# Reasoning About Commitments in the Event Calculus: An Approach for Specifying and Executing Protocols

Pınar Yolum (pyolum@eos.ncsu.edu) and Munindar P. Singh (singh@ncsu.edu)
*Department of Computer Science, North Carolina State University, Raleigh, NC 27695-7535, USA*

**Abstract.** Commitments among agents are widely recognized as an important basis for organizing interactions in multiagent systems. We develop an approach for formally representing and reasoning about commitments in the event calculus. We apply and evaluate this approach in the context of protocols, which represent the interactions allowed among communicating agents. Protocols are essential in applications such as electronic commerce where it is necessary to constrain the behaviors of autonomous agents. Traditional approaches, which model protocols merely in terms of action sequences, limit the flexibility of the agents in executing the protocols. By contrast, by formally representing commitments, we can specify the content of the protocols through the agents' commitments to one another. In representing commitments in the event calculus, we formalize commitment operations and domain-independent reasoning rules as axioms to capture the evolution of commitments. We also provide a means to specify protocol-specific axioms through the agents' actions. These axioms enable agents to reason about their actions explicitly to flexibly accommodate the exceptions and opportunities that may arise at run time. This reasoning is implemented using an event calculus planner that helps determine flexible execution paths that respect the given protocol specifications.

**Keywords:** Commitments; agent communication languages and protocols; methodologies

## 1. Introduction and Motivation

The design and analysis of multiagent systems comes down, to a large extent, to the design and analysis of interactions among agents. Interactions among agents must be modeled and reasoned about in such a manner as to respect the open, dynamic nature of interactions by accommodating the key aspects of autonomy, heterogeneity, opportunities, and exceptions.

- *Autonomy:* Promoting the participants' autonomy is crucial for creating effective systems in open environments. The participants should be able to exercise their autonomy in deciding what actions they want to perform, who they want to interact with, or how they want to carry out their tasks. Thus, participants must be constrained in their interactions only to the extent necessary to carry out the given protocol, and no more.
- *Heterogeneity:* Participants can be of diverse designs and may adopt different strategies to carry out their interactions. To interact in open systems, participants need to accommodate heterogeneity in others (Singh,

2003). Current protocol specifications do not allow this. For example, many e-commerce protocols assume that all participants are untrustworthy, and each step ensures that appropriately safe actions are taken by the various participants. This unnecessarily degrades performance where trust has been, or can be, established.

– *Opportunities:* Participants should be able to take advantage of opportunities to improve their choices or to simplify their interactions. Depending on the situation, certain steps in a protocol can perhaps be skipped. A participant may take advantage of domain knowledge and jump to a state in a protocol without explicitly visiting one or more intervening states, since visiting each state may require additional messages and cause delays.

– *Exceptions:* Participants must be able to modify their interactions to handle unexpected conditions. Unlike programming or networking exceptions such as loss of messages or network delays, the exceptions of interest here are higher-level exceptions that result from unexpected behaviors of the participants. For example, a deadline may be renegotiated at a discount. This would obviously involve domain knowledge, but the protocol representation should allow it.

Multiagent protocols regulate the interactions between agents. Current formalisms used in modeling network protocols, such as finite state machines (FSMs) and Petri Nets, specify protocols merely in terms of legal sequences or concurrent combinations of actions without regard to the meanings of those actions. When directly applied to multiagent settings, the above approaches leads to protocols that are over-constrained (Hutchison and Winikoff, 2002; Fisher and Wooldridge, 1997). However, protocol representations should not only constrain the actions of the participants, but also accommodate the essential properties of multiagent interactions listed above.

We develop an approach for specifying and executing protocols that accounts for these properties of open systems. Figure 1 gives an overview of our approach. We begin with a finite state machine representation of a protocol. From this representation, we develop a declarative protocol specification by capturing the intrinsic meaning of actions. We model these intrinsic meanings through *social commitments* or simply, *commitments*. Conceptually, commitments capture the obligations from one party to another (Castelfranchi, 1995; Walton and Krabbe, 1995). We define operations to create and manipulate commitments. We view each action in the protocol as an operation on commitments. In other words, by following the protocol, each agent creates and manipulates commitments, e.g., by fulfilling or canceling them. In addition to providing a protocol specification that defines the actions as op-

erations on commitments, we provide reasoning rules to operationalize the commitments.

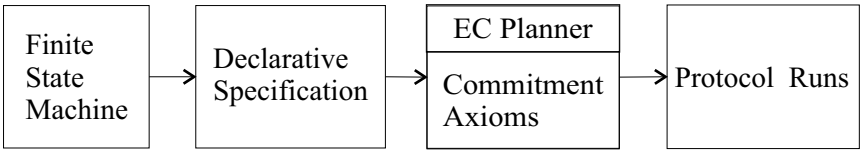


*Figure 1.* A schematic view of our approach

We formalize these reasoning rules and the operations on commitments in a variant of the event calculus that allows capturing of meanings of actions easily. Denecker *et al*. previously showed that the event calculus could be used to represent traditional network protocols accurately and succinctly (1996). We extend this result to multiagent protocols by incorporating commitments into the formalism. Capturing the intrinsic meaning of the actions through commitments and explicitly representing them as part of the protocol improves flexibility, permitting the agents to reason about their and others' behavior during the execution of the protocol, and enabling them to modify their actions as best suits them.

The protocol specifications developed by our approach can be used in two ways:

- *Execution:* The protocol specification can be used at run time if the agents have the necessary resources to process logical formulae. Essentially, each agent can logically compute its transitions using the operations and the reasoning rules. After each action, the agent is aware of its pending commitments and knows what additional commitments it has to make to get to a final state. Hence, the agent can follow the protocol by generating paths that will take it from its current state to a desired final state. Section 6 gives a detailed description of path generation.

- *Compilation:* If the agents cannot reason logically, then they have to follow a formalism that does not require them to compute the transitions. One such formalism is of traditional finite state machines. Even though FSMs are easy to execute, they are not easy to design in the first place. Without capturing the meaning of transitions, redundant transitions may be added or necessary transitions may be omitted. On the other hand, specifying a protocol using our approach is easy. Compiling this specification into an FSM ensures that all necessary transitions are captured.

This paper studies the framework for protocol specification and execution in detail. Compilation of a commitment-based protocol into an FSM is discussed elsewhere (Yolum and Singh, 2001). The rest of this paper is organized as

follows: Section 2 introduces our running example with pointers to different scenarios that may arise. Section 3 describes the necessary background about event calculus. Section 4 describes the operations and reasoning rules on commitments. Section 5 explains the specification of protocols. Section 6 depicts how planning can be used to generate scenarios of a protocol, and Section 7 discusses our work with respect to the literature.

## 2. Running Example

As a running example, we study the NetBill protocol developed for buying and selling of encrypted software goods on the Internet (Sirbu, 1998).

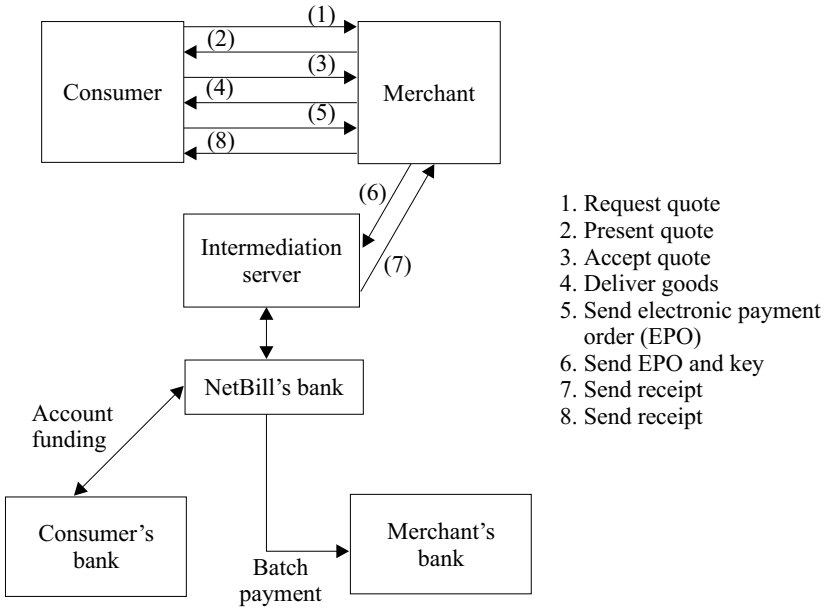


*Figure 2.* The NetBill payment protocol

EXAMPLE 1. As shown in Figure 2, the protocol begins with a customer requesting a quote for some desired goods, followed by the merchant sending the quote. If the customer accepts the quote, then the merchant delivers the goods and waits for an electronic payment order (EPO). The goods delivered at this point are encrypted, that is, not usable. After receiving the EPO, the merchant forwards the EPO and the key to the intermediation server, which handles the funds transfer. When the funds transfer completes, the intermediation server sends a receipt back to the merchant. The receipt contains the decryption key for the sold goods. As the last step, the merchant forwards the receipt to the customer. After the customer gets the receipt, he can decrypt

and use the goods. The intermediation server acts as a trusted party between the consumer and the merchant. It ensures that the funds are transferred between the consumer's and the merchant's banks and holds a copy of the receipt. Hence, if the EPO of the consumer does not clear or if the consumer does not receive a receipt, the intermediation server can be contacted. For our present purposes, we omit the banking procedures, thus simplifying the protocol with the assumption that if a merchant gets an EPO, he can take care of it successfully. ▮

