Agent Communication Frameworks and Verification

Frank Guerin and Jeremy Pitt
Intelligent and Interactive Systems,Department of Electrical & Electronic Engineering,
Imperial College of Science, Technology & Medicine,Exhibition Road, London, SW7 2BT.
Phone: +44 20 7594 6318 / Fax: 6274
{f.guerin,j.pitt}@ic.ac.uk

ABSTRACT

This paper develops a general agent communication framework which allows us to define several different notions of verification and to investigate if an agent communication language is verifiable. The framework is sufficiently general to accommodate communication languages based on agents' mental states as well as those based on social states of the multi-agent system. For this purpose an existing computational model is employed to represent agent states and extended to represent social states. We use this framework to identify the types of languages that are verifiable in open systems where agents' internals are kept private.

1. INTRODUCTION

This paper develops a general agent communication framework which allows us to define several different notions of verification and to investigate if an agent communication language (ACL) is verifiable. An ACL is one component of an agent communication framework. A framework may additionally include agents' names, programs, states and whatever languages are needed to provide a well defined relationship between these components. An ACL typically defines a specification (for each communicative act) which must be satisfied by the system of agents using that language, we say that it defines a semantics for each communicative act. It is important to note that there is a distinction between *program semantics* for communication statements in an agent's program and *ACL semantics* (which constitute specifications) for communicative acts. If an agent is ACL-compliant then its program semantics will satisfy the semantics defined by the ACL specification. ACL semantics may be defined in different ways and each way implies different notions of verification. Our framework builds on the framework presented by Wooldridge [16]; we attempt to make the framework sufficiently general to allow ACLs with various semantic definitions to be accommodated.

Communicating agents operate in a certain context, the entire context includes the private states and programs of agents as well as the publicly observable state of the society. ACL semantics for communicative acts must specify something about the state of this context. ACLs based on mental states typically specify semantics by means of preconditions and/or postconditions [8] which must be true before or after the communicative act is performed. ACLs based on social states typically specify social facts that are created or modified by the performance of a communicative act [13]. Thus an agent communication framework will need to include a representation of the multi-agent system which captures information about the internal states of agents in the system as well as observable (social) states. In addition to an agent's current state, we also need a representation of the agent's program because we might need to know what the agent is going to do. The ACL specification for the semantics of communicative acts may refer to future actions and we may need to verify that an agent will do them. For example, in order to specify that an agent holds a certain intention as a precondition to sending a promise act, the specification for the act may require that the agent's program eventually executes the intention. A computational model can represent the programs and states of the agents in a multi-agent system; the model we choose is a fair transition system [9], with some extensions to represent social states of the system.

We begin by describing the agent programs which a multi-agent system in composed of (§ 2); followed by a computational model for multi-agent systems (§ 3). We then make some extensions to capture observable states of an open system through a representation of the agents' social context (§ 4) and the states of this context (§ 5). After defining the ACL component of the framework (§ 6) we present the general framework (§ 7) and define several notions of verification (§ 8). We use the framework to analyse existing ACLs to determine if they are verifiable (§ 8.5) and we identify the type of ACL which would be appropriate in an open system. Finally we compare and contrast our framework with two other frameworks for agent communication (§ 9) and conclude with future work (§ 10).

2. AGENT PROGRAMS

We use the Simple Programming Language (SPL) of [9] to describe the agent programs in our multi-agent systems. The entire multi-agent system can be described by a single program P.

$$P :: \Big\| _{i \in Ag} M_i$$

That is, a cooperation statement (for parallel execution)

AAMAS 2002 Bologna, Italy

where each top level process M_i is viewed as a module which represents the program of agent i. Where i ranges across the set of agent names Ag which uniquely identify agents. Each module M_i (representing the program of an agent i of the system) has a set of asynchronous buffered input channels on which it can receive messages from other agents and a set of asynchronous buffered output channels on which it sends messages to other agents. A channel is a variable whose value is a list of messages. We identify channel variables using an α with a subscript. Agent i receives messages from agent j on the channel $\alpha_{i,j}$ and sends them on channel $\alpha_{j,i}$. We need separate channel identifiers for each communication link so that we can distinguish between communicative events from or to different agents. In practical systems agents may not need to use more than two channels (one input, one output) since many agent systems have a platform which deals with the distribution of messages. In such cases we can model the functions of the platform as a special *facilitator* module which can accept messages from all agents and distribute them to the intended recipients. This would also enable agents to be added or removed in a dynamic system without having to add new channels to all the existing agents, the existing agents would just need to be informed of the identity of the new agent and then they could send messages to it on their regular output channel.

3. COMPUTATIONAL MODEL

A computational model is a well defined mathematical structure which can represent the behaviour of a program. The behaviour of a program can be described as a sequence of states that the program could produce, where a state gives a value to all the variables in the program including the control variable π which describes the location of the next statement. In addition to the internal variables of agents we will model the social states of the system.

3.1 Variables and States

The computational model we choose for multi-agent systems is a fair transition system [9]. This is a system which contains variables and transitions. Variables represent the states of the agents and transitions represent the state changes caused by statements in the agents' programs. The variables come from a universal set of typed variables $\mathcal{V}$, called the vocabulary; from this we can construct assertions (for example $\forall x : \exists y : \ y > x$).

A *state* s is an interpretation of $\mathcal{V}$, assigning each variable $u \in \mathcal{V}$ a value $s[u]$ over its domain. As a program executes it passes through a sequence of states in which variables may take on different values. We sometimes find it convenient to refer to the value of a variable u and to its previous value u^-. The asynchronous communication predicate $[\alpha{\leqslant} \, m]$ denotes a sending event i.e. when a message is sent to a channel α; it is defined as $\neg \mathit{first} \ \wedge \ \alpha = \alpha^- \bullet m$. That is, a sending event has occurred on channel α if this is not the first state and if variable α is equal to what it was before (α^-) with message m appended to the end of the list. To specify that an input or output has occurred on a channel without specifying the value communicated we use $[\alpha{\leqslant}]$ and $[\alpha{\geqslant}]$. Often when an agent A sends a message m, we write $[A{\leqslant} \, m]$ instead of specifying the channel; since our message is a tuple of which the first part is always the sender and the second the receiver, $[A{\leqslant} \, m]$ is an abbreviation for $\alpha_{m\downarrow 2, m\downarrow 1}{\leqslant} \, m$. Where $\downarrow i$ denotes the operation such that $(a_1, a_2, \ldots, a_n)\downarrow i = a_i$.

