Denotational Semantics for Agent Communication Languages

Frank Guerin and Jeremy Pitt
Intelligent and Interactive Systems
Department of Electrical & Electronic Engineering,
Imperial College of Science, Technology & Medicine,
Exhibition Road, London, SW7 2BZ.
+44 20 7594 6331
{f.guerin,j.pitt}@ic.ac.uk

ABSTRACT

The dilemma encountered in the design of an agent communication language (ACL) for an open society is that it should be based on externally observable phenomena yet it should capture something of the intuitions behind the high level abstractions typically found in internal mental states. Our solution treats an ACL message as a declarative statement that is given a procedural interpretation by a denotational semantics. This defines a speech act as a function between states. These states are *social states* which store public information including expressed mental attitudes and control variables. Expressed mental attitudes are externally observable and capture the conventional public meaning of communication. The variables control the flow of conversation in a protocol. We conclude firstly that since the denotational semantics is based on externally observable phenomena, it is possible to verify compliance and prove properties of protocols. Secondly, since the semantics is more expressive than behavioural specifications, it lays the foundation for high-level communication between intelligent agents.

1. INTRODUCTION

We are concerned with specifying communication languages for agents in an open society. In such systems the internal states of agents are not accessible so the semantic definition should be based on externally observable phenomena. This poses problems for semantics based on declarative statements about mental states. In previous work [1], we analysed an auction protocol based on a procedural approach, using expected replies for external semantics and further specifications for decision-making based on state variables and mental attitudes. This work revealed firstly the existence of a richer state description commonly known to all agents participating in the auction, which was changed by their speech acts. Secondly, the speech acts had consequences beyond just the intended replies. Agents were publicly committed to certain other acts (auctioneer commits to sell, bidder commits to buy). However, these state variables and commitments were specified in the logical specification of the protocol and not accommodated in the general semantic framework. This paper shifts all the public knowledge into an external semantics for agent communication. It therefore advances the semantic framework while being complementary to and consistent with our work on intentional specifications of internals. We treat an ACL message as a declarative statement that is given a procedural interpretation by means of denotational semantic evaluations. We introduce the notion of expressed mental attitudes which are externally observable and capture the conventional public meaning of communication. We define a *social state* which holds all public information and is explicitly modified by the semantic definitions of speech acts. The social state stores propositions and control variables and defines commitments for future acts.

We begin with an analysis of the chief problems with existing ACLs and list some desiderata. We then describe our semantic framework. This is followed by the formal semantics of the framework which includes an auction protocol specification. Finally we evaluate the framework and outline areas for further research. Since the formal semantics are based on externally observable phenomena, it is possible to verify compliance and prove properties such as the possible outcomes of protocols. Furthermore, since the semantics is more expressive than behavioural specifications, it lays the foundation for intelligent communication between agents in the long term.

AGENTS'01, May 28-June 1, 2001, Montréal, Québec, Canada.

2. PROBLEMS AND REQUIREMENTS

Agent communication languages (ACLs) are typically designed by either a *procedural* or *declarative* approach [4]. In the *procedural* approach agents exchange procedural directives, a message defines the appropriate response from the recipient. Behaviour based approaches extend this to larger sections of conversation, identifying certain patterns of communication (auction, contract-net, negotiation) that arise in many situations and designing protocols for them. Speech acts are given a behavioural meaning in the context of a conversation in that only certain responses are appropriate at any stage of a conversation. Procedural communication languages are more oriented toward computer implementation, but there are some difficulties:

1. Behavioural meanings are low level and are more appropriate for computational entities which lack the rationality to plan their own behaviour. Multiagent systems are supposed to provide higher level abstractions than traditional distributed programming [8].
2. If the protocol is the unit of communication and speech acts are defined only in terms of possible replies then communication becomes an ordered exchange of meaningless tokens and the language is not sufficiently expressive.
3. By specifying exactly how a speech act should be used in a conversation there is a lack of flexibility to use it in different situations and one needs a library of predefined protocols to handle every possible conversation.

For example, if the meaning of *request* is defined behaviourally as an intention to reply with *agree* or *refuse*, then a helpful intelligent agent could not suggest an alternative source in response to a request for a service it is incapable of providing. To enable such cooperation, the semantics of request should capture the intuitive meaning i.e. the requestor expresses a desire. The behaviour based approach is adequate for simple reactive agents, but a more meaningful semantics is vital for rational agents to be able to plan their communications intelligently.

The *declarative* approach is akin to natural language. Communicative acts are given a meaning compatible with human intuitions and agents are expected to have a sufficiently advanced level of rationality to interpret received acts and plan new acts. The semantics of acts is typically specified in terms of mental attitudes of participants [2],[5] (also called intentional semantics). Such languages are expressive but have the following drawbacks:

1. It is difficult to verify compliance with a semantics which is based on internal states if the internal states are inaccessible.
2. An agent's autonomy is limited. The example most frequently cited [8] is the inability of FIPA [2] agents to operate in a setting where sincerity cannot be taken for granted.
3. Meaning is specific to a certain mental state thus the speech acts are not flexible for use in a different context. Ideally the meaning of an act should depend on the context.
4. It is not clear how the agent's reasoning should be designed so that desired behaviour results, especially when protocol specification is attempted. Many such languages specify protocols without considering the speech act semantics [2].

A significant advancement to the declarative approach is presented in Singh [7], where semantics are declarative, describing social commitments which are externally observable thus making verification possible.

