Verifying Compliance with Commitment Protocols

Mahadevan Venkatraman and Munindar P. Singh
Department of Computer Science,
North Carolina State University,
Raleigh, NC 27695, USA

mvenkat@eos.ncsu.edu, singh@ncsu.edu

TR 98-13

Abstract

Interaction protocols are specific, often standard, constraints on the behaviors of the autonomous agents in a multiagent system. Protocols are essential to the functioning of open systems, such as those that arise in most interesting web applications. A variety of common protocols in negotiation and electronic commerce are best treated as *commitment protocols*, which are defined, or at least analyzed, in terms of the creation, satisfaction, or manipulation of the commitments of the various agents to one another.

When protocols are employed in open environments, such as the Internet, they must be executed by agents that behave more or less autonomously and whose internal designs are not known. In such settings, therefore, there is a risk that the participating agents may fail to comply with the protocol. Without a rigorous means to verify compliance, the very idea of protocols for interoperation is subverted. We develop a novel approach for testing the compliance of agents with respect to a commitment protocol. Our approach requires the specification of commitment protocols in temporal logic, and involves a novel way of synthesizing and applying ideas from distributed computing and logics of program.

1 Introduction

Interaction among agents is the distinguishing property of multiagent systems. However, ensuring that only the desirable interactions occur is one of the most challenging aspects of multiagent system analysis and design. This is especially so when the given multiagent system is meant to be used as an open system, for example, in web-based applications. In such a system, the member agents are contributed by several sources and serve different interests. Thus, these agents must be treated as

- *autonomous*—with few constraints on behavior, reflecting the independence of their users, and
- *heterogeneous*—with few constraints on construction, reflecting the independence of their designers.

Effectively, the multiagent system is specified as a kind of standard that its member agents must respect. In other words, the multiagent system can be thought of as specifying a protocol that governs how its member agents must act. For our purposes, the standard may be *de jure* as created by a standards body, or *de facto* as may emerge from practice or even because of the arbitrary decisions of a major participant. All that matters is that a standard imposes some restrictions on the agents. Consider the fish-market protocol as an example of such a standard protocol [14].

Example 1 In the fish-market protocol, we are given agents of two roles: a single auctioneer and one or more potential bidders. The fish-market protocol is designed to sell fish. The seller or auctioneer announces the availability of a bucket of fish at a certain price. The bidders gathered around the auctioneer can scream back *Yes* if they are interested and *No* if they are not; they may also stay quiet, which is interpreted as a lack of interest or *No*. If exactly one bidder says *Yes*, the auctioneer will sell him the fish; if no one says *Yes*, the auctioneer lowers the price; if more than one bidder says *Yes*, the auctioneer raises the price. In either case, if the price changes, the auctioneer announces the revised price and the process iterates. ▮

Because of its relationship to protocols in electronic commerce and because it is more general than the popular English and Dutch auctions, the fish-market protocol has become an important one in the recent multiagent systems literature. Accordingly, we use it as our main example in this paper.

Because of the autonomy and heterogeneity requirements of open systems, compliance testing can neither be based on the internal designs of the agents nor on concepts such as beliefs, desires, and intentions that map to internal representations [20]. The only way in which compliance can be tested is based on the behavior of the participating agents—this may be by a central authority or any of the participating agents. However, the requirements for behavior in multiagent systems can be quite subtle. Thus, along with languages for specifying such requirements, we need corresponding techniques to test compliance.

1.1 Commitments in an Open Architecture

There are three levels of architectural concern in a multiagent system. One deals with individual agents; another deals with the systemic issues of how different services and brokers are arranged. Both of these have received much attention in the literature. In the middle is the multiagent *execution* architecture, which has not been as intensively studied within the community. An execution architecture must ultimately be based on distributed computing ideas albeit with an open flavor, e.g., [1, 5, 11]. A well-defined execution functionality can be given a principled design, and thus facilitate the construction of robust and reusable systems. Some recent work within multiagent systems has begun to address this level, e.g., Ciancarini *et al.* [8, 9] and Singh [17].

Much of the work on this broad theme, however, focuses primarily on coordination, which we think of as the lowest level of interaction. Coordination deals with how autonomous agents may align their activities in terms of what they do and when they do it. However, there is more to interaction in general, and the issues of compliance in particular. Specifically, interaction must include some consideration of the commitments that the agents enter into with each other. The commitments of the agents are not only base-level commitments dealing with what actions they must or must not perform, but also meta-commitments dealing with how they will adjust their base-level commitments [19]. Commitments provide a layer of coherence to the agents' interactions with each other. They are especially important in environments where we need to model any kind of contractual relationships among the agents.

Such environments are crucial wherever open multiagent systems must be composed on the fly, e.g., in electronic commerce of various kinds on the Internet. The addition of commitments as an explicit first-class object results in considerable flexibility of how the protocols can be realized in changing situations. We term such augmented protocols *commitment protocols*.