3.2 Fair Transition Systems

Our multi-agent system is represented by the fair transition system $\langle V, \Theta, \mathcal{T}, \mathcal{J}, \mathcal{C} \rangle$ which represents the states and programs of the agents in the system.

- $V \subseteq \mathcal{V}$ is a set of system variables, these represent data variables (i.e. data within an agent's internal state) and the location of control within an agent's program.
- Θ is an assertion characterising initial states, i.e. if a state s of the system satisfies the assertion Θ, then it is a state from which the system can start running.
- $\mathcal{T}$ is a set of transitions. Each transition maps each state onto a set of possible successor states. So $\mathcal{T}$ represents the agents' programs, each statement in an agent's program is associated with a transition in $\mathcal{T}$.
- $\mathcal{J}, \mathcal{C} \subseteq \mathcal{T}$ are the sets of just and compassionate transitions, they ensure that each parallel process is executed fairly and that one process is not waiting forever while another continues to execute (more detail in [9]).

The agent programs (written in SPL) which constitute our multi-agent system have a semantics [9] which identifies each of the components of the fair transition system we are using to represent the multi-agent system.

4. REPRESENTING SOCIAL CONTEXT

The model thus far presented models the internal programs and states of agents. This can completely describe the system, but we would additionally like to have a convenient representation of the externally observable phenomena including social facts. These facts include the social facts created by interactions and the rules governing their creation. The governing rules are in fact the social meanings of the events in the interaction (these are what Searle calls constitutive rules, see § 6); for simplicity we will assume that they are fixed by the ACL specification and do not change during an execution. The only events we consider here are communicative events. As stated in § 1, the semantics of communicative acts may create or modify social facts. Additionally, certain aspects of the context may affect the meaning of a communicative act. Aspects of context which affect meaning include

- The history of communicative acts in the interaction.
- The domain in which the interaction takes place.
- The social facts holding between agents in the system, for example, the status and authority of participants or social commitments.

We treat all these aspects of the context as public knowledge and include them in the *social state*.

The observable social states can be determined by observing communications: we assume that a message being added to an agent's channel can be observed. The social state is described by a set of variables $\mathcal{F} \subset \mathcal{V}$ that is disjoint from the fair transition system variables V. Each agent i may have a differing view of the social context, since it may not have received all events (communications) occurring in the system. We use the variable $f_i \in \mathcal{F}$ to denote the social state observable to agent i. Corresponding to Θ for the fair

transition system, we define an initial condition Φ for the social state variables $\mathcal{F}$. Thus the initial[1] observable social state f_i^0 of an agent i must satisfy Φ. The type of each social facts variable is a mapping from well formed formulae of the social facts language $\mathcal{L}_f$ to true or false values.

$$f_i : \quad wff(\mathcal{L}_f) \rightarrow \{true, false\}$$

Social facts describe role relationships, commitments to perform actions and publicly expressed attitudes.

4.1 Governing Rules

These rules define how communications create and modify social facts (according to social conventions). The ACL specification (described in § 6) defines the governing rules with the second part of the semantic function $[\![-]\!]_c$, where c is the communication language. If a message m is communicated then $[\![m]\!]_c{\downarrow}2$ gives the resultant social state change; this takes as input an observable social state f_i and returns the new social state. The following formula characterises the observable social states f_i :

$$O_i : \Phi \wedge \widehat{\Box} \left[\begin{array}{c} \forall j : \left[\begin{array}{c} \neg[\alpha_{i,j} \lessdot\ ] \wedge \neg[\alpha_{j,i} \lessdot\ ] \\ \wedge\ f_i = f_i^- \end{array} \right] \\ \vee \\ \exists\, m, j : \left[\begin{array}{c} ([\alpha_{i,j} \lessdot m] \vee [\alpha_{j,i} \lessdot m]) \\ \wedge\ f_i = [\![m]\!]_c{\downarrow}2\, f_i^- \end{array} \right] \end{array} \right] \quad (1)$$

That is, Φ is initially satisfied and in all subsequent states (represented by $\widehat{\Box}$) either

- there is no communication on $\alpha_{i,j}$ (i.e. $\neg[\alpha_{i,j} \lessdot\ ]$) or $\alpha_{j,i}$ and f_i is equal to what it was in the previous state (f_i^-) **or**
- the message m is sent to $\alpha_{i,j}$ (i.e. $[\alpha_{i,j} \lessdot m]$) or $\alpha_{j,i}$ and f_i is determined by applying the state change function $[\![m]\!]_c{\downarrow}2$ to the old state f_i^-.

Essentially O_i says that the social state is unchanged if no communication occurs; or if a communication does occur the social state is modified according to the state change function described by the ACL. We show how such a function can be formally specified in [5]. For example, if an agent i promises something to another agent, the ACL semantics of the communicative act for *promise* may require that a new social commitment proposition becomes true in f_i.

Note that given an initial state Φ and a certain sequence of messages, we can work out the values of the social facts; i.e. apart from their value in the initial state, they do not capture any additional information; they are merely a convenience which we will use when proving that an agent satisfies the social facts observable by it. The social state is not a part of our fair transition system since it need not be explicitly represented anywhere in a real multi-agent system, parts of it may or may not be represented by the the local variables of agents in the system. Each agent should store a copy of all the information in the social state that might be relevant to its interactions so that it may correctly interpret context dependent communicative acts and keep track of its social commitments. The complete social state is then implicit. As an external observer we need to know the social state if we wish to understand the interactions taking place in the system. We need to know the social knowledge observable to each agent in a system in order to determine if it is complying with the social conventions to the best of its knowledge.

[1]We use subscripts to identify which variables of $\mathcal{F}$ correspond to which agents and superscripts to enumerate the states in a sequence where 0 enumerates the initial state.

5. STATES AND COMPUTATIONS

Recall that a state s is an interpretation of $\mathcal{V}$, assigning each variable $u \in \mathcal{V}$ a value $s[u]$ over its domain. Σ is the set of all states s. An infinite sequence of states

$$\sigma : s^0, s^1, s^2, s^3, \ldots$$

is called a *system model*. A system model is a *computation of the program* P (which identifies our fair transition system) if s^0 satisfies the initial condition Θ and if each state s^{j+1} is accessible from the previous state s^j via one of the transitions $\mathcal{T}$ in the system and the requirements of justice and compassion are respected. A computation is a sequence of states that could be produced by an execution of the program. All computations are system models but not vice versa. Thus the agent programs identify the components of a fair transition system and the fair transition system describes all the possible computations that a program could produce. This is how we say what a program means, mathematically: it is described as the set of all the sequences it could produce.