Traditional representations of protocols are inadequate in settings where autonomous agents must flexibly interact, e.g., to handle exceptions and exploit opportunities.

EXAMPLE 2. Consider the following scenarios that may arise in the NetBill protocol:

– Before the customer asks for a quote, the merchant wants to advertise his goods by sending a quote to the customer.

– The customer may send an "accept" message without first exchanging explicit messages about a price. This situation would reflect the level of trust the customer places in the merchant. If the customer trusts the merchant to give him the best quote, he may accept the price without a prior announcement or quote. Alternatively, this action could result from the customer's lack of interest in the price, the emergency of the transaction, the insignificance of money to the customer, and so on.

– As shown in Figure 3, a merchant may send the goods without an explicit price quote. Sending unsolicited goods would correspond to "try before you buy" deals, common in the software industry. After a certain period, the customer is expected to pay to continue using the software.

– After receiving the goods, the customer may send the EPO to the bank instead of the merchant. By delegating the payment to the bank, the customer makes the bank responsible for ensuring that the money gets to the merchant. ▮

The scenarios depicted in Example 2 cannot be handled by a protocol representation that specifies the legal sequences of actions but does not define the content of the actions or of the intervening states.

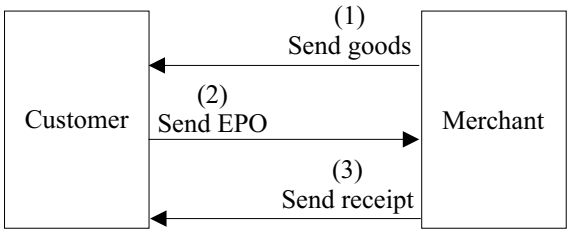


*Figure 3.* An alternative scenario

## 3. Event Calculus

The event calculus (EC), introduced by Kowalski and Sergot (1986), is a formalism to reason about events. EC involves *events* and *fluents*. Fluents are properties that may have different values at different time points. Events manipulate fluents. A fluent starts to hold after an event occurs that can initiate the fluent. Similarly, it ceases to hold when an event occurs that can terminate the fluent.

The event calculus used in this paper is a subset of Shanahan's circumscriptive event calculus (1997). It is based on many-sorted first-order predicate calculus, with the addition of eight predicates to reason about events. We now introduce these predicates and the axioms with which to reason about them.

In the following, $a, b, \ldots$ refer to events; $f, g, \ldots$ refer to fluents; and $t, t_1, t_2, \ldots$ refer to time points. The variables that are not explicitly quantified are assumed to be universally quantified. $\leftarrow$ denotes implication and $\wedge$ denotes conjunction. The time points are ordered by the $<$ relation, which is defined to be transitive and asymmetric.

1. *Initiates*$(a, f, t)$ means that $f$ holds after event $a$ at time $t$.
2. *Terminates*$(a, f, t)$ means that $f$ does not hold after event $a$ at time $t$.
3. *Initially*$_P(f)$ means that $f$ holds from time 0.
4. *Initially*$_N(f)$ means that $f$ does not hold from time 0.
5. *Happens*$(a, t_1, t_2)$ means that event $a$ starts at time $t_1$ and ends at $t_2$.
6. *HoldsAt*$(f, t)$ means that $f$ holds at time $t$.
7. *Clipped*$(t_1, f, t_2)$ means that $f$ is terminated between $t_1$ and $t_2$.
8. *Declipped*$(t_1, f, t_2)$ means that $f$ is initiated between $t_1$ and $t_2$.

Based on the language of EC, the following axioms are defined (Shanahan, 1997):

EC AXIOM 1.

$\mathit{HoldsAt}(f, t) \leftarrow \mathit{Initially}_P(f) \wedge \neg \mathit{Clipped}(0, f, t)$ ∎

All fluents that hold initially and are not terminated by any event from time 0 to time $t$ continue to hold at time $t$.

EC AXIOM 2.

$\mathit{HoldsAt}(f, t_3) \leftarrow \mathit{Happens}(a, t_1, t_2) \wedge \mathit{Initiates}(a, f, t_1) \wedge (t_2 < t_3) \wedge \neg \mathit{Clipped}(t_1, f, t_3)$ ∎

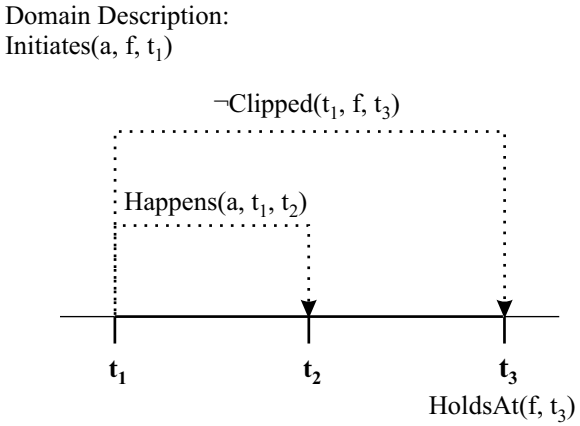


*Figure 4.* Axiom 2

As shown in Figure 4, after an event initiates a fluent, the fluent continues to hold if no other event that can terminate it occurs at a later time.

EC AXIOM 3.

$\mathit{Clipped}(t_1, f, t_4) \leftrightarrow \exists a, t_2, t_3\ [\mathit{Happens}(a, t_2, t_3) \wedge (t_1 < t_2) \wedge (t_3 < t_4) \wedge \mathit{Terminates}(a, f, t_2)]$ ∎

As shown in Figure 5, Axiom 3 states that a fluent is said to be *clipped* if and only if an event occurs to terminate it.

Axioms 4 and 5 represent the duals of Axioms 1 and 2, respectively.

EC AXIOM 4.

$\neg \mathit{HoldsAt}(f, t) \leftarrow \mathit{Initially}_N(f) \wedge \neg \mathit{Declipped}(0, f, t)$ ∎

Axiom 4 states that a fluent does not continue to hold, if initially it did not hold, and there does not occur any event that initiates it.

Domain Description:
Terminates(a, f, $t_2$)

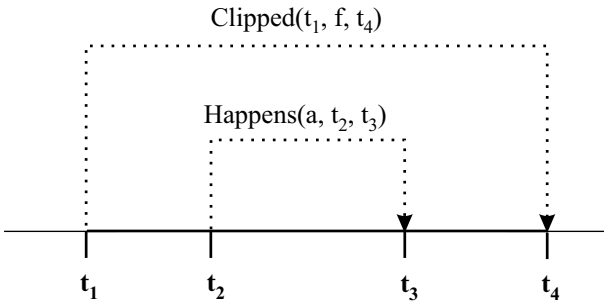


*Figure 5.* Axiom 3

Domain Description:
Terminates(a, f, $t_1$)

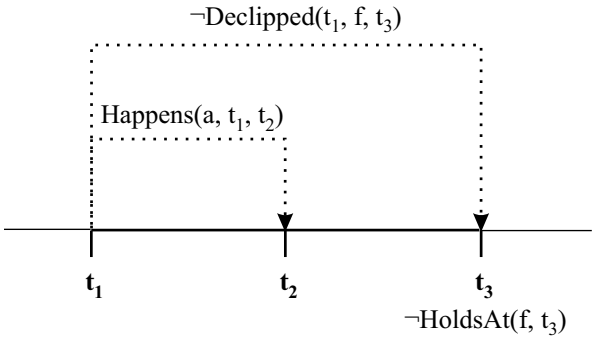


*Figure 6.* Axiom 5

EC AXIOM 5.