Example 2 We informally describe the protocol of Example 1 in terms of commitments. When a bidder says *Yes*, he commits to buying the bucket of fish at the advertised price. When the auctioneer advertises a price, he commits that he will sell fish at that price if he gets a unique *Yes*. Neither commitment is irrevocable. For example, if the fish smell bad, the auctioneer releases the bidder from paying for them. Specifying all possibilities in terms of irrevocable commitments would complicate each commitment, but would still fail to capture the practical meanings of such a protocol. For instance, the auctioneer may not honor his offering price if a sudden change in weather indicates that fishing will be harder for the next few days. ▮

1.2 Compliance in Open Systems

The existence of standardized protocols is necessary but not sufficient for the correct functioning of open multiagent systems. We must also ensure that the agents behave according to the protocols. This is the issue of *compliance*. However, unlike in traditional closed systems, verifying compliance in open systems is practically and even conceptually nontrivial.

Preserving the autonomy and heterogeneity of agents is crucial in an open environment. Otherwise, many applications would become infeasible. Consequently, protocols must be specified as flexibly as possible without making untoward requirements on the participating agents. Similarly, an approach for testing compliance must not require that the agents are homogeneous or impose stringent demands on how they are constructed.

Consequently, in open systems, compliance can be meaningfully expressed only in terms of observable behavior. This leads to two subtle issues. One, although we talk in terms of behavior, we must still consider the high-level abstractions that differentiate agents from other active objects. The focus on behavior renders approaches based on mental concepts ineffective [20]. However, well-framed social constructs can be used. Two, we must clearly delineate the role of the observer who assesses compliance.

1.3 Contributions

The approach developed here functions under some fairly flexible assumptions. It allows a fully distributed system on which the agents exist. There is an underlying messaging layer, which delivers messages asynchronously and, for now, reliably. However, the approach assumes for simplicity that the agents are not malicious and don't forge the timestamps on the messages that they send or receive.

The compliance testing is performed by any observer of the system—typically, a participating agent. Our approach is to evaluate temporal logic specifications with respect to locally constructed models for the given observer. The model construction proposed here employs a combination of potential causality and operations on social commitments (both described below). Our contributions are in

- incorporating potential causality in the construction of local models
- identifying patterns of messages corresponding to different operations on commitments.

Our approach also has important ramifications on agent communication in general, which we discuss in Section 4.

Organization. The rest of this paper is organized as follows. Section 2 presents our technical framework, which combines commitments, potential causality, and temporal logic. Section 3 presents our approach for testing (non-)compliance of agents with respect to a commitment protocol. Section 4 concludes with a discussion of our major themes, the literature, and the important issues that remain outstanding.

2 Technical Framework

Commitment protocols as defined here are a multiagent concept. They are far more flexible and general than commitment protocols in distributed computing and databases, such as *two-phase commit* [12, pp. 562–573]. This is because our underlying notion of commitment is flexible, whereas traditional commitments are rigid and irrevocable. However, because multiagent systems are distributed systems, and commitment protocols are protocols, it is natural that techniques developed in classical computer science will apply here. Accordingly, our technical framework integrates approaches from distributed computing, logics of program, and distributed artificial intelligence.

2.1 Potential Causality

The key idea behind potential causality is that the ordering of events in a distributed system can be determined only with respect to an observer [13]. If event e precedes event f with respect to an observer, then e may *potentially* cause f. The observed precedence suggests the possibility of an information flow from e to f, but without additional knowledge of the internals of the agents, we can't be sure that true causation was involved. It is customary to define a *vector clock* as having an entry for the local time of each communicating agent. A vector v is considered later than a vector u if v is later on some, and not sooner on any, element.

Definition 1 A clock over n agents is an n-ary vector $v = \langle v_1 \ldots v_n \rangle$ of natural numbers. The starting clock is $\vec{0} \triangleq \langle 0 \ldots 0 \rangle$. ∎

Definition 2 Given n-ary vectors u and v, $u \prec v$ if and only if $(\forall i : 1 \leq i \leq n : u_i \leq v_i)$ and $(\exists i : 1 \leq i \leq n : u_i < v_i)$. ∎

Each agent starts at $\vec{0}$. It increments its entry in that vector whenever it performs a local event [15]. It attaches the entire vector as a timestamp to any message it sends out. When an agent receives a message, it updates its vector clock to be the element-wise max of its previous vector and the vector timestamp of the message it received. Intuitively, the message brings news of how far the system has progressed; for some agents, the recipient may have better news already. However, any message it sends after this receive event will have a later timestamp than the message just received. The following assumes that $timestamp$ and $content$ of each message are defined.