This constrains only the interpretations of variables in V. We specify additional constraints on the interpretations of the variables in $\mathcal{F}$ so that the social state changes in response to communicative acts between agents. We define a *computation of the multi-agent system* S with initial social fact Φ to be a system model σ which is a computation of the program P and which also satisfies O_i for all agents i.

5.1 System for a Single Agent

If we are the designers of a single agent and only have access to that agent's internals we can construct a new fair transition system S_i where the variables, initial condition and transition sets of the system are just the same as if i was the only agent in the system [9]. The initial condition for social facts Φ will include social facts that will be true for the system we intend to allow our agent to run in. We add one extra transition τ_E, the environmental transition which represents all the things other agents could do and is included in the justice set; τ_E cannot modify any variables in agent i's program apart from the communication channels; the outbound communication channels can be modified by the removal of a message and the inbound ones can be modified by addition of a message. Other variables in $\mathcal{V}$ may be modified arbitrarily. S_i represents all the possible behaviours of agent i in any multi-agent system.

5.2 External System

If we do not have access to the internals of any agent, but can detect each message being sent, we can construct a fair transition system S_E which represents all possible observable sequences. The variables of S_E are simply the communication channels we can observe and there are only two transitions, the idling transition τ_I (preserves all variables and does nothing) and the environmental transition τ_E. The environmental transition allows arbitrary modification of any variable outside of the observable channels and

allows a channel to be modified by adding a message to the end or removing one from the front. The initial condition Θ requires that all channels are empty. In order to complete the social states in computations of the multi-agent system S_E we must also know the initial social facts for the initial condition Φ. A computation of the multi-agent system S_E does not care about how its states interpret variables which are not observable, it only cares about channels and social facts variables.

6. AGENT COMMUNICATION LANGUAGE

When agents communicate they exchange messages which are well-formed formulae of a language $\mathcal{L}_c$. Agents pass messages in order to perform communicative acts and these acts must have a well defined semantics which is a part of the ACL specification. Semantics can be based on constitutive or regulative rules.

6.1 Constitutive and Regulative Rules

There are two broad classes of ACL which differ by specifying either regulative or constitutive rules. Regulative rules define certain conditions that should hold in the system if a certain communication takes place. Constitutive rules define that a certain communication constitutes the creation of certain conditions in the system. Languages using constitutive rules are *social* languages; languages using regulative rules are *mental* languages (because they refer to agent's mental states). This regulative/constitutive classification is due to Searle, who states:

> "Regulative rules regulate a pre-existing activity, an activity whose existence is logically independent of the existence of the rules. Constitutive rules constitute (and also regulate) an activity the existence of which is logically dependent on the rules." [12]

Searle cites an example regulative rule: "when cutting food hold the knife in the right hand"; and an example constitutive rule: "a checkmate is made if the king is attacked in such a way that no move will leave it unattacked". Thus constitutive rules create new forms of behaviour by stating what activity constitutes the new behaviour. It is Searle's contention that "...the semantics of a language can be regarded as a series of systems of constitutive rules...". However, many existing ACLs are based on regulative rules and we wish to make our framework sufficiently general to accommodate both types of rule.

In our framework, ACLs which deal with observable social states will be considered to define constitutive rules and will define (for each message) a social state change function. ACL's which refer to agents' internal states will be considered to define regulative rules and will define an assertion (for each message) which should be true of the system if agents are compliant. The specification for the semantics of communicative acts is a function $[\![-]\!]_c$ from a message to a tuple $\langle mental, social \rangle$. For languages based on mental states the second part of the tuple will be ignored; for languages based on social states the first part of the tuple will be ignored.

$$[\![-]\!]_c \ : \quad \mathit{wff}(\mathcal{L}_c) \rightarrow \mathit{wff}(\mathcal{L}_s) \times (\Omega \rightarrow \Omega)$$

Where Ω is the set of all possible observable states.

6.2 Mental Part

The first part of the tuple returned by $[\![-]\!]_c$ is a formula in the semantic language $\mathcal{L}_s$. The formula specifies properties of the system, for example, it may describe pre and/or postconditions which must be true of sender or receiver or some other element of the fair transition system. Preconditions should be true when the message is sent, postconditions should be applied after i.e. they define things that should become true after the message is passed. For example, a precondition might require that a certain mental state exist in the sender or that the receiver has performed some action before a message can be sent. A postcondition might assert that the receiver is obliged to adopt a certain mental state upon receiving the message. Whether it is required that postconditions become true immediately after a message is passed or eventually depends on how the semantics are specified. In KQML semantics, postconditions "...describe the states of agents after the utterance of a performative (for the sender) and after the receipt (but before a counter utterance) of a message (by the receiver)" [8].

The formula is given a semantics in terms of the set of models where the formula is satisfied.

$$[\![-]\!]_s : \mathit{wff}(\mathcal{L}_s) \rightarrow \wp(\mathit{mod}(\mathcal{L}_s))$$

If $\mathcal{L}_s$ is a temporal logic, this semantics can be given for the models of the system [9]. For many existing ACLs this is the missing part, i.e. they do not define how their semantic language relates to a computational model.

6.3 Social Part

The second part of the tuple returned by $[\![-]\!]_c$ is a function from social state to social state. The function describes the change to the social state caused by the message transmitted and this constitutes the governing rules described in § 4.1. Since the ACL is responsible for defining the changes that messages cause in the social state, the ACL should also define the language $\mathcal{L}_f$ for social facts whose semantics $[\![-]\!]_f$ is interpreted over an observable model (i.e. a model where we are concerned only with how states interpret observable variables f_i). Thus $[\![a, i]\!]_f$, the semantics of a social facts assertion a for agent i describes the set of models where the formula a is satisfied by agent i.

$$[\![-]\!]_f \ : \ (\mathit{wff}(\mathcal{L}_f) \times Ag) \rightarrow \wp(\mathit{mod}(\mathcal{L}_f))$$

Many social facts may be simply satisfied in all situations, but some such as a commitment to do an action may be satisfied only in those models where the action is eventually done by the agent. The social facts semantics function $[\![-]\!]_f$ is allowed to make use of channel variables and any of the observable state variables f_i but it cannot place constraints on agent internals.