$\neg HoldsAt(f, t_3) \leftarrow Happens(a, t_1, t_2) \wedge Terminates(a, f, t_1) \wedge (t_2 < t_3) \wedge \neg Declipped(t_1, f, t_3)$ ∎

Following the same intuition, Axiom 5 states that if an event occurs and terminates a fluent, and no other event occurs to initiate it, then the fluent continues not to hold. This axiom is illustrated in Figure 6.

EC AXIOM 6.

$Declipped(t_1, f, t_4) \leftrightarrow \exists a, t_2, t_3\ [Happens(a, t_2, t_3) \wedge (t_1 < t_2) \wedge (t_3 < t_4) \wedge Initiates(a, f, t_2)]$ ∎

A fluent is said to be *declipped* in a time period if and only if there exists an event that occurs and either initiates or releases the fluent in that time period. This axiom is illustrated in Figure 7.

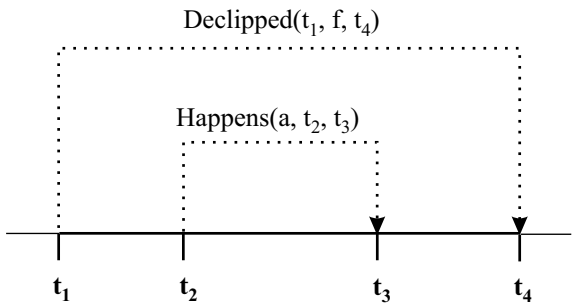


*Figure 7.* Axiom 6

EC AXIOM 7.

$\mathit{Happens}(a, t_1, t_2) \rightarrow t_1 \leq t_2$ ∎

Axiom 7 ensures that no event takes a negative amount of time.

DEFINITION 1. We introduce a two-argument *Happens* predicate to reason about events that start and end at the same time point. For simplicity, we will use this version of the *Happens* predicate hereafter.

$\mathit{Happens}(a, t) \equiv_{def} \mathit{Happens}(a, t, t)$

## 4. Commitments

As defined here, commitments are made from one agent to another agent to bring about a certain property. Commitments result from communicative actions. That is, agents create commitments and manipulate them through the protocol they follow. We represent commitments as properties in the event calculus, and develop a scheme where we model the creation and manipulation of commitments as a result of performing actions. Further, by allowing preconditions to be associated with the initiation and termination of properties, different commitments can be associated with communicative acts to model the communications among agents more concretely.

DEFINITION 2. A base-level commitment C*(x, y, p)* is a commitment from a debtor *x* to a creditor *y* to bring about a condition *p* (Singh, 1999).

When a commitment of this form is created, *x* becomes responsible to *y* for satisfying *p*, i.e., *p* holds sometime in the future. The condition *p* does not involve other fluents or commitments.

DEFINITION 3. 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.

EXAMPLE 3. Given that *goods* means that goods are delivered, a base-level commitment C*(merchant, customer, goods)* shows that the merchant is committing to the customer that goods will be delivered. Given that *pay* means that payment is done, the conditional commitment, CC*(merchant, customer, pay, goods)* specifies that the merchant will commit to send the goods only if the customer pays the money. ∎

**Persistent Commitments** The commitments discussed so far are for bringing about a proposition. These commitments are discharged by bringing about the committed proposition. However, some commitments are persistent; they hold even though the committed proposition is brought about. For example, in the NetBill protocol, the merchant may commit to the customer that it will always honor returns. Honoring one return will not discharge this commitment. The commitment will persist and be valid whenever the customer wants to return an item. In order to denote persistent commitments, we define an operator on propositions, $G(p)$, which is based on the *always* operator in temporal logic (Emerson, 1990). Informally, this operator means that $p$ will hold on all future time points. Axiom 1 shows that $G(p)$ holds only if $p$ holds now and remains to hold for all future points, $t_2$. That is, $p$ is not terminated between the current time point $t_1$ and a later time point $t_2$.

COMMITMENT AXIOM 1.

$$\mathit{HoldsAt}(G(p), t_1) \leftarrow \mathit{HoldsAt}(p, t_1) \wedge (t_1 < t_2) \wedge \neg \mathit{Clipped}(t_1, p, t_2)$$ ∎

DEFINITION 4. A persistent commitment $\mathsf{C}(x, y, G(p))$ is defined as a commitment from a debtor $x$ to a creditor $y$ to ensure that $p$ holds on all future time points.

**Operations** Below, we present a formal account of the operations that can be performed to create and manipulate commitments (Venkatraman and Singh, 1999; Singh, 1999) and show how these operations can be formalized in the event calculus. These operations apply to base-level commitments as well as to the persistent commitments, with only one exception. Persistent commitments can never be discharged. Thus, the discharge operation described below does not apply to persistent commitments. In the following discussion, $x, y, z$ denote agents, $c, c'$ denote commitments, and $e$ denotes an event.

1. *Create(e, x, c)* establishes the commitment $c$. The *create* operation can only be performed by the debtor of the commitment $x$. When the event $e$ is performed, the commitment $c$ is initiated.

   COMMITMENT AXIOM 2.

   *Initiates(e,* C*(x, y, p), t)* $\leftarrow$ *Happens(e, t)* $\wedge$ *Create(e, x,* C*(x, y, p))* ∎

2. *Discharge(e, x, c)* resolves the commitment $c$. Again, the *discharge* operation can only be performed by the debtor of the commitment to mean that the commitment has successfully been carried out. Discharging a commitment terminates that commitment.

   COMMITMENT AXIOM 3.

   *Terminates(e,* C*(x, y, p), t)* $\leftarrow$ *Happens(e, t)* $\wedge$ *Discharge(e, x,* C*(x, y, p))* ∎

3. *Cancel(e, x, c)* cancels the commitment $c$. The cancel operation is performed by the debtor of the commitment. Usually, the cancellation of a commitment is followed by the creation of another commitment to compensate for the former one. This ensures that the debtor does not back out of a commitment without a punishment. When $x$ performs the event $e$, the commitment $c$ is terminated.

   COMMITMENT AXIOM 4.

   *Terminates(e,* C*(x, y, p), t)* $\leftarrow$ *Happens(e, t)* $\wedge$ *Cancel(e, x,* C*(x, y, p))* ∎

4. *Release(e, y, c)* releases the debtor from the commitment $c$. It can be performed by the creditor to mean that the debtor is no longer obliged to carry out his commitment.

   COMMITMENT AXIOM 5.

   *Terminates(e,* C*(x, y, p), t)* $\leftarrow$ *Happens(e, t)* $\wedge$ *Release(e, y,* C*(x, y, p))* ∎

5. *Assign(e, y, z, c)* assigns a new agent as the creditor of the commitment. More specifically, the commitment $c$ is eliminated, and a new commitment $c'$ is created for which $z$ is appointed as the new creditor.

   COMMITMENT AXIOM 6.

*Create(e, x,* C*(x, z, p))* ← *Happens(e, t)* ∧ *Assign(e, y, z,* C*(x, y, p))*
*Release(e, y,* C*(x, y, p))* ← *Happens(e, t)* ∧ *Assign(e, y, z,* C*(x, y, p))* ∎

6. *Delegate(e, x, z, c)* replaces the debtor of the commitment with another agent $z$ so that $z$ becomes responsible to carry out the commitment. Similar to the previous operation, the commitment $c$ is eliminated, and a new commitment $c'$ is created in which $z$ is the debtor.

COMMITMENT AXIOM 7.

*Create(e, z,* C*(z, y, p))* ← *Happens(e, t)* ∧ *Delegate(e, x, z,* C*(x, y, p))*
*Cancel(e, x,* C*(x, y, p))* ← *Happens(e, t)* ∧ *Delegate(e, x, z,* C*(x, y, p))* ∎

The creation and manipulation of commitments is handled with the above operations. In addition to these operations, we formalize reasoning rules on commitments that capture the operational semantics of our approach. Some of these operations require additional domain knowledge to reason about. For example, canceling a commitment might be constrained differently based on the domain. The reasoning rules we provide here only pertain to the create and discharge operations and the conditional commitments.

