Towards Design Tools for Protocol Development

Pınar Yolum
Department of Computer Engineering
Boğaziçi University
TR-34342 Bebek, Istanbul, Turkey
pinar.yolum@boun.edu.tr

ABSTRACT

Interaction protocols enable agents to communicate with each other effectively. Whereas several approaches exist to specify interaction protocols, none of them has design tools that can help protocol designers catch semantical protocol errors at design time. As research in networking protocols has shown, flawed specifications of protocols can have disastrous consequences. Hence, it is crucial to systematically analyze protocols in time to ensure correct specification. This paper studies and formalizes important generic properties of commitment protocols that can ease their correct development significantly. Since these properties are formal, they can easily be incorporated in a software tool to (semi-)automate the design and specification of commitment protocols. Where appropriate we provide algorithms that can directly be used to check these properties in such a design tool.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence; D.2.2 [**Software Engineering**]: Design Tools and Techniques

General Terms

Design, Verification

Keywords

Commitments, protocols, tools

1. INTRODUCTION

Multiagent systems consist of autonomous, interacting agents. For the agents to interact effectively, their interactions should be regulated. Multiagent interaction protocols provide a formal ground for realizing this regulation. However, developing effective protocols that will be carried out by autonomous agents is challenging [12, 13].

Similar to the protocols in traditional systems, multiagent protocols need to be specified rigorously so that the agents can interact successfully. Some important properties of network protocols have been studied before, where a protocol was represented as a finite state machine (FSM) [11, 10]. However, FSMs are not well-suited for dynamic environments of multiagent systems [16, 3, 20]. Contrary to the protocols in static systems, multiagent protocols need to be specified flexibly so that the agents can exercise their autonomy by making choices or by dealing with exceptions as best suits them.

Recently, social constructs are being used to specify agent interactions. These approaches advocate declarative representations of protocols and give semantics to protocol messages in terms of social (and thus observable) concepts. Alberti *et al.* specify interaction protocols using social integrity constraints and reason about the expectations of agents [1]. Fornara and Colombetti base the semantics of agent communication on commitments, such that the meanings of messages are denoted by commitments [9]. Yolum and Singh develop a methodology for specifying protocols wherein protocols capture the possible interactions of the agents in terms of the commitments to one another [20, 21].

In addition to providing flexibility, these approaches make it possible to verify compliance of agents to a given protocol. Put broadly, commitments of the agents can be stored publicly and agents that do not fulfill their commitments at the end of the protocol can be identified as non-compliant. In order for these approaches to make use of all these advantages, the protocols should be designed rigorously. For example, the protocol should guarantee that, if an agent does not fulfill its commitment, it is not because the protocol does not specify how the fulfillment can be carried out. The aforementioned approaches all start with a manually designed, correct protocol. However, designing a correct protocol in the first place requires important correctness properties to be established and applied to the protocol. A correct protocol should define the necessary actions (or transitions) to lead a computation to its desired state. Following a protocol should imply that progress is being made towards realizing desired end conditions of the protocol. The followed actions should not yield conflicting information and lead the protocol to unrecoverable errors. That is, the protocol should at least allow a safe execution.

This paper develops and formalizes design requirements for developing correct and consistent *commitment protocols* [19, 20]. However, the underlying ideas are generic and can be applied to other social approaches as well. These requirements detect inconsistencies as well as errors during design time. These requirements can easily be automated in a design tool to help protocol designers to develop protocols.

The rest of the paper is organized as follows. Section 2 gives a technical background on event calculus and commitments. Section 3 reviews commitment protocols. Sections 4 and 5 develop correctness and consistency requirements, respectively. Section 6 shows how these requirements can be implemented in a design tool. Section 7 discusses the recent literature in relation to our work.

2. TECHNICAL BACKGROUND

We first give a brief overview of event calculus, which we use to formalize the design requirements. Next, we summarize Yolum and Singh's formalization of commitments and their operations.

2.1 Event Calculus

The event calculus (EC) is a formalism based on many-sorted first order logic [14]. The three sorts of event calculus are *time points* (T), *events* (E) and *fluents* (F). Fluents are properties whose truth values can change over time. Fluents are manipulated by initiation and termination of events. Table 1 supplies a list of predicates to help reason about the events in an easier form. Below, events are shown with $a, b, \ldots$; fluents are shown with $f, g, \ldots$; and time points are shown with t, t_1, and t_2.

Table 1: Event calculus predicates

Initiates(a, f, t)	f holds after event a at time t.
Terminates(a, f, t)	f does not hold after event a at time t.
Initially$_P(f)$	f holds at time 0.
Initially$_N(f)$	f does not hold at time 0.
Happens(a, t_1, t_2)	event a starts at time t_1 and ends at t_2.
Happens(a, t)	event a starts and ends at time t.
HoldsAt(f, t)	f holds at time t.
Clipped(t_1, f, t_2)	f is terminated between t_1 and t_2.
Declipped(t_1, f, t_2)	f is initiated between t_1 and t_2.