To summarise, the semantic definition of an ACL should have the following features:

1. Formal definition. If it is specified informally, with a natural language description, it is possible that different vendors will take different interpretations of the specification and produce agents which cannot interoperate.
2. Verifying compliance with the semantics should be possible.
3. State what a speech act means rather than how it should be used. This makes the language more expressive and makes speech acts flexible so they can be used in different contexts.
4. Intuitive or human-like semantics. To facilitate high level communication for intelligent agents.
5. Context dependence. The context in which a speech act is uttered should be a factor in determining its meaning.
6. Based on conventional meaning (public perspective) rather than the inferences of an individual (private perspective).
7. Design Autonomy. Compliant agents should not be forced to adopt a certain architecture, e.g. they should not be forced to have an explicit representation of beliefs, desires or intentions.
8. Consideration of the social aspects of communication and the perspective of the society of agents. It should be possible to specify (observable) commitments to actions.

An ACL should also have a formal protocol description language which can cope with multi-party protocols, nested protocols, parallel conversations and different agent roles within a conversation. There should be a precise relationship between the semantics of protocols and the semantics of the speech acts of which it is composed. We address these issues in the next section.

3. SEMANTIC FRAMEWORK

An ACL message is treated as a declarative statement that is given a procedural interpretation by means of denotational semantic evaluations. We define a speech act as a function from context onto context [3]. Hamblin (cited in Gazdar [3]) suggests that the context can take the form of a list of propositions representing commitments. We extend this to include propositions representing expressed mental attitudes, control variables for the conversation and the history of speech acts. We distinguish between an agent's publicly expressed mental attitudes and its personal internal mental attitudes. These can be different (if agents are not sincere). For example, an agent may express a desire to have an action performed (when it does have that desire in its internal state) in order to test the willingness of another agent to comply. This means that an agent does not need to hold a mental attitude as a precondition to expressing it. Thus meanings are specified from a public perspective, rather than the private perspective of a single individual whose personal inferences are subjective. This public meaning together with social relations captures the conventional meaning of acts within the society.

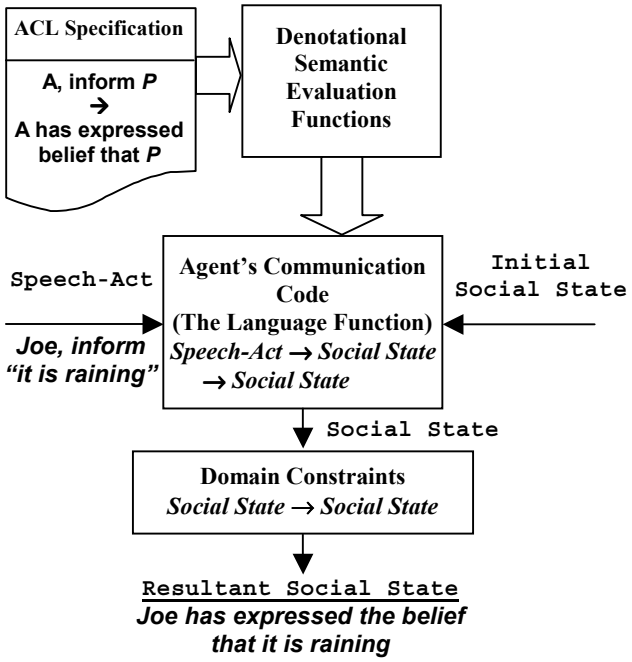

Figure 1 Communication Model

3.1 Public Information

We represent the context as a *social state* which is used to hold information that is externally visible and known to all participants in a social interaction. Within the social state is the state of *persistent social relations* (dealing with long term commitments for example) and the *conversation states* (dealing with all public information relating to the current conversations). Each conversation state can be further broken down into a set of

(1)variables and (2)propositions describing the conversation, a set of (3)*contingent social relations* and the (4)history of speech acts in the conversation. We distinguish between contingent and persistent social relations as follows: contingent relations are solely concerned with the permissions and obligations (to perform subsequent speech acts in the current conversation) that arise as a result of the current conversation state. Each agent keeps a copy of the social state and updates it as speech acts are sent and received (both sender and receiver change state). The conversation variables and propositions control the conversation, although they are labelled with beliefs, desires and intentions, they need not have any direct relationship with the internal states of the participants. The use of intentional labels makes the state description more intuitive for the agent designer. Agents play certain roles in a conversation and these roles may define permissions and obligations at any point in the protocol. Conversation variables keep track of which agents occupy which roles at any time during the conversation.

As shown in Figure 1, The ACL specification is treated like a computer program which is turned into a function (compiled) by the denotational semantics. The function maps a speech act to a change in the state of the society. The ACL specification has three parts: (1)the *Converse Function* gives permissions and obligations for subsequent speech acts based on the current conversation state; (2)the *Protocol Semantics* gives the meanings of speech acts in context of the current protocol and (3)the *Speech-Act Semantics* gives the protocol independent elements of meaning. For example, announcing may have a protocol independent meaning which makes some fact public as well as an additional meaning which depends on whether it is used in an auction or a negotiation. Protocol specific meanings may override the speech act semantics. These three functions are incorporated in the *language function*. The language function matches incoming speech acts with the correct conversation state (based on the conversation identifier parameter) and updates the history in that state.

We do not define the appropriate replies to an act along with the semantics of the act. We consider the separation between the converse function and the semantic function to be important since the determination of the possible replies to an act may not depend on that act alone, but also on the history of previous acts.

