

---

# A Protocol-Based Semantics for FIPA'97 ACL and its Implementation in JADE

**Jeremy Pitt**

Intelligent & Interactive Systems Group, Department of Electrical & Electronic Engineering,  
Imperial College of Science, Technology & Medicine, Exhibition Road, London, SW7 2BZ  
j.pitt@ic.ac.uk <http://www-ics.ee.ic.ac.uk/>

**Fabio Bellifemine**

CSELT, Centro Studi e Laboratori Telecomunicazioni S.p.A.,  
Turin, Italy  
bellifemine@cse.lt.it

---

## ABSTRACT

There are fundamental limitations on using mental attitudes to formalise the semantics of an Agent Communication Language (ACL). In this paper, we define a general semantic framework for a class of ACLs in terms of protocols, and develop a method for designing and specifying a member of this class, and configuring it for a particular application. We then analyse the performatives and protocols of FIPA'97 ACL from this point of view. We show its usage in the agent execution model of JADE, a software framework to develop agent applications in compliance with the FIPA specifications for interoperable intelligent multi-agent systems.

---

## 1 Introduction

The growing *ease* of network connectivity of computers provided the enabling technology for the agent paradigm. From the *computing* perspective, agents are autonomous, asynchronous, communicative, distributed and possibly mobile processes. From the *AI* perspective, they are communicative, intelligent, rational, and possibly intentional entities.

The common feature of communication has determined that some kind of message passing between agents is required. To provide inter-operability between heterogeneous agents, a commonly understood agent communication language (ACL) is used: examples include KQML [Finin *et al*, 1995], Arcol [Breiter and Sadek, 1996], and FIPA's ACL [FIPA, 1997]. To ensure that it is commonly understood, a formal semantics for the ACL is required. From the AI perspective, the semantics has typically been characterised in terms of speech act theory and framed in terms of the intentional stance (i.e. mentalistic notions such as beliefs, desires and intentions) [Cohen and Levesque, 1995; Breiter and Sadek, 1996; FIPA, 1997; Labrou and Finin, 1998].

However, any formalisation of communication based on the mental states of the participants must deal with what is fundamentally a process of revision and updating of those states. It is unlikely that a single set of axioms will cover all eventualities because communication is inherently context-dependent. What works well in one application may then be inappropriate in another. Therefore there are considerable limitations of intentionality as a basis for defining the semantics of an ACL (see also [Singh, 1998]). Accordingly, in this paper, we define in Section 2 a general semantic framework for a class of ACLs in terms of protocols. Section 3 develops a method for designing and specifying a member of this class, and configuring it for a particular application. In Section 4 we analyse the performatives and protocols of FIPA'97 ACL from this point of view. Then Section 5 shows its usage in the agent execution model of JADE, a software framework to develop agent applications in compliance with the FIPA specifications for interoperable intelligent multi-agent systems.

## 2 A General Semantic Framework for ACLs

Consider the situation illustrated in Figure 1. It shows two agents, each embedded in an environment, which partially overlap. However, rather than communicating by changing the environments, they can communicate by using speech acts and the ACL protocols.

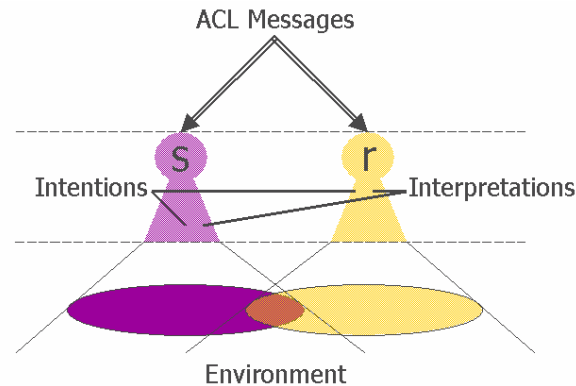


Figure 1: Communicating Agents

We would argue that there are three layers of semantics here:

1. The content level semantics, which is concerned with understanding the content of a message (i.e. the content meaning), and *is internal to an agent*;
2. The action level semantics, which is concerned with replying in appropriate ways to received messages, and *is external to the agents*;
3. The intentional semantics, which is concerned with making a communication and how to react to a received message (i.e. the representational meaning), and again *is internal to the agent*.

We would argue that the current FIPA ACL semantics [FIPA, 1997], for example, is level 3, and because it is *internal* to an agent, its usefulness in standardisation has been questioned [Wooldridge, 1998; Singh, 1998]. The only part of the communication that is amenable to standardisation is the observable tip of the iceberg: namely the communication itself. Note that this properly includes ontologies, so that there may be a ‘standard interpretation’, but the actual interpretation of the content of the message is once again *internal* to the agent.