Axiom 8 states that a commitment is no longer in force if the condition committed to holds. For the condition $p$ to hold, an event must occur to initiate it. In Axiom 8, when the event $e$ occurs at time $t$, it initiates the property $p$, thereby discharging the commitment $\mathsf{C}(x, y, p)$.

COMMITMENT AXIOM 8.

*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 conditional commitment is resolved based on the temporal ordering of the commitments it refers to.

COMMITMENT AXIOM 9.

*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 conditional commitment $\mathsf{CC}(x, y, p, q)$ holds, if $p$ becomes true, then the original commitment ceases to exist but a new base-level commitment is created, since the debtor $x$ is now committed to bring about $q$. More specifically, in Axiom 9, when the event $e$ occurs, it initiates $p$, which results in the termination of the original commitment, and the initiation of $\mathsf{C}(x, y, q)$. This is similar to Colombetti's treatment of conditional commitments (2000), but here we capture how conditional commitments are resolved into base-level commitments rather than how they can be violated.

COMMITMENT AXIOM 10.

*Terminates(e,* CC*(x, y, p, q), t)* $\leftarrow$ *HoldsAt(*CC*(x, y, p, q), t)* $\wedge$ *Happens(e, t)* $\wedge$ *Initiates(e, q, t)* ∎

Again, when the conditional commitment $\mathsf{CC}(x, y, p, q)$ holds, if an event $e$ that can initiate $q$ occurs, $q$ begins to hold and the original commitment is terminated. Since the creditor $y$ has not committed to anything, no additional commitments are created.

**Communicative Acts** Many communicative acts (Austin, 1962) can be defined in terms of operations on commitments. Here, we study two such acts: prohibit and permit. Intuitively, prohibitions forbid a party from bringing out a proposition. For example, if a client prohibits a merchant to send advertisements, the merchant becomes committed to not sending the advertisements from then on. In many settings, we would expect prohibitions to be constrained, so that only authoritative roles can issue prohibitions. Here, we do not explicitly model the relationships between roles. Even though modeling the relations between different roles is crucial (Damianou et al., 2001), a detailed analysis is beyond the scope of this paper. We define prohibitions through persistent commitments.

DEFINITION 5. If a party is prohibited to bring about a proposition $p$, then it has a commitment to ensure that $p$ never holds.
*Create(prohibit(x, y, p), y,* C*(y, x, G(*$\neg$*p)))*

Permissions, on the other hand, allow a party to bring about a proposition, i.e., to make a commitment. We define permissions as negations of prohibitions.

DEFINITION 6. A party is permitted to bring about a proposition if it has not been prohibited from it.
*Release(permit(x, y, p), x,* C*(y, x, G(*$\neg$*p)))*

## 5. Specifying Protocols

To represent a protocol, we need to represent the flow of execution within the protocol. In EC, two predicates are used to specify how the execution can evolve: *Initiates* and *Terminates*. In addition to defining which fluents they *initiate* or *terminate*, the required preconditions for activating these predicates can be specified. Therefore, the possible transitions in a protocol can be specified in terms of a set of *Initiates* and *Terminates* axioms (to manipulate propositions) and the commitment axioms (to manipulate commitments).

DEFINITION 7. A *protocol specification* is a set of protocol axioms that define which properties pertaining to the protocol are initiated and terminated by each action.

Next we define how a protocol run is structured, which we represent by a set of actions that take place at specific time points. We assume that only one action can happen at one time point. Hence, we are not concerned with concurrent actions.

DEFINITION 8. A *protocol run* is a set of *Happens* clauses along with an ordering of the time points referred to in the *Happens* clauses.

We assume uniqueness-of-names axioms such that each distinct token refers to a unique fluent or a unique event. Given a conjunction of *Initiates* and *Terminates* clauses for domain description, a conjunction of $Initially_N$ and $Initially_P$ clauses, a conjunction of unique-name-axioms, and the EC axioms, we desire a protocol run that will derive a given goal state. The frame problem (McCarthy and Hayes, 1969) is handled through circumscription as shown by Shanahan (1997). Through circumscription, the set of *Initiates* and *Terminates* clauses is minimized. That is, all possible effects of the actions are assumed to be known. Similarly, by minimizing the protocol run, we assume that no other action happens. Since no action other than the ones in the generated protocol run takes place, and since all the changes of these actions are known, what cannot be derived from the the logical framework is assumed not to hold.

Notice that a protocol specified as in Definition 7 does not indicate any starting states, final states, or transitions among executions states. An agent can start a protocol by performing any of the actions whose preconditions match the current state of the execution. By appropriately increasing or decreasing the preconditions of the actions, a protocol can be abbreviated or enhanced to allow a broader range of interactions. Although we do not represent the final states of a protocol explicitly, we can examine a protocol run to determine if any agent has backed out of its commitment.

A violation of the commitments results in the violation of the protocol. The existence of open base-level commitments at the end of the protocol execution shows that a participant has not fulfilled some commitment. Unlike for base-level commitments, some violation of a persistent commitment can be detected before the end of the protocol. At any state in the protocol, if the negated proposition of the persistent commitment holds, then the persistent commitment is violated. Conditional commitments can never be violated directly. Detecting these violations is crucial for proper execution of protocols. Although we do not investigate the issue of protocol compliance by agents in this work, violated commitments provide evidence to identify non-compliant agents (Venkatraman and Singh, 1999).

To continue our treatment of the NetBill protocol, we first define the fluents used in that protocol and then provide the protocol specification.

EXAMPLE 4. The messages of Figure 2 can be assigned content based on the following definitions. Since each action can be performed by only one party, we do not represent the performers explicitly.

– **Roles:**

- *MR* represents the merchant.
- *CT* represents the customer.

– **Domain-specific fluents:**

- *request(i):* a fluent meaning that the customer has requested a quote for item $i$.
- *goods(i):* a fluent meaning that the merchant has delivered the goods $i$.
- *pay(m):* a fluent meaning that the customer has paid the agreed upon amount $m$.
- *receipt(i):* a fluent meaning that the merchant has delivered the receipt for item $i$.

– **Commitments:**

- *accept(i, m):* an abbreviation for CC*(CT, MR, goods(i), pay(m))* meaning that the customer is willing to pay if he receives the goods.
- *promiseGoods(i, m):* an abbreviation for CC*(MR, CT, accept(i, m), goods(i))* meaning that the merchant is willing to send the goods if the customer promises to pay the agreed amount.
- *promiseReceipt(i, m):* an abbreviation for CC*(MR, CT, pay(m), receipt(i))* meaning that the merchant is willing to send the receipt if the customer pays the agreed-upon amount.
- *offer(i, m):* an abbreviation for *promiseGoods(i,m)* $\wedge$ *promiseReceipt(i, m)*

– **Initiates Axioms in the NetBill Protocol:**

PA1. *Initiates(sendRequest(i), request(i), t)*

PA2. *Initiates(sendGoods(i, m), goods(i), t)*

PA3. *Initiates(sendEPO(i, m), pay(m), t)*

PA4. *Initiates(sendReceipt(i, m), receipt(i), t)*

– **Create Axioms in the NetBill Protocol:**

PA5. *Create(sendQuote(i, m), MR, promiseGoods(i, m))*

PA6. *Create(sendQuote(i, m), MR, promiseReceipt(i, m))*

PA7. *Create(sendAccept(i, m), MR, accept(i, m))*

PA8. *Create(sendGoods(i, m), MR, promiseReceipt(i, m))*

∎

The following example describes an example protocol run, and illustrates how commitments are created and then resolved.

EXAMPLE 5. The protocol run shown in Figure 3 can be formalized by the following facts:

F1. *Happens(sendGoods(i, m),* $t_1$*)*

F2. *Happens(sendEPO(m),* $t_2$*)*

F3. *Happens(sendReceipt(i),* $t_3$*)*

F4. $t_1 < t_2$

F5. $t_2 < t_3$

F6. $t_3 < t_4$

Now we look at how the commitments among the participants evolve in the given protocol run. We also assume that initially no commitments or predicates hold.

– When the goods are sent at time $t_1$, the fluent *goods(i)* is initiated. Further, by Axiom PA8, the commitment CC*(MR, CT, pay(m), receipt(i))* is created. So now the goods have been delivered, and the merchant is willing to send the receipt if the customer pays.