Domain constraints further modify the social state. For example, we could specify for some domain that if an agent *A* expresses a desire for an agent *B* to perform an action, then the agent *B* is obliged to do it. Domain constraints are external to the ACL specification, this allows for the same ACL to be used in different domains with different constraints. These rules could appear in the converse function or in the specification of protocols, but this would lead to a specification which is locked in context.

3.2 Private Information

The ACL specification describes how the social state is to change as a result of speech acts and thereby defines the permissions and obligations for participants to perform subsequent acts. It remains to describe how the agents update their internal information states and how they select the next act to perform. Agents in a heterogeneous system may be implemented in diverse ways and may not even have an explicit representation of desires and intentions, therefore the description here is purely informative. The internal code of the agent can be implemented as two functions: (1)The *add* function has as inputs the internal state, social state and speech act and outputs a new internal state. This function updates the agent's mental state when a speech act arrives. (2)The *select* function has as inputs the social state and the internal state and outputs a new internal state. The agent reasons about what acts to perform next and adds them to the its intentions, obviously the permissions and obligations defined in the social state should be a major factor in determining the output if the agent wishes to be ACL compliant. The *add* and *select* functions are implemented after the public information is updated.

String Lists
  Domain l ∈ *String-List* = *String* → *Tr*
  Operations
    newlist : *variables*
      newlist = λ*s*.*false*
    checklist : *String* → *String-List* → *Tr*
      checklist = λ*s*.λ*l*.*l*(*s*)
    updatelist : *String* → *String-List* → *String-List*
      updatelist = λ*s*.λ*l*.[*s* ↦*true*]*l*

Variables
  Domain *v* ∈ *variables* = *Id* → (*Nat* + *String*)
  Operations
    newvars : *variables*
      newvars = λ*i*.(*one*,"null")
    access : *Id* → *variables* → (*Nat* + *String*)
      access = λ*i*.λ*v*.*v*(*i*)
    update : *Id* → (*Nat* + *String*) → *variables* → *variables*
      update = λ*i*.λ*n*.λ*v*.[*i* ↦*n*]*v*

History
  Domain *v* ∈ *History* = *seq* → *Speech-Act*
  Operations
    newHistory : *History*
      newHistory = λ*i*.(*null*,*null*,*null*,*null*,*null*)
    accessHistory : *seq* → *History* → *Speech-Act*
      accessHistory = λ*i*.λ*v*.*v*(*i*)
    updateHistory : *seq* → *Speech-Act* → *History*
      updateHistory = λ*i*.λ*n*.λ*v*.[*i* ↦*n*]*v*

Conversation State
  Domain *s* ∈ *Conversation-State*
    = *variables* × *String-List* × *String-List* × *History*
  Operations
    newconversation: *Conversation-State*
      newconversation = (*newvars*, *newlist*, *newlist, newHistory*)

Speech Act
  Domain *a* ∈ *Speech-Act* =
    Name × *Name* × *perf* × *content* × *cid* × *seq*

Cases of Speech Acts
  Domain *c* ∈ *Speechactcase* =
    Speech-Act → *Conversation-State*→ *Conversation-State*
  Operations
    nostatechange : *Speechactcase*
      nostatechange = λ*a*. λ*s*. *s*
    errorstate : *Speechactcase*
      errorstate = λ*a*. λ*s*.(*error*, *error*)

Figure 2 Semantic Algebras

3.3 Development Method

The expressive power of natural language is achieved through words or phrases that can take different meanings depending on the context in which they are uttered. This gives flexibility to each natural language word or phrase, and an economy to the number of words needed. Meaning is built from a group of words or phrases in a particular order and in a particular pragmatic context, rather than from a single all-meaningful performative. This concept of building complex meaning from simple building blocks is what we would like to borrow from natural language. Our ACL building blocks are [(1)]speech act specification, [(2)]protocol specification (via protocol semantics and converse functions) and [(3)]domain constraints. The speech acts themselves should carry as much useful meaning as possible, while still being flexible, so as they lend themselves to use in many different scenarios. They should not be specific to one protocol, one domain or one ACL. This design philosophy follows through in all specifications, for example, domain specific aspects should be kept out of protocol specifications. The specifications for generic speech acts and protocols can be published. An ACL can then be developed by downloading speech acts and protocols, and if necessary, designing new ones. ACL specifications can be published for designers and also so that foreign agents can compile them into code.

4. DENOTATIONAL SEMANTICS

We now present a method for defining the language function precisely. Essentially we treat propositions in the social state as strings and define a language function mapping a speech act onto a state change. We follow the notation of Schmidt [6] where a function f is written with the lambda calculus abstraction $\lambda a.e$ and the function argument appears after the function. To evaluate the function, the argument replaces occurrences of a in e. For example if $f(y) = y^2$ then we write $f = \lambda a.a^2$ and $f(y) = \lambda a.a^2\ y$. If f is a function, then $[x \mapsto y]f$ denotes the function that maps x to y, but behaves exactly like f for any other argument. $\downarrow i$ denotes the operation such that $(a_1, a_2, \ldots, a_n)\downarrow i = a_i$. Let $a \rightarrow b \square c$ take the value b if a is true, or c if a is false. For $x \in R$ and $y \in S$, we tag the members of each set so they can be distinguished: in$R(x)$ = (*zero*,x) and in$S(y)$ = (*one*,y). To remove the tags for any $m \in R+S$, the expression:

cases m of is$R(x) \rightarrow f(x) \square$ is$S(y) \rightarrow g(y)$ end

evaluates to $f(x)$ when m=(*zero*, x) and evaluates to $g(y)$ when m=(*one*, y). (*a strequals b*) returns true if a and b are identical strings and *cons* concatenates strings. For brevity, we omit the abstract syntax definitions and the standard semantic algebras for truth values, strings, etc. Many of the trivial evaluation functions dealing with low-level string manipulations are omitted. We first present a simplified version of the semantics and illustrate how it works with a simple ACL containing a single speech act which is not part of any protocol. We then move on to the complete semantics and animate an ACL containing an auction protocol.