We introduce the subset of the EC axioms that are used here; the rest can be found elsewhere [17]. The variables that are not explicitly quantified are assumed to be universally quantified. The standard operators apply (i.e., $\leftarrow$ denotes implication and $\wedge$ denotes conjunction). The time points are ordered by the $<$ relation, which is defined to be transitive and asymmetric.

1. $\mathit{HoldsAt}(f, t_3) \leftarrow \mathit{Happens}(a, t_1, t_2) \wedge \mathit{Initiates}(a, f, t_1) \wedge (t_2 < t_3) \wedge \neg\, \mathit{Clipped}(t_1, f, t_3)$
2. $\mathit{Clipped}(t_1, f, t_4) \leftrightarrow \exists a, t_2, t_3\ [\mathit{Happens}(a, t_2, t_3) \wedge (t_1 < t_2) \wedge (t_3 < t_4) \wedge \mathit{Terminates}(a, f, t_2)]$
3. $\neg \mathit{HoldsAt}(f, t) \leftarrow \mathit{Initially}_N(f) \wedge \neg \mathit{Declipped}(0, f, t)$
4. $\neg \mathit{HoldsAt}(f, t_3) \leftarrow \mathit{Happens}(a, t_1, t_2) \wedge \mathit{Terminates}(a, f, t_1) \wedge (t_2 < t_3) \wedge \neg \mathit{Declipped}(t_1, f, t_3)$

2.2 Commitments

Commitments are obligations from one party to another to bring about a certain condition [4]. A base-level commitment C*(x, y, p)* binds a debtor *x* to a creditor *y* to bring about a condition *p* [18]. When a base-level commitment is created, *x* becomes responsible to *y* for satisfying *p*, i.e., *p* should hold sometime in the future. The condition *p* does not involve other conditions or commitments.

A conditional commitment CC(x, y, p, q) denotes that if the condition *p* is satisfied, *x* will be committed to bring about condition *q*. Conditional commitments are useful when a party wants to commit only if a certain condition holds or only if the other party is also willing to make a commitment. It is easy to see that a base-level commitment is a special case of a conditional commitment, where the condition is set to true. That is, $\mathsf{C}(x, y, p)$ is an abbreviation for $\mathsf{CC}(x, y, true, p)$. Commitments are represented as fluents in the event calculus. Hence, the creation and the manipulation of the commitments are shown with the *Initiates* and *Terminates* predicates.

Compared to the traditional definitions of obligations, commitments can be carried out more flexibly [18]. By performing operations on an existing commitment, a commitment can be manipulated (e.g., delegated to a third-party). We summarize the operations to create and manipulate commitments [18, 20]. In the following discussion, x, y, z denote agents, c, c' denote commitments, and e denotes an event.

1. *Create*$(e, x, \mathsf{C}(x, y, p)$*)*: When x performs the event e, the commitment c is created.

   $\{\mathit{Happens}(e, t) \wedge \mathit{Initiates}(e, \mathsf{C}(x, y, p), t)\}$
2. *Discharge*$(e, x, \mathsf{C}(x, y, p))$: When x performs the event e, the commitment c is resolved.

   $\{\mathit{Happens}(e, t) \wedge \mathit{Initiates}(e, p, t)\}$
3. *Cancel*$(e, x, \mathsf{C}(x, y, p))$: When x performs the event e, the commitment c is canceled. Usually, the cancellation of a commitment is followed by the creation of another commitment to compensate for the former one.

   $\{\mathit{Happens}(e, t) \wedge \mathit{Terminates}(e, \mathsf{C}(x, y, p), t)\}$
4. *Release*$(e, y, \mathsf{C}(x, y, p))$: When y performs the event e, x no longer need to carry out the commitment c.

   $\{\mathit{Happens}(e, t) \wedge \mathit{Terminates}(e, \mathsf{C}(x, y, p), t)\}$
5. *Assign*$(e, y, z, \mathsf{C}(x, y, p)$*)*: When y performs the event e, commitment c is eliminated, and a new commitment c' is created where z is appointed as the new creditor.

   $\{\mathit{Happens}(e, t) \wedge \mathit{Terminates}(e, \mathsf{C}(x, y, p), t) \wedge \mathit{Initiates}(e, \mathsf{C}(x, z, p), t)\}$
6. *Delegate*$(e, x, z, \mathsf{C}(x, y, p)$*)*: When x performs the event e, commitment c is eliminated but a new commitment c' is created where z is the new debtor.

   $\{\mathit{Happens}(e, t) \wedge \mathit{Terminates}(e, \mathsf{C}(x, y, p), t) \wedge \mathit{Initiates}(e, \mathsf{C}(z, y, p), t)\}$

The following rules operationalize the commitments. Axiom 1 states that a commitment is no longer in force if the condition committed to holds. In Axiom 1, when the event e occurs at time t, it initiates the fluent p, thereby discharging the commitment $\mathsf{C}(x, y, p)$.

COMMITMENT AXIOM 1. *Discharge(e, x,* C*(x, y, p))* $\leftarrow$ *HoldsAt(*C*(x, y, p), t)* $\wedge$ *Happens(e, t)* $\wedge$ *Initiates(e, p, t)*

The following axiom captures how a conditional commitment is resolved based on the temporal ordering of the commitments it refers to. When the conditional commitment $\mathsf{CC}(x, y, p, q)$ holds, if p becomes true, then the original commitment is terminated but a new commitment is created, since the debtor x is now committed to bring about q.