D1. *HoldsAt(goods(i),* $t_2$*)* (By Axiom PA2)

D2. *HoldsAt(*CC*(MR, CT, pay(m), receipt(i)),* $t_2$*)* (By Axiom PA8)

– By sending the EPO at time $t_2$, the customer *initiates* the fluent *pay(m)* at time $t_3$. By Axiom 9, this ends the commitment CC*(MR, CT, pay(m), receipt(i))* and creates the commitment C*(MR, CT, receipt(i)).* Since no event occurred to terminate *goods(i),* it continues to hold.

D3. *HoldsAt(pay(m),* $t_3$*)* (By Axiom PA3)

D4. *HoldsAt(*C*(MR, CT, receipt(i)),* $t_3$*)* (By D2, Axiom 9, and D3)

D5. ¬*HoldsAt(*CC*(MR, CT, pay(m), receipt(i)),* $t_2$*)* (By D2, Axiom 9, and D3)

– At time $t_3$ the third *happens* clause (F3) is applicable, which initiates the fluent *receipt(i).* Following Axiom 8, this discharges the commitment C*(MR, CT, receipt(i)).* By Axiom 3, a discharged commitment is terminated. Thus, we reach the state where the merchant has delivered the goods and the receipt, and the customer has paid.

D6. *HoldsAt(receipt(i),* $t_4$*)* (By Axiom PA4)

D7. ¬*HoldsAt(*C*(MR, CT, receipt(i)),* $t_4$*)* (By D5, Axiom 8, and Axiom 3)

– Thus, at time $t_4$, the following holds:
*HoldsAt(goods(i),* $t_4$*)* ∧ *HoldsAt(pay(m),* $t_4$*)* ∧ *HoldsAt(receipt(i),* $t_4$*)* ∎

## 6. Executing Protocols

Planning is the construction of plans for automatic or semiautomatic execution by an agent. Planning has been used in multiagent systems to aid different tasks, including resource allocation and mixed symbolic and numerical reasoning (Atkins et al., 2001; Dix et al., 2000), but has not been used for protocol execution.

Here we consider plans that would lead from an initial protocol state to a final state by applying the commitment axioms to transition among states. The form of logical reasoning that is commonly used in building event calculus planners is abduction (Eshghi, 1988; Denecker et al., 1992). One such abductive event calculus planner is due to Shanahan (2000); we use this event calculus planner here to demonstrate how possible paths can be generated between an initial state and a goal state given a protocol specification defined based on the preconditions and the effects of actions as in Definition 7.

We now walk through the steps to put together the protocol specification of the NetBill protocol and an initial state in the protocol to produce protocol runs that lead from the initial state to a sample final state. The existing EC planner is coupled with the commitment axioms and the protocol axioms. The commitment axioms are represented as state constraints in the planner. The fluents used in the protocol axioms are the same as the ones used in Example 4, except that here we add a transaction identifier to each fluent (as the last argument). This identifier is used to ensure that the participants, by bringing about properties, resolve commitments with the corresponding transaction ids. Since the debtor and the creditor of the commitments are clear from the context, the arguments corresponding to the debtor and the creditor are dropped.

In the event calculus, the initial states are represented by conjunctive expressions consisting of *Initially*$_P$ and *Initially*$_N$ clauses to represent which

fluents hold or do not hold at the beginning. However the planner implements only one predicate, *Initially*. In order to indicate negative fluents, it introduces a new predicate, *neg*. Figure 8 gives some example clauses to set up the initial state of the NetBill protocol. For the initial state of the NetBill protocol, none of the possible fluents and commitments hold. Hence, the first clause states that initially no goods ($I$) for none of the transactions ($D$) have been delivered. Similarly, the second clause states that there do not exist any commitments for payments ($M$) for any transactions ($D$).

```
axiom(initially(neg(goods(I, D))),[]).
axiom(initially(neg(c(pay(M, D)))),[]).
```

*Figure 8.* Example clauses of an initial state

Operators are the actions in the domain. These actions are defined in the program through the *executable* clause. Figure 9 gives such an example. This clause means that the *sendRequest* action can be used in constructing a plan.

```
executable(sendrequest(I, D)).
```

*Figure 9.* Example clause of executable events

Figure 10 gives an example *Initiates* axiom. The first argument to the axioms is the *Initiates* clause, and the second argument is the set of preconditions needed for the *Initiates* clause to be applicable.

The example axiom in Figure 10 says that the action *sendEPO* initiates the fluent *pay* with the requirement that *goods* must have been delivered. The precondition section is optional. Here, it is added to ensure that the payment is always made after the goods are delivered. We will discuss the effects of preconditions shortly.

```
axiom(initiates(sendepo(M, D), pay(M, D),T),
              [holds_at(goods(I, D),T)]).
```

*Figure 10.* Example of an *Initiates* clause

Figure 11 shows that the *sendAccept* action creates the conditional commitment *cc(goods(I, D), pay(M, D))*. Again, the precondition is optional. Here, it is used to enforce the requirement that the *sendAccept* action will only be invoked if the *goods* have not been sent.

Given the initial states, the goal states, and the domain description, an event calculus planner can generate protocol runs that contain *Happens* clauses and an ordering of time points of the actions that take place (Shanahan, 2000).

In order to walk through the code, we provide the description of an example final state. Figure 12 gives such an example. The final state depicted

```
axiom(create(sendaccept(I, M, D),
             cc(goods(I, D), pay(M, D))),
            [holds_at(neg(goods(I, D)), T)]).
```

*Figure 11.* Example of a *Create* clause

in Figure 12 means that goods (software) and the receipt have been sent with transaction identifier 51 and the customer sent the money for this transaction. This yields the result shown in Figure 13, where each R is a possible protocol run starting from the initial state where no commitments or fluents hold, and ending in the final state depicted in Figure 12.

```
abdemo([holds_at(goods(software, 51),t),
        holds_at(pay(price, 51),t),
        holds_at(receipt(software, 51),t)],R).
```

*Figure 12.* Example description of a final state

**Additional Paths** As described in Section 5, each protocol run consists of *Happens* clauses and an ordering of time points. The last argument in each *Happens* clause denotes a time point at which the event happens. The ordering of these time points is then shown with the *before* clauses. The time points generated by the planner have been replaced with simpler time points to ease the following of the time line.

The three protocol runs in Figure 13 correspond to the first three scenarios described in Example 2. Figure 14 shows these extra paths derived by the planner.

– The first protocol run starts with the merchant sending a quote for the software. Hence, the conditional commitments *promiseGoods(i, m)* and *promiseReceipt(i, m)* are initiated. Next, the customer sends an accept message. By Axiom 9, the merchant becomes committed to sending the goods. Once the merchant sends the goods, the customer becomes committed to sending a payment due to its conditional commitment, *accept*. Next, the customer sends the payment, which triggers the merchant's commitment to send the receipt. Finally, merchant sends the receipt. There are no pending commitments in the system and the final state is reached. This protocol run corresponds to the path that contains the states 1, 3, 4, 5, 6, and 7 in Figure 14.