We are therefore inclined (as with many others, e.g. Smith *et al* [1998]; Singh [1998]) to take a more protocol-oriented view of the ACL semantics. From this point of view, the communication between two (or more) agents can be viewed as a conversation. Individual speech acts therefore take place in the context of a conversation. We then specify the meaning of performatives by describing an input-output relationship. In our case, we define *the meaning of a speech act (as input) to be the intention to perform another speech act (as output)*. The performative used in the output speech act will be determined and constrained by the context (i.e. the conversation state) in which the input speech act occurred, while the content of the output speech act will be determined by the context (the agent’s information state) in which the input was received.

We therefore base this input-output relation on how the object-level content of a message (content\_meanings) induces a change in the information state (info\_state) of the receiver, and how the meta-level action descriptor of a message (the performative) induces a response from the receiver.

Our proposal is that the semantics of performatives can be characterised in these terms, and that this is the semantics that should be specified for a standard agent communication language. This means specifying, for the content, what it ‘means’ for an agent (receiving a message) to add its interpretation of that content to its current information state, and for the speech act, the space of possible responses an agent may make, one of which it is obliged to make. We require that agents use agreed protocols to communicate, and that agents involved in a conversation are behaving according to a protocol (i.e. a conversation policy), and maintain the history of the ongoing dialogue (cf. Bradshaw *et al* [1997]).

An agent  $s$  communicates with (and communicates information to) an agent  $r$  via a speech act. This (possibly infinite) set is denoted by  $\text{speech\_acts}$ , a single member of which is represented by:

$$\langle s, \text{perf}(r, (C, L, O, \text{cp}, i, t_s)) \rangle$$

This is saying that  $s$  does (communicates with) performative  $\text{perf}$  with content  $C$  in language  $L$  using ontology  $O$  in the context of protocol  $\text{cp}$  as part of conversation (identified by)  $i$  at time of sending  $t_s$ . Given any speech act  $\text{sa} = \langle s, \text{perf}(r, (C, L, O, \text{cp}, i, t_s)) \rangle$ , define a function  $f$  such that  $f(\text{sa}) = \text{perf}$ .

For the specification that follows in this section, we first need some sets and functions. Let  $c$  and  $\sigma$  be sets of integers. Then an ACL is a 3-tuple  $\langle \text{Perf}, \text{Prot}, \text{reply} \rangle$ , where  $\text{Perf}$  is a set of performative names,  $\text{Prot}$  is a set of protocol names, and  $\text{reply}$  is a partial function given by:

$$\text{reply}: \text{Perf} \times \text{Prot} \times \sigma \rightarrow \mathcal{P}(\text{Perf})$$

Note that in  $\text{Perf}$  we include the **null** performative which is a ‘do nothing’ (no reply) performative (cf. “silence” as used in Smith *et al* [1998]), and in  $\text{Prot}$  require an empty protocol **no\_protocol** – agents should be able to communicate using one-shot speech acts irrespective of a particular protocol (see the next section). Each element of  $\text{Prot}$  names a finite state diagram which is a pictorial representation of a protocol (or conversation policy). Then  $\text{reply}$  is a (partial) function from performatives, protocols and protocol states to the power set of performatives. Intuitively this states for each performative, ‘uttered’ in the context of a conversation being conducted according to a specific protocol, what performatives are acceptable replies.

These first three items are standard for all agents using the ACL. To fully characterise the semantics, we need three further items which are *relative* to an agent  $a$ , and specify what an agent does with a message, not how it does it:

$$\begin{aligned} \text{add}_a: \text{Perf} \times \text{Prot} \times \text{info\_states} \times \text{content\_meanings} &\rightarrow \text{info\_states} \\ \text{select}_a: \mathcal{P}(\text{Perf}) \times \text{info\_states} &\rightarrow \text{speech\_acts} \\ \text{conv}_a: c &\rightarrow \sigma \end{aligned}$$

Firstly,  $\text{add}_a$  is agent  $a$ ’s own procedure for computing the change in its information state from the content of an incoming message using a particular performative ‘uttered’ in the context of a particular protocol.  $\text{add}_a$  then converts  $a$ ’s (a-type) information state into a new (a-type) information state. Secondly,  $\text{select}_a$  is agent  $a$ ’s own procedure for selecting a performative from a set of performatives (valid replies), and from its current information state generating a complete speech act for this performative which will be its (intended) reply. Finally,  $\text{conv}_a$  maps a conversation identifier  $i$ ,  $i \in c$  onto the current state  $s$  of the protocol,  $s \in \sigma$ , i.e. an agent may be involved in more than one conversation and uses the identifier to distinguish between them. (Note FIPA ACL has message attributes to serve such purposes, but it is not accommodated in the formal semantics.)

