Correctness Requirements for Multiagent Commitment Protocols

Pınar Yolum

Department of Artificial Intelligence, Vrije Universiteit Amsterdam
De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands
pyolum@few.vu.nl

Abstract. Commitments are a powerful abstraction for representing the interactions between agents. Commitments capture the content of the interactions declaratively and allow agents to reason about their actions. Recent work on multiagent protocols define agent interactions as the creation and manipulation of commitments to one another. As a result, commitment protocols can be executed flexibly, enabling the agents to cope with exceptions that arise at run time.
We study the correctness requirements of commitment protocols that are necessary to ensure correct specification and coherent execution. We analyze and formalize various requirements for commitment protocols and draw relations among them. The main contribution of this analysis is that it allows protocol designers to develop correct protocols by signaling possible errors and inconsistencies that can possibly arise at run time. Since the requirements are formal, they can be incorporated in a software tool to automate the design and specification of commitment protocols.

1 Introduction

Multiagent systems consist of autonomous, interacting agents. To operate effectively, the interactions of the agents should be appropriately regulated. Multiagent interaction protocols provide a formal ground for enabling this regulation. However, developing effective protocols that will be carried out by autonomous agents is challenging [8]. Similar to the protocols in traditional systems, multiagent protocols need to be specified rigorously so that the agents can interact successfully. Contrary to the protocols in traditional systems, multiagent protocols need to be specified flexibly so that the agents can exercise their autonomy to make choices as best suits them or to handle unexpected situations that arise at run time. This basic requirement rules out many traditional formalisms, such as finite state machines (FSMs) or Petri nets. These formalisms only specify sequences of actions and leave no room for the agents to act flexibly.

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 [6]. 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 [14].

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 commitment protocols. 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 discusses commitment protocols. Section 4 develops correctness requirements for commitment protocols. Section 5 discusses the recent literature in relation to our work.

2 Technical Background

We use event calculus to study commitment protocols and their design requirements. Since event calculus can operate on actions well, it provides a useful way to manage the creation and manipulation of commitments through actions. Next, we give a brief overview of event calculus and then summarize our previous formalization of commitments and their operations in event calculus [13].

2.1 Event Calculus

The event calculus is a formalism based on many-sorted first order logic [9]. 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. Initiating and termination of events allow manipulation of fluents. 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.

We introduce the subset of the EC axioms that are used here; the rest can be found elsewhere [10]. The variables that are not explicitly quantified are assumed to be universally quantified. The standard operators apply (i.e., $\leftarrow$ denotes implication and $\wedge$

Table 1. The 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 from time 0.
Initially$_N(f)$	f does not hold from 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.

denotes conjunction). The time points are ordered by the $<$ relation, which is defined to be transitive and asymmetric.

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

2.2 Commitments

Commitments are obligations from one party to another to bring about a certain condition [3]. However, compared to the traditional definitions of obligations, commitments can be carried out more flexibly [11]. By performing operations on an existing commitment, a commitment can be manipulated (e.g., delegated to a third-party).

Definition 1 A unilateral commitment C*(x, y, p)* is an obligation of *x* to *y* to bring about a condition *p* [11]. *x* is the debtor and *y* is the creditor of the commitment. ∎

When a commitment of this form is created, *x* becomes responsible to *y* for satisfying *p*, i.e., *p* holds sometime in the future. Notice that we do not differentiate whether *p* is brought about deliberately or accidentally. The condition *p* does not involve other conditions or commitments.

Definition 2 A bilateral commitment CC*(x, y, p, q)* denotes that if the condition *p* is satisfied, *x* will be committed to bring about condition *q*. ∎

Bilateral 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 unilateral commitment is a special case of a bilateral commitment, where the condition p is set to true. That is, $C(x, y, q)$ is an abbreviation for $CC(x, y, true, q)$. 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.

Operations We summarize the operations to create and manipulate commitments [12, 11]. In the following discussion, x, y, z denote agents, c, c' denote commitments, and e denotes an event.

1. *Create(e, x, c):* When x performs the event e, the commitment c is created.
   Create$(e, x, \mathsf{C}(x, y, p))$: {*Happens*$(e, t) \wedge$ *Initiates*$(e, \mathsf{C}(x, y, p), t)$}
2. *Discharge(e, x, c):* When x performs the event e, the commitment c is resolved.
   Discharge$(e, x, \mathsf{C}(x, y, p))$: {*Happens*$(e, t) \wedge$ *Initiates*(e, p, t)}
3. *Cancel(e, x, c):* 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.
   Cancel$(e, x, \mathsf{C}(x, y, p))$: {*Happens*$(e, t) \wedge$ *Terminates*$(e, \mathsf{C}(x, y, p), t)$}
4. *Release(e, y, c):* When y performs the event e, x no longer need to carry out the commitment c.
   Release$(e, y, \mathsf{C}(x, y, p))$: {*Happens*$(e, t) \wedge$ *Terminates*$(e, \mathsf{C}(x, y, p), t)$}
5. *Assign(e, y, z, c):* 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.
   Assign$(e, y, z, \mathsf{C}(x, y, p))$: {*Happens*$(e, t) \wedge$ *Terminates*$(e, \mathsf{C}(x, y, p), t) \wedge$ *Initiates*$(e, \mathsf{C}(x, z, p), t)$}
6. *Delegate(e, x, z, c):* When x performs the event e, commitment c is eliminated but a new commitment c' is created where z is the new debtor.
   Delegate$(e, x, z, \mathsf{C}(x, y, p))$: {*Happens*$(e, t) \wedge$ Terminates$(e, \mathsf{C}(x, y, p), t) \wedge$ 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)) ← HoldsAt(*C*(x, y, p), t) ∧ Happens(e, t) ∧ Initiates(e, p, t)*