6.4 Complete ACL

Therefore a complete ACL for our general framework is a 3-tuple:

$$\mathcal{ACL} = \langle \mathcal{L}_c, \mathcal{L}_s, \mathcal{L}_f \rangle$$

Where $\mathcal{L}_c$ is the communication language itself, $\mathcal{L}_s$ is the semantic language and $\mathcal{L}_f$ is the language for social facts which can define permissions and obligations to perform actions for example. Each of these languages can by specified by a tuple (for example $\mathcal{L}_c = \langle \mathit{wff}(\mathcal{L}_c), [\![-]\!]_c \rangle$) where the first part of the tuple gives the set of well formed formulae

of the languages and the second part gives the semantics. The semantic function $[\![-]\!]_f$ maps $\mathcal{L}_f$ expressions to formulae in some language (for example temporal logic) which only make use of observable variables (variables in $\mathcal{F}$, not V). For simplicity we use the same language $\mathcal{L}_s$ to give semantics to both $\mathcal{L}_c$ and $\mathcal{L}_f$ and our model σ is a sequence of states which interpret both internal variables of agents and observable social variables. We could make our framework more general by allowing a different model for observable social phenomena and a different semantic language, but this is unnecessary for our purpose. A language which describes states must make some assumptions about those states, so by defining the semantics of $\mathcal{L}_s$ the ACL is implicitly defining the computational model to be used for verification. We note that many existing ACLs are not complete because they have not formally specified all of the elements above.

7. GENERAL FRAMEWORK

An agent communication framework is a 4-tuple:

$$\langle Ag, \langle V, \Theta, \mathcal{T}, \mathcal{J}, \mathcal{C}\rangle, \mathcal{ACL}, \Phi\rangle$$

- Ag is a set of agent names, $Ag = \{1, \ldots, n\}$;
- $\langle V, \Theta, \mathcal{T}, \mathcal{J}, \mathcal{C}\rangle$ is the fair transition system representing all the programs of all the agents in the multi-agent system (described in § 3);
- $\mathcal{ACL} = \langle \mathcal{L}_c, \mathcal{L}_s, \mathcal{L}_f\rangle$ is an ACL including mental and social components (described in § 6);
- Φ is the initial assertion for social states.

Using this framework we can define what verification means.

		mental language			social language		
		access to internals	external observation	specification only	access to internals	external observation	specification only
design time verification	Prove a property for agent programs	•	-	-	•	-	-
	Mental semantics are always respected	•	-	-	-	-	-
	Social facts are always respected	-	-	-	•	-	-
	Verify the outcome of a system	•	-	-	•	-	-
	Assume unknown agents are compliant	-	-	•	-	-	•
	Prove a protocol property	-	-	•	-	-	•
run time verification	Verify semantic formula holds	•	-	-	-	-	-
	Verify via history						
	Verify social commitments by history	-	-	-	-	•	-
	Verify protocols by history	-	-	-	-	•	-

• verification is possible and appropriate

Table 1: Types of Verification

8. VERIFICATION

Several different types of verification are possible depending on the *type of ACL* used, the *information available* and whether we wish to verify at *design time* or at *run time*. Design time verification is important when we want to prove some properties (of an agent or the entire system) to guarantee certain behaviours or outcomes in a system. Run time verification is used to determine if agents are misbehaving in a certain run of the system. Run time verification is important in an open system because it may be the only way to identify rogue agents. We must be able to identify misbehaving agents if we are to take action against them and hence guarantee that they will not prevent the society from functioning in the desired way.

Type of ACL: Our general framework allows the semantics of an act to include both a formula which must be satisfied in the system and a social state change function (see § 6), in which case both would need to be verified. In practise there exist no ACLs which include both parts and ACLs can partitioned into *mental languages* and *social languages*.

Information available: There are three relevant types of information that might be available for verification.

1. *Internal States*: The agent designer or system designer will typically have access to internal states (the agents' program code and state during execution). This is usable for both design time and run time verification.
2. *External States*: In an e-commerce scenario different vendors may contribute their own agents to the system, in such cases the system administrator has to perform some type of verification which works with observable social states. We assume that the communications occurring in a run of the system can be observed and so verification is only performed at run time.
3. *Language specification*: With only the language specification available we can still prove certain properties. For example by assuming that all agents respect the language's semantics during the execution of a protocol we can verify that certain outcomes will result. In this case we have no information about runs of the system so only design time verification is possible.

Table 1 shows the types of verification that are possible and appropriate based the information available. Note how the only hope for run-time verification in an open system (where agent internals are inaccessible) is with a social language. We now give a more detailed explanation of each kind of verification in terms of our general framework. We will assume that $\mathcal{L}_s$ is linear temporal logic [9] whenever we specify a formula to describe a certain type of verification.

8.1 Prove a Property for Agent Programs

This entails ensuring that some property holds for the system at design time. In relation to communicating agents this verification has been used by van Eijk [2] where a certain property is specified and proven to hold for a certain system of communicating agents, no ACL is used. The system of agents in van Eijk's example consists of agent S_1 and agent S_2 whose programs are as follows:

$$\begin{aligned} S_1 &\equiv \texttt{query}(\varphi) \cdot \texttt{tell}(c, \varphi) \\ S_2 &\equiv (\texttt{ask}(c, \psi_1) \cdot \texttt{update}(\psi_1))+ \\ &\quad\ (\texttt{ask}(c, \psi_2) \cdot \texttt{update}(\psi_2)) \end{aligned}$$

Where c is a common synchronous communication channel and φ, ψ_1 and ψ_2 are constraints. The program of agent S_1 will only execute the `tell` command if the `query`(φ) is successful i.e. the information store of agent S_1 contains φ. Agent S_2 executes a non-deterministic choice between asking for ψ_1 or ψ_2 along the channel. If the information (φ) being told by agent S_1 entails ψ_1 then the first `ask` is successful and the subsequent `update` executes. If φ entails ψ_2 then the second `ask` (and subsequent `update`) is successful. Van Eijk proves that if φ is a constraint that entails ψ_1 but not ψ_2 then this system of agents satisfies the specification $\mathbf{B}_1\varphi \wedge \mathbf{B}_2\psi_1$. Meaning that agent S_1 believes φ and agent S_2 believes ψ_1 for all terminating computations.