COMMITMENT AXIOM 2. *Initiates(e,* C*(x, y, q), t)* $\wedge$ *Terminates(e,* CC*(x, y, p, q), t)* $\leftarrow$
*HoldsAt(*CC*(x, y, p, q), t)* $\wedge$ *Happens(e, t)* $\wedge$ *Initiates(e, p, t)*

3. COMMITMENT PROTOCOLS

A commitment protocol is a set of actions such that each action is either an operation on commitments or brings about a proposition. Agents create and manipulate commitments they are involved in through the protocol they follow. An agent can start a protocol by performing any of the actions that is allowed by the role it is playing. The transitions of the protocol are computed by applying the effect of the action on the current state. In most cases this correspond to the application of commitment operations. Figure 1 gives an overview of the possible transitions. Given a protocol specification, actions of the protocol can be executed from an arbitrary initial state to a desired final state. A protocol run can be viewed as a series of actions; each action happening at a distinct time point.

EXAMPLE 1. *We consider the Contract Net Protocol (CNP) as our running example [8]. CNP starts with a manager requesting proposals for a particular task. Each participant either sends a proposal or a reject message. The manager accepts one proposal among the submitted proposals and (explicitly) rejects the rest. The participant with the accepted proposal informs the manager with the proposal result or the failure of the proposal.*

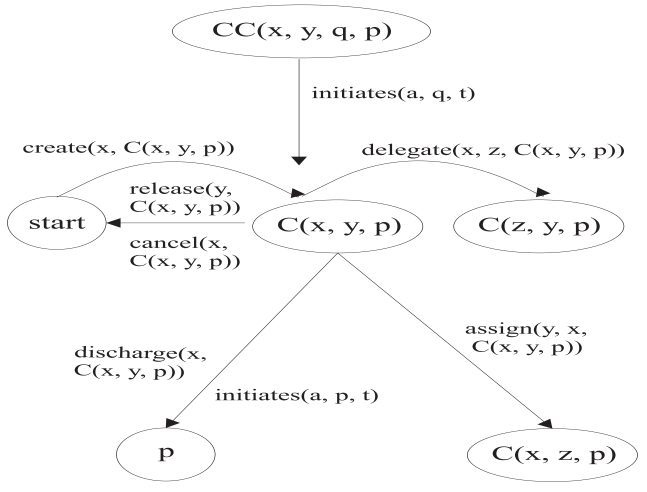

Figure 1: Commitment transitions

EXAMPLE 2. *By sending a proposal to the manager, a participant creates a conditional commitment such that if the manager accepts the proposal, then the participant will deliver the result of the proposal (e.g.,* CC*(participant, manager, accepted, result). If the manager then sends an accept message, this conditional commitment will cease to exist but the following base-level commitment will hold:* C*(participant, manager, result). Since the commitments can be easily manipulated, the participant can manipulate its commitment in the following ways: (1) it can discharge its commitment by sending the result as in the original CNP (*discharge*), (2) it can delegate its commitment to another participant, who carries out the proposal (*delegate*), or (3) it can send a failure notice as in the original protocol (*cancel*). Meanwhile, if for some reason, the manager no longer has a need for the proposed task, (1) it can let go of the participant (*release*) or (2) let another agent benefit from the proposal (*assign*).*

4. PROTOCOL CORRECTNESS

Analysis of commitment protocols poses two major challenges. One, the states of a commitment protocol are not given *a priori* as is the case with FSMs. Two, the transitions are computed at run time to enable flexible execution. To study a commitment protocol, we study the possible protocol runs that can result. A protocol run specifies the actions that happen at certain time points. We base the definition of a protocol state on these time points. More specifically, a state of the protocol corresponds to the set of propositions and commitments that hold at a particular time point in a particular run.

To ease the explanation, we introduce the following notation. Let F be the set of fluents in the protocol. F is $CS \cup CCS \cup PS$ such that CS is the set of base-level commitments, CCS is the set of conditional commitments and PS is the set of propositions in the protocol. Let c be a commitment such that $c \in CS$ then $O(c)$ is the set of operations allowed on the commitment c in the protocol and $O = \{O(c) : c \in CS\}$. Since a commitment cannot be part of a protocol if it cannot be created, we omit the create operation from the set. Hence, $O(c)$ can contain five types of operations in Section 2.2, namely, discharge, cancel, release, delegate, and assign. We assume that all the propositions referred by the commitments in CS and CCS are in PS.

DEFINITION 1. *A protocol state $s(t)$ captures the content of the protocol with respect to a particular time point t. A protocol state $s(t)$ is a conjunction of $HoldsAt(f, t)$ predicates with a fixed t but possibly varying f. Formally, $s(t) \equiv \bigwedge_{f \in F'} HoldsAt(f, t)$ such that $F' \subseteq F$.*

Two states are equivalent if the same fluents hold in both states. Although the two states are equivalent, they are not strictly the same state since they can come about at different time points.

