

Abstract

YOLUM, PINAR. Flexible Execution of E-Commerce Protocols: A Commitment-Based Approach (Under the direction of Munindar P. Singh).

Protocols represent the allowed interactions among communicating components. Protocols are essential in electronic commerce to constrain the behaviors of autonomous entities. Traditional representations of protocols include the allowed actions, but not their content. This prevents them from being applied in settings where autonomous entities must flexibly interact to handle exceptions and exploit opportunities.

We develop an agent-based approach in which we model the communication protocols via a commitment-based representation. This formalism provides a content to the protocols. The content can be reasoned about, thereby enabling flexible execution. We provide reasoning rules to capture the evolution of commitments and show how an existing protocol can be systematically enhanced to yield a protocol that allows the given actions as well as other legal moves. We also show how a commitment-based representation can be compiled into a finite state machine for efficient execution.

FLEXIBLE EXECUTION OF E-COMMERCE PROTOCOLS: A COMMITMENT-BASED APPROACH

BY

PINAR YOLUM

A THESIS SUBMITTED TO THE GRADUATE FACULTY OF
NORTH CAROLINA STATE UNIVERSITY
IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

RALEIGH

JULY 2000

APPROVED BY:

CHAIR OF ADVISORY COMMITTEE

Biography

Pinar Yolum was born on April 23rd, 1976 in Istanbul, Turkey. She lived in Istanbul until August, 1998. She got her Bachelor of Engineering degree in Computer Engineering from Marmara University, Istanbul, Turkey in 1998. She was a masters student in North Carolina State University in the Department of Computer Science from August 1998 to July 2000.

Acknowledgements

I would like to thank my advisor Dr. Munindar Singh for his constant support and guidance through out this research. I would also like to thank my committee members Dr. James Lester and Dr. Peter Wurman. This research was supported by the National Science Foundation under grant IIS-9624425 (Career Award). I would also like to thank Mike Eberdt for numerous proof readings, S. Ilker Birbil for always helping out when things get rough, and finally my family for always being there.

Table of Contents

List of Figures	vi
1 Introduction	1
1.1 Dynamic Properties	2
1.2 Semantic Analysis	3
1.3 Approach	6
1.4 Organization	7
2 Technical Framework	8
2.1 Social Commitments	8
2.2 Event Calculus	10
2.3 Finite State Machines	16
3 Commitment Machines	17
3.1 Formalization	19
3.2 Compiling CMs to FSMs	25
3.2.1 Compilation Formalized	28
3.2.2 Configuration	33
3.3 Reasoning with CMs	35
3.3.1 Opportunism	35
3.3.2 Negotiation	35
3.3.3 Composition	36
3.3.4 Factoring and Regrouping	36
4 Commitments in Event Calculus	38
4.1 Representing Commitments in Event Calculus	38
4.2 Representing Protocols in Event Calculus	41
4.3 Planning	45
4.3.1 Planning in Event Calculus	46

4.3.2	Generating Paths	46
5	Conclusions	51
5.1	Literature	52
5.2	Future Directions	54

List of Figures

1.1	The NetBill payment protocol	4
1.2	Simplified version of the NetBill protocol	5
1.3	Alternative execution where the customer accepts any price first	6
1.4	Alternative execution where the merchant sends the goods first	6
2.1	Holding of fluents	12
2.2	Axiom 1	12
2.3	Axiom 2	13
2.4	Axiom 3	13
2.5	Axiom 4	14
2.6	Axiom 5	14
2.7	Axiom 6	15
2.8	Axiom 7	15
2.9	FSM representation of the NetBill Protocol	16
3.1	Example transition	26
3.2	A sample computation	29
3.3	Enhanced CM	33
4.1	Description of an initial state	47
4.2	Description of <i>Initiates</i> clauses	48
4.3	Description of <i>Terminates</i> clauses	49
4.4	Description of executable events	49
4.5	Description of a final state	50
4.6	Protocol runs	50

Chapter 1

Introduction

Commerce, electronic or otherwise, is based on communication among interacting participants. Protocols streamline this communication. Although e-commerce promises to be more flexible than traditional commerce, current techniques for specifying and enacting e-commerce protocols result in more rigid protocols than traditional, human-oriented protocols. This rigidity subverts some of the motivations behind e-commerce [Kimbrough and Lee, 1996].

