Verification and Compliance Testing

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.
{f.guerin,j.pitt}@ic.ac.uk
Phone: +44 20 7594 6318 / Fax: 6274

Abstract. Verification and compliance testing are required if agents are to be delegated responsibility for legally binding contracts, for example in electronic markets. This paper describes a general agent communication framework which allows several different notions of verification and compliance testing to be described. In particular we consider what type of verification or testing may be possible depending on the information which may be available (agent internals, observable behaviour, normative specifications) and the semantic definition of the communication language. We use this framework to identify the types of languages which will permit verification and testing in open systems where agents' internals are kept private. This analysis gives some ideas about how compliance might be enforced in an open system.

1 Introduction

Verification means checking the specifications or programs of a multi-agent system at design time to ensure that the system will behave as desired in all possible runs. Compliance testing means checking the behaviour of the system at run time to determine if it behaves as desired. Desired behaviour may mean compliance with some normative specification, for example honouring contracts in an e-commerce system. Such a normative specification must specify something about required states of the multi-agent system, these states may be agent states or states of the society. Before we can investigate different notions of verification and compliance testing we must have a frame of reference: a general agent communication framework which describes the normative constraints as well as the states of the system.

We begin by describing a general agent communication framework (§ 2). This framework must describe the agent programs which a multi-agent system in composed of; these can be represented by a computational model for the multi-agent system (§ 2.1). We then specify additional variables to capture observable states of an open system through a representation of the agents' social context (§ 2.2) and the states of this context (§ 2.3). Next we define the ACL component of the framework (§ 2.6) which can accomodate languages based on mental or social

M.-P. Huget (Ed.): Communications in Multiagent Systems, LNAI 2650, pp. 98–112, 2003.

states and we summarise the general framework (§ 2.7). Using this framework as a reference we describe several notions of verification and testing (§ 3). We use the framework to analyse existing ACLs to determine if they are verifiable (§ 4) and we identify the type of ACL which would be appropriate in an open system. Finally we discuss how compliance might be enforced in an open system (§ 5) and conclude (§ 6).

2 A General Agent Communication Framework

Before discussing verification and compliance testing we need to describe a general agent communication framework containing the necessary components of a multi-agent system. Our framework builds on the framework presented by Wooldridge [14]; we attempt to make the framework more general to allow ACLs with social semantics to be accommodated.

An agent communication language (ACL) typically has two functions[1]:

- To specify the meaning of messages; this is useful to an agent designer who is deciding when an agent will plan to send a message and also what way the agent will update its internal state upon receiving a message.
- To provide a normative specification for communication in the system; this is necessary to ensure that the multi-agent system does not become dysfunctional, for example to ensure that agents respond when spoken to and that they honour their commitments.

We say that the first part defines a semantics for each communicative act while the second (normative) part defines a specification which must be satisfied by the system of agents using that language. 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.

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 [7] 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 [11]. 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.

[1] The first agent communication languages confused these two functions (for example FIPA [3] and KQML [7]).

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.

2.1 Computational Model

Each agent in a multi-agent system can be described as a module. An entire multi-agent system can be described by a single program P executing all these modules in parallel. Each module 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.

A computational model is a 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 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.

The computational model we choose for multi-agent systems is a fair transition system [8]. This is a system which contains variables and transitions and an initial comdition specifying initial values for the variables. Variables represent the states of the agents and transitions represent the state changes caused by statements in the agents' programs. A *state* is an interpretation of the variables, assigning each variable a value over its domain. As a program executes it passes through a sequence of states in which variables may take on different values.

2.2 Representing Social Context

A language $\mathcal{L}_F$ is introduced to describe social facts. Social facts describe role relationships, commitments to perform actions and publicly expressed attitudes. Each agent may have a differing view of the social context, since it may not have received all events (communications) occurring in the system; therefore for each agent we use a unique variable to describe the social state observable to it. The type of each social state variable is a mapping from well formed formulae of the social facts language $\mathcal{L}_F$ to true or false values. We also define an initial condition Φ for the social state variables.