The meaning of a speech act is then given by:

$$\|\langle s, \text{perf}(r, (C, L, O, \text{cp}, i, t_s)) \rangle\| = I_r \langle r, \text{sa} \rangle \text{ such that } f(\text{sa}) \in \text{reply}(\text{perf}, \text{cp}, \text{conv}_r(i))$$

This defines the meaning of the speech act to be an intention of  $r$  to perform some other speech act: in effect, it a commitment of  $r$  to  $s$ . The performative used in the speech act as the response is selected from  $\text{reply}(\text{perf}, \text{cp}, \text{conv}_r(i))$ , i.e. it is constrained to be one of the valid performatives according to the protocol. The speech act is generated from  $r$ ’s current information state at the time of selection  $\mathcal{O}_{r@t_{\text{now}}}$  and the state of the conversation  $i$  according to the (agreed) protocol (conversation policy)  $\text{cp}$ , using  $r$ ’s  $\text{select}_r$  function, i.e.:

$$\text{sa} = \text{select}_r(\text{reply}(\text{perf}, \text{cp}, \text{conv}_r(i)), \mathcal{O}_{r@t_{\text{now}}})$$

where  $\mathcal{O}_{r@t_{\text{now}}}$  is given by:

$$\mathcal{O}_{r@t_{\text{now}}} = \text{add}_r(\text{perf}, \text{cp}, \mathcal{O}_{r@t_r}, L_r \| C \|_{L,O})$$

This equation states that  $r$ ’s information state after receiving the message is whatever the result is of adding, using  $r$ ’s own procedure  $\text{add}_r$ ,  $r$ ’s own interpretation (content meaning) of the content  $C$  in language  $L$  using ontology  $O$  to its database at the time of receipt,  $\mathcal{O}_{r@t_r}$ .

### 3 A Method for Designing ACLs

The definition of a particular ACL by a 3-tuple  $\langle \text{Perf}, \text{Prot}, \text{reply} \rangle$  effectively defines a set of finite state diagrams with arcs labeled by performatives and states labeled by which agent's turn it is to 'speak' in the conversation. As we show in the next section, it is relatively straightforward to do this for, for example, the FIPA ACL, which has a relatively small set of primitive communicative acts and a small number of given protocols, most of which have fewer than 10 states. Some attention needs to be paid to protocols where there are more than two participants, the sequence of speech acts is not simple turn-taking, there are timing constraints which affect which actions that may be taken, and so on. It may also be that a protocol, if not infinite, may nevertheless have a 'very large' number of states. Therefore a further generalisation of the specification may be required, by defining rules for generating speech acts between two players (i.e. a game). A further generalisation would be possible if we identify the actual Searle speech act category [Singh, 1998].

However, in general our approach allows us to define a *class* of ACLs, rooted on a common core. We envisage starting from a set of core performatives and protocols (i.e. performatives whose intended meaning is intuitively clear and protocols from some common functions). This would be the 'standard' ACL, the root of this class of ACLs, which could then be extended within the same semantic framework for particular applications. There are three directions in which the core ACL could be extended, as illustrated in Figure 2:

1. *Generalization*. New performatives and new protocols can be defined to create a new ACL where there were general interactions not covered by the standard. For example, auction protocols are a common requirement in some multi-agent systems, and new protocols could be added for the particular type of auction demanded by an application (Dutch, English, etc.).
2. *Specialization*. Additional constraints can be added to transitions, and/or richer description of states can be specified. For example, in the development of a FIPA'97-compliant multi-agent system for real-time load control [Pitt and Prouskas, 1998], we required some interactions to be 'timed out'. Also, a state could be associated with a set of conversation variables that determine the conversation state, and there could be a complex interaction of state changes according to certain transitions given a particular conversation state.
3. *Instantiation*. The specific decision-making functionality for deciding which reply from a set of allowed replies can also be specified. For example, the same protocol may be used in quite different application domains. The decision-making that characterises whether to reply with performative X or performative Y from the same state may then be dependent on very different functions. In principle, application designers can take an ACL "off the shelf", if it contains a protocol that serves the task required, and specify the application-specific functions which the system developers can then implement.