DEFINITION 2. *The $\equiv$ operator defines an equivalence relation between two states $s(t)$ and $s(t')$ such that $s(t) \equiv s(t')$ if and only if $\forall f \in F : (HoldsAt(f, t) \iff HoldsAt(f, t'))$.*

Protocol execution captures a series of operations for making and fulfilling of commitments. Intuitively, if the protocol executes successfully, then there should not be any open base-level commitments; i.e., no participant should still have commitments to others. This motivates the following definition of an end-state.

DEFINITION 3. *A protocol state $s(t)$ is a proper end-state if no base-level commitments exist. Formally, $\forall f \in F : HoldsAt(f, t) \Rightarrow f \notin CS$.*

Generally, if the protocol ends in an unexpected state, i.e., not a proper end-state, one of the participants is not conforming to the protocol. However, to claim this, the protocol has to ensure that participants have the choice to execute actions that will terminate their commitments. The following analysis derives the requirements for correct commitment protocols.

Holzmann labels states of a protocol in terms of their capability of allowing progress [11]. Broadly put, a protocol state can be labeled as a progressing state if it is possible to move to another state. For a protocol to function correctly, all states excluding the proper end-states should be progressing states. Otherwise, the protocol can move to a state where no actions are possible, and hence the protocol will not progress and immaturely end.

DEFINITION 4. *A protocol state $s(t)$ is progressing if both of the following hold:*

- *$s(t)$ is not a proper end-state (e.g., $s(t) \Rightarrow \exists f \in CS : HoldsAt(f, t)$).*
- *there exists an action that if executed creates a transition to a different state. (e.g., $s(t) \Rightarrow \exists t' : t < t' \wedge s(t) \not\equiv s(t')$)* ▌

At every state in the protocol, either the execution should have successfully completed (i.e., proper end-state) or should be moving to a different state (i.e., progressing state).

DEFINITION 5. *A protocol $\mathcal{P}$ is progressive if and only if each possible state in the protocol is either a proper end-state or a progressing state.*

This follows intuitively from the explanation of making progress. Lemma 1 formalizes a sufficient condition for ensuring that a commitment protocol is progressive,

LEMMA 1. *Let $\mathcal{P}$ be a commitment protocol and c be a base-level commitment. If $\forall c \in CS : O(c) \neq \emptyset$, then $\mathcal{P}$ is progressive.*
Proof. *By Definition 5, every state in $\mathcal{P}$ should be a proper end-state or a progressing state. If a state does not contain open commitments then it is a proper end-state (Definition 3). If the state does contain a base-level commitment, then since at least one operation exists to manipulate it, the protocol will allow a transition to a new state. Thus, the state is a progressing state (Definition 4).*

Ensuring a progressing protocol is the first step in ensuring correctness. If a protocol is not progressing, then the participants can get stuck in an unexpected state and not transition to another state. However, progress by itself does not guarantee that the interactions will always lead to a proper end-state. This is similar in principle to livelocks in network protocols, where the protocol can transition between states but never reach a final state [11, p.120].

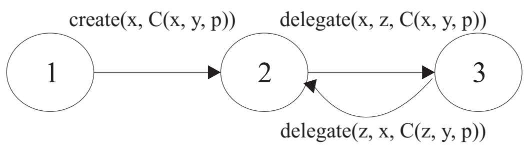

Figure 2: Infinitely delegating a commitment

EXAMPLE 3. *Consider a participant x whose proposal has been accepted (hence,* C*(x, manager, result). Next, the participant delegates its commitment to another participant z (hence,* C*(z, manager, result)). Next, participant z delegates the commitment back to participant x and thus the protocol moves back to the previous state (*C*(x, manager, result)). Participants x and z delegate the commitment back and forth infinitely.*

Obviously, the situation explained in Example 3 is is not desirable. It is necessary to ensure progress but this is not sufficient to conclude that the protocol is making *effective* progress.

DEFINITION 6. *A cycle in a protocol refers to a non-empty sequence of states that start and end at equivalent states. A cycle can be formalized by the content of the beginning and ending states. That is, an execution sequence is a cycle if: $\exists t, t', t'' \in T : (s(t) \equiv s(t')) \wedge (t < t'' < t') \wedge (s(t) \not\equiv s(t''))$.*

DEFINITION 7. *An infinitely repeating cycle is a cycle with progressing states such that if the protocol gets on to one of the states then the only possible next transition is to move to a state in the cycle [11].*

In Example 3, the two delegate actions form an infinitely repeating cycle. Once the protocol gets into either state 2 or state 3, it will always remain in one of these two states.

LEMMA 2. *An infinitely repeating cycle does not contain any proper end-states.*
Proof. *By Definition 7 an infinitely repeating cycle only contains progressing states and by Definition 4, a progressing state cannot be an end-state.*

Given a cycle, it is easy to check if it is infinitely repeating. Informally, for each state in the cycle, we need to check if there is a possible transition that can cause a state outside the cycle. This can be achieved by applying all allowed operations (by the proposition) to the commitments that exist in that state. As soon as applying a commitment operation to a state in the cycle yields a state not included in the cycle, the procedure stops, concluding that the cycle is not infinitely repeating.