Given an initial social state Φ and a certain sequence of messages, we can work out the values of the social facts in subsequent states as follows: 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. 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 the social state.

The social state 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.

2.3 States and Computations

A state s is an interpretation of the variables, assigning each variable a value over its domain. 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 and if each state s^{j+1} is accessible from the previous state s^j via one of the transitions in the system. 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 the agent programs. We specify additional constraints on the interpretations of the social state variables 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 where the social facts variables are also updated according to the ACL specification each time a communication occurs.

2.4 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 [8]. 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; 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 may be modified arbitrarily. S_i represents all the possible behaviours of agent i in any multi-agent system.

2.5 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. 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.

2.6 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. The specification for the semantics of communicative acts is a function $[\![-]\!]_C$ which varies depending on whether the ACL is based on mental or social states.

Mental: The function $[\![-]\!]_C$ returns a formula in temporal logic $\mathcal{L}_T$. 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. The formula is given a semantics $[\![-]\!]_T$ in terms of the set of models where the formula is satisfied.

Social: The function $[\![-]\!]_C$ returns a function from social state to social state. The function describes the change to the social state caused by the message

transmitted. Since the ACL is responsible for defining the changes that messages cause in the social state, the ACL must also define the language $\mathcal{L}_F$ for social facts whose semantics $[\![-]\!]_F$ returns a temporal formula in $\mathcal{L}_T$ which is interpreted over an observable model (i.e. a model where we are concerned only with how states interpret observable variables). 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. 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 but it cannot place constraints on agent internals.

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

$$\mathcal{ACL} = \langle \mathcal{L}_C, \mathcal{L}_F, \mathcal{L}_T \rangle$$

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.

2.7 Agent Communication Framework

An agent communication framework is a 4-tuple:

$$\langle Ag, \mathit{Compmodel}, \mathcal{ACL}, \Phi \rangle$$

- Ag is a set of agent names, $Ag = \{1, \ldots, n\}$;
- *Compmodel* is the fair transition system representing all the programs of all the agents in the multi-agent system;
- $\mathcal{ACL} = \langle \mathcal{L}_C, \mathcal{L}_F, \mathcal{L}_T \rangle$ is an ACL including mental and social components;
- Φ is the initial assertion for social states.

Using this framework we can define a few notions of verification and compliance testing.

3 Types of Verification and Testing

Several different types of verification and compliance testing are possible depending on the *type of ACL* used, the *information available* and whether we wish to verify at *design time* or test 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 testing is used to determine if agents are misbehaving in a certain run of the system. Run time compliance testing 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.

		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	-	-	•	-	-	•
compliance testing	Test mental semantic formula	•	-	-	-	-	-
	Test via observed history						
	Test social constraints by history	-	-	-	-	•	-
	Test protocols by history	-	-	-	-	•	-

• verification is possible and appropriate

Table 1. Types of Verification and Compliance Testing

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 § 2.6), 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 and testing.

1. *Internal States*: The agent designer or system designer will typically have access to internal states. By internal states we mean the agents' program code as well as their state during execution. Knowledge of the agents' code permits verification of future behaviour. This type of knowledge is usually necessary to verify compliance with mental languages. When a social language is used, the agent designer who has access to internal states of the agents may still wish to verify at design time that the agent will always respect the semantics. We may use information about internal states for both design time verification and run time testing.
2. *External States*: In an e-commerce scenario different vendors contribute their own agents to the system; these vendors might not desire to publicise the internal code of their agents. Even if the code is publicised, the agents might be based on very different architectures making it too difficult to verify their behaviour by analysis of the code. In such cases the system administrator has to perform some type of compliance testing which works with observable social states. If we assume that the communications occurring in a run of the system can be observed then testing based on external states is possible at run time for languages based on social states.
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 and testing that are possible and appropriate based the information available. Note how the only hope for run time compliance testing in an open system (where agent internals are inaccessible) is with a social language. We now give a more detailed explanation in terms of our general framework.