The above verification does not necessarily imply the use of any communication language. In our framework we have a communication language which may be social or mental, corresponding to these possibilities there are two special cases of proving properties that are of particular interest to us.

8.1.1 Verify Mental Semantics are Always Respected

Given a mental language we can verify at design time that the semantics of the communication language are always respected by the agents in all possible computations of the system. We can specify this property with the formula

$$\forall \alpha : \forall m : \Box ([\alpha \lessdot m] \rightarrow [\![m]\!]_c \downarrow 1)$$

which must be P-valid over the program P which is a program composed of the programs of all the agents in the system (see § 3). This means that for all communication channels α and for all messages m, whenever a message m is sent on an asynchronous channel then the property specified by the first part of the message semantics for m (i.e. $[\![m]\!]_c \downarrow 1$) must hold. The property may be a precondition in the case of the FIPA ACL or a conjunction of a precondition and postcondition in the case of KQML. Let us take a KQML example. Following the KQML semantics defined by [8] there exist preconditions **Pre**(**A**) for the sender and **Pre**(**B**) for the receiver and postconditions **Post**(**A**) for sender and **Post**(**B**) for receiver. We are given a precise definition of when **Post**(**B**) should be true i.e. "after the receipt (but before a counter utterance) of a message (by the receiver)"[8]. We assume that a counter utterance is the next message sent from the original receiver to the original sender. For **Post**(**A**) we are only told that it should be true of the sender after the utterance of the performative, we are not told how much time the sending agent has after the utterance to change its internal state to agree with the postcondition. Let us assume it is before it (sender) sends another message on this channel. Let us attempt to formalise the KQML semantics temporal logic. Thus when a message m is sent to $\alpha_{i,j}$, the channel on which agent i receives messages from agent j, the KQML semantics would give us a property of the form

$$\begin{aligned} p: \quad & \mathbf{Pre}(\mathbf{A}) \wedge \Diamond \left(\mathbf{Post}(\mathbf{A}) \wedge (\neg[\alpha_{i,j} \lessdot]) \; \mathcal{S} \; [\alpha_{i,j} \lessdot m]\right) \wedge \\ & \mathbf{Pre}(\mathbf{B}) \wedge \Diamond \left(\mathbf{Post}(\mathbf{B}) \wedge (\neg[\alpha_{j,i} \lessdot]) \; \mathcal{S} \; [\alpha_{i,j} \lessdot m]\right) \end{aligned}$$

The property means that both preconditions hold at the time of message sending and eventually **Post**(**A**) will hold and when it does there will have been no communication on channel $\alpha_{i,j}$ since the last time message m was sent to it. Also **Post**(**B**) will eventually hold and when it does there will have been no communication on channel $\alpha_{j,i}$ since the last time message m was sent to channel $\alpha_{i,j}$. The pre- and postconditions for a particular performative should be expanded to expressions in our semantic language, however the KQML specification describes these conditions using a semantic language which includes operators **Know**, **Want** and **Intend** whose meaning has not been defined, so we can go no further.

8.1.2 Verify Social Facts are Always Respected

With social languages acts create or modify social facts and we may discuss whether or not agents respect the social facts. If we have access to an agent's internals, we can verify at design time that the agent will always respect its social commitments regardless of what other agents in a system do. From the agent's code we construct the transition system S_i as described in § 5.1; to verify that agent i always respects its social commitments we need to prove that the following property holds over all computations of the multi-agent system S_i:

$$\forall x : \Box \; (f_i[x] \rightarrow [\![x, i]\!]_f) \tag{2}$$

Where x is a variable denoting a well formed formula of the social facts language $\mathcal{L}_f$. Thus for all social facts x that are true for agent i we require that the semantic formula corresponding to x holds in our model. $[\![x, i]\!]_f$ gives us the set of models where agent i has satisfied its part of the social facts, i.e. some facts x may be commitments for other agents, not satisfiable by i. Agent i should not be deemed non-compliant if some other agent has dishonoured a commitment to i. In practice we will not need to check all possible well formed formulae of the social facts language, inspection of the ACL specification can allow us to identify the set of social facts that may arise. This is provided that our ACL satisfies certain reasonable requirements, for example an agent should not be able to create commitments for another agent without notifying the other. If an agent is implemented by a finite state program[2] then we can use a model checking algorithm to perform the verification, it is less complex than proof theoretic verification.

8.2 Verify the Outcome of a System

The designer of a multi-agent system may want to verify that a certain outcome will occur given a certain initial state. If the internals of all agents are known this is simply a matter of proving that a property holds eventually in all computations of the system. This is independent of any communication language. If we don't know the internals of all the agents in the system, we cannot say much about the outcome unless we make some assumptions about unknown agents.

8.2.1 Verify Outcome for Compliant Agents

Supposing we have designed an agent (whose internals are known to us) and we wish to verify at design time that a certain outcome is guaranteed when we let our agent run in a system of agents whose internals we do not have access to. We construct a fair transition system S_i which represents all the possible behaviours of our agent in any environment as described in § 5.1. We constrain these possibilities by imposing the requirement that other agents in the system must be compliant. Let $\mathcal{D}$ be a formula characterising our desired outcome state.

[2]A finite state program is one where each system variable assumes only finitely many values in all computations.

Then if the formula

$$[(\forall j \in Ag : \forall x : \ \Box\,(f_j[x] \rightarrow [\![x, j]\!]_f)] \rightarrow \Diamond\,\mathcal{D} \qquad (3)$$

holds over all computations of the multi-agent system S_i, the outcome $\mathcal{D}$ is guaranteed to eventually occur in a system of compliant agents. This type of verification is possible both with mental and social languages. We can split the proof into two stages: firstly to show that the constraints on the system are sufficient to ensure that a certain agent strategy $\mathcal{S}$ will result in an outcome $\mathcal{D}$; secondly to show that an agent's code implements the strategy $\mathcal{S}$.