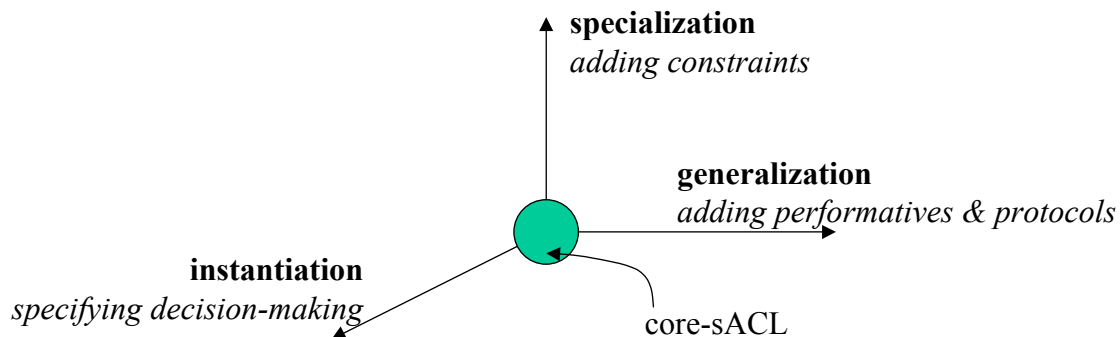


Figure 2: Extending core-sACL

We are currently working on developing tool support to assist in the ACL development process, and a middleware extension that we call *brown pages* that enable one agent to publish a 'non-standard' protocol and other agents to discover how to use this protocol (cf. Bradshaw *et al* [1997]).

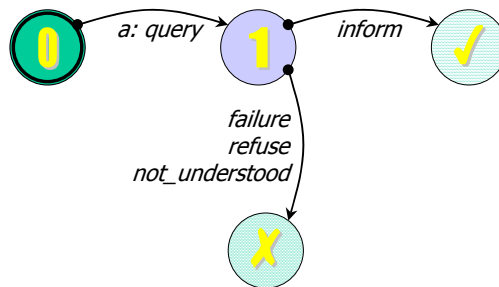
## 4 A Protocol Semantics for FIPA'97 ACL

In this section, we give a protocol-based semantics to FIPA'97 ACL using the method described in the previous section. Assuming we just start with the *inform* performative at the core, we extend FIPA'97 ACL to be the 3-tuple  $\langle \text{Perf}, \text{Prot}, \text{reply} \rangle$  where:

Perf = {*accept\_proposal, agree, cancel, cfp, confirm, disconfirm, failure, inform, inform\_if, inform\_ref, not\_understood, propose, query\_if, query\_ref, refuse, reject\_proposal, request, request\_when, request\_whenever, subscribe, null*}

Prot = {*FIPA\_request, FIPA\_query, FIPA\_request\_when, FIPA\_contract\_net, FIPA\_iterated\_contract\_net, FIPA\_Auction\_English, FIPA\_Auction\_Dutch, no\_protocol*}

The reply function can be determined by turning the FIPA protocol description notation into a Finite State Diagram. This is straightforward for the protocol, *FIPA\_query*, as shown in Figure 3:



<i>query</i>	<i>FIPA_query</i>	0	{ <i>inform, failure, not_understood, refuse</i> }
<i>inform</i>	<i>FIPA_query</i>	1	{ <b>null</b> }
<i>failure</i>	<i>FIPA_query</i>	1	{ <b>null</b> }
<i>not_understood</i>	<i>FIPA_query</i>	1	{ <b>null</b> }
<i>refuse</i>	<i>FIPA_query</i>	1	{ <b>null</b> }

Figure 3: *FIPA\_query* and reply function

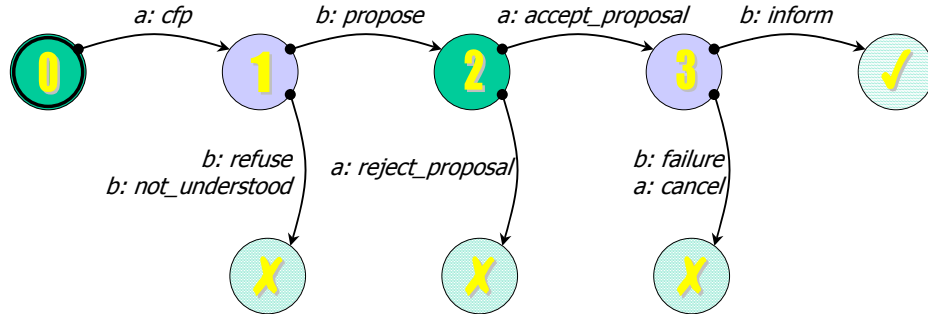
For the other protocols we have to work a little harder. For the *FIPA\_contract\_net* protocol the complication is the *cancel* action from the initiator (i.e. agent *a*, the agent that has the role of manager) of the protocol in state 3. The action is specified so that the manager can cancel the contract due to a change in situation. It does not perturb the reply function, though: it just means that there is an extra action available to agent *a* in this state of the protocol.