3.1 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; this 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.

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. This means that for any computation, we verify that for each state which involves message passing, the semantics of that message are satisfied. The semantic 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.

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 facts regardless of what other agents in a system do. From the agent's code we construct the transition system S_i as described in § 2.4; to verify that agent i always respects its social commitments we need to prove that in all computations of the multi-agent system S_i, whenever a social fact x is true for agent i, the semantic formula corresponding to x holds in the model.

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.

3.2 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.

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 which represents all the possible behaviours of our agent in any environment as described in § 2.4. Then we prove that our desired outcome is guaranteed if all external transitions in the environment are compliant with the normative constraints of the ACL. This type of verification is possible both with mental and social languages.

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

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

- 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 § 2.5).
- Prove the following over all computations of the multi-agent system S_E: if all agents are compliant then property p will hold.

The antecedant "all agents are compliant" requires that all social facts are respected by all agents; in practice and we need only consider social facts that can arise in the protocol under consideration; likewise we need only 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 [5].

3.3 Test Mental Semantic Formula at Run Time

This type of compliance testing 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 check if the semantics of the communication language are satisfied for that communication. We check that the mental semantic formula is satisfied on all possible paths from this point. This type of testing 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. Then we check for this system that the mental semantic formula for message m holds. The type of verification discussed by Wooldridge [14] falls in this category.

3.4 Testing Using an Observable History

This is used to determine if an agent 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 testing possible in open systems.

Test social constraints by history: This is the type of compliance testing discussed by Singh, where "agents could be tested for compliance on the basis of their communications"[11]. 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. 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 § 2.5. We then find the set of all models which match the observed finite sequence up to its final state 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 testing.

Now we can interpret the semantics of each of our observed agent's social facts over these *observed* models. Certain social facts in states of an observed model may already have their semantics satisfied before the state where the last observed message was passed (i.e. satisfied in the sequence which proceeds after that state 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 be possible that they will be satisfied after and do not yet constitute a violation.

Thus we wish to check if there exists an observed model in which the semantic formulae for all social facts in all states are satisfied; i.e. it is possible that the observed sequence is part of a model where the observed agent is compliant.

Test Protocols by History: With a protocol based language it may be possible to test 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 [9] which defines the semantics of an act as a postcondition which is an intention for the receiver to reply, given a predefined possible set of replies. Given an observable history as described above (§ 3.4), we can see 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 tested for a mental language by observing a history.

4 Verifiability

Table 1 has shown what types of verification are possible for mental and social languages; however, some languages may not be verifiable at all if certain components of the communication framework are missing. Typically we describe this in terms of missing language components; recall that we had three languages as part of our ACL: A communication language, a social facts language and temporal logic. In the more general case the temporal logic language $\mathcal{L}_T$ can be replaced by any semantic language $\mathcal{L}_s$ which will provide a relationship between the semantics of communication (either mental states or social facts) and

a grounded computational model. Table 3 shows what language components are present in several different languages. For mental languages both $wff(\mathcal{L}_s)$ and $[\![-]\!]_s$ must be present to allow any type of verification at all. While FIPA has specified a semantic language $wff(\mathcal{L}_s)$, it has given it a semantics using modal operators;[3] 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 QUETL language of Wooldridge [13] 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***					
	$wff(\mathcal{L}_C)$	$[\![-]\!]_C$	$wff(\mathcal{L}_s)$	$[\![-]\!]_s$	$wff(\mathcal{L}_F)$	$[\![-]\!]_F$
FIPA	✓	✓	✓	-	-	-
KQML Cognitive States	✓	✓	✓	-	-	-
QUETL Wooldridge '99 [13]	✓	✓	✓	✓	-	-
Singh Commitments 2000 [11]	✓	✓	✓	✓	✓	✓
Necessary for:	mental languages					
	social languages					

Table 2. Some ACLs and their Constituent Language Components.