8.2.2 Prove a Protocol Property

Proving properties of protocols at design time is possible for both mental and social languages even when the internals of agents are not accessible. If a property p holds for any system of compliant agents executing a protocol *prot*, then we say that protocol *prot* has property p. With a social language, the proof is carried out as follows

- Let p be an assertion characterising the desired property to be proved for protocol *prot*.
- Set the initial condition Φ to an assertion characterising a social state where protocol *prot* has started.
- Construct a fair transition system S_E which represents all possible observable sequences of states (see § 5.2).
- Prove the following over all computations of the multi-agent system S_E:

$$[\forall i \in Ag : \forall x : \ \Box\,(f_i[x] \rightarrow [\![x, i]\!]_f)] \rightarrow p \qquad (4)$$

This states that if all agents are compliant then property p will hold. The quantifier over social facts x can be simplified in practice and we need only consider social relations that can arise in the protocol under consideration. The quantifier over agent identifiers needs to consider agents that are involved in the protocol and these agents must be specified in the initial condition Φ as they will occupy certain roles in the protocol. For a worked example of this type of verification see [6].

8.3 Verify Mental Semantics at Run Time

This type of verification is performed at run time with a mental language. Given that the system is in a certain state s where a communication has just taken place (by passing a message m), we wish to verify that the semantics of the communication language are satisfied for that communication. We check that the semantic formula (the first part of the tuple returned by $[\![m]\!]_c$) is satisfied on all possible paths from this point. This type of verification allows for the possibility that the semantics are respected in this instance but may not always be respected by the agents of the system. We set the initial assertion to an assertion characterising the state s.

$$\Theta = \bigwedge_{v \in V} (v = s[v])$$

Then we verify for the system that $[\![m]\!]_c \downarrow 1$, the semantic condition for message m holds. The type of verification discussed by [16] falls in this category.

8.4 Verification Using an Observable History

This is used to determine if an agent i is compliant by observing its external behaviour at run time. We assume that we have access to the ACL specification, an initial description of social facts and an observable history which takes the form of a history of messages exchanged by one agent or by the entire system. With this information it may be possible to determine if agents have complied with the ACL thus far, but not to determine if they will comply in future. However, this is probably the only kind of verification possible in open systems.

8.4.1 Verify Social Facts by History

This is the type of verification discussed by Singh, where "agents could be tested for compliance on the basis of their communications"[13]. Recall that a history of messages and an initial social state description Φ can uniquely describe a sequence of observable states. From information of sending events we construct social states which are consistent with the sequence of sending events, but where no messages need to be removed from channels. Let the observed communications take the form of an ordered sequence

$$m^1, m^2, m^3, \ldots, m^t.$$

We construct a temporal formula

$$\mathcal{M} : \quad \bigwedge_{j=1}^{t} \bigcirc^j [\alpha_{m^j \downarrow 2, m^j \downarrow 1} \Subset m^j] \qquad (5)$$

where $\bigcirc^j$ stands for j applications of the *next* operator; for example the intended meaning of $\bigcirc^3$ is $\bigcirc\bigcirc\bigcirc$. The subscript on the channel identifier takes its values from the first and second parts of the message tuple m; these will be the agent identifiers of sender and receiver.[3]

We then construct a fair transition system S_E which represents all possible observable sequences of states. Φ is the initial condition of S_E; the variables are the channels of agents present in Φ and the social state variables; the transitions are τ_I and τ_E as described in § 5.2. We now say that a model

$$\sigma_h : s_h^0, s_h^1, s_h^2, s_h^3, \ldots, s_h^t, \ldots$$

is a possible model of our observed system if σ_h is a computation of multi-agent system S_E and if $(\sigma_h, 0) \vDash \mathcal{M}$. This gives us the set of models $\mathcal{E}$ which match the observed finite sequence up to state s_h^t and thereafter take all possible paths by taking the idling or environmental transitions. Note that the models constructed here do not coincide with models of the entire system where the transitions of agent programs are considered and many transitions do not involve message passing; however, the semantics of social facts will never refer to an absolute number of states, so this model is sufficient for verification.

Now we can interpret the semantics of each of i's social facts over these models. Certain social facts in states of a model σ_h may already have their semantics satisfied before s_h^t (i.e. satisfied in the sequence which proceeds after s_h^t by infinite applications of the idling transition) for example obligations which have been fulfilled. Certain other facts may not have their semantics satisfied yet, though it may

[3]We are assuming that agents do indeed place their own identifier first in the message tuple; this could easily be enforced at the agent platform level if need be.

be possible that they will be satisfied after s_h^t and do not yet constitute a violation. We cannot simply check each fact one by one as there may be two social facts in σ_h which have not yet been satisfied by s_h^t, each of which could be satisfied in a model of $\mathcal{E}$, but which could not both be satisfied in the same model of $\mathcal{E}$. We must therefore search for the existence of a model where all the social facts are satisfied. We will also require that the states subsequent to s_h^t satisfy their social semantics; it would not suffice to find a model where agent i satisfies the semantic formula for a social fact in a state of the observed sequence only by performing a non compliant action after s_h^t.

Thus we wish to check if there exists a model σ_h in $\mathcal{E}$ in which the semantic formulae for all social facts in all states are satisfied; i.e. it is possible that the observed sequence σ_h is part of a model where i is compliant. We say that an agent i is compliant if the following formula holds:

$$\exists\ \sigma_h \in \mathcal{E} :\ \forall x \in \mathit{wff}(\mathcal{L}_f) :\ (\sigma_h, 0) \vDash \Box\,(f_i[x] \rightarrow [\![x, i]\!]_f) \quad (6)$$

8.4.2 Verify Protocols by History

With a protocol based language it may be possible to verify compliance with a protocol by observing a history of communications if the semantics of acts define obligations to perform observable actions. This is the case with sACL [10] which defines the semantics of an act as a postcondition which is an intention for the receiver (written I_r) to reply, given a predefined possible set of replies.

$$[\![< s, \mathit{perf}(r, \mathit{content}) >]\!] = \mathrm{I}_r < r, \mathit{sa} >$$

Where s is the sender, *perf* the the performative, r the receiver and *content* the message content for the outgoing message. The message *sa* which the receiver intends to reply with must use a performative given by the *reply* function which encodes the protocols and returns performatives appropriate to the current stage of this conversation. Given an observable history as described above (§ 8.4), we can verify if each agent respects the protocol by checking that an agent does send the message he is obliged to send after receiving a message. This approach is effectively giving a social semantics to the intention to reply by interpreting it as an observable obligation, hence we are really creating a new language which is no longer entirely mental. This is why table 1 states that protocols cannot be verified for a mental language by observing a history.