This is an indicator that although the FIPA standard protocols are well-motivated and a good starting point, much more work is still needed. For example, why should this be the only occasion when a *cancel* can be issued – what if “the situation” changes directly after the *cfp* has been issued? Why is the *cancel* not available in other protocols? For example, suppose after a *query* in *FIPA\_query*, that the agent finds out the answer from another source. Presumably it should then be able to cancel the query, but not according to the *FIPA\_query* protocol.

The *FIPA\_contract\_net* protocol as a finite state diagram and with associated functions is illustrated in Figure 4.

Note also that the FIPA'97 specification notes that the transition from state 2 to state 3 is subject to some deadline, i.e., that all proposals must be received by a certain time otherwise the manager could wait idly indefinitely. However, this deadline is enormously domain-dependent. In the EU GOAL project, where we developed a agent-based system for distributed document review, the deadline could be days [Pitt *et al*, 1996]. However, in the EU MARINER project, where we are developing an agent-based system for load control in Intelligent Network, the deadline was seconds [Pitt and Prouskas, 1998]. However, the same basic pattern of interaction was common to both systems. This is

why we propose to extend a core ACL with common protocols, and specialise a protocol to suit the particular application requirements with application-dependent constraints.



<i>cfp</i>	<i>FIPA_contract_net</i>	0	{ <i>propose, not_understood, refuse</i> }
<i>propose</i>	<i>FIPA_contract_net</i>	1	{ <i>accept_proposal, reject_proposal</i> }
<i>accept_proposal</i>	<i>FIPA_contract_net</i>	2	{ <i>inform, failure</i> }
<i>refuse</i>	<i>FIPA_contract_net</i>	1	{ <b>null</b> }
<i>not_understood</i>	<i>FIPA_contract_net</i>	1	{ <b>null</b> }
<i>reject_proposal</i>	<i>FIPA_contract_net</i>	2	{ <b>null</b> }
<i>inform</i>	<i>FIPA_contract_net</i>	3	{ <b>null</b> }
<i>failure</i>	<i>FIPA_contract_net</i>	3	{ <b>null</b> }
<i>cancel</i>	<i>FIPA_contract_net</i>	3	{ <b>null</b> }

Figure 4: *FIPA\_contract\_net* and reply function

The *FIPA\_request* and *FIPA\_request\_when* protocols are examples of the sort of instantiation required to choose between alternative speech acts in choosing a reply. To fully specify the *FIPA\_request* protocol according to the FIPA'97 specification, first we have to deal with the specification that one agent make two consecutive speech acts, without any intervening action from the other agent. Again, we could handle it as above, or in this case we could use protocol composition, as shown below in Figure 5:

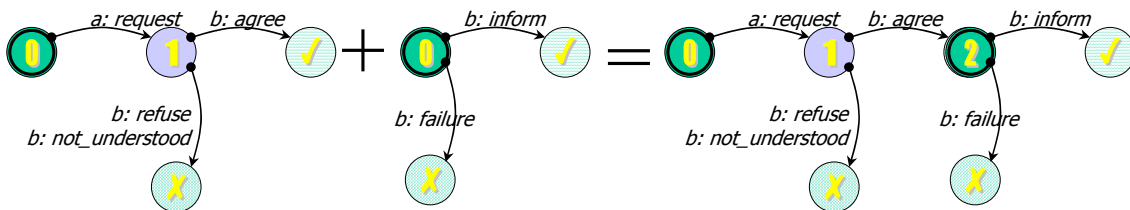


Figure 5: *FIPA\_request* Protocol

The second part of the specification requires specialising the protocol with the general conditions which enable the agent to choose between the different alternatives. Of course, other applications may need to specify this instantiation differently. Here we complete the specification in terms of a BDI (Beliefs-Desires-Intentions) agent architecture [Kinny et al, 1995; Pitt and Mamdani, 1999].

The BDI specification for initiating a conversation according to the *FIPA\_request* protocol could be:

$$D_s \text{DONE}(A) \ \& \ B_s \text{Capable}(r,A) \ > I_s \langle s, \text{request}(r, A, \text{FIPA\_request}) \rangle$$

This states that if agent *s* desires (D) that action *A* be done, and believes that *r* is capable of *A*, then *s* will form the intention to request *s* to do *A*, using the *FIPA\_request* protocol. After performing the speech act, so *r* receives the message, *r*'s behaviour can be specified as:

```

[s,request(r, A, FIPA_request)]
  (Capable(r,A) >
    (ok_to_do(A) > I_r<r,agree(s, A, FIPA_request)>
      & I_r<r, A>
      & B_r[r, A] (success(A) > I_r<r,inform(s, DONE(A), FIPA_request)>
        + (failure(A) > I_r<r,failure(s, A, FIPA_request)>))
      + (-ok_to_do(A) > I_r<r,refuse(s, A, FIPA_request)>))
    + (-Capable(r, A) > I_r<r,not_understood(s, A, FIPA_request)>))