4.1 Simplified Definition (Speech Acts Only)

We wish to define the language function as a mapping from speech act to state to state. Therefore we must first define the domains for speech acts and states, see Figure 2. The domains [(1)]*seq*, [(2)]*Name*, [(3)]*perf*, [(4)]*content* and [(5)]*cid* are domains of [(1)]sequence numbers for conversations, [(2)]agent names, [(3)]performative names, [(4)]message contents and [(5)]conversation identifiers respectively. A speech act has six parts: sender, receiver, performative, content, conversation identifier and sequence number within this conversation. The conversation state has four parts: [(1)] variables (string or numerical values) which may define roles or other relevant information for the execution of a conversation, [(2)]expressions describing the mental attitudes expressed by speakers (treated as strings), [(3)]contingent social relations (described in section 3.1) and [(4)]the conversation history. The *speechactcase* domain is necessary to handle more than one speech act as will be seen in section 4.1.3. In the simplified version of the evaluation functions (Figure 3) we only consider the speech act's effect on the conversation state with no protocol.

L: Language →
Speech-Act → *Conversation-State* → *Conversation-State*
L[[**speech-act-semantics** S]] = **S**[[S]] *nostatechange*

S: Speech-Act-Semantics → *Speechactcase* → *Speechactcase*
S[[S_1; S_2]] = λc. **S**[[S_2]] (**S**[[S_1]]c)
S[[**[** F **]** M]] = λc. λa. λq. a↓3 *strequals* **F**[[**[** F **]**]]
→ **M**[[M]] a q □ c a q

M: Semantics →
Speech-Act → *Conversation-State* → *Conversation-State*
M[[M_1; M_2]] = λa. λq. **M**[[M_2]] a (**M**[[M_1]] a q)
M[[R]] = λa. λq. (q↓1, *updatelist* (**R**[[R]]a) q↓2, q↓3, q↓4)

R: Proposition → *Speech-Act* → *String*
R[[X R]] = λa. (**X**[[X]]a) *cons* (**R**[[R]]a)
R[[**C**]] = λa. a↓4

X: Proposition → *Speech-Act* → *String*
X[[Y Z]] = λa. **Y**[[Y]] *cons* (**Z**[[Z]]a)

Y: Modality → *String*
Y[[**believe**]] = B **Y**[[**desire**]] = D
Y[[**intend**]] = I **Y**[[**know**]] = K

Z: Actor → *Speech-Act* → *String*
Z[[**R**]] = λa. a↓2 **Z**[[**S**]] = λa. a↓1

F[[**[** F **]**]] simply maps the speech act name to its denotable value

Figure 3 Simplified version of the valuation functions.

4.1.1 Evaluation of a simple ACL

We now evaluate l_1, a sample fragment of an ACL which has only one speech act (*query*) and no protocols, see Figure 4. This defines the semantics of a query as the sender's expression of a desire to know if the receiver believes the content. We want to be able to turn this ACL into a function from a speech act to a change in conversation state. If the performative of the incoming speech act matches the one in our ACL (*query*), then our function should add the sender's expressed desire to the state.

```
speech-act-semantics
[query]
desire S know S believe R C
```

Figure 4 Specification of a simple ACL.

Applying the **L** valuation function to l_1 we have:

L[[l_1]]= **S**[[s_1]] *nostatechange*

Where s_1 = `[query] desire S know S believe R C`

S[[s_1]]=λc.λa.λq.a↓3 *strequals* **F**[[[f_1]]]→ **M**[[m_1]] a q c a q

Where m_1 = `desire S know S believe R C`,
and f_1 = `query`

F[[[`query`]]] = *query*

M[[m_1]] = λa. λq. (q↓1, *updatelist* (**R**[[r_1]]a) q↓2, q↓3, q↓4)

Where $r_1 = m_1$ = `desire S know S believe R C`

All that remains is to simplify **R**[[r_1]] :

R[[r_1]] = **R**[[`desire S know S believe R C`]]

= λa. (**X**[[`desire S`]]a)
cons (**R**[[`know S believe R C`]]a)

= λa. (**X**[[`desire S`]]a) *cons* (λa'. (**X**[[`know S`]]a')
cons (**R**[[`believe R C`]]a') a)

= λa. (**X**[[`desire S`]]a) *cons* (λa'. (**X**[[`know S`]]a')
cons (λa^2. (**X**[[`believe R`]]a^2) *cons* (**R**[[C]]a^2)
a') a)

X[[`desire S`]] = λa. **Y**[[`desire`]] *cons* (**Z**[[S]]a)
= λa. *D cons* ((λa'. a'↓1) a)
= λa. *D cons* a↓1

and similarly for the other **X** valuations. Note that superscripts are used merely to distinguish between separate identifiers represented by the same letter in nested abstractions.

R[[C]]a = λa. a↓4

The full valuation of r_1 is :

R[[r_1]]
=λa. (λa'. *D cons* a'↓1 a)
cons (λa'. (λa^2. *K cons* a^2↓1 a')
cons (λa^2. (λa^3. *B cons* a^3↓2 a^2)
cons (λa^3. a^3↓4 a^2)
a')
a)

=λa.*D cons* a↓1 *cons K cons* a↓1 *cons B cons* a↓2 *cons* a↓4

R is a function from speech act to string, effectively it takes the speech act *a* and puts the sender, receiver and content in the right places in the string. This completes the denotation of l_1.