  D1. *HoldsAt(*CC*(MR, CT, accept(i, m), goods(i)), t2)* (By Axiom PA5)

  D2. *HoldsAt(*CC*(MR, CT, pay(m), receipt(i)), t2)* (By Axiom PA8)

  D3. *HoldsAt(*CC*(CT, MR, goods(software), pay(price)), t3)* (By Axiom PA7)

```
R =
[[happens(sendquote(software, price, 51), t1),
  happens(sendaccept(software, price, 51), t2),
  happens(sendgoods(software, price, 51), t3),
  happens(sendepo(price, 51), t7),
  happens(sendreceipt(software, 51), t8)],
 [before(t1, t), before(t1, t8),
  before(t1, t7), before(t1, t3),
  before(t1, t2), before(t2, t),
  before(..., ...)|...]] ;

R =
[[happens(sendaccept(software, price, 51), t5),
  happens(sendgoods(software, price, 51), t6),
  happens(sendepo(price, 51), t7),
  happens(sendreceipt(software, 51),t8)],
 [before(t5, t), before(t5, t8),
  before(t5, t7), before(t5, t6),
  before(t6, t), before(t6, t8),
  before(..., ...)|...]] ;

R =
[[happens(sendgoods(software, price, 51), t4),
  happens(sendepo(price, 51), t7),
  happens(sendreceipt(software, 51), t8)],
 [before(t4, t), before(t4, t8),
  before(t4, t7), before(t7, t),
  before(t7, t8), before(t8, t)]] ;
```

*Figure 13.* Protocol runs computed by the planner

D4. *HoldsAt(*C*(MR, CT, goods(software)), t3)* (By D3 and D1)

D5. ¬*HoldsAt(*CC*(MR, CT, accept(software, price), goods(software)), t7)* (By D1, Axiom 10)

D6. *HoldsAt(goods(software), t7)* (By Axiom PA2)

D7. *HoldsAt(*C*(CT, MR, pay(price)), t7)* (By Axiom 9, D3, and D6)

D8. *HoldsAt(pay(price), t8)* (By Axiom PA3)

D9. ¬*HoldsAt(*C*(CT, MR, pay(price)), t8)* (By Axiom 8, D7, and D8)

D10. *HoldsAt(*C*(MR, CT, receipt(software)), t8)* (By D2, D8, and Axiom 9)

D11. ¬*HoldsAt(*CC*(MR, CT, pay(m), receipt(i)), t8)* (By D2, D8, and Axiom 9)

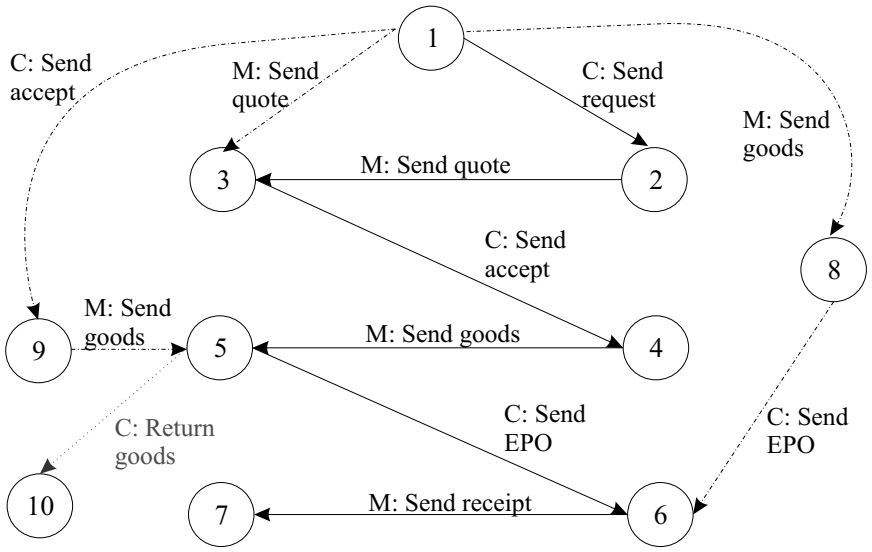


*Figure 14.* Additional paths generated by the planner

D12. *HoldsAt(receipt(i), t)* (By Axiom PA4)

D13. ¬*HoldsAt(*C*(MR, CT, receipt(software)), t)* (By D12 and Axiom 8)

– The second protocol run starts off with the customer sending an accept message, creating a conditional commitment that when the merchant sends the goods, then it (the customer) will pay. Next, the merchant sends the goods, making the customer committed to pay and creating a conditional commitment to send the receipt after the payment. After the customer sends the payment, its commitment to send the payment is discharged. But, now the merchant becomes committed to send the receipt. After the merchant sends the receipt, there are no pending commitments in the system and the final state is reached. This protocol run captures the second scenario described in Example 2. This protocol run corresponds to the path that contains the states 1, 9, 5, 6, and 7 in Figure 14.

D1. *HoldsAt(*CC*(CT, MR, goods(software), pay(price)), t6)* (By Axiom PA7)

D2. *HoldsAt(goods(software), t7)* (By Axiom PA2)

D3. *HoldsAt(*C*(CT, MR, pay(price)), t7)* (By D1, D2, and Axiom 9)

D4. ¬*HoldsAt(*CC*(CT, MR, goods(software), pay(price)), t6)* (By D1 and Axiom 9)

D5. *HoldsAt(*CC*(MR, CT, pay(price), receipt(software)), t7)* (By Axiom PA8)

D6. *HoldsAt(pay(price), t8)* (By Axiom PA3)

D7. *HoldsAt(*C*(MR, CT, receipt(software)), t8)* (By D5, D6, and Axiom 9)

D8. ¬*HoldsAt(*CC*(MR, CT, pay(price), receipt(software)), t8)* (By D5, D6, and Axiom 9)

D9. *HoldsAt(receipt(i), t)* (By Axiom PA4)

D10. ¬*HoldsAt(*C*(MR, CT, receipt(software)), t)* (By D9 and Axiom 8)

– The third protocol run is identical to Example 5. This protocol run corresponds to the path 1, 8, 6, and 7 in Figure 14.

These protocol runs can be used by an agent at run time to decide if an action is appropriate at a particular state of the execution. This enables the agents to cope with the exceptions by reconstructing plans as necessary. Thus, agents can execute protocols flexibly by taking advantage of opportunities and handling exceptions.

**Safe Paths** All generated protocol runs ensure that if followed correctly, the protocol ends in a desired final state. But, the actions in the protocol runs are usually taken by different parties. The generation of a path by an agent does not guarantee that another agent will follow it. The agents are only responsible for honoring their commitments. For example, an execution of the three actions (*sendGoods*, *sendEPO*, and *sendReceipt*) in any order takes the protocol to the desired final state (Figure 12), since these actions initiate *goods*, *pay*, and *receipt*; respectively. Although all orderings of these actions achieve the final state, some orderings of these actions can be more preferable than others because they may enforce creation of commitments.

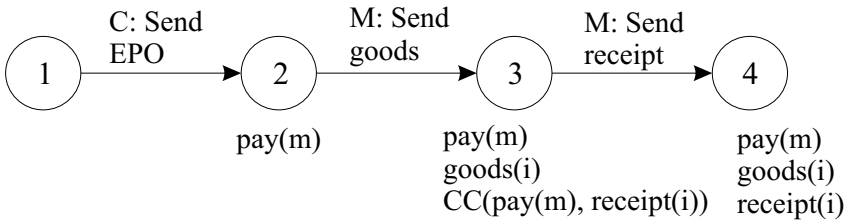


*Figure 15.* Possibly an unsafe path from the customer's side

EXAMPLE 6. We compare two orderings of these actions.

1. The first case is shown in Figure 15. The customer starts the protocol by paying (*sendEPO*), then the merchant sends the goods (*sendGoods*) and the receipt (*sendReceipt*). By definition of *sendGoods*, the merchant makes a conditional commitment that it will send the receipt if the customer pays, but since the customer has already paid at that moment, the

conditional commitment stays until the merchant sends the receipt. Even though this is a valid protocol run, after the customer pays, the merchant has no obligation to send the goods or the receipt, since it has not created any commitments. An example protocol run of this scenario would be as follows.
*happens(sendEPO(price, 51),* $t_1$*), happens(sendGoods(software, price, 51),* $t_2$*), happens(sendReceipt(software, 51),* $t_3$*), before(*$t_1$*,* $t_2$*), before(*$t_2$*,* $t_3$*), before(*$t_3$*,* $t$*)*

2. Figure 16 shows a different ordering where the merchant starts the protocol by sending the goods. Again, this initiates a conditional commitment such that the merchant will deliver the receipt if it gets a payment. This time, when the customer sends a payment, the merchant becomes committed to send the receipt. Thus, from the customer's point of view, this protocol run is safer in that the customer does not need to pay until it gets the goods and when it pays, there is a guarantee (through a commitment) that the merchant will deliver the receipt. An example protocol run of this scenario would be as follows.
*happens(sendGoods(software, price, 51),* $t_1$*), happens(sendEPO(price, 51),* $t_2$*), happens(sendReceipt(software, 51),* $t_3$*), before(*$t_1$*,* $t_2$*), before(*$t_2$*,* $t_3$*), before(*$t_3$*,* $t$*)*

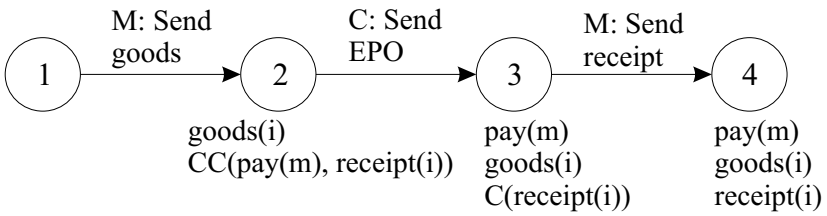


*Figure 16.* A more secure path

Even though the second protocol run provides additional security, the choice between the two protocol runs is left to the individual agents. A customer who trusts the merchant to deliver the goods without a commitment can still choose to follow the first run.