8.5 Verifiability

Table 1 has shown what types of verification are possible for mental and social languages; however, we have seen that some languages may not be verifiable at all if certain components of the framework are missing (§ *8.1.1*). Table 2 shows what language components are present in several different languages. For mental languages both $\mathit{wff}(\mathcal{L}_s)$ and $[\![-]\!]_s$ must be present to allow any type of verification at all. These languages provide the relationship between the communication language semantics and a grounded computational model. Viewed in this way, the problem of verifiability is often a problem of missing language components. While FIPA [4] has specified a semantic language $\mathit{wff}(\mathcal{L}_s)$, it has given it a semantics using modal operators;[4] it has not attempted to give it a grounded semantics in terms of a computational model, and this is what we require of the component $[\![-]\!]_s$ in our framework. In contrast, the language of Wooldridge [15] includes all four components necessary for a mental language to be verifiable. Although it defines the semantics of an *inform* in terms of an agent's knowledge, this knowledge operator is grounded in terms of states of the agent program.

ACL	***ACL Components***					
	$\mathit{wff}(\mathcal{L}_c)$	$[\![-]\!]_c$	$\mathit{wff}(\mathcal{L}_s)$	$[\![-]\!]_s$	$\mathit{wff}(\mathcal{L}_f)$	$[\![-]\!]_f$
FIPA	✓	✓	✓	-	-	-
KQML	✓	✓	✓	-	-	-
Wooldridge [15]	✓	✓	✓	✓	-	-
Singh [13]	✓	✓	✓	✓	✓	✓
Necessary for:	mental languages					
	social languages					

Table 2: ACLs and their Constituent Components.

A social language must specify all six components if it is to be verifiable:

- Messages are written in $\mathcal{L}_c$ and $[\![-]\!]_c$ defines how they create or modify social facts (such as commitment, authority, power).
- $[\![-]\!]_f$ is necessary to provide a mapping from social facts to social facts semantics, i.e. expressions in some language $\mathit{wff}(\mathcal{L}_s)$ (such as temporal logic).
- $[\![-]\!]_s$ is necessary to give these expressions a grounding in the computational model.

We see that the language of Singh [13] does meet these requirements; let us look at Singh's *request* and its objective meaning as an example: $\mathit{request}(x, y, p)$ is a message of $\mathcal{L}_c$; this is given a semantics $[\![-]\!]_c$ which maps it to C(x, y,G,RFp), a commitment in a social language $\mathcal{L}_f$ (this is the objective meaning, there is also a subjective and practical meaning for each act). The expression C(x, y,G,RFp) means that x commits that he expects y to make p true. This expression is in turn given a semantics using $\mathcal{L}_s$, a variant of Computation Tree Logic (CTL). CTL formulas have a semantics $[\![-]\!]_s$ in terms of the system models where they are satisfied. Singh has in fact put together the languages $\mathcal{L}_s$ and $\mathcal{L}_f$ by extending the syntax and semantics of CTL so that commitments can be specified within it, and their semantics given in terms of the other CTL primitives.

As mentioned earlier, in an open system it may only be possible to make external observations and if so, as shown in table 1, the only verification possible will be by observing a history with a social language. The only language in our table, which could be verifiable in an open system, is Singh's language; this is because the semantics of communication is grounded in *social states* (which are observable in an open system), in contrast the semantics of a mental language is grounded in *program states* (which may not be accessible in an open system).

[4]The semantics of these operators is not given in any of the FIPA documents, see also [11].

8.6 Verification in an Open System

We now enumerate the four types of verification which are useful in an open system.

1. Verify that an agent always satisfies its social facts (equation 2, § *8.1.2*).

2. Verify the outcome of a system, assuming unknown agents are compliant (equation 3, § *8.2.1*).

3. Prove a property of a protocol (equation 4, § *8.2.2*).

4. Determine if an agent is not respecting its social facts at run time (equation 6, § *8.4.1*).

These types support each other, for example, proving properties of open systems requires three verification types: agent designers must be able to prove that individual agents are compliant (type 1); the protocol designer must be able to prove properties for a system of compliant agents using the protocol (type 3); and the system itself needs to determine if agents do comply with social commitments at run time (type 4) in order to police the society and guarantee that rogue agents cannot damage the system's properties.

8.7 Policing an Open Society

With reference to the enumerated verification types for an open system above, types 2 and 3 require that all agents comply. To be able to use these in an open system there must be some way to enforce compliance. The issue of policing a society can be tackled in one of the following three ways.

Sentinel agents may monitor the observable behaviour of agents and have the capability to place sanctions or to evict or terminate offending agents. If we guarantee that all violators are evicted then the system progresses as if all agents complied; however, we must design protocols in such a way that an eviction cannot destroy the desirable properties of the system.

If the society has to police itself we may introduce notions like trust and politeness, whereby agents violating certain commitments or conventions of the society are branded as untrustworthy or antisocial and are ostracised by the rest of the society. Prisoner's dilemma experiments [1] have shown that a strategy of reciprocating (rewarding good behaviour and punishing bad behaviour) has the effect of policing the society because agents will not tend to misbehave if they cannot thereby gain an advantage. If we want self policing we must consider this in the design of protocols so that all agents participating can observe enough information to determine if an agent complies.

Yet another possibility is that agent owners will be legally responsible for the behaviour of their agents. Agents will not be allowed to participate in a system unless their owner guarantees that they are compliant. Then if such an agent misbehaves at run time some sanction (such as a fine) can be placed on the agent owner. This approach has the drawbacks that it requires some centralised authority and the practicality of policing a system as distributed as the internet might be questionable [14]. However, if the exchange of real money is to be carried out by agents, there will inevitably be some human or institution who is liable.

9. RELATED WORK

Complete agent communication frameworks are not common in the literature, most work being aimed at low level specification of protocols or approaches to semantic specification, as discussed in [5]. We now review two explicitly specified agent communication frameworks for comparison with our framework. Firstly there is the framework by Wooldridge [16], which provided the starting point for this paper; it is quite similar but for the following differences. Wooldridge has represented agent programs and states and a semantics for these, while we have used a computational model of the system which represents the programs and states. Wooldridge's framework describes the state of the multi-agent system at some instant of time, possibly during an execution. Our framework describes the multi-agent system as designed, before it starts running. If we wish to investigate properties of the system at run time we must specify both the components of the framework and a particular state during the execution. Wooldridge's ACL component is based on FIPA where the semantics of a message defines a constraint that the sender of the message must satisfy; we extend this to allow message to define constraints for the sender or receiver or any other variables of the system. These constraints are regulative rules (see § 6); an additional component allows for the definition of constitutive rules in terms of social facts.