4.1.2 Testing sample speech act inputs:

L[[l_1]] is a function, given a speech act and conversation state, it returns the new conversation state. Looking inside **L**[[l_1]] we find **S**[[s_1]], which is a function of *Speechactcase* onto *Speechactcase.* We see that if the performative of the speech act input to **L**[[l_1]] is anything other than *query*, the returned *Speechactcase* is *nostatechange* and hence the returned state is the same as the original:

L[[l_1]] =λc.λa.λq.(a↓3 *strequals* **F**[[[`query`]]] →
M[[m_1]] a q c a q) *nostatechange*
=λa. λq. (a↓3 *strequals* **F**[[[`query`]]] →
M[[m_1]] a q *nostatechange* a q)
=λa. λq. (a↓3 *strequals* **F**[[[`query`]]] →
M[[m_1]] a q λ*a*'. λ*s*'. *s*' a q)

Let a_1 be a speech act with a performative which is not *query*:

L[[l_1]] a_1 = λa. λq. (a↓3 *strequals* **F**[[[`query`]]] →
M[[m_1]] a q λ*a*'. λ*s*'. *s*' a q) a_1
= λq. (a_1↓3 *strequals* **F**[[[`query`]]] → **M**[[m_1]]
a_1 q λ*a*'. λ*s*'. *s*' a_1 q)
= λq. (λ*a*'. λ*s*'. *s*' a_1 q)
= λq. (q)

This is a mapping from conversation state to conversation state which leaves the state unchanged. Now consider a speech act a_2=($sender_2$, $receiver_2$, *query*, $content_2$, id_2) :

L[[l_1]]a_2 = λa. λq. (a↓3 *strequals* **F**[[[`query`]]] →
M[[m_1]] a q λ*a*'. λ*s*'. *s*' a q) a_2
= λq. (a_2↓3 *strequals* **F**[[[`query`]]] →
M[[m_1]] a_2 q λ*a*'. λ*s*'. *s*' a_2 q)
= λq. (**M**[[m_1]] a_2 q)
= λq. (λa'. λq'.
(q'↓1, *updatelist* (**R**[[r_1]]a') q'↓2 a_2 q, q↓3, q↓4)
= λq. (q↓1, *updatelist* (**R**[[r_1]] a_2) q↓2 , q↓3, q↓4)

R[[r_1]] a_2= *D cons* (a_2↓1) *cons K cons* (a_2↓1)
cons B cons (a_2↓2) *cons* (a_2↓4)
= *D* $sender_2$ *K* $sender_2$ *B* $receiver_2$ $content_2$

Let us call this string $string_2$

L[[l_1]] a_2 =λq. (q↓1, *updatelist* $string_2$ q↓2, q↓3, q↓4)

This is a function from conversation state to state which adds the proposition contained in $string_2$ to the state (as desired).

4.1.3 The evaluation of two or more speech acts

Note how **S** handles semantics for two speech acts:

L[[**speech-act-semantics** S_1; S_2]]
= **S**[[S_1; S_2]] *nostatechange*
= λc. **S**[[S_2]] (**S**[[S_1]]c) *nostatechange*
= **S**[[S_2]] (**S**[[S_1]] *nostatechange*)
= **S**[[S_2]] (λc. λa. λq. a↓3 *strequals* **F**[[[f_1]]] → **M**[[m_1]] a q
 c a q *nostatechange*)

Where f_1 and m_1 are the performative name and semantics of S_1.

= **S**[[S_2]] (λa. λq. a↓3 *strequals* **F**[[[f_1]]] → **M**[[m_1]] a q
 nostatechange a q)

= λc. λa. λq. a↓3 *strequals* **F**[[[f_2]]] → **M**[[m_2]] a q c a q
(λa'. λq'. a'↓3 *strequals* **F**[[[f_1]]] → **M**[[m_1]] a' q'
 nostatechange a' q')

= λa. λq. a↓3 *strequals* **F**[[[f_2]]] → **M**[[m_2]] a q
(λa'. λq'. a'↓3 *strequals* **F**[[[f_1]]] → **M**[[m_1]] a' q'
 nostatechange a' q') a q

The functionality of this expression is:

Speech-Act → *Conversation-State* → *Conversation-State*

Consider an incoming speech act a_1. If the performative name of this incoming act matches f_2, we get **:**

λa. λq. (a↓3 *strequals* **F**[[[f_2]]] → **M**[[m_2]] a q .....) a q a_1

= λq. **M**[[m_2]] a_1 q

i.e. the state change defined by semantics m_2 when a_1 is the speech act. If the performative name does not match f_2, we get **:**

$\lambda q.\ (a_1\downarrow 3 = \mathbf{F}[[[\ f_2\ ]]] \rightarrow \mathbf{M}[[m_2]]\ a_1\ q \ \square$
$(\lambda a'.\ \lambda q'.\ a'\downarrow 3\ \mathit{strequals}\ \mathbf{F}[[[\ f_1\ ]]] \rightarrow \mathbf{M}[[m_1]]\ a'\ q' \ \square\ \mathit{nostatechange}\ a'\ q')\ a_1\ q)$