LEMMA 3. *Let l be a cycle. Let $c \in CS$ be a commitment that holds at a state $s(t)$ on this cycle at any time t. If* discharge, cancel *or* release $\in O(c)$ *then cycle l is not infinitely repeating.*
Proof. *A cycle is not infinitely repeating if there is a path from a state in the cycle to a state outside the cycle. Discharging, canceling, or releasing a commitment will lead the protocol to go to a proper end-state. Since no proper end-state is on an infinitely repeating cycle, the cycle will not repeat (Lemma 2).*

EXAMPLE 4. *In Example 3, if either participant could discharge the commitment or could have been released from the commitment, then there need not have been an infinitely repeating cycle.*

DEFINITION 8. *A protocol $\mathcal{P}$ is effectively progressive if and only if and only if (1) $\mathcal{P}$ is progressive and (2) $\mathcal{P}$ does not have infinitely repeating cycles.*

THEOREM 1. *$\mathcal{P}$ is an effectively progressive protocol if for any commitment $c \in CS$ either (1)* discharge $\in O(c)$ *or* cancel $\in O(c)$ *or* release $\in O(c)$ *or (2) by applying finite number of operations a commitment c' is reached for which* discharge $\in O(c')$ *or* cancel $\in O(c')$ *or* release $\in O(c')$.
Proof. *In both cases, for all commitments in $\mathcal{P}$, there is at least one operation defined. Hence, by Lemma 1, $\mathcal{P}$ is progressive. Assume that $\mathcal{P}$ has an infinite cycle. By Lemma 3, there has to be a commitment c'' holding in some state on the cycle for which none of the operations lead to a state with* discharge, cancel, *or* release *operators. Since $\mathcal{P}$ does not allow such a state, $\mathcal{P}$ does not contain an infinitely repeating cycles.*

EXAMPLE 5. *The protocol P contains three actions: accept a proposal (*create*(acceptProposal, participant,* C*(participant, manager, proposal))), authorize a subcontractor to carry out the proposal (*delegate*(authorize, participant, subcontractor,* C*(participant, manager, proposal))), and carry out the proposal (*discharge*(carryOut, subcontractor,* C*(subcontractor, manager, proposal))).*

The protocol in Example 5 is effectively progressive since the commitment C(participant, manager, proposal) can be delegated to someone who can apply one of the discharge, cancel, or release operations. An algorithm that checks for an effectively progressive protocol is given in Section 6.

5. PROTOCOL CONSISTENCY

In Section 4 we have defined the requirements to guarantee that a protocol can effectively progress. However, in addition to effective progress, a protocol should always preserve a consistent computation. In other words, a protocol that functions correctly does not allow creation of conflicting information. Following the CNP example, a participant cannot both refuse to send a proposal and send a proposal at the same time. That is, the available information that is created by the protocol should be consistent at every time point of the protocol. To explain the consistency requirements for a commitment protocol, we again start with studying individual states. Since each state is defined in terms of holding commitments and propositions, we start by defining when the commitments and propositions are inconsistent.

DEFINITION 9. *Let p and r be two propositions such that $p, r \in PS$. If p entails the negation of r, that is,* false$\leftarrow HoldsAt(p,t) \wedge HoldsAt(r,t)$ *then p and r are conflicting. A protocol state $s(t)$ is consistent if $s(t) \not\equiv$* false.

Obviously, the protocol should never enter an inconsistent state. The set of operations defined for a commitment should ensure that only consistent states are realized. Notice that we allow two base-level commitments to exist together even if the propositions that need to be brought out by these commitments are conflicting. That is, a state could contain two commitments $C(x,y,p)$ and $C(x,y,r)$ such that p and r are conflicting. Obviously, both commitments cannot be satisfied simultaneously. Hence, discharging one commitment restricts the discharging of the second commitment.

DEFINITION 10. *A protocol $\mathcal{P}$ is consistent if and only if $\mathcal{P}$ is progressive and each possible state in the protocol is consistent.*

LEMMA 4. *Let $\mathcal{P}$ be a commitment protocol and c and c' be two base-level commitments in CS such that c and c' have conflicting propositions. If $O(c) = O(c') = \{$discharge$\}$, then $\mathcal{P}$ is not consistent.*
Proof. *Let $C(x,y,p)$ and $C(x,y,r)$ be any two commitments in CS with conflicting propositions. If either of them is not discharged, then the protocol state will contain a base-level commitment. By Definition 3, it will not be a proper end-state. If both of them discharge, the protocol will move to the* false *state. Thus, by Definition 10, it will not be consistent.*

THEOREM 2. *Let $\mathcal{P}$ be an effectively progressive commitment protocol, and c and c' be two base-level commitments in CS with conflicting propositions. If either* release$\in O(c')$ *or* cancel$\in O(c')$ *then $\mathcal{P}$ is consistent.*
Proof. *If* discharge $\notin O(c)$*, then $\mathcal{P}$ can never move into the* false *state and hence will be consistent. If* discharge $\in O(c)$*, by Lemma 4, c' needs to define an operation other that* discharge *to avoid the* false *state. By Theorem 1, commitment c' should define at least one of* discharge, release, *or* cancel. *Since* discharge *is eliminated by Lemma 4, at least* release, *or* cancel *should be defined.*

EXAMPLE 6. *Assume that a participant commits to send a proposal and at the same time refuses to send a proposal (commits not to send a proposal). Then the participant will not be able to discharge both of its commitments. On the other hand, if the participant can cancel one of its commitment or if the manager releases the participant from one of them, then the protocol can continue consistently.*

6. ALGORITHMS

The results of the previous sections can be implemented in a design tool. This section provides algorithms to compute the derived correctness and consistency requirements of Theorems 1 and 2.