```

This specification states that:

```

after s requests r to do A, then
  if r is capable of A then
    if it is ok for r to do A then
      r intends to tell s that it agrees to do A
      r intends to do A
      r believes that after r does A then
        if A is done successfully, then
          inform s that DONE(A) is true,
          else inform s that the attempt to do A failed
        else tell s that r refuses to do A
      else tell s that r does not understand the request

```

We note that the implementation of Capable, ok\_to\_do, and success or failure of actions are all application dependent. Therefore we can specify the actual decision making needed to compute the truth- or false-hood of these predicates. This is the process we call instantiation, and it is now possible to completely implement the agent. The actual agent implementation can be done using JADE.

## 5 Implementation in JADE

JADE is a software framework to develop agent applications in compliance with the FIPA specifications for interoperable intelligent multi-agent systems. The goal is to simplify development while ensuring standard compliance through a comprehensive set of system services and agents. JADE can then be considered an agent middle-ware that implements an Agent Platform and a development framework. It deals with all those aspects that are not peculiar of the agent internals and that are independent of the applications, such as message transport, encoding and parsing, or agent life-cycle. The JADE features and its internal architecture are described in [Bellifemine et al, 1999].

The JADE implementation of the agent execution model encourages agent programmers to put in practice the protocol-based semantics formalized in this paper.

A distinguishing property of a software agent is its *autonomy*: an agent is not limited to react to external stimuli, but it is also able to autonomously start new activities. This requires each agent to have an internal thread of control; however, an agent can engage multiple simultaneous conversations, besides carrying on other activities not involving message exchanges. JADE uses the *Behaviour* abstraction to model the tasks that an agent is able to perform and agents instantiate their behaviours according to their needs and capabilities. From a concurrent programming point of view an agent is an active object, holding inside a thread of control. JADE uses a *thread-per-agent* concurrency model instead of a *thread-per-behaviour* model in order to keep small the number of threads required to run the agent platform. This means that, while different agents run in a preemptive multithreaded environment, two behaviours of the same agent are scheduled cooperatively. Apart from preemption, behaviours work just like co-operative threads, but there is no stack to be saved. A scheduler, implemented by the base Agent class and hidden to the programmer, carries out a round-robin non-preemptive policy among all behaviours available in the ready queue, allowing the execution of a Behaviour-derived class until it will release the execution control by itself. If the task relinquishing the control has not yet completed, it will be rescheduled the next round unless it is blocked; in fact, a

behaviour can block itself, for instance while waiting for messages to avoid wasting CPU time and doing busy waiting.