$= \lambda q.\ (\ (\lambda a'.\ \lambda q'.\ a'\downarrow 3\ \mathit{strequals}\ \mathbf{F}[[[\ f_1\ ]]] \rightarrow \mathbf{M}[[m_1]]\ a'\ q' \ \square\ \mathit{nostatechange}\ a'\ q')\ a_1\ q)$

$= \lambda q.\ (a_1\downarrow 3\ \mathit{strequals}\ \mathbf{F}[[[\ f_1\ ]]] \rightarrow \mathbf{M}[[m_1]]\ a_1\ q \ \square\ \mathit{nostatechange}\ a_1\ q)$

Which checks whether or not the performative name matches f_1, if it does we get **:**

$\lambda q.\ \mathbf{M}[[m_1]]\ a_1\ q$

i.e. the state change defined by semantics m_1 when a_1 is the speech act. Otherwise we get **:**

$\lambda q.\ (\mathit{nostatechange}\ a_1\ q) = \lambda q.\ (\lambda a.\ \lambda s.\ s\ a_1\ q) = \lambda q.\ (q)$

This function leaves the state unchanged. The denotation of the semantics for three or more acts can proceed in a similar fashion. The semantic function **S** effectively passes on the parameters to the next nesting of the function if the speech act doesn't match.

Persistent Social Relations
  Domain $v \in$ *Persistent-Soc-Rel* = *String-List*
  Operations
    newpersistentsocrel: *Persistent-Soc-Rel*
      newpersistentsocrel = *newlist*

Conversation Array
  Domain $a \in$ *Conv-Array* = *cid*→ *Conversation-State*
  Operations
    newArray: *Conv-Array*
      newArray = $\lambda i.$*newconversation*
    accessArray : *cid* → *Conv-Array* → *Conversation-State*
      accessArray = $\lambda i.\lambda a.a(i)$
    updateArray : *Id* → *Conversation-State* →
        Conv-Array → *Conv-Array*
      updateArray = $\lambda i.\lambda c.\lambda a.[i \mapsto c]a$

Social State
  Domain $s \in$ *Social-State* = *Persistent-Soc-Rel* × *Conv-Array*
  Operations
    newsocial: *Social-State*
      newsocial = (*newpersistentsocrel*, *newArray*)

Current Social State
  Domain $s \in$ *Cur-Social-State* =
    Persistent-Soc-Rel × *Conversation-State*

Cases of Speech Acts
  Domain $c \in$ *Speechactcase* =
    Speech-Act → *Cur-Social-State*→ *Cur-Social-State*
  Operations
    nostatechange : *Speechactcase*
      nostatechange = $\lambda a.\ \lambda s.\ s$
    errorstate : *Speechactcase*
      errorstate = $\lambda a.\ \lambda s.$(*error*, *error*)

Figure 5 Additional semantic algebras.

4.2 Complete Definition

The additional semantic algebras are shown in Figure 5. We now have a new structure: the *Social-State* as described in section 3.1, it contains a dynamic array of conversation states where the speech act's *cid* selects the current conversation. In the evaluation of the language (Figure 6 – only the more important valuation functions are included) we have replaced the *Conversation-State* with the *Social-State*. We have also added the converse function, it is a *Conversation-State* to *Conversation-State* mapping and it looks at the input state to determine the updated contingent social relations for the output state. The converse function is incorporated in the **L** evaluation function so that contingent social relations are updated after the semantic functions are processed. The converse function can also stand alone, and this is used if an agent is initiating a new protocol, in which case a state containing only the protocol variable and the string "initiator" is passed into the function and the appropriate initial speech acts are returned.

L: Language → *Speech-Act* → *Social-State* → *Social-State*
  L[[**converse-function** C **protocol-semantics** P
  speech-act-semantics S]] = $\lambda a.\ \lambda q.$
  let z = (**P**[[P]] proterrstate a ((**S**[[S]] *nostatechange*) a (q↓1, *accessArray* a↓5 q↓2)))
  in let y = **C**[[C]] (z↓1,(z↓2↓1, z↓2↓2, *newlist*, *updateHistory* a a↓6 z↓2↓4))
  in let x = (y↓1, *updateArray* a↓5 y) in
    q↓2↓3 ("obliged " *cons* **F**[[a↓1]] *cons* **F**[[a↓3]] *cons* "(" *cons* **F**[[a↓2]] *cons* ")") → x
    □ (q↓2↓3 ("permitted " *cons* **F**[[a↓1]] *cons* **F**[[a↓3]] *cons* "(" *cons* **F**[[a↓2]] *cons* ")") → x
    □ (q↓1, *updateArray* a↓5 (*update violator* a↓1 q↓2↓1, *updatelist* "violation" q↓2↓2, *newlist*, *updateHistory* a a↓6 z↓2↓4)))

P: Protocol-Semantics → *Speechactcase* → *Speechactcase*
  P[[P_1**;** P_2]] = λ p. **P**[[P_2]] (**P**[[P_1]]p)
  P[[**[**T**]** S]] = λp. λa. λq. cases (*access* "protocol" q↓2↓1) of
    (is*String*(x)→x □ is*Nat*(y)→ *error*) end
    strequals **T**[[**[** T **]**]] → **S**[[S]] *errorstate* a q □ p a q

C: Converse-Function → *Cur-Social-State*$_\perp$ → *Cur-Social-State*$_\perp$
  C[[C_1**;** C_2]] = λ q. **C**[[C_2]] (**C**[[C_1]]q)
  C[[**[** T **]** O]] = λq. cases (*access* "protocol" q↓2↓1) of
    (is*String*(x)→x □ is*Nat*(y)→ *error*) end
    strequals **T**[[**[** T **]**]] → **O**[[O]]q □ q

Figure 6 Updated and additional evaluation functions.