A commitment graph $G = (V, E)$ consists of a set of nodes V and a set of edges E. Each node denotes a single possible base-level commitment in a given protocol. A directed edge between node u to v denotes an operation applied on the commitment at node u, yielding node v. A commitment graph contains two designated nodes, namely RC and D. These nodes do not contain any commitments. RC is used as a sink node for all commitments for which a release or a cancel operation is defined. In other words, if a node u is connected to node RC then the operation on edge (u, RC) could only be a release or a cancel operation (since these operations resolve the commitment, and do not create other commitments). Similarly, node D is a sink node for commitments for which discharge is defined. If a node u is connected to node D then the operation on edge (u, D) could only be a discharge. If there is an edge (u, v) such that v is not the RC or the D node, then the operation associated with the edge is either a delegate or an assign.

Algorithm 1 Build-commitment-graph(CS: Set of base-level commitments; O: Set of operations on base-level commitments)

```
 1: Create a new node $RC$ {$RC$ stands for a sink node for release
    and cancel}
 2: Create a new node $D$ {$D$ stands for a sink node for discharge}
 3: possible-commitments = $CS$
 4: while (possible-commitments ! = $\emptyset$) do
 5:   Remove a commitment $c$
 6:   Add a new node $c$ to $V$
 7:   for $i = 1$ to $|O(c)|$ do
 8:     if (O(c)[i] == delegate) then
 9:       Add a new node $c.delegate$ to $V$
10:       Add $(c, c.delegate)$ to $E$
11:       Add $c.delegate$ to possible-commitments
12:     else if (O(c)[i] == assign) then
13:       Add a new node $c.assign$ to $V$
14:       Add $(c, c.assign)$ to $E$
15:       Add $c.assign$ to possible-commitments
16:     else if (O(c)[i] == release) || (O(c)[i] == cancel) then
17:       Add $(c, RC)$ to $E$
18:     else if (O(c)[i] == discharge) then
19:       Add $(c, D)$ to $E$
20:     end if
21:   end for
22: end while
```

Algorithm 1 takes as input the base-level commitment set CS and operations set O and builds a commitment graph. The algorithm starts by creating the RC and the D nodes. Then, the algorithm iterates over the set of possible commitments that can be created by the protocols ($possible - commitments$) and adds a new node for each commitment. After adding a node for a commitment, it goes through the operations set of the commitment and adds an edge between the node and the RC state for cancel and release operations and an edge between the node and the D state for discharge operation. If there is an assign or a delegate operation, the algorithm applies the operation on the commitment and creates a new node with the resulting commitment. The resulting commitment corresponds to the initiated commitment as explained in Section 2.2. The new commitment is added to the set of possible commitments.

We assume that the graph contains a standard adjacency matrix that can determine if a node has an edge to another node. In the commitment graph, this shows whether applying a single action can transform the commitment either to another commitment or lead it to one of the discharge, cancel, or release states. The $adjacentTo$ method serves this purpose. If a commitment node has at least one outgoing edge, then the commitment is said to have a neighbor (i.e., $hasNeighbors()$ method is true).

Algorithm 2 checks if all the commitments in the commitment graph can be resolved. To do this, it functions like a search algorithm. Algorithm 2 takes as input a commitment graph and visits

Algorithm 2 Color-graph(G:Commitment Graph)

```
1: $visited = \emptyset$
2: $whiteList = \emptyset$
3: $blackList = \emptyset$
4: for $i = 1$ to $|V|$ do
5:    if (V(i) $\notin visited$) then
6:       visit(V(i))
7:    end if
8: end for
```

each node (with Algorithm 3) to color each node. If a node satisfies the properties in Theorem 1, then it is colored white, if not black. The algorithm terminates when all nodes are colored.

Algorithm 3 takes as input the node u that will be visited, goes through the nodes as in depth first search (DFS), and assigns a color. White nodes are stored in the $whiteList$ and the black nodes are stored in the $blackList$. All visited nodes are stored in the $visited$ set. Initially, nodes do not have any color. The node u is first added to the $visited$ set.