Protocols for e-commerce have traditionally been modelled by the same formalisms used to specify network protocols. Although protocols in both areas are based on the idea of modelling communication between components, the formalisms used for network protocols are not adequate to represent the interactions in e-commerce [Fisher and Wooldridge, 1997]. As a consequence of this, most e-commerce protocols suffer from unneeded rigidity in execution, resulting in redundant interactions and unnecessary failures. In order to deal with this problem, we first investigate the desired properties of the components and then describe a new formalism that can accommodate them.

1.1 Dynamic Properties

Current formalisms used in modelling networking protocols, such as finite state machines and Petri Nets, specify protocols merely in terms of legal sequences of actions without regard to the meanings of those actions. For this reason, traditional protocols are often over-constrained to the level of specific sequences of actions.

However, e-commerce protocols should not only constrain the actions of the participants, but also recognize the open, dynamic nature of e-commerce interactions by accommodating the key aspects of autonomy, heterogeneity, opportunities, and exceptions.

- *Autonomy*: Enabling participants to exercise autonomy as much as they can in the social environment in which they are situated is a crucial factor in creating effective e-commerce context. The components should be able to exercise 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, in an e-commerce setting, components must retain their autonomy and be minimally constrained in their interactions, that is, constrained only to the extent necessary to carry out the given protocol.
- *Heterogeneity*: Components can be diverse in their identity and adopt different strategies to carry out their interactions. In current protocol specifications the identities of the participants are not taken into account. All those playing a given role are assumed to have identical qualities. This especially stands out in the case of trust. In most e-commerce protocols, all participants are assumed to be untrustworthy, and each step of the protocol ensures that appropriately safe actions are taken by the various participants. This unrealistic assumption degrades the performance of the protocols since the actions performed could

be greatly varied according to the correspondents' identities. In particular, participants that have established trust with each other could safely skip certain steps in the protocol.

- *Opportunities*: Components should be able to take advantage of opportunities to improve their choices or to simplify their interactions. Depending on the situation or the specific participant, certain steps in a protocol can be skipped. A participant may take advantage of either personal knowledge, or a convenience provided by other participants, and jump to a state in a protocol without visiting one or more intervening states.
- *Exceptions*: Components must be able to modify their interactions to handle any unexpected conditions. The sort of exceptions referred to here are not programming or networking exceptions such as loss of messages, network delays, and so on, but are higher-level exceptions that result from the unexpected behavior of the participating parties.

1.2 Semantic Analysis

We now analyze the concepts and challenges underlying communication, especially with regard to the execution of activities. As a running example, we consider the NetBill protocol that has been developed to handle the buying and selling of electronic goods, such as software and electronic documents, over the Internet [Sirbu, 1998].

Example 1 As shown in Figure 1.1, the protocol starts 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