4.3 Specifying an ACL with auction protocol

This is an English auction protocol, the price is increased at each iteration until no more bidders are prepared to bid, the last successful bidder being the winner. We use only three speech acts. The main iteration consists of an $^{(1)}$*announce* of the new price from auctioneer to bidders, and an $^{(2)}$*accept* of a price from bidder to auctioneer. The auction terminates with a $^{(3)}$*declare* from auctioneer to all participants. There are three roles, a member of the role $^{(1)}$*Bidder* becomes $^{(2)}$*Buyer* when its accept causes the $^{(3)}$*Auctioneer*'s next announce. We assume there is a commonly agreed global timer variable that can be referred to in any conversation.

```
converse-function
[auction]
Initiator:
announce(#Bidder,{protocol=auction,
 item=#item,price=#price,timeout=#timeout});
if #time>#timer then obliged #Auctioneer,
 declare (#Bidder U #Buyer,null);
if intend Auctioneer Auctioneer,sell
 (#item,member of #Bidder,>=#startprice)
 then permitted #Bidder,
 accept(#Auctioneer U #Bidder U #Buyer);
if intend #Buyer
 #Buyer,Buy(#item,#Auctioneer,#price) then
 obliged #Auctioneer, announce(#Bidder U
 #Buyer, price=#price+#increment));
protocol-semantics
[auction]
[announce]
timer:=#time+#timeout;
if #Auctioneer=null then
 (Auctioneer:=S;Bidder:=R);
intend #Auctioneer #Auctioneer,sell
 (#item,member of #Bidder,>=#price);
intend #Auctioneer #Auctioneer,
 (wait(#timeout) then declare (#Bidder U
 #Buyer,{protocol=auction,item=#item,
 price=#price,timeout=#timeout}))
proposal=Buy(#item,#Auctioneer,#price);
desire #Auctioneer know #Auctioneer
 intend #Bidder #proposal
[accept]
if #Buyer!=null then not intend #Buyer
 #Buyer,Buy(#item,#Auctioneer,#price);
Bidder:= #Bidder U #Buyer;Buyer:= S;
Bidder:= #Bidder \ S;oldprice:=#price;
[declare]
p.obliged #Buyer,buy(#item,S,#oldprice);
p.obliged S,sell(#item,#Buyer,#oldprice);
terminate;
speech-act-semantics
[announce] C
[declare] C
[accept]
if #proposal!=null then intend S,#proposal;
```

Figure 7 ACL specification containing an auction protocol

Figure 7 shows the ACL specification for an ACL containing this protocol. When rules are applied, variables prefixed with a # are replaced by their actual value. Note in particular the protocol specific semantics specified for the opening announce: the auctioneer expresses the intention to sell the item *item* to a member of the group of *bidders* at a price not less than *price* and to wait until the time *timeout* has elapsed for a response. These italicised variables do not yet exist in the conversation state, the auctioneer must choose values for them. These rich semantics are vital to enable the auctioneer to reason about how the values will be chosen and to enable the receiver to reason about how to update its internal state. For example when choosing a value for *startprice*, the auctioneer must know that *startprice* means the minimum selling price for the item. Note how the context independent semantics of announce and declare are identical, it is only in the context of the protocol that they are given different meanings. This is similar to the way in which many words in natural language are synonymous in some contexts and not in others. The additional protocol specific meaning of declare is the creation of two persistent social relations: (1)the bidder is committed to buying the item and (2)the auctioneer is committed to selling it to him (at the previously announced price). A new accept in the next iteration of the auction revokes the commitments of the previous iteration. The *accept* has a general context independent meaning which is the expression of an intention to perform an action that the intended recipient of the *accept* has previously proposed. This is included to show the kinds of complex intuitive meanings which can be specified in this framework, and which can be given an exact semantics by means of a function which is easily implemented in any procedural language. An animation of the protocol is shown in Table 1. Only the more interesting propositions and variables are shown (to save space).

5. EVALUATION AND CONCLUSIONS

The practical engineering aspects of developing and implementing agent communication languages were considered in the design of this framework. Emphasis was therefore placed on minimising the effort of the developer by separating concerns and facilitating a modular development. In particular we cater separately for the following four aspects of context, i.e. the meaning of an act can depend on:

1. The protocol being followed in the conversation.
2. The relationship between this speech act and the remainder of the discourse, for example, in section 4.3 the semantics of *accept* and the previous *announce* are related.
3. The domain in which the conversation takes place.
4. The status or authority of the participants (via roles).

Separate specifications constitute flexible building blocks which can be re-used for specific solutions. Designers can extend the set of primitive building blocks, adding speech acts and protocols which can be published and used by others for different protocols. Although these elements of meaning have been separated, they are brought together in the semantic evaluation functions and are thereby interlinked. In particular, there is a well defined relationship between the semantics of protocols and the acts they are composed of since protocols are defined using the state created by the individual acts.

The formal semantics and public perspective mean that it is possible for an outside observer to determine when agents are not complying with the specifications. Enforcing compliance can be achieved by: (1)Sentinel agents may be placed in the society to observe interactions and punish offending agents (by evicting or terminating them for example). (2)We may introduce notions like politeness and trustworthiness. Agents that consistently violate contingent commitments and speak out of turn may be branded as impolite and possibly ostracised by the society. Similarly, agents that are known to violate persistent commitments may be deemed untrustworthy and may not be offered contracts.