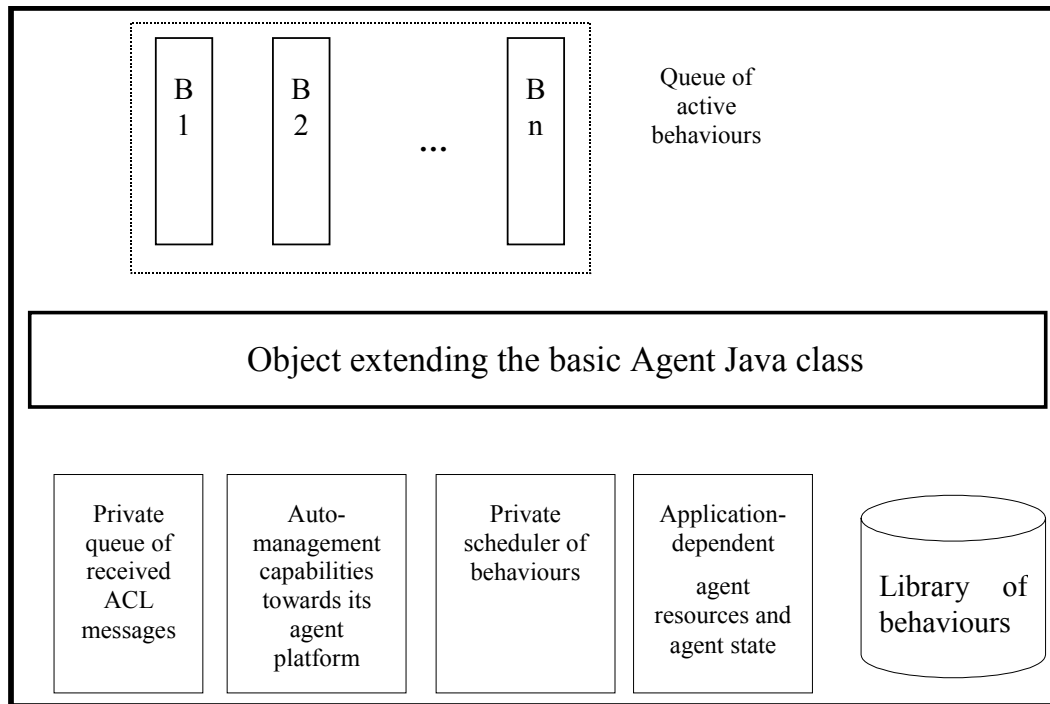


Figure 6 - Internal view of a JADE Agent

The *Agent* class (here the term class denotes a Java class) provided by the JADE framework, represents a common superclass for user defined agents. Therefore, from the point of view of the programmer, a JADE agent is simply a Java class that extends the base Agent class. It allows inheritance of:

- the capability of sending and receiving ACL messages, a private queue for received messages, and the capability of filtering and managing this private queue;
- some basic capabilities that deal with all those management tasks (mostly interfacing directly with an agent platform, such as registration, configuration, remote management, ...),
- and a scheduler of behaviours, as later explained.

Further to that, in order to exploit all the JADE capabilities, a developer should implement all the agent-specific tasks through a set of *Behaviour* classes. One of the JADE packages already provides the skeleton of all the interaction protocols that have been standardised by FIPA, such that the programmer can extend a protocol with its application-dependent handlers, one handler for each state of the protocol. These application-aware interaction protocols form the Library of Behaviours shown in Figure 6.

In this way, the programmer is encouraged to think in terms of interaction protocols, rather than single communicative acts. An agent that, based on its internal state, takes the intention to reach a goal, does not select the sequence of single communicative acts that allow to reach that goal. Rather, it selects the interaction protocol that includes that goal in one of its ending states. All the available protocols are in the library of behaviours, they are selected, instantiated, appropriately customized (e.g. by passing some parameters) and moved into the queue of the behaviours (i.e. the agent tasks) being currently executed.

The *Agent* class includes, and hides to the programmer, a round-robin scheduler of behaviours (that is agent activities). Finally, the *Agent* class provides the single point of contact for all those application-dependent resources and agent state.

In order to make things rather simple and fast, the private scheduler discretizes the time as follows: at each quantum of time the next behaviour is retrieved from the queue of active ones, and its *action()* method is executed. Performing this action, may change the environment, establish new goals and beliefs, possibly adding new behaviours to the active ones or removing existing ones (all these being done through the capabilities of the *Agent* class). This pretty empirical notion of time allows simplifying the implementation of the scheduler and, mostly, allows making very fast the change of context between behaviours. On the other hand, it puts some burden to the programmers that, for instance, must absolutely avoid an *action()* method entering into an infinite loop, otherwise the cooperative scheduler would completely fails. Because this feature can be overridden by instantiating Java Threads rather than JADE Behaviours, the JADE philosophy "pay as you go" is still implemented, that is, user-defined agents incur in runtime costs associated with a JADE feature only when they actually use that feature.

Reconsidering the theoretical framework, JADE fixes the 3-tuple  $\langle Perf, Prot, reply \rangle$  and does not allow an agent to extend it and create new dialects. In fact, as pointed in section 3, this 3-tuple is the necessary standard ACL: (1) the *Perf* set is fixed because if an agent tries to send a performative that does not belong to the FIPA set, an exception is thrown; (2) in a certain sense, also the *Prot* set is fixed because JADE provides behaviours only for those interaction protocols that have been implemented by FIPA. Of course, the programmer is still free to extend this set by implementing new protocols. It is under consideration also the implementation of a new feature that would allow an agent to construct at run-time a new protocol; and (3) for each element of the *Prot* set implemented by JADE, the *reply* function is also fixed and cannot be changed by the programmer.

For each agent, and for each instantiation of an interaction protocol, the programmer needs instead to implement the three application-dependent functions:  $add_a$ ,  $select_a$ ,  $conv_a$  by appropriately implementing the method that handles each state of the protocol.

## 6 Conclusions and Further Work

The specification of standards are extremely important engineering tool for achieving interoperability in heterogeneously designed and implemented multi-agent systems. The FIPA standardization body has produced a specification of a syntax and semantics for a 'standard' ACL to address the essential requirement for communication between agents. This is loosely based on speech act theory, and defined in terms of a set of performatives or communicative acts. A communicative act occurs whenever one agent sends a message to another.

At present, though, it is not clear whether the specification of the formal semantics is intended to be normative or informative, so that: (A) it is not clear how the semantics and axioms should be used: when they are providing guidance for the developers and when they are requirements that the agents themselves are responsible for satisfying; (B) the information states of both sender and receiver are being updated, and, according to context, more axioms are needed to infer that some condition holds in either of the participants after a communicative act has been either performed or observed; and (C) the FIPA performatives or protocols contained in the specification are not exhaustive, and while there is scope for extending the ACL, it is not clear what the mechanism is for documenting and distributing a proposed extension nor the implication for compliance.