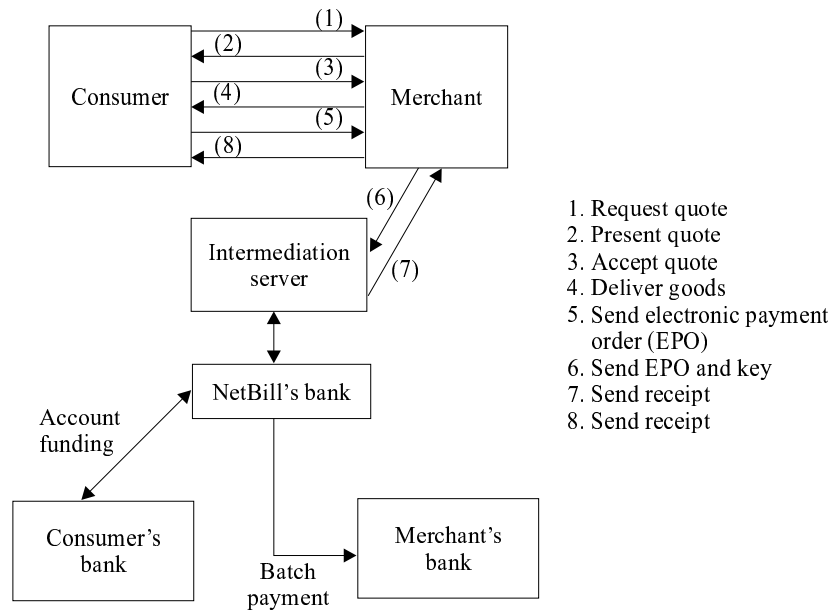


Figure 1.1: The NetBill payment protocol

receiving the EPO, the merchant forwards the EPO and the key to the Intermediation Server, which then contacts the bank to take care of the banking process. Once the debit-credit operations are handled, 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, who can successfully decrypt and use the goods. ■

For our present purposes, we are concerned neither about the details of the actual transactions that take place among the banks nor about the underlying security and encryption mechanisms. Therefore, we simplify the protocol to the point where we assume that once the merchant receives an EPO, he can take care of the banking services successfully. Figure 1.2 shows this simplified version. We use this simplified version of the protocol as our main example throughout this thesis.

The participating parties in an e-commerce protocol are self-interested and eager to practice any of a variety of interactions that would benefit them. Thus, in an e-commerce

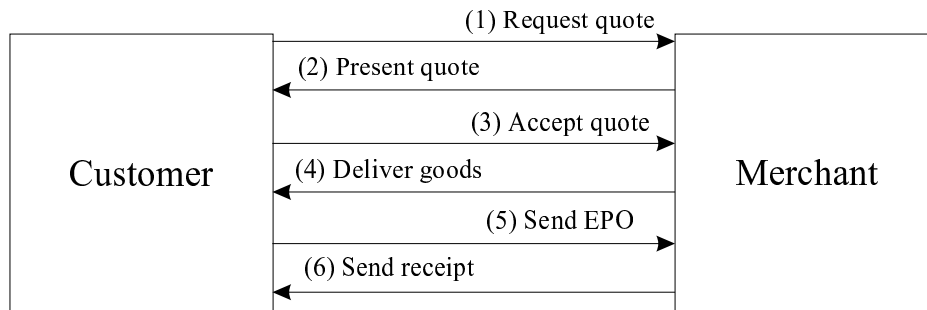


Figure 1.2: Simplified version of the NetBill protocol

setting, parties should be permitted a choice of actions, and be able to select the actions that benefit them the most.

Example 2 The rigid specification in Figure 1.2 cannot handle some of the natural situations that arise in e-commerce:

- Instead of waiting for a customer to request a quote, a merchant may proactively send a quote, mimicking the idea of advertising.
- The customer may send an "accept" message without first exchanging explicit messages about a price. This could very well reflect the level of trust between the parties. That is, the customer (who trusts the merchant to give him the best quote) 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. This scenario is shown in Figure 1.3.
- As shown in Figure 1.4, a merchant may send the goods without an explicit price quote. Such goods could represent the trial versions in the software industry, where after a certain period of time the customer is expected to pay to continue using the software.

