN-server. This in turn, creates the probabilistic model of user's preferences. UPA finally sends to UIA either an *inform* message containing the new identifier, or, in case something wrong happened, a *failure* message containing the error reason.

6. The role of FIPA in order to achieve MAS Cooperation

In this section we consider the broader picture of MAS cooperation requirements and how the FIPA standards may be used and extended. A recent view of agent communication languages provided by Singh (see [16]) illustrates the problems encountered when using KQML and ACL semantics. The main criticism is the communication language focuses on the internal agent view of the world making interoperability between MAS systems almost impossible. This is due to the co-operative behaviour of an agent system relying heavily on the implicit reasoning of an agent which is devised carefully by the developers. It is possible to overcome some of these restriction within KQML and ACL.

In the AVEB application co-operation relies upon FIPA protocols. An extension by the application was made to improve error handling when using the protocols. Also, the current content language is not SL as proposed by FIPA. A number of interpretation problems where encountered with the language and to pursue development a simpler content language was supported by JADE. The ambiguity of the language semantics is well illustrated by Wooldridge (see [17]).

Both KQML and FIPA ACL semantics is based on speech acts [4]. These communication languages encourage mental agency of belief desire and intention model. However, this model does not provide the means to interpret the beliefs and intentions of a random agent [16]. Further to this we encounter other problems with BDI models [18]. Fischer observed that a BDI architecture, in practice, has little fundamental difference between desires and intentions. Also, while practical systems incorporate elements termed beliefs and intentions, these are distinct from the formally defined beliefs and intentions model. Further to this most languages used for implementing agents do not provide an obvious representation for linking beliefs to intentions. This needs to be programmed by the developer.

To summarise, there are three key interrelated problems:

1. the key communication languages' semantics used in agent architectures are based on this mental agency which represents the private context of an agent,
2. the BDI formal semantics often suffers from only a tenuous link to the systems that are implemented and
3. a limited distinction between intentions and desires for an implementation.

Essentially, a developer often treats the communication language as a software interface. The interactions, dialogues or protocols are a set of calls to a particular program. Often an internal model of an agent interprets the input from a message, using both its current beliefs and the message to create a context in which to determine what to do next. The agent developer, like with most software development, will have a knowledge of the expected input (communications received) and the expected output (communications sent). Protocols, communicative acts and content definitions are driven by the application requirements. Certain degree of separation can be achieved e.g. through the use of FIPA protocols to obtain a level of architectural openness. But much is still tied implicitly to the internal reasoning of the agent community to ensure coherence of interaction and service delivery. This limits interoperability and flexibility of an implementation

6.1 Social Agency and commitments.

Singh [16] proposes a shift from this mental agency model to a social agency model in order to develop a framework of interoperability. As discussed in the previous section there are current limitations when using current languages, such as, KQML or FIPA ACL. Part of the solution to creating flexibility but co-ordinated systems is the use of protocols. However, much of the MAS' social behaviour and an agent's commitments are based on the application/domain specifics. The design of a multi-agent system which supports social agency and commitments through protocols will separate out domain independent concerns from domain specific concerns.

The approach is to achieve cooperation of a set of agents which have committed to achieving a joint goal. There are number of approaches to reasoning about the communication and cooperation in agent systems (see [19,20,21]). Haddadi proposes incorporating cooperation with reasoning about actions. Jennings' solution is based on negotiation as an approach to generalise the cooperative process and Lesser illustrates the inclusion of organisation design into an agent framework.

There exist a number of pre-defined high level protocols, such as, contract net protocol [22] for negotiation. These naturally require some form of commitment agreement. Haddadi defines pre-commitment rules to establish a potential commitment. In order to establish commitments among a set of agents the agents will have to announce their role(s) which must be interpreted into what an agent will commit to. This mapping from roles to commitments essentially establishes a set of cooperative policies, which can be implemented as a set of protocols.

It is worth noting that natural conceptual clustering occurs in the design of agent systems even when an explicit model of social agency does not exist. However, without an explicit model of cooperative policies then interoperability between agent architectures is at best only at the sending and receiving of messages. In order to support these cooperative policies means representing an agent with both an internal organisation and an external organisational view. An agent's belief database must hold two sets of belief states: (a) about itself and (b) the organisation. The representation of the internal reasoning model of an agent appears to favour a belief and intentional model (and perhaps desires). The organisational representation seems to favour the explicit modelling of roles, commitments and cooperative policies.