The following two axioms capture how a bilateral commitment is resolved based on the temporal ordering of the commitments it refers to.

Commitment Axiom 2 *Initiates(e,* C*(x, y, q), t) ←*
HoldsAt(CC*(x, y, p, q), t) ∧ Happens(e, t) ∧ Initiates(e, p, t)*
Terminates(e, CC*(x, y, p, q), t) ← HoldsAt(*CC*(x, y, p, q), t) ∧ Happens(e, t) ∧ Initiates(e, p, t)*

When the bilateral 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.

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.

Definition 3 A *commitment protocol* $\mathcal{P}$ is a set of *Initiates* and *Terminates* clauses that define which fluents in the protocol are initiated and terminated by each action. A *protocol run* is a set of *Happens* clauses along with an ordering of the referred time points [13]. ∎

Hence, following a protocol is executing actions which in return create or manipulate fluents. As said before, these fluents can be atomic propositions or commitments. An agent can start a protocol by performing any of the actions that suits itself. The transitions of the protocol are computed by applying the effect of the action on the current state. In most cases this corresponds to the application of commitment operations. Figure 1 gives an overview of the possible commitment 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.

To ease the explanation, we introduce the following notation. Let F be the set of fluents referred to with *Initiates* and *Terminates* clauses in the protocol. F is $CS \cup CCS \cup PS$ such that CS is the set of unilateral commitments, CCS is the set of bilateral 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. 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 at most contain five 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.

Example 1 We consider the Contract Net Protocol (CNP) as our running example [5]. 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.

Specifying a protocol in terms of commitments is significantly different from specifying a protocol in terms of an FSM. One major advantage of the commitment-based approach is that the execution of the protocol becomes more flexible than the execution of an FSM. Whereas in an FSM the execution is restricted to the predefined sequences, with commitment protocols participants are free to execute actions as long as they fulfill their commitments.

Example 2 By sending a proposal to the manager, a participant creates a bilateral 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 bilateral commitment will cease to exist but the following unilateral 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

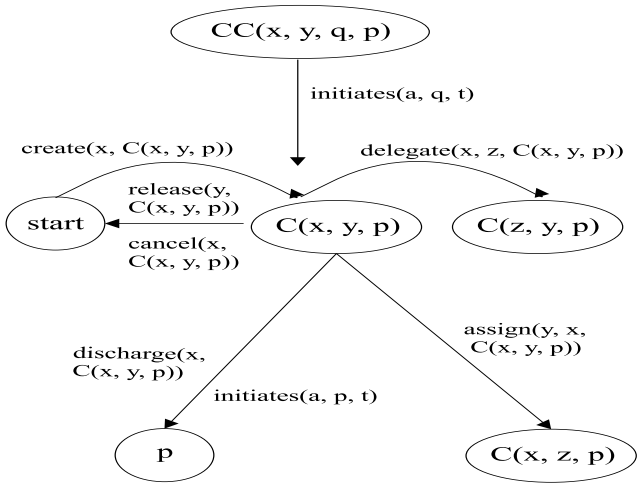

Fig. 1. Transitions in commitment protocols

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

Analyzing a commitment protocol is significantly more difficult than analyzing an FSM. First, 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.

Definition 4 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, $s(t) \equiv \bigwedge_{f \in F} HoldsAt(f,t)$. ∎

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 5 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 $\forall f \in F : (HoldsAt(f,t) \Rightarrow HoldsAt(f,t'))$ and $\forall f \in F : (HoldsAt(f,t') \Rightarrow HoldsAt(f,t))$. ∎

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

Definition 6 A protocol state $s(t)$ is a *proper* end-state if no unilateral 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. What are the requirements to establish this? The following analysis derives the requirements for correct commitment protocols.

Holzmann labels states of a protocol in terms of their capability of allowing progress [7]. 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 7 A protocol state $s(t)$ is progressing if

- $s(t)$ is not a proper end-state (e.g., $s(t) \Rightarrow \exists f \in CS : HoldsAt(f,t)$), and
- 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., progress state).

Definition 8 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. A progressive protocol exhibits the liveness property of traditional network protocols. 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 unilateral commitment. If $\forall c \in CS : O(c) \neq \emptyset$, then $\mathcal{P}$ is progressive.