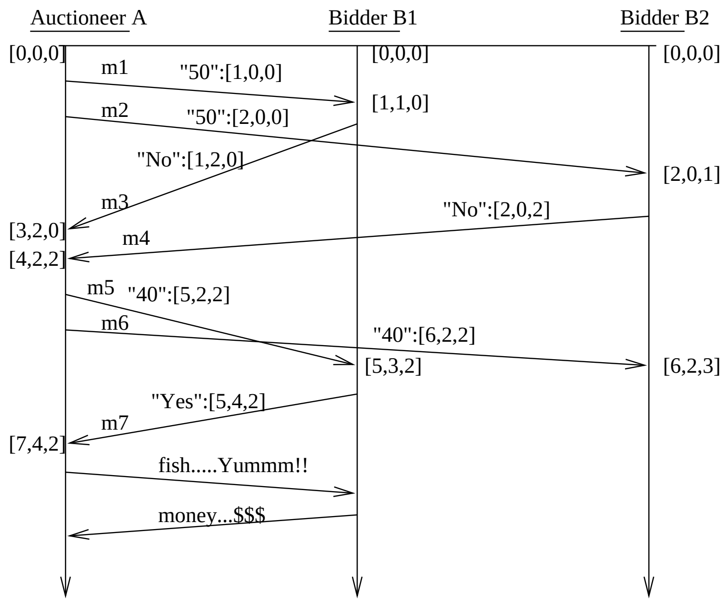

Figure 1: Vector Clocks in the Fish-Market Protocol

Example 3 Figure 1 illustrates the evolution of vector timestamps for one possible run of the fish-market protocol. In the run described here, the auctioneer (A) announces a price of 50 for a certain bucket of fish. Bidders B1 and B2 both decline. A lowers the price to 40 and announces it. This time B1 says *Yes*, leading A to transfer the fish to B1 and B1 to send money to A. For uniformity, the last two steps are also modeled as communications. The messages are labeled m_i to facilitate reference from the text. ▮

2.2 Temporal Logic

The progression of events, which is inherent in the execution of any protocol, suggests the need for representing and reasoning about time. Temporal logics provide a well-understood means of doing so, and have been applied in various subareas of computer science. Because of their naturalness in expressing properties of systems that may evolve in more than one possible way and for the efficiency of reasoning that they support, the branching-time logics have been especially popular in this regard [10]. Of these, the best known is Computation Tree Logic (CTL), which we adapt here in our formal language $\mathcal{L}$. Conventionally, a model of CTL is expressed as a tree. Each node in the tree is associated with a state of the system being considered; the branches of the tree or *paths* thus indicate the possible courses of events or ways in which the system's state may evolve. CTL provides a natural means by which to specify acceptable behaviors of the system.

The following Backus-Naur Form (BNF) grammar with a distinguished start symbol *L* gives the syntax of $\mathcal{L}$. $\mathcal{L}$ is based on a set Φ of atomic propositions. Below, *slant* typeface indicates nonterminals; $\longrightarrow$ and | are metasymbols of BNF specification; $\ll$ and $\gg$ delimit comments; the remaining symbols are terminals. As is customary in formal semantics, we are only concerned with abstract syntax.

L1. $L \longrightarrow$ *Prop* $\ll$atomic propositions: members of Φ $\gg$

L2. $L \longrightarrow \neg\, L$ $\ll$negation$\gg$

L3. $L \longrightarrow L \wedge L$ $\ll$conjunction$\gg$

L4. $L \longrightarrow \mathsf{A}\ P$ $\ll$universal quantification over paths$\gg$

L5. $L \longrightarrow \mathsf{E}\ P$ $\ll$existential quantification over paths$\gg$

L6. $P \longrightarrow L\ \mathsf{U}\ L$ $\ll$until: operator over a single path$\gg$

The meanings of formulas generated from L are given relative to a model and a state in the model. The meanings of formulas generated from P are given relative to a path and a state on the path. The boolean operators are standard. Useful abbreviations include $\mathsf{false} \equiv (p \wedge \neg p)$, for any $p \in \Phi$, $\mathsf{true} \equiv \neg\mathsf{false}$, $p \vee q \equiv \neg p \wedge \neg q$ and $p \to q \equiv \neg p \vee q$. The temporal operators A and E are quantifiers over paths. Informally, $p\mathsf{U}q$ means that on a given path from the given state, q will eventually hold and p will hold until q holds. $\mathsf{F}q$ means "eventually q" and abbreviates $\mathsf{true}\mathsf{U}q$. $\mathsf{G}q$ means "always q" and abbreviates $\neg\mathsf{F}\neg q$. Therefore, $\mathsf{E}p\mathsf{U}q$ means that on some future path from the given state, q will eventually hold and p will hold until q holds.

$M = \langle \mathbf{S}, <, \mathbf{I} \rangle$ is a formal model for $\mathcal{L}$. $\mathbf{S}$ is a set of states; $<\subseteq S \times S$ is a partial order indicating branching time, and $\mathbf{I} : \mathbf{S} \mapsto \Phi$ is an interpretation, which tells us which atomic propositions are true in a given state. For $t \in \mathbf{S}$, $\mathbf{P}_t$ is the set of paths emanating from t. $M \models_t p$ expresses "M satisfies p at t" and $M \models_{P,t} p$ expresses "M satisfies p at t along path P."