6. FURTHER RESEARCH

With a formal semantics based on externals, verification is possible, but not trivial. This is the subject of ongoing work. When both the ACL specification and the internals are well defined mathematical functions, further functions can be derived which can prove a certain outcome will result, given a certain initial state. We are currently working on developing a design tool and a compiler for ACLs which uses a diagrammatical representation of protocols based on UML diagrams.

Table 1 Animation of a sample Auction

Speech act (*sender, receiver, performative, content, conversation-identifier, sequence*)	Social State			
	Conversation State (*history omitted*)			Persistent social relations
	Variables	Propositions	Contingent social relations	
Joe,{Bob,Pat},announce, {protocol=auction, item=book, price=40,timeout=3},6,1	Auctioneer=Joe Bidder={Bob,Pat} protocol=auction item=book; price=40 timeout=3; timer=37 time=34; proposal= Buy(book,Joe,40)	*I* Joe Joe,sell(book,member of {Bob,Pat},>=40) *I* Joe Joe,(wait(3) then declare({Bob,Pat})) *D* Joe *K* Joe *I* {Bob,Pat} {Bob,Pat},Buy(book,Joe,40)	permitted Bob,accept({Joe,Pat}) permitted Pat,accept({Joe,Bob})	
Bob,{Joe,Pat}, accept,null,6,2	Bidder=Pat Buyer=Bob	*I* Bob Bob, Buy(book,Joe,40)	obliged Joe,announce({Bob,Pat}, price=40+null)	
Joe, {Bob,Pat},announce, price=43,6,3	time=36; price=43 timer=39	*I* Joe Joe,sell(book,member of {Pat},>=43)	permitted Pat,accept(Joe,Bob)	
Bob,{Joe,Pat},accept, null,6,4	Bidder=Bob Buyer=Pat	*I* Pat Pat,Buy(book,Joe,40)	obliged Joe,announce({Bob,Pat}, price=43+null)	
Joe, {Bob,Pat},announce, price=46,6,5	time=37; price=46 timer=40	*I* Joe Joe,sell(book, member of {Bob},>=46)	permitted Bob,accept(Joe,Pat)	
	time=41; timer=40		obliged Joe,declare {Bob,Pat},null)	
Joe,{Bob,Pat},declare, null,6,6		terminate		obliged Pat,buy(book,Joe,43) obliged Joe,sell(book,Bob,43)

It is apparent that some of the variables that we have placed in the conversation state are really external to the conversation, for example the timer variable. Also there are exogenous inputs that are not speech acts which do have an effect on the social state. Communication is but one component of social interaction and it is dependent on other components. We have already made this explicit by extending the conversation state to a social state which subsumes it. The next step is to do the same for the inputs that effect changes in this state, (i.e. speech acts are only one type of input) and also for the specification which defines the effect of actions on the state of the society (i.e. the ACL specification is but one component of this).

In human societies characteristics such as being helpful are typically determined uniformly by different observers (because there are certain conventions of society by which behaviour is judged) i.e. it is not a subjective opinion, it becomes public knowledge. We would like the social state to include such public information. For this purpose we intend to make a specification of the rules governing the conventions of social behaviour, and to extend the language evaluation function (maps a speech act onto a change in the social state) so that it can infer the changes in the states of these high level parameters. An explicit representation of these high level concepts would allow us to specify a domain constraint which requires that agents are helpful, and to enforce sanctions on those who violate the constraint.

We have proposed a development method where the rules agents use to update their internal states are specified at design time by a human. In practice this may be difficult or impossible. Agents may move to new domains and learn new protocols via published specifications. Such specifications will not include *add* and *select* functions, nor should they, since different agents should have the freedom to choose their own strategies. This is analogous to a human who has not been to an auction before, after reading the rules, the individual can decide upon a strategy for participating.

If we consider that an agent has some rationality, and can access formal specifications of protocols, there should be a way for the agent to automatically produce a strategy of its own. For this reason we are working on specifications of meta-rules that an agent can use to decide how to update its internal state in a new protocol. This research has once again highlighted the importance of rich semantics capturing the intuitive meaning of acts so that an agent can reason intelligently to interpret and respond to them.

7. REFERENCES

[1] Artikis, A., Guerin, F., Pitt, J. Integrating Interaction Protocols and Internet Protocols for Agent-Mediated E-Commerce, In Dignum, F., Cortes, U., *Agent-Mediated Electronic Commerce III*. Springer-Verlag. (2001).

[2] FIPA, [OC00003] FIPA 97 Part 2 Version 2.0: Agent Communication Language Specification. Foundation for Intelligent Physical Agents. (1997).

[3] Gazdar, G. Speech Act Assignment. *Elements of Discourse Understanding* . Cambridge University Press. (1981).

[4] Genesereth, M., Ketchpel, S., Software Agents. *Communications of the ACM*, (July 1994).

[5] Cohen, P. and Levesque, H. Communicative Actions for Artificial Agents. ICMAS'95. MIT Press, Cambridge, Massachusetts, USA; 1995; p. 65-72. (1995).

[6] Schmidt, D. A. Denotational Semantics: A Methodology for Language development. Allyn and Bacon Inc. (1986)

[7] Singh, M. A Social Semantics for Agent Communication Languages. *Proceedings of the IJCAI Workshop on Agent Communication Languages*, Springer-Verlag. (2000).

[8] Singh, M. Agent Communication Languages: Rethinking the Principles. *IEEE Computer.* vol.31, no.12; p.40-7. (1998).