Proof. By Definition 8, every state in $\mathcal{P}$ should be a proper end-state or a progress state. If a state does not contain open commitments then it is a proper end-state (Definition 6). If the state does contain a unilateral 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 7). ∎

Example 3 If a participant has a commitment to the manager to deliver the result (C(participant, manager, result)), then there needs to be at least one operation on this commitment so that the commitment can be terminated.

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 [7, p.120].

Example 4 Consider a participant x whose proposal has been accepted. Hence, C(x, manager, result) holds (state 1). Next, the participant delegates its commitment to another participant z C(z, manager, result) (state 2). Next, participant z delegates the commitment back to participant x and thus the protocol moves back to the previous state where C(x, manager, result) holds. Participants x and z delegate the commitment back and forth infinitely.

Obviously, the situation explained in Example 4 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 9 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 path 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 10 An infinitely repeating cycle is a cycle with progressive states such that if the protocol gets on to one of the states then the only execution sequence is to move to a state in the cycle [7]. ∎

In Example 4, the two delegate actions form an infinitely repeating cycle. Once the protocol gets into state 1 or state 2, 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 10 an infinitely repeating cycle only contains progressing states and by Definition 7, 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 5 In Example 4, 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 11 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 Let $\mathcal{P}$ be commitment protocol such that for all the commitments $c \in CS$, discharge$\in O(c)$ or cancel$\in O(c)$ or release$\in O(c)$. Then, $\mathcal{P}$ is effectively progressive.

Proof. By Lemma 1, any one of the commitment operations is sufficient to make the protocol progressive. By Lemma 3, any of the three operations will avoid infinite cycles. ∎

Note that we are not concerned about the choices of the agents in terms of which actions to take. Looking back at Example 4, 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 and delegating. 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 best suits it.

5 Discussion

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 [6]. In addition to unilateral and bilateral 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 as we have done here.

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.

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. Alberti *et al.* does not provide any design rules to ensure the correctness of their interaction protocols.

Endriss *et al.* study protocol conformance for interaction protocols that are defined as deterministic finite automaton (DFA) [4]. 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. DFAs have been analyzed extensively and many correctness properties have been formalized. However, DFAs represent sequences of actions and thus does not allow the agents to flexibly execute a protocol.

In our future work, we plan to work on other design requirements for commitment protocols, such as those for avoiding deadlocks. Identifying and formalizing these requirements will facilitate correct commitment protocols to be designed.

6 Acknowledgments

I have benefited from discussions with Munindar Singh and Ashok Mallya on commitments. The reviewers' comments have greatly improved the paper.

References

1. M. Alberti, D. Daolio, and P. Torroni. Specification and verification of agent interaction protocols in a logic-based system. In *Proc. ACM Symp. on Applied Computing (SAC)*. 2004. To appear.
2. A. Artikis, J. Pitt, and M. Sergot. Animated specifications of computational societies. In *Proc. Intl. Joint Conf. on Autonomous Agents and MultiAgent Syst. (AAMAS)*, pages 1053–1061. 2002.
3. C. Castelfranchi. Commitments: From individual intentions to groups and organizations. In *Proc. Intl. Conf. on Multiagent Systems*, pages 41–48, 1995.
4. U. Endriss, N. Maudet, F. Sadri, and F. Toni. Protocol conformance for logic-based agents. In *Proc. Intl. Joint Conf. on Artificial Intelligence (IJCAI)*, pages 679–684. 2003.
5. Foundation for Intelligent Physical Agents (FIPA). Contract net interaction protocol specification, 2002. Number 00029.
6. N. Fornara and M. Colombetti. Operational specification of a commitment-based agent communication language. In *Proc. Intl. Joint Conf. on Autonomous Agents and MultiAgent Syst. (AAMAS)*, pages 535–542. 2002.
7. G. J. Holzmann. *Design and Validation of Computer Protocols*. Prentice-Hall, NJ, 1991.
8. N. R. Jennings. On agent-based software engineering. *Artificial Intelligence*, 177(2):277–296, 2000.
9. R. Kowalski and M. J. Sergot. A logic-based calculus of events. *New Generation Computing*, 4(1):67–95, 1986.
10. M. Shanahan. *Solving the Frame Problem: A Mathematical Investigation of the Common Sense Law of Inertia*. MIT Press, Cambridge, 1997.
11. M. P. Singh. An ontology for commitments in multiagent systems: Toward a unification of normative concepts. *Artificial Intelligence and Law*, 7:97–113, 1999.
12. 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.
13. P. Yolum and M. P. Singh. Flexible protocol specification and execution: Applying event calculus planning using commitments. In *Proc. Intl. Joint Conf. on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 527–534. 2002.
14. P. Yolum and M. P. Singh. Reasoning about commitments in the event calculus: An approach for specifying and executing protocols. *Annals of Math. and AI*, 42(1-3), 2004. To appear.