M1. $M \models_t \psi$ iff $\psi \in \mathbf{I}(t)$, where $\psi \in \Phi$

M2. $M \models_t p \wedge q$ iff $M \models_t p$ and $M \models_t q$

M3. $M \models_t \neg p$ iff $M \not\models_t p$

M4. $M \models_t \mathsf{A}p$ iff $(\forall P : P \in \mathbf{P}_t \Rightarrow M \models_{P,t} p)$

M5. $M \models_t \mathsf{E}p$ iff $(\exists P : P \in \mathbf{P}_t$ and $M \models_{P,t} p)$

M6. $M \models_{P,t} p\mathsf{U}q$ iff $(\exists t' : t \leq t'$ and $M \models_{P,t'} q$ and $(\forall t'' : t \leq t'' \leq t' \Rightarrow M \models_{P,t''} p))$

The above is an abstract semantics. In Section 3.3, we specify the concrete form of Φ, $\mathbf{S}$, $<$, and $\mathbf{I}$, and can be used in our computations.

3 Approach

In their generic forms, both causality and temporal logic are well-known. However, applying them in combination and in the particular manner suggested here is novel to this paper.

Temporal logic model checking is usually applied for design-time reasoning [10, pp. 1042–1046]. We are given a specification and an implementation, i.e., program, that is supposed to meet it. A model is generated from the program. A model checking algorithm determines whether the specification is true in the generated model. However, in an open, heterogeneous environment, a design may not be available at all. For example, the vendors who supply the agents may consider their designs to be trade secrets.

By contrast, ours is a run-time approach, and can meaningfully apply model checking even in open settings. This is because it uses a model generated from the joint executions of the agents involved. Model checking in this context simply determines whether the present execution satisfies the specification. Model checking of this form may be applied by any observer in the multiagent system. A useful case is when the observer is one of the participating agents. Another useful case is when the observer is some agent dedicated to the task of managing or auditing the interactions of some of the agents in the multiagent system.

Potential causality is most often applied in distributed systems to ensure that the messages being sent in a system satisfy causal ordering [3]. Causality motivates vector clocks and vector timestamps on messages, which help ensure correct ordering by having the messaging subsystem reorder and retransmit messages as needed. This application of causality can be important, but is controversial [4, 6], because its overhead may not always be justifiable.

By contrast, in our approach, the delivery of messages may be noncausal. However, causality serves the important purpose of yielding accurate models of the observations of each agent. These are needed, because in a distributed system, the global model is not appropriate. Creating a monolithic model of the execution of the entire system requires imposing a central authority through which all messages are routed. Adding such an authority would take away many of the advantages that make distributed systems attractive in the first place. Consequently, our method of constructing and reasoning with models should

- not require a centralized message router
- work from a single vantage of observation, but can naturally consider the cases where some agents pool evidence.

Such a method turns out to naturally employ the notion of potential causality.

3.1 Models from Observations

The observations made by each agent are essentially a record of the messages it has sent or received. Since each message is given a vector timestamp, the observations can be partially ordered. In general, this order is not total, because messages received from different agents may be mutually unordered.

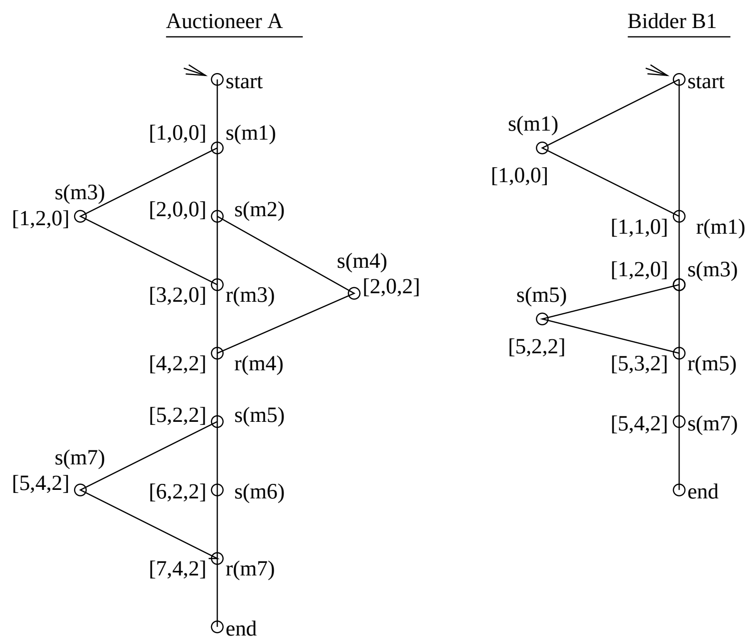

Figure 2: Observations for Auctioneer and a Bidder in the Fish-Market Protocol

Example 4 Figure 2 shows the models constructed locally from the observations of the auctioneer and a bidder in the run of Example 3. ▮

Although a straightforward application of causality, the above shows how the local models may be constructed. Some subtleties are discussed next.

As remarked above, commitments give the real meaning of a protocol. Our approach builds on a flexible and powerful variety of social commitments, which are the commitments of one agent to another [19]. These commitments are defined relative to a *context*, which is typically the multiagent system itself. The *debtor* refers to the agent that makes a commitment, and the *creditor* to the agent who receives the commitment. Thus we have the following logical form.