The security of the paths can be adjusted by modifying the preconditions of the actions. For example, to generate the second protocol run, it is enough to constrain the *sendEPO* action by enforcing the precondition *goods* as was the case in Figure 10. This way, the customer will only be able to initiate a *sendEPO* action after it receives the goods.

**Flexibility During Execution** As the previous examples illustrate, our approach can handle different types of exception, including different orderings of actions, or skipping actions as necessary. These exceptions can be handled both at compile time and at run time. In other words, additional protocol

paths can be generated to account for these types of exceptions. Our approach offers another kind of flexibility, which is caused by an unexpected action from a participant. We illustrate this point with an example. Consider an additional action *returnGoods(i, m)* meaning that the customer is returning goods $i$ for which it has promised to pay an amount $m$. We can show this with the following protocol axioms,

- *Terminates(returnGoods(i, m), goods(i), t)*
- *Release(returnGoods(i, m), MR,* C*(pay(m)))*

These two axioms say that after the *returnGoods* action takes place, the proposition *goods* will not hold and this action will release the customer from paying for the goods.

In Figure 14, in state 5 the following holds:

- *HoldsAt(goods(software), t)*
- *HoldsAt(*C*(CT, MR, pay(price)), t)*
- *HoldsAt(*CC*(MR, CT, pay(m), receipt(i)), t)*

If at state 5, the customer invokes the *returnGoods* action, the commitment C*(CT, MR, pay(price))* as well as the proposition *goods(i)* is terminated. Thus, the protocol moves to a new state (shown as 10). Even though this is not the desired final state, the protocol can end here since no base-level commitments hold. Since our operations on commitments are rich, the protocol execution can accommodate this type of exceptions at run time.

## 7. Discussion

Traditionally, protocols have been specified using formalisms such as finite state machines, or Petri Nets, that only capture the legal orderings of actions. The main advantage of these formalisms is that they are easy to implement and can be trivially followed by reactive agents. However, since the semantic content of the actions is not captured, the agents cannot reason about their actions, which means that they cannot take advantage of opportunities that arise or handle unexpected situations at run time. To remedy this situation, we develop an approach for protocol specification that embodies the commitments of agents to one another. Commitments have been studied before (Castelfranchi, 1995; Gasser, 1998) but have not been used for protocol specification as we have done here. The specification of protocols in terms of commitments allows agents to reason about their actions, enabling them to take care of the unexpected situations that may arise at run time. As we demonstrated, agents

that follow these protocols can decide on the actions they want to take by generating protocol runs with a planner.

In designing protocols, we can exploit the strengths of the event calculus to reason about actions and commitments. The event calculus provides an elegant way to represent the changes of the world through the actions in a protocol, and enables us to uniformly represent commitments, operations on them, and reasoning rules about them. Based on this formal grounding, multiagent protocols can be specified rigorously yet flexibly.

The event calculus has been theoretically studied, but has not been used for modeling commitments or commitment-based protocols. Denecker *et al.* (1996) use event calculus for specifying process protocols. Their specification also captures the preconditions of actions as well as the execution of the protocol through actions. Since Denecker *et al.* specify process protocols only, they use domain propositions to denote the meanings of actions. In multiagent systems, in addition, protocols should respect agents autonomy and enable them to interact flexibly to exploit opportunities and to handle exceptions. In order to achieve this, we use commitments to denote the meanings of the actions.

Representations of multiagent interactions should satisfy three desirable criteria (Singh, 2000). First we describe these criteria and show how our approach satisfies them. Next we review the recent literature, considering how these systems satisfy the same criteria.

– *Meaningful.* The messages should be represented with their content instead of being treated as mere tokens. Our approach is meaningful since we capture the meanings of the actions via commitments.

– *Verifiable.* The protocol representation should allow detection of agents that are not compliant with a given protocol without requiring that we examine the internal reasoning (or the source code) of the agents, which would violate our requirement of heterogeneity. Commitments inherently support verifiability. By keeping track of the commitments that are created and resolved in the system, we can infer whether an agent has acted according to its commitments.

– *Declarative.* A declarative, as opposed to a procedural, representation should specify what conditions should be brought out in the protocol, rather than how they are brought out. Our approach is declarative in that it specifies the contents of actions and states rather than specifying procedurally how agents can get from a start state to a final state.

It is worth noting that designing protocols that ensure meaningful conversations among agents is necessary but not sufficient to support the dynamic interactions among agents. For example, even though a protocol may enable

an exception to be handled, if doing so is not favorable to an agent, the protocol may not proceed along that run. Hence, a full-blown treatment of interactions would also involve considerations of the strategic reasoning of agents, which is beyond the scope of this paper. Below we discuss some leading approaches that are most related to our work.

The protocols defined using FSMs are verifiable, since all transitions are public, i.e., externally visible. However, FSMs procedurally specify the sequences of actions that reach a goal state when executed in the described sequence. That is, if the transitions from a start state to a final state are followed, then the protocol is executed correctly. However, if an exception occurs that is not handled by the FSM, then the agents cannot reason about the individual transitions of the FSM. In other words, FSMs specify how a certain goal state can be reached rather than specifying what each action brings out. In addition to not being declarative, FSM representation is not meaningful, since each message is treated as an arbitrary token without considering the meaning attached to it. As we showed above, representations that are not meaningful are not adequate to accommodate flexible interactions among autonomous agents, since agents cannot exercise their autonomy by choosing among actions without knowing what each action means.

d'Inverno *et al.* (1998) develop interaction protocols for the multiagent framework, Agentis. They model protocols as a composition of various services and tasks requested and offered among agents. d'Inverno *et al.*'s protocol model consist of four levels: registration, service, task, and notification. At each level of Agentis, the protocols are specified with FSMs. The FSM representations allow Agentis to specify the protocols formally. However, the protocols suffer from the shortcomings of the FSMs.

Lespérance *et al.* (2000) develop a tool, ConGolog, for developing reactive control mechanisms that are capable of handling exceptions. Their system's execution is based on a declarative domain description, which is similar to our definition of a protocol specification in that it specifies the preconditions and effects of the actions in the domain. We share their intuition in reconstructing plans at run time to handle exceptions. The main difference between Lespérance *et al.*'s approach and ours is that they define the meanings of actions in terms of only domain propositions, whereas we define them in terms of domain properties and commitments. In this respect, Lespérance *et al.*'s representation is not verifiable. Considering our example protocol, assume a protocol run where a customer agent performs a *sendAccept* action after receiving a *sendQuote* action—and no other actions take place. In our approach, we can easily determine that the merchant must send the goods to the customer, and the customer must send the EPO. By contrast, Lespérance *et al.*'s approach simply determines that the merchant sent a quote to the customer and the customer accepted the quoted price. To decide if the protocol

is in a good state requires examining the individual agent programs, i.e., the internals of the agents.

Smith *et al.* (1998) develop protocols in which actions are given a content based on joint intentions. We concur on the necessity of declarative content. However, Smith *et al.* model the content of actions via mental attributes. Since the mental attributes (such as beliefs, intentions, or individual commitments) are conceptually within an agent, there is no fundamental verification possible of them. That is, an agent may be insincere or mistaken about its beliefs. We cannot determine whether or not it is so without somehow examining the internals of the agent. By contrast, social commitments are made from one agent to another. Since social commitments are conceptually public, we can potentially judge if an agent has violated its commitments.

Pitt and Mamdani (1999) develop an agent communication language (ACL) framework in terms of protocols. Each protocol is associated with a set of performatives and a reply function that constrains possible performatives for replies. Pitt and Mamdani give content to messages based on social constructs, similar to the present approach. Whereas our focus in this work is to generate flexible protocols, their focus is to show how an agent designer can build agents that can follow such an ACL specification.