Secondly there is Van Eijk's "Verification Framework for Agent Communication" [3]. Van Eijk's framework provides a method by which an agent program can be shown to satisfy a certain specification; where the specification is in van Eijk's assertion language which includes expressions such as $\mathbf{B}_n\psi$, meaning that agent n has ψ in its information store. With van Eijk's framework we can write an assertion such as S *sat* Φ meaning that the agent program S satisfies the assertion Φ. In the context of agent communication, if the agent program S consists of a communication statement, we can use the assertion language to specify a constraint that the communication should satisfy.

The fundamental difference lies in this: Van Eijk's framework can provide a relationship between communication statements in an agent's program S and a separate specification Φ. In contrast our framework has divorced the semantics of communication statements in an agent's program from the semantics of the messages being sent. In our framework the messages sent have a semantics of their own and this is what is said to constitute an agent communication language. Our framework allows us to look at a message that is sent and to assign a meaning to it without any knowledge of the agent program which sent the message. We can then provide a relationship between a message and a separate specification without any knowledge of any agent program.

Van Eijk's framework provides a complete verification calculus for showing that a system of agents satisfies a specification. We can easily create an agent communication language using the framework by associating an agent program statement sending a particular message with a specification for the conditions of its correct use. For example, the sending of a $tell(\varphi)$ message could be related to a $\mathbf{B}_n\varphi$. Treated in this way van Eijk's assertions would constitute a mental language since the satisfaction of an assertion Φ by an agent is based on the agent's mental state. The type of verification presented by van Eijk has already been described in § 8.1, and our first type of verification is a special case of this.

The concept of histories as an observable sequence of messages "to enable a proof system which does not assume any knowledge about the internal structure of agents"[3] is the same in both frameworks. However, van Eijk goes on to define an observable behaviour that includes the agent's information store as well as its local communication history. This observable behaviour is calculated from the agent's internals; thus we would conclude that van Eijk's framework would be appropriate in closed systems where an agent's internal code and state can be determined.

Our framework has not specified a language for social facts, or for message contents; Van Eijk tackles the issue of the content language using the Concurrent Constraint Programming paradigm which defines a language of logical assertions which can be sent as message contents. Thus it is conceivable that the two frameworks could complement one another.

10. CONCLUSIONS AND FUTURE WORK

We have developed a general agent communication framework which includes a computational model and an agent communication language. This allowed us to investigate different types of verification and to identify which components must be present in the framework to facilitate each type of verification. We have identified the need for a language based on social conventions for applications with open systems of agents, such as e-commerce applications. We have also described the types of verification that are possible in such systems and the types of policing which could be used to enforce compliance. This theoretical framework could be used to implement useful tools for such systems. For example, an agent platform could automatically monitor the messages exchanged and identify (and take action against) rogue agents. A tool could aid a designer by automatically checking if an agent's code is compliant and checking if protocols give the expected outcomes.

The use of temporal logic for the specification of social facts allows many properties to be specified but does not allow an absolute time frame to be referenced; this could be achieved by moving to a clocked transition system [7].

Future work involves applying a model checking algorithm to each type of verification; this will use protocol diagrams as state transition diagrams for observable systems, some of this is described in [6].

11. REFERENCES

[1] R. Axelrod. *The evolution of cooperation*. Basic Books, New York, 1984.

[2] R. M. v. Eijk. *Programming Languages for Agent Communication*. PhD thesis, Department of Information and Computing Sciences, Utrecht University, 2000.

[3] R. M. v. Eijk, F. de Boer, W. van der Hoek, and J.-J. Meyer. A verification framework for agent communication. *Journal of Autonomous Agents and Multi-Agent Systems*, to appear, 2001.

[4] FIPA. [FIPA OC00003] FIPA 97 part 2 version 2.0: Agent communication language specification. In *Foundation for Intelligent Physical Agents. http://www.fipa.org/specs/fipa2000.tar.gz*, 1997.

[5] F. Guerin and J. Pitt. Denotational semantics for agent communication languages. In *Autonomous Agents 2001, Montreal*, pages 497–504. ACM Press, 2001.

[6] F. Guerin and J. Pitt. Guaranteeing properties for e-commerce systems. In *Autonomous Agents 2002 Workshop on Agent Mediated Electronic Commerce IV: Designing Mechanisms and Systems, Bologna*, 2002.

[7] Y. Kesten, Z. Manna, and A. Pnueli. Verifying clocked transition systems. In *Hybrid Systems III, LNCS vol. 1066*, pages 13–40. Springer-Verlag, Berlin, 1996.

[8] Y. Labrou and T. Finin. A semantics approach for kqml – a general purpose communication language for software agents. In *Third International Conference on Information and Knowledge Management (CIKM'94)*, pages 447–455, 1994.

[9] Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems (Safety), vol. 2*. Springer-Verlag, New York, Inc., 1995.

[10] J. Pitt and A. Mamdani. A protocol-based semantics for an agent communication language. In *Proceedings 16th International Joint Conference on Artificial Intelligence IJCAI'99, Stockholm*, pages 486–491. Morgan-Kaufmann Publishers, 1999.

[11] J. Pitt and A. Mamdani. Some remarks on the semantics of FIPA's agent communication language. *Autonomous Agents and Multi-Agent Systems*, 4:333–356, 1999.

[12] J. R. Searle. What is a speech act ? In *Philosophy of Language. edited by A.P. Martinich, Third edition.* 1996, Oxford University Press, 1965.

[13] M. Singh. A social semantics for agent communication languages. In *IJCAI Workshop on Agent Communication Languages, Springer-Verlag, Berlin.*, 2000.

[14] M. Wooldridge. Verifiable semantics for agent communication languages. In *ICMAS'98*, 1998.

[15] M. Wooldridge. Verifying that agents implement a communication language. In *Sixteenth National Conference on Artificial Intelligence (AAAI-99), Orlando, FL, (July 1999)*, 1999.

[16] M. Wooldridge. Semantic issues in the verification of agent communication languages. *Journal of Autonomous Agents and Multi-Agent Systems*, 3(1):9–31, 2000.