Definition 3 A commitment is an expression $\mathsf{C}(x, y, p, G)$, where x is the debtor, y the creditor, G the context, and p the condition committed to. ▮

Definition 4 A commitment $c = \mathsf{C}(x, y, p, G)$ is *base-level* if p does not refer to any other commitments; c is a *metacommitment* if p refers to a base-level commitment (we do not consider higher-order commitments here). ∎

Intuitively, a protocol definition is a set of metacommitments for the different roles (along with a mapping of the message tokens to operations on commitments). In combination with what the agents communicate, these lead to base-level commitments being created or manipulated, which is primarily how a commitment may be referred to within a protocol. The violation of a base-level commitment can give us proof or a "smoking gun" that an agent is noncompliant.

The following *operations* on commitments define how they may be created or manipulated. Each operation is realized through a simple message pattern, which we describe. Following the specified patterns ensures that the local models have the information necessary for testing compliance.

O1. *Create* instantiates a commitment; it is typically performed as a consequence of the (new) debtor promising something contractually or by exercising a metacommitment. *Create* requires a message from the debtor to the creditor.

O2. *Discharge* satisfies the commitment; it is performed by the debtor concurrently with the actions that lead to the given condition being satisfied, e.g., the delivery of promised goods or funds. We model the discharge as a single message from the debtor to the creditor.

O3. *Cancel* revokes the commitment. It can be performed by the debtor as a single message. Depending on the existing metacommitments, the cancel of one commitment may simultaneously be the create of another.

O4. *Release* essentially eliminates the commitment. This is distinguished from both *discharge* and *cancel*, because *release* does not mean success or failure, although it lets the debtor off the hook. The *release* action may be performed by the context or the creditor of the given commitment.

O5. *Delegate* shifts the role of debtor to another agent within the same context, and can be performed by the new debtor or the context. However, to prevent the risk of miscommunication, we require the old debtor to also be involved in the message pattern, as shown in Figure 3.

O6. *Assign* transfers a commitment to another creditor within the same context, and can be performed by the present creditor or the context. Here we require that the new creditor and the debtor are also involved as shown in Figure 3. The figure shows only the general pattern. A potential situation is if A discharges the commitment even as

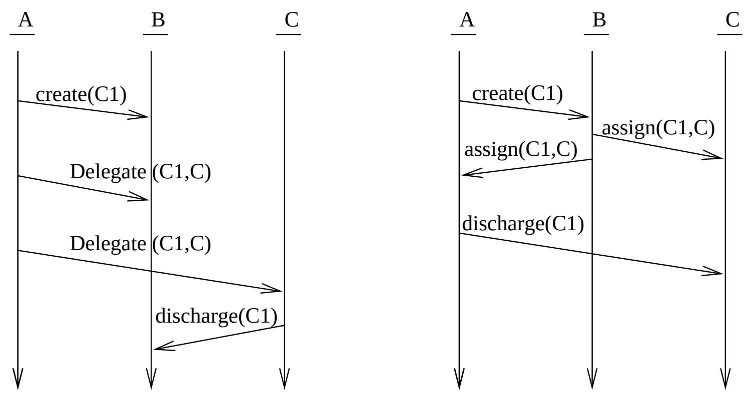

Figure 3: Message Pattern for Delegate (l) and Assign (r)

B is assigning it to C. In this case, we require B to discharge the commitment to C—essentially by forwarding the contents of the message from A.

For simplicity, we write the operations as propositions indicating successful execution, and we don't consider actions performed by the context.

Definition 5 A commitment c, once created, is *resolved* through a *release* or *discharge*, or through the resolution of any commitments created by the *cancel*, *delegate*, or *assign* of c. ∎

Theorem 1 essentially states that the creation and resolution of a commitment occur along the same path. This legitimizes significant optimization below. Theorem 2 states that the creditor knows the disposition of any commitments due to it. This result helps establish that the creditor, or anyone with the same information, can always determine compliance of others relative to what was due to it.

Theorem 1 If message m_i creates commitment c and message m_j resolves c, then $m_i \prec m_j$.
Proof. By inspection of the message patterns constructed for the various operations on commitments. ∎

Theorem 2 If message m_i creates commitment c and message m_j resolves c, then the creditor of c sees both m_i and m_j.
Proof. By inspection of the message patterns constructed for the various operations on commitments. ∎

3.2 Specifying Protocols

We first consider the coordination and then the commitment aspects of compliance. A *skeleton* is a coarse description of how an agent may behave [17]. A skeleton would be associated with each role in the given multiagent system to specify how an agent playing that role may behave in order to coordinate with others. Coordination includes the simpler aspects of interaction, e.g., turn-taking. Coordination is required so that agents' commitments make sense. For instance, a bidder shouldn't make a bid prior to the advertisement, or the commitment content of the bid won't be fully defined.

The skeletons may be constructed by introspection or through the use of a suitable methodology [18]. No matter how they are created, the skeletons are the first line of compliance testing, because an agent that doesn't comply with the skeleton for its role is automatically in violation. So as to concentrate on commitments, we postulate that a "proxy" is interposed between an agent and the rest of the system and ensures that the agent follows the dictates of the skeleton of its role.

We now define the syntax of the specification language simply by extending that of $\mathcal{L}$ with respect to atomic propositions.