We therefore contend that the formal semantics is too strict to be normative, in that intentional conditions on the performance of speech acts do not generalise across all agents and all applications. While well-motivated under certain assumptions (e.g. co-operation), it can at best provide only informative guidelines to an agent developer.

Our approach allows us instead to identify a *class* of ACLs. We envisage starting from a set of core performatives and protocols (i.e. performatives whose intended meaning is intuitively clear and protocols from some common functions). This would be the 'standard' ACL, the root of the class of ACLs, which could then be extended within the same semantic framework for particular applications. New performatives and protocols could then be defined to create a new ACL where there were general interactions not covered by the standard. The standard protocols could also be reified and

specialized for particular tasks, i.e. any time constraints or decision-making needed to determine actions in a conversation for a particular application domain (cf. Pitt *et al* [1996]). There is then a direct mapping into the implementation model of JADE, leaving only code stubs for the application-specific requirements for the agent developer to implement.

## ACKNOWLEDGEMENTS

This work has been undertaken in the context of the EU ACTS Projects FACTS (AC317) and MARINER (AC333), and UK-EPSC/Nortel Networks funded Project CASBAh (Common Agent Service Brokering Architecture, EPSC Grant No. GR/L34440), and support from these funding bodies is gratefully acknowledged.

JADE is a trademark registered by CSELT and is mainly the result of a research activity. The authors wish to thank Agostino Poggi and Giovanni Rimassa, from the University of Parma, for the significant collaboration into the analysis and implementation of JADE, and Abe Mamdani and Frank Guerin for discussion on the framework and method.

## REFERENCES

- Bellifemine, F., Poggi, A. & Rimassa G., JADE – A FIPA-compliant agent framework. In *Proceedings of PAAM99*. London, 1999.
- Bradshaw, J., Dutfield, S., Benoit, P. & Woolley, J. KAoS: Toward an industrial-strength open agent architecture. In J.M. Bradshaw (Ed.) *Software Agents*. AAAI/MIT Press, 1997.
- Breiter, P. & Sadek, M., A Rational Agent as a Kernel of a Co-operative Dialogue System: Implementing a Logical Theory of Interaction. In *Proceedings ECAI'96 Workshop on Agent Theories, Architectures and Languages*, pp261–276, Springer-Verlag, 1996.
- Cohen, P. & Levesque. H., Communicative Actions for Artificial Agents. *Proceedings ICMAS'95*, AAAI Press, 1995.
- Finin, T., Labrou, Y. & Mayfield, J., KQML as an Agent Communication Language. In J. Bradshaw (ed.), *Software Agents*, MIT Press, 1995.
- FIPA, FIPA 97 Specification Part 2: Agent Communication Language. FIPA (Foundation for Intelligent Physical Agents), <http://www.fipa.org/>, 28<sup>th</sup> November, 1997.
- Kinny, D., M. Georgeff, J. Bailey, D. Kemp and K. Ramamohanarao. Active Databases and Agent Systems: A Comparison, *Proceedings Second International Rules in Database Systems Workshop*, Athens, 1995.
- Labrou, Y. & Finin, T., Semantics for an Agent Communication Language. In M. Singh, A. Rao & M. Wooldridge (eds.), *Intelligent Agents IV*, LNAI1365, Springer-Verlag, 1998a.
- Pitt, J., Anderton, M. & Cunningham, J., Normalized Interactions between Autonomous Agents. *Computer-Supported Collaborative Work*, 5, pp201–222, 1996.
- Pitt, J. & Mamdani, A., A Protocol Based Semantics for an Agent Communication Language. *Proceedings IJCAI'99*, to appear, 1999.
- Pitt, J. & Prouskas, K. (eds). Initial Specification of Multi-Agent System for Realisation of Load Control and Overload Protection Strategy, MARINER Project (EU ACTS AC333) Deliverable D3, <http://www.teltec.dcu.ie/mariner>, 1998.
- Singh, M. Agent Communication Languages: Rethinking the Principles. *IEEE Computer*, pp40–47, December, 1998.
- Smith, I., Cohen, P., Bradshaw, J., Greaves, M. & Holmback, H. Designing Conversation Policies using Joint Intention Theory. In V. Lesser (ed.), *Proceedings ICSMAS'98*, 1998.
- Wooldridge, M. Verifiable Semantics for Agent Communication Languages. In V. Lesser (ed.), *Proceedings ICMAS-98*, 1998.