The organisational model explicitly represents the social agency. As a minimum this agency defines a possible team or cluster of agents explicitly that are involved in an overall service delivery/access. Currently, in many systems most of the social agency is implicit, that is the agents are assumed to cooperate as the explicit computational behaviour of a MAS is defined by the set of services it supports. It is deemed successful if the behaviour exhibited provides the desired functional support of these services. Hence the social agency has a dependency on the application or

service it is supporting. However, as we have seen some common software patterns in current MAS are already emerging. It is quite possible that another common component would be the explicit support of the organisational model that represents in affect this concept of social agency. Already mentioned in this paper are a few approaches to achieving cooperation explicitly which is the first step to gaining social agency. However, there is not, currently, an accepted standard (formal or otherwise) to including such a concept into computational MAS. This limits the potential for scaleablility, robustness, autonomy and flexibility to the implicit assumptions made by the development of each agent acting in a particular MAS making interoperability between MAS dependent on these implicit assumptions.

In order to incorporate a set of software patterns that can express the social agency of each agent explicitly requires a language in which to describe the terms and a way of communicating these terms. There are a number of ways that a developer can extend the current set of components in order to gain a more explicit model of the social agency. An obvious point in which to include this explicit model is in the agent's communication language. Potentially the outer language parameters could be extended to include a specific attribute called for example social-policies. The content property in the communication language can be used to express a particular instance of a policy that the agent is agreeing to. Another feature, which can be extended, is the set of communicative acts and/or protocols to convey the social policy of a particular communication.

Just as there are ontology models, which define objects, such as a user model object which implements the user model ontology, there are social policy models which define a set of objects that can be implemented. These sets of objects define the social policy ontology. The explicitness of the ontology depends on how much assumption is placed on understanding the meaning of a single term representing the policies. For example, there may be a set of social policies that defines the way a team of agents operate when a new user logs on to a particular MAS. A policy could, for example be given the label new-user-policy. This might be the limit to the policy definition in the run time system. To actually understand the meaning of this policy the developer of a new agent, which would like the agent to participate in the policy, would refer to the specifications or implementation details of this policy to find out what the attribute values are and their meaning. However, it is possible to make more aspects about this new-user-policy explicit. We can define the attribute values of such a policy. These attributes might be, simply, size of team (number of agents allowed to participate to limit the group size), language the agents communicate in, the location of the agents, supports a friendly interface, has on-line help etc. An agent can commit to a particular new-user-policy by stating that it only works with agents that operate on a particular platform and/or there are other agents that can support on-line help etc. An agent can register its policies and in what context it will commit to these policies just as it registers its service support. Another aspect to consider is the relationship between the policies and the protocols supported in the communication language. Do different policies demand different sets of protocols?

To summarise, in order to address some of the limitations discussed, we need to establish a set of cooperative policies that can be formulated as protocols in an explicit manner. Roles and commitments need to be defined which support the cooperative policies. Finally, these two orthogonal views of an internal agent's reasoning behaviour and an external organisational model need to be integrated in a structured way.

7. Conclusion

In this paper we have analysed pros and cons of using FIPA standards for building MAS on the base of the experience gained in developing the AVEB application.

The basic claim is that the FIPA specifications are very useful for the design aspects. As far as the implementation issues are concerned, some improvements could be useful; we expect FIPA to evolve in this direction, now that agent platforms compliant with FIPA have been developed and are under testing. Although FIPA provides a number of key concepts in which to develop agent systems and development of complex distributed systems is reduce due to the use of agent platforms the actual compliance testing is still very rudimentary. The main level of testing is through platform registration and the use of FIPA protocols and ACL parsers.

Some of the drawbacks outlined in Section 4.2 could be overcome in future specifications – this is the case of error detection and management. Other limitations may not be considered true drawbacks since they are a direct consequence of FIPA's philosophy and therefore structural within the framework – for example limiting the effect the standardisation has on the specifications about the internal of agents.