A social language must specify all six components if it is to be verifiable: messages are written in $\mathcal{L}_C$ and $[\![-]\!]_F$ defines how they create or modify social facts; both $wff(\mathcal{L}_F)$ and $[\![-]\!]_F$ are necessary to provide a mapping from social facts to social facts semantics; both $wff(\mathcal{L}_s)$ and $[\![-]\!]_s$ are necessary to give that semantics a grounding in the computational model. We see that the language of Singh [11] does have all six components, let us look at a *request* and its objective meaning as an example:

- $\mathcal{L}_C$ is the language in which messages are written, such as $request(x, y, p)$; this is given a semantics $[\![-]\!]_C$ which maps it to $\mathrm{C}(x, y, \mathrm{G}, \mathrm{RF}p)$, an expression in a social language $\mathcal{L}_F$ (this is the objective meaning, there is also a subjective and practical meaning for each act).
- $\mathcal{L}_F$ is a language of commitments, the expression $\mathrm{C}(x, y, \mathrm{G}, \mathrm{RF}p)$ means that x commits that he expects y to make p true. This expression is in turn given a semantics $[\![-]\!]_F$ as an expression in $\mathcal{L}_s$.

[3] The semantics of these operators is not given in any of the FIPA documents, see also [10].

– $\mathcal{L}_s$ is 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. The semantics of the objective and practical meanings are grounded in observable social states of the system.

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 might not be accessible in an open system).

5 Verification, Testing and Enforcement in an Open System

The following are the most important types of verification in an open system.

1. Verify that an agent always satisfies its social facts.
2. Prove a property of a protocol.
3. Determine if an agent is not respecting its social facts at run time.

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 2); and the system itself needs to determine if agents do comply with social commitments at run time (type 3) in order to police the society and guarantee that rogue agents cannot damage the system's properties.

5.1 Policing an Open Society

With reference to the enumerated verification types for an open system above, type 2 requires that all agents comply. To be able to use this 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 [12]. 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.

6 Conclusions and Future Work

We have described a general agent communication framework which includes a computational model and an agent communication language component which can accomodate an ACL based on mental or social states. 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 [6].

Current and 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, much of this is described in [4].

References

[1] R. Axelrod. *The evolution of cooperation*. Basic Books, New York, 1984. 111

[2] R. M. v. Eijk. *Programming Languages for Agent Communication*. PhD thesis, Department of Information and Computing Sciences, Utrecht University, 2000. 105

[3] FIPA. [FIPA OC00003] FIPA 97 Part 2 Version 2.0: Agent Communication Language Specification. In *Website of the Foundation for Intelligent Physical Agents. http://www.fipa.org/specs/fipa2000.tar.gz*, 1997. 99

[4] F. Guerin. *Specifying Agent Communication Languages*. PhD thesis, Department of Electrical and Electronic Engineering, Imperial College, UK, 2002. 111

[5] 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. 107

[6] 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. 111

[7] 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. 99, 99

[8] Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems (Safety), vol. 2*. Springer-Verlag, New York, Inc., 1995. 100, 101

[9] 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. 108

[10] 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. 109

[11] M. Singh. A social semantics for agent communication languages. In *IJCAI Workshop on Agent Communication Languages, Springer-Verlag, Berlin.*, 2000. 99, 107, 109, 109

[12] M. Wooldridge. Verifiable semantics for agent communication languages. In *ICMAS'98*, 1998. 111

[13] M. Wooldridge. Verifying that agents implement a communication language. In *Sixteenth National Conference on Artificial Intelligence (AAAI-99), Orlando, FL, (July 1999)*, 1999. 109, 109

[14] M. Wooldridge. Semantic issues in the verification of agent communication languages. *Journal of Autonomous Agents and Multi-Agent Systems*, 3(1):9–31, 2000. 99, 107