L7. *Prop* $\longrightarrow$ *BaseAct* $|$ *Meta* $\ll$only talk about commitments here$\gg$

L8. *Meta* $\longrightarrow$ C(*Debtor*, *Creditor*, *Context*, AG[*BoolAct* $\rightarrow$ AF*BaseAct*])

L9. *BoolAct* $\longrightarrow$ $\ll$Boolean combinations of$\gg$ *Action*

L10. *Action* $\longrightarrow$ *BaseAct* $|$ *OtherCommAct* $\ll$e.g., a request$\gg$

L11. *BaseAct* $\longrightarrow$ *Operation*(C(*Debtor*, *Creditor*, *Context*, *Dom*))

L12. *Operation* $\longrightarrow$ $\ll$the six operations of Section 3.1$\gg$

L13. *Dom* $\longrightarrow$ *OtherExpression* $\ll$domain-specific concepts$\gg$

For the above syntax to work properly, we need additional restrictions on how it is processed. We assume for simplicity that *discharge* actions are *detached*, meaning that the proposition of the commitment being discharged can be treated as `true`. Similarly, the $content$ of a message is not only the direct action it connotes, but also the implied actions caused by the discharge of the applicable metacommitments. Thus, if A has a metacommitment that it will honor B's bid, then B's bid will create the commitment to honor it. Example 5 applies the above language on the fish-market.

Example 5 The messages in Figure 1 can be given a content based on the following. Here FM refers to the fish-market context. We are given *Dom* propositions $fish$—meaning the fish is delivered and $money_i$—meaning that the appropriate money is paid: subscripted to allow different prices. $Bid_i(B_j)$ abbreviates $create(B_j, \mathsf{C}(B_j, A, FM, \mathsf{AG}[fish \rightarrow create(B_j, \mathsf{C}(B_j, A, FM, \mathsf{AF}money_i))]))$—meaning the bidder promises to pay $money_i$ if given the fish.

- Paying up: $discharge(B_1, \mathsf{C}(B_1, A, FM, money_i))$
- Delivering fish: $discharge(A, \mathsf{C}(A, B_1, FM, fish))$
- Yes from B_j (for $money_i$): $Bid_i(B_j)$
- Advertise (to B_1): $create(A, \mathsf{C}(A, B_1, FM, \mathsf{AG}[Bid_i(B_1) \wedge \neg Bid_i(B_2) \rightarrow create(A, \mathsf{C}(A, B_1, FM, \mathsf{AF}fish))]))$
- Bad fish: $release(A, \mathsf{C}(B_1, A, FM, money_i))$

In this scheme, the *No* messages have no significance on commitments. They serve only to assist in the coordination so the auctioneer can determine if enough bids are received. Coordination is not being studied in this paper. Notice that the seller can go away or adjust the price in any direction if a unique *Yes* is not received for the current price $money_i$. It would neither be rational for the auctioneer to raise the price if there are no takers at the present price, nor to lower the price if takers are available, but the protocol *per se* does not legislate against either behavior. ▮

3.3 Reasoning with the Concrete Model

Now we explain the main reasoning steps in our approach and show why they are sound. The main reasoning with models applies the CTL model-checking algorithm on a model and a formula denoting the conjunction of the specifications. The algorithm evaluates whether the formula holds in the initial state of the model. Thus a concrete version of the model M (see Section 2.2) is essential. Various such models are possible. For the purposes of the semantics, we must define a global model with respect to which commitment protocols may be specified. Intuitively, a protocol specification tells us which behaviors of the entire system are correct. Thus, it corresponds naturally to a global model in which those behaviors can be defined.

Our specific concrete model identifies states with messages. The states are ordered according to the timestamps. The proposition true in a state is the one corresponding to the operation that is performed by the message.

Definition 6 $\mathbf{Q} = \{m : m \text{ is a message}\} \cup \{\vec{0}\}$ ▮

Definition 7 For $s, t \in \mathbf{Q}$, $s < t$ iff $timestamp(s) \prec timestamp(t)$ ▮

Definition 8 For $s \in \mathbf{Q}$, $\mathbf{I}(s) = \{content(s)\}$ ▮

The structure $M_Q = \langle \mathbf{Q}, <, \mathbf{I} \rangle$ is a *quasimodel*. (Here and below, we assume that $<$ and $\mathbf{I}$ are appropriately projected to the available states.) M_Q is not a model, because the branches in it are concurrent events and do not individually correspond to a single path. A quasimodel can be mapped to a model, $M_S = \langle \mathbf{S}, <, \mathbf{I} \rangle$ with an initial state $\vec{0}$, by including all possible interleavings of the transitions. However, there is potentially an exponential blowup in the size of the resulting model.

Theorem 3 shows that naively treating a quasimodel as if it were a model is correct. Thus, the above blowup can be eliminated entirely. Our construction ensures that all the events relevant to another event are totally ordered with respect to each other. Notice that, as showing in Figure 3, the construction may appear to require one more message than necessary for the *assign* and *delegate* operations. This linear amount of extra work (for the entire set of messages), however, pays off in reducing the complexity of our reasoning algorithm. In the following, p refers to a metacommitment.