If u does not have any outgoing edges, then it is a singleton in the graph and is not connected to the rest of the graph. Hence, the commitment has no operations defined and thus cannot be resolved. Such nodes are labeled as black and put into $blackList$. If the commitment at node u has one of the discharge, cancel, or release operations defined (there is an edge between u and RC or u and D), then the color of the node u becomes white. This means that the protocol allows commitment node u to be resolved. Otherwise, the neighbors of the node u are analyzed. If any one

Algorithm 3 visit(u: node)

```
1: Add $u$ to $visited$
2: if ($u$.adjacentTo($D$ OR $CR$)) then
3:    Add $u$ to $whiteList$
4: else if ($u$.hasNeighbors()) then
5:    while ($u \notin whiteList$) AND ($\exists\, E(u,v)$: $v \notin visited$) do
6:       if ($v \notin$ visited) then
7:          visit($v$)
8:       end if
9:       if ($v \in whitelist$) then
10:         Add $u$ to $whiteList$
11:      else
12:         Add $u$ to $blackList$
13:      end if
14:   end while
15: else
16:   Add $u$ to $blackList$
17: end if
```

neighbor node v is already white, then u is also labeled as white. The intuition is that if the commitment at v can be resolved and if the commitment at u can be transformed (by delegate or assign) to v, then v can be resolved, too. If no neighbor node is already white, then the algorithm visits neighbor nodes that are not already visited. The aim is to find a directed path from the current node to a white node. When a white node is found, then all nodes on the path become white and are inserted into $whiteList$. If a white node cannot be reached by a directed path, then all nodes on the path become black and are added to the $blackList$. Algorithms 2 and 3 are a variant of DFS and thus computes the set of unresolvable commitments in $O(|E|)$ [5]. The protocol designer can modify the protocol until the $blackList$ computed by this algorithm is empty.

Algorithm 4 checks the protocol consistency (Theorem 2). The

Algorithm 4 Check-consistency(G: Commitment Graph)

```
1: $inconsistentList$=$\emptyset$
2: for $i = 1$ to $|V|$-1 do
3:    for $j = i + 1$ to $|V|$ do
4:       Determine if $V(i)$ and $V(j)$ are conflicting
5:       if conflicting($V(i)$ and $V(j)$) then
6:          if ($\nexists\, E(V(i), RC)$) AND ($\nexists\, E(V(j), RC)$) then
7:             Add $V(i)$ and $V(j)$ to $inconsistentList$
8:          end if
9:       end if
10:   end for
11: end for
```

algorithm compares all commitments to each other to see if they have conflicting propositions. If so, the algorithm checks if either of the commitments can be released or canceled.
The $inconsistentList$ keeps the pairs of commitments that fail the test. Algorithm 4 computes the set of inconsistent commitments in $O(|V|^2)$. After this set is computed, a protocol designer can modify the protocol until the set of inconsistent commitments is empty.

7. DISCUSSION

This work derives some design-time requirements for commitment protocols. These requirements are concerned with allowing sufficient actions for agents to carry out their actions. However note that we are not concerned about the choices of the agents in terms of which actions to take. Looking back at Example 3, assume that agent x could also execute an action that could discharge its commitment (to carry out the proposal), but choose instead to delegate it to agent z. The protocol then would still loop infinitely. However, our purpose here is to make sure that agent x has the choice of discharging. The protocol should allow an agent to terminate its commitment by providing at least one appropriate action. It is then up to the agent to either terminate it or delegate it as

The algorithms given in Section 6 can be implemented in a design tool. The design tool should be fed with a description of the protocol, which contains the actions and the commitment operation each action corresponds to as specified in Section 3. The commitment and operation set of the protocol can then be easily formed and fed into Algorithm 1 for creating a commitment graph. Once, there is a commitment graph both Algorithms 2 and 4 can be applied to check correctness and consistency, respectively.

We review the recent literature with respect to our work. Fornara and Colombetti develop a method for agent communication, where the meanings of messages denote commitments [9]. In addition to base-level and conditional commitments, Fornara and Colombetti use precommitments to represent a request for a commitment from a second party. They model the life cycle of commitments in the system through update rules. However, they do not provide design requirements on correctness or consistency as we have done here. The requirements and algorithms developed here can easily be applied to their framework.

Artikis *et al.* develop a framework to specify and animate computational societies [2]. The specification of a society defines the social constraints, social roles, and social states. Social constraints define types of actions and the enforcement policies for these actions. A social state denotes the global state of a society based on the state of the environment, observable states of the involved agents, and states of the institutions. Our definition of a protocol state is similar to the global state of Artikis *et al.*. The framework of Artikis *et al.* does not specify any design rules to establish the correctness of the executed societies. It would be interesting to apply the design ideas here to their setting where in addition there are social constraints.

Dignum *et al.* formalize interaction protocols within an organization through contracts [6]. They develop a language for specifying contracts that can capture various contracts and their deadlines. They use the interaction protocols to realize the objectives of the organization that the agents are situated in. However, they do not provide a methodology for analyzing contracts defined in that language as we have done here.

Alberti *et al.* specify interaction protocols using social integrity constraints [1]. Given a partial set of events that have happened, each agent computes a set of expectations based on the social integrity constraints; e.g., events that are expected to happen based on the given constraints. If an agent executes an event that does not respect an expectation, then it is assumed to have violated one of the social integrity constraints. We have studied the violating of commitments in richer time structure elsewhere [15]. Alberti *et al.* does not provide any design rules to ensure the correctness of their interaction protocols. Since the commitments and their operations are more flexible than the expectations defined by Alberti *et al.*, our requirements can also be applied to their framework.