Fornara and Colombetti (2002) develop a method for agent communication, where the meanings of messages denote commitments. In addition to base-level and conditional commitments, Fornara and Colombetti use pre-commitments to represent a request for a commitment from a second party. They model the life cycle of commitments in the system through update rules. Based on these update rules, a commitment can either be fulfilled, violated, or canceled. Our approach can handle other operations on commitments, such as delegate and assign.

Flores and Kremer (2002) develop a model to specify conversation protocols using conversation policies. They define conversation policies, e.g., always answering a proposal, as norms in the society. As in our approach, Flores and Kremer model actions as creating and discharging commitments. In addition to following the operations on commitments, agents also follow the conversation policies to ensure desired sequences of actions. Flores and Kremer only model base-level commitments, whereas our approach accommodates conditional commitments and reasoning rules for these commitments. As a result, their approach does not allow any additional flexibility to the protocol.

In our future work, we plan to study various properties of protocols produced with this approach in more detail. In this paper, we have conceptually studied safe protocol runs. It would be interesting to study generation of paths that satisfy certain constrains, such as the mentioned safe protocol runs. Protocols in the networking community have been studied in detail, and validation models and correctness requirements have been identified (Holzmann,

1991). Similar analysis need to be applied to multiagent protocols, and design tools need to be developed to ensure correct protocols.

## Acknowledgements

A previous version of this paper appears in the First International Conference on Autonomous Agents and Multiagent Systems (Yolum and Singh, 2002). We thank Jürgen Dix, Peter Wurman, Matt Stallman, Michael Winikoff, Mehdi Dastani, and the anonymous reviewers for useful comments.

This work was supported by the National Science Foundation under grants IIS-9624425 (Career Award) and DST-0139037.

## References

Atkins, E. M., T. F. Abdelzaher, K. G. Shin, and E. H. Durfee: 2001, 'Planning and Resource Allocation for Hard Real-Time, Fault-Tolerant Plan Execution'. *Autonomous Agents and Multi-Agent Systems* **4**(1-2), 57–78.

Austin, J. L.: 1962, *How to Do Things with Words*. Oxford: Clarendon Press.

Castelfranchi, C.: 1995, 'Commitments: From Individual Intentions to Groups and Organizations'. In: *Proceedings of the International Conference on Multiagent Systems*. pp. 41–48.

Colombetti, M.: 2000, 'A Commitment-Based Approach to Agent Speech Acts and Conversations'. In: *Proceedings of the Workshop on Agent Languages and Conversation Policies*.

Damianou, N., N. Dulay, E. Lupu, and M. Sloman: 2001, 'The Ponder Policy Specification Language'. In: *Proceedings of the Workshop on Policies for Distributed Systems and Networks*, Vol. 1995 of *LNCS*. Bristol, UK, pp. 17–28, Springer-Verlag.

Denecker, M., K. V. Belleghem, G. Duchatelet, F. Piessens, and D. D. Schreye: 1996, 'A Realistic Experiment in Knowledge Representation in Open Event Calculus : Protocol Specification'. In: *Proceedings of the Joint International Conference and Symposium on Logic Programming*. pp. 170–184.

Denecker, M., L. Missiaen, and M. Bruynooghe: 1992, 'Temporal reasoning with abductive event calculus'. In: *Proceedings of the 10th European Conference on Artificial Intelligence (ECAI)*. pp. 384–388, John Wiley & Sons.

d'Inverno, M., D. Kinny, and M. Luck: 1998, 'Interaction Protocols in Agentis'. In: *Proceedings of the 3rd International Conference on Multiagent Systems (ICMAS)*. pp. 112–119, IEEE Computer Society Press.

Dix, J., H. Muñoz-Avila, and D. Nau: 2000, 'IMPACTing SHOP: Planning in a Multi-Agent Environment'. In: F. Sadri and K. Satoh (eds.): *Proceedings of Computational Logic in Multi-Agent Systems (CLIMA)*. pp. 30–42, Imperial College.

Emerson, E. A.: 1990, 'Temporal and Modal Logic'. In: J. van Leeuwen (ed.): *Handbook of Theoretical Computer Science*, Vol. B. Amsterdam: North-Holland, pp. 995–1072.

Eshghi, K.: 1988, 'Abductive planning with event calculus'. In: R. A. Kowalski and K. A. Bowen (eds.): *Proceedings of the Fifth International Conference on Logic Programming (ICLP)*. pp. 562–579, MIT Press.

Fisher, M. and M. Wooldridge: 1997, 'On the Formal Specification and Verification of Multi-Agent Systems'. *International Journal of Intelligent and Cooperative Information Systems* **6**(1), 37–65.

Flores, R. A. and R. C. Kremer: 2002, 'To Commit or Not to Commit: Modelling Agent Conversations for Action'. *Computational Intelligence* **18**(2), 120–173.

Fornara, N. and M. Colombetti: 2002, 'Operational Specification of a Commitment-Based Agent Communication Language'. In: *Proceedings of the 1st International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*. pp. 535–542, ACM Press.

Gasser, L.: 1998, 'Social Conceptions of Knowledge and Action: DAI Foundations and Open Systems Semantics'. In: *(*Huhns and Singh, 1998*)*. pp. 389–404. (Reprinted from *Artificial Intelligence, 1991*).

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

Huhns, M. N. and M. P. Singh (eds.): 1998, *Readings in Agents*. San Francisco: Morgan Kaufmann.

Hutchison, J. and M. Winikoff: 2002, 'Flexibility and Robustness in Agent Interaction Protocols'. In: *Proceedings of the Workshop on Challenges in Open Agent Systems*.

Kowalski, R. and M. J. Sergot: 1986, 'A Logic-Based Calculus of Events'. *New Generation Computing* **4**(1), 67–95.

Lespérance, Y., K. Tam, and M. Jenkin: 2000, 'Reactivity in a Logic-Based Robot Programming Framework'. In: *Intelligent Agents VI: Agent Theories, Architectures, and Languages*. pp. 173–187.

McCarthy, J. and P. J. Hayes: 1969, 'Some Philosophical Problems from the Standpoint of Artificial Intelligence'. In: *Machine Intelligence 4*. American Elsevier.

Pitt, J. and A. Mamdani: 1999, 'A Protocol-Based Semantics for an Agent Communication Language'. In: *Proceedings of the International Joint Conference on Artificial Intelligence*. pp. 486–491.

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

Shanahan, M.: 2000, 'An Abductive Event Calculus Planner'. *Journal of Logic Programming* **44**, 207–239.

Singh, M. P.: 1999, 'An Ontology for Commitments in Multiagent Systems: Toward a Unification of Normative Concepts'. *Artificial Intelligence and Law* **7**, 97–113.

Singh, M. P.: 2000, 'A Social Semantics for Agent Communication Languages'. In: F. Dignum and M. Greaves (eds.): *Issues in Agent Communication*, LNAI 1916. Springer, pp. 31–45.

Singh, M. P.: 2003, 'The future of agent communication'. In: M.-P. Huget (ed.): *Communication in Multiagent Systems: Background, Current Trends and Future*. Springer Verlag. To appear.

Sirbu, M. A.: 1998, 'Credits and Debits on the Internet'. In: *(*Huhns and Singh, 1998*)*. pp. 299–305. (Reprinted from *IEEE Spectrum, 1997*).

Smith, I. A., P. R. Cohen, J. M. Bradshaw, M. Greaves, and H. Holmback: 1998, 'Designing Conversation Policies using Joint Intention Theory'. In: *Proceedings of the 3rd International Conference on Multiagent Systems (ICMAS)*. pp. 269–276, IEEE Computer Society Press.

Venkatraman, M. and M. P. Singh: 1999, 'Verifying Compliance with Commitment Protocols: Enabling Open Web-Based Multiagent Systems'. *Autonomous Agents and Multi-Agent Systems* **2**(3), 217–236.

Walton, D. N. and E. C. W. Krabbe: 1995, *Commitment in Dialogue: Basic Concepts of Interpersonal Reasoning*. Albany: State University of New York Press.

Yolum, P. and M. P. Singh: 2001, 'Commitment Machines'. In: *Intelligent Agents VIII: Proceedings of the 8th International Workshop on Agent Theories, Architectures, and Languages*. pp. 235–247, Springer-Verlag.

Yolum, P. and M. P. Singh: 2002, '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)*. pp. 527–534, ACM Press.