Theorem 3 $M_Q \models_{\vec{0}} p$ iff $M_S \models_{\vec{0}} p$.
Proof. From Theorem 1 and the restricted syntax of metacommitments. ■

The above results show that compliance can be tested and without blowing up the model unnecessarily. However, we would like to test for compliance based on local information—so that any agent can decide for itself whether it has been wronged by another. For this reason, we would like to be able to project the global model onto local models for each agent, while ensuring that the local models carry enough information so they are indeed usable in isolation from other local models. Accordingly, we can define the construction of local models corresponding to an agent's observations. This is simply by defining a subset of $\mathbf{S}$ for a given agent a.

Definition 9 $\mathbf{S}_a = \{m : m \text{ is a message from or to a}\}$. $M_a = \langle \mathbf{S}_a, <, \mathbf{I} \rangle$. ▮

Theorem 4 shows that the projected quasimodel never yields false conclusions relative to the global quasimodel. Theorem 5 shows that it yields all the correct conclusions relative to the global quasimodel about the commitments for which the given agent is creditor. Thus, if the interested party is vigilant, it can check if anyone else violated the protocol.

Theorem 4 $M_a \models_{\vec{0}} p$ only if $M_Q \models_{\vec{0}} p$.
Proof. From Theorem 1 and the fact that metacommitments include AG expressions, so truth in a fragment entails truth in the entire model. ■

Theorem 5 $M_a \models_{\vec{0}} p$ if $M_Q \models_{\vec{0}} p$, provided a is creditor of all commitments mentioned in p.
Proof. From Theorem 2 and the construction of M_a. ■

Example 6 If one of the bidder backs down from a successful bid, the auctioneer immediately can establish that he is cheating, because the auctioneer is the creditor for the bidder's commitment. However, a bidder cannot ordinarily decide whether the auctioneer is noncompliant, because he is not the creditor for commitments under which which the auctioneer may cancel commitments to him. ∎

Theorem 6 lifts the above results to sets of agents. Thus, a set of agents may pool their evidence in order to establish whether a third party is noncompliant. Thus, in a setting with two bidders, a model that includes all their evidence can be used to determine whether the auctioneer is noncompliant. Ordinarily, the bidders would have to explicitly pool their information to do so. However, in a broadcast-based protocol (like a traditional fish-market in which everyone is screaming), the larger model can be built by anyone who hears all the messages. Let A be a set of agents.

Definition 10 $\mathbf{S}_A = \bigcup_{a \in A} \mathbf{S}_a$. $M_A = \langle \mathbf{S}_A, <, \mathbf{I} \rangle$. ∎

Theorem 6 Let A include the creditors for all the commitments in p. Then $M_A \models_{\vec{0}} p$ iff $M_Q \models_{\vec{0}} p$.
Proof. From Theorem 2 and the construction of M_A. ∎

Information about commitments that have been resolved, i.e., are not pending, is not needed in the algorithm, and can be safely deleted from each observer's model. This is accomplished by searching backward in time whenever something is added to the model. Pruning extraneous messages from each observer's model reduces the size of the model and facilitates reasoning about it. This simplification is sound, because the CTL specifications do not include nested commitments.

The above approach loses some of the agents' knowledge, but has all the details we need to assess compliance. Mapping from an event-based to a state-based representation, we should consider every event as potentially corresponding to a state change. This approach would lead to a large model, which accommodates not only the occurrence of public events such as message transmissions, but also local events. Such an approach would thus capture the evolution of the agent's knowledge about the progress of the system, which would help in handling unreliable messaging.

4 Discussion

The compliance checking procedure can be used by any agent who participates in, or observes, a commitment protocol. There are two obvious uses. One, the agent can track which of the commitments made by others are pending. Two, it might track which of its own commitments are pending or whose satisfaction has not been acknowledged by others.

Given the autonomy and heterogeneity of agents, the most natural way to treat interactions is as communications. A communication protocol involves the exchange of messages with a streamlined set of tokens. Traditionally, these tokens are not given any meaning except through reference to the beliefs or intentions of the communicating agents. By contrast, our approach assigns *public*, i.e., observable, meanings in terms of social commitments. Viewed in this light, *every communication protocol is a commitment protocol*.

Formulating and testing compliance of autonomous and heterogeneous agents is a key prerequisite for the effective application of multiagent systems in open environments. As asserted by Chiariglione, minimal specifications based on external behavior will maximize interoperability [7]. The research community has not paid sufficient attention to this issue. A glaring shortcoming of the present semantics for agent communication languages is their fundamental inability to allow testing for the compliance of an agent [20, 22]. The present approach shows how that might be carried out.

Some of the important strands of research of relevance to commitment protocols have been carried out before. However, the synthesis and application of these techniques on multiagent commitment protocols is a novel contribution of this paper. Interaction (rightly) continues to draw much attention from researchers. However, most current approaches do not consider an explicit execution architecture as in [8, 9, 17]. Other approaches lack a formal underpinning; still others focus primarily on monolithic finite-state machine representations for protocols. Such representations can capture only the lowest levels of a multiagent interaction, and their monolithicity does not accord well with distributed execution and compliance testing. Model checking has recently drawn much attention in the multiagent community, e.g., [2, 16]. However, these approaches consider knowledge and related concepts and are thus not directly applicable for behavior-based compliance.