Endriss *et al.* study protocol conformance for interaction protocols that are defined as deterministic finite automaton (DFA) [7]. The set of transitions of a DFA are known *a priori*. If an agent always follows the transitions of the protocol, then it is compliant to the given protocol. Hence, the compliance checking can be viewed as verifying that the transitions of the protocol are followed correctly.

McBurney and Parsons propose posit spaces protocol to handle e-commerce transactions of agents [16]. The protocol consists of five locutions: propose, accept, delete, suggest_revoke, and ratify_revoke. The usage of propose and accept locution resembles the conditional commitments in commitment protocols. The delete locution corresponds to the release, or discharge operation. Suggest_revoke and ratify_revoke enable canceling of posits. McBurney and Parsons do not provide any design rules to develop posit space protocols as we have done here. The analysis constructed in this paper may be applied in the posit space framework.

In our future work, we plan to work on other design criteria for commitment protocols, such as requirements for avoiding possible deadlocks as well as requirements for conditional commitments.

8. ACKNOWLEDGMENTS

This paper extends an earlier version that appeared in the 19th International Symposium on Computer and Information Sciences (ISCIS-04).

This research has been supported by Boğaziçi University Research Fund under grant BAP05A104. I thank the reviewers whose suggestions have improved the paper considerably.

9. REFERENCES

[1] M. Alberti, D. Daolio, and P. Torroni. Specification and verification of agent interaction protocols in a logic-based system. In *Proceedings of the ACM Symposium on Applied Computing (SAC)*, pages 72–78. ACM Press, Mar. 2004.

[2] A. Artikis, J. Pitt, and M. Sergot. Animated specifications of computational societies. In *Proceedings of the 1st International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 1053–1061. 2002.

[3] J. Bentahar, B. Moulin, J.-J. C. Meyer, and B. Chaib-draa. A logical model for commitment and argument network for agent communication. In *Proceedings of the 3rd International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 792–799. 2004.

[4] C. Castelfranchi. Commitments: From individual intentions to groups and organizations. In *Proc. of the Intl Conf. on Multiagent Systems*, pages 41–48, 1995.

[5] T. H. Cormen, C. E. Leiserson, and R. Rivest. *Design and Analysis of Algorithms*. MIT Press, 1990.

[6] V. Dignum, J.-J. Meyer, F. Dignum, and H. Weigand. Formal specification of interaction in agent societies. In *2nd Goddard Workshop on Formal Approaches to Agent-Based Systems (FAABS)*, Maryland, Oct 2002.

[7] U. Endriss, N. Maudet, F. Sadri, and F. Toni. Protocol conformance for logic-based agents. In *Proc. of Intl. Joint Conf. on AI (IJCAI)*, pages 679–684. 2003.

[8] FIPA. Contract net interaction protocol specification, 2002. Number 00029.

[9] N. Fornara and M. Colombetti. Operational specification of a commitment-based agent communication language. In *Proc. of 1st Intl. Joint Conf. on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 535–542. 2002.

[10] M. G. Gouda. Protocol verification made simple: a tutorial. *Computer Networks and ISDN Systems*, 25:969–980, 1993.

[11] G. J. Holzmann. *Design and Validation of Computer Protocols*. Prentice-Hall, New Jersey, 1991.

[12] M.-P. Huget and J.-L. Koning. Requirement analysis for interaction protocols. In *Proc. of the Central and Eastern European Conf. on Multiagent Systems (CEEMAS)*, LNAI 2691, pages 404–412. Springer-Verlag, 2003.

[13] N. R. Jennings. On agent-based software engineering. *Artificial Intelligence*, 177(2):277–296, 2000.

[14] R. Kowalski and M. J. Sergot. A logic-based calculus of events. *New Generation Computing*, 4(1):67–95, 1986.

[15] A. U. Mallya, P. Yolum, and M. P. Singh. Resolving commitments among autonomous agents. In M.-P. Huget and F. Dignum, editors, *Proceedings of the AAMAS Workshop on Agent Communication Languages and Conversation Policies, LNAI 2922*, pages 166–182. Springer Verlag, 2003.

[16] P. McBurney and S. Parsons. Posit spaces: A performative model of e-commerce. In *Proceedings of the 2nd International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 624–631. 2003.

[17] M. Shanahan. *Solving the Frame Problem: A Mathematical Investigation of the Common Sense Law of Inertia*. MIT Press, Cambridge, 1997.

[18] M. P. Singh. An ontology for commitments in multiagent systems: Toward a unification of normative concepts. *Artificial Intelligence and Law*, 7:97–113, 1999.

[19] M. Venkatraman and M. P. Singh. Verifying compliance with commitment protocols: Enabling open Web-based multiagent systems. *Autonomous Agents and Multi-Agent Systems*, 2(3):217–236, Sept. 1999.

[20] P. Yolum and M. P. Singh. Flexible protocol specification and execution: Applying event calculus planning using commitments. In *Proceedings of the 1st International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 527–534. ACM Press, July 2002.

[21] P. Yolum and M. P. Singh. Reasoning about commitments in the event calculus: An approach for specifying and executing protocols. *Annals of Mathematics and Artificial Intelligence*, 42(1-3):227–253, 2004.