Finally some problems are intrinsic in MAS and in distributed systems in general, like the problem of the global consistency and coordination [10,11]. FIPA protocols are a first convincing answer to this need, even though specifications about cooperation and coordination policies would be useful. As far as the problem of understanding the message content is

concerned, FIPA approach to adopt ontologies is again a first convincing step; nevertheless it is still an open issue how to balance explicitness of information with code manageability and efficiency [12,13]. In order to arrive at further interoperability the policies and commitments require explicit representation and interpretation. Otherwise the problem will not be addressed but just moved to another set of implicit assumptions.

Acknowledgements

We would like to thank CSELT for providing their agent platform [8] within the FACTS project. We would like also to acknowledge Bruno Caprile and Luciano Serafini for their active contributions.

References

[1] FIPA - Foundation for Intelligent Physical Agents, http://www.fipa.org/.
[2] FIPA97 Specification, version 2.0, Part 1, *Agent Management*
[3] FIPA97 Specification, version 2.0, Part 2, *Agent Communication Language*
[4] J. R. Searle. *Speech Acts* (Cambridge University Press, Cambridge, 1969).
[5] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, Morgan Kaufmann (1988)
[6] D. Heckerman, *A tutorial on learning bayesian networks*, Technical Report MSR-TR-95-06, Microsoft Research (1995)
[7] Digital Video Broadcasting (DVB), *Specification for Service Information (SI) in DVB system*, ETS 300 468 (1997)
[8] F. Bellifemine, G. Rimassa, A. Poggi, *JADE - a FIPA-compliant Agent Framework*, Proceedings of PAAM99, London, UK, (1999)
[9] Message Passing Interface Forum, *MPI: A Message-Passing Interface Standard*, TR CS-94-230, University of Tennessee, Knoxville (1996)
[10] N. Jennings, Coordination Techniques for Distributed Artificial Intelligence, in *Foundations of Distributed Artificial Intelligence*, Wiley Interscience, pp. 187-229 (1997)
[11] M. Wooldridge and N. Jennings, Formalizing the Cooperative Problem Solving Process, in *Readings in Agents*, Ed: M. Huhns & M. Singh, pages 430-432 (1998)
[12] Y. Labrou & T. Finin, Semantics and Conversations for an Agent Communication Language, in *Readings in Agents*, Ed: M. Huhns & M. Singh, pages 235-242 (1998)
[13] M. Singh, A Semantics of Speech Acts*, in *Readings in Agents*, Ed: M. Huhns & M. Singh, pages 458-470 (1998)
[14] T. Finin & R. Fritzson; KQML: A Language & Protocol for Knowledge & Information Exchange, In Proc. of 19th Intl. Distributed Artificial Intelligence Workshop, pages 127–136. Seatle, USA, 1994.
[15] R. McGuigan, P. Delorme, J. Grimson, P. Charlton, & Y. Arafa, The Reuse of Multimedia Objects by Software in the KIMSAC System, In proceedings of Object Oriented Information Systems, OOIS'98.
[16] Munindar p. Singh, "Agent Communication Languages: Rethinking the Principles", In *IEEE Computer*, November 1998, Pages 40-47
[17] M. Wooldridge, "Verifiable Semantics for Agent Communication Languages" in Proceedings ICMAS 98, pg 349-356
[18] M. Fisher "Implementing BDI-like Systems by Direct Execution" In IJCAI 97, Japan, pg. 316-321, 1997
[19] A Haddadi "Communication and Cooperation in Agent Systems: A Pragmatic Theory" In Springer-Verlag, Berlin, 1996.
[20] N. Jennings "Co-ordination Techniques for Distributed Artificial Intelligence" In *Foundations of Distributed Artificial Intelligence*, Wiley Interscience, pp. 187-229
[21] V. Lesser "Reflections on the Nature of Multi-Agent Co-ordination and Its Implications for an Agent Architecture" In Autonomous Agents and Multi-Agent Systems vol 1. No.1, pg 89-111, 1998.
[22] B. Burmeister, A. Haddadi & K. Sundermeyer "Generic, Configurable, Cooperation Protocols for Multi-Agent Systems", In LNAI, From Reaction to Cognition, MAAMAW 93, pg 157-171, 1993