The present approach highlights the synergies between distributed computing and multiagent systems. Since both fields have advanced in different directions, a number of important technical problems can be addressed by their proper synthesis. One aspect relates to situations where the agents may suffer a Byzantine failure or act maliciously. Such agents may fake messages or deny receiving them. How can they be detected by the other agents? Another aspect is to capture additional structural properties of the interactions so that noncompliant agents can be more readily detected. These properties would accompany enhancements and refinements of the specification language so that additional efficiencies may be obtained. Alternatively, we might offer an assistance to designers by synthesizing skeletons of agents who participate properly in commitment protocols. Lastly, it is well-known that there can be far more potential causes than real causes [15]. Can we analyze conversations or place additional, but reasonable, restrictions on the agents that would help focus their interactions on the true relationships between their respective computations? We defer these topics to future research.

Acknowledgments

This work is supported by the National Science Foundation under grants IIS-9529179 and IIS-9624425, and IBM corporation. We are indebted to Feng Wan and Sudhir Rustogi for useful discussions.

References

[1] Gul A. Agha and Nadeem Jamali. Concurrent programming for distributed artificial intelligence. In *[21]*, chapter 12, pages 505–534. 1998.

[2] Massimo Benerecetti, Fausto Giunchiglia, and Luciano Serafini. Model checking multiagent systems. *Journal of Logic and Computation*, 8(3):401–423, June 1998.

[3] Kenneth P. Birman. The process group approach to reliable distributed computing. *Communications of the ACM*, 36(12):36–53, December 1993.

[4] Kenneth P. Birman. A response to Cheriton and Skeen's criticism of causal and totally ordered communication. *Operating Systems Review*, 28(1):11–21, 1994.

[5] Nicholas Carriero and David Gelernter. Coordination languages and their significance. *Communications of the ACM*, 35(2):97–107, February 1992.

[6] David R. Cheriton and Dale Skeen. Understanding the limitations of causally and totally ordered communication. In *Proceedings of the 14th ACM Symposium on Operating System Principles (SOSP)*, pages 44–57. ACM Press, December 1993.

[7] Leonardo Chiariglione. Foundation for intelligent physical agents (FIPA) scope, 1998. http://drogo.cselt.stet.it/fipa/fipa_scope.htm.

[8] Paolo Ciancarini, Andreas Knoche, Robert Tolksdorf, and Fabio Vitali. PageSpace: An architecture to coordinate distributed applications on the web. *Computer Networks and ISDN System*, 28(7–11):941–952, 1996.

[9] Paolo Ciancarini, Robert Tolksdorf, Fabio Vitali, Davide Rossi, and Andreas Knoche. Coordinating multiagent applications on the WWW: A reference architecture. *IEEE Transactions on Software Engineering*, 24(5):362–375, May 1998.

[10] E. Allen Emerson. Temporal and modal logic. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, vol. B, pages 995–1072. North-Holland, Amsterdam, 1990.

[11] Nissim Francez and Ira R. Forman. *Interacting Processes: A Multiparty Approach to Coordinated Distributed Programming*. ACM Press, New York, 1996.

[12] Jim Gray and Andreas Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, San Mateo, 1993.

[13] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.

[14] Juan A. Rodríguez-Aguilar, Francisco J. Martín, Pablo Noriega, Pere Garcia, and Carles Sierra. Towards a test-bed for trading agents in electronic auction markets. *AI Communications*, 11(1):5–19, 1998.

[15] Reinhard Schwarz and Friedemann Mattern. Detecting causal relationships in distributed computations: In search of the holy grail. *Distributed Computing*, 7(3):149–174, 1994.

[16] Munindar P. Singh. Applying the mu-calculus in planning and reasoning about action. *Journal of Logic and Computation*, 8(3):425–445, June 1998.

[17] Munindar P. Singh. A customizable coordination service for autonomous agents. In *Intelligent Agents IV: Proceedings of the 4th International Workshop on Agent Theories, Architectures, and Languages (ATAL-97)*, pages 93–106. Springer-Verlag, 1998.

[18] Munindar P. Singh. Developing formal specifications to coordinate heterogeneous autonomous agents. In *Proceedings of the 3rd International Conference on Multiagent Systems (ICMAS)*, pages 261–268. IEEE Computer Society Press, July 1998.

[19] Munindar P. Singh. An ontology for commitments in multiagent systems: Toward a unification of normative concepts. *Artificial Intelligence and Law*, 1998. In press.

[20] Munindar P. Singh. Principles of agent communication. *IEEE Computer*, 1998. In press.

[21] Gerhard Weiß, editor. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. MIT Press, Cambridge, MA, 1998.

[22] Michael J. Wooldridge. Verifiable semantics for agent communication languages. In *Proceedings of the 3rd International Conference on Multiagent Systems (ICMAS)*, pages 349–356. IEEE Computer Society Press, July 1998.