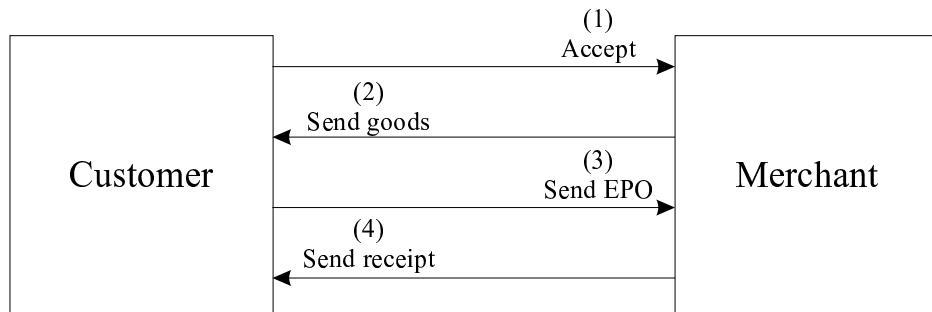


Figure 1.3: Alternative execution where the customer accepts any price first

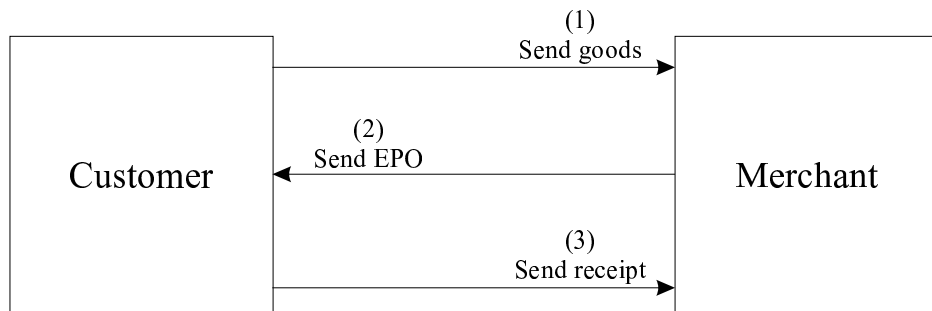


Figure 1.4: Alternative execution where the merchant sends the goods first

- 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.

■

1.3 Approach

In order to handle the aforementioned dynamic aspects of e-commerce, we develop an agent-based approach that incorporates the key abstraction of *commitments*. *Agents* are persistent computations that can perceive, reason, act, and communicate. Agents can be

autonomous and heterogeneous, and can represent different interacting components. The agents' communications affect, and are affected by, their commitments. In turn, the agents' commitments reflect the protocols they are following and the communications they have already made.

In what follows, we propose a new formalism, the *commitment machine (CM)*, to be used in the formal specification and execution of protocols. Instead of specifying protocols merely in terms of legal sequences of actions, we attach specific meanings, based on commitments of the participating agents, to states and actions. When states and actions are formally defined in terms of valid meanings, the possible paths between the states can be logically inferred. This permits an enhanced protocol with additional transitions, allowing a broader range of interactions. This, in turn, enables agents to exercise their autonomy, and to exploit opportunities and handle exceptions.

We provide reasoning rules that formalize the evolution of commitments. These rules, and the operations to create and to manipulate commitments are all formalized in event calculus. By stepping through a running example, we show how an e-commerce protocol can be specified using a CM and how it can be executed flexibly to yield an enhanced protocol.

1.4 Organization

The rest of this thesis is organized as follows: Chapter 2 explains the technical background dealing with communications. Chapter 3 introduces commitment-based finite state machines and discusses its applicability. Chapter 4 formalizes the reasoning rules and operations on commitments. Chapter 5 describes our contributions with respect to the most relevant literature, and the future directions.

Chapter 2

Technical Framework

Our approach to accommodating flexible execution of e-commerce protocols is based on the key abstraction of *social commitments*. In what follows, we first introduce social commitments, and then explain the concepts of finite state machines and the event calculus on which we build our approach.

2.1 Social Commitments

Starting with the rationalistic approaches, language has been thought to be a tool to relate the internal world of a being to the external world. As a counter to the language-as-description model, the language-as-action model emerged where each utterance in a language is considered not just a description but an action. More specifically, by making an utterance, an agent does not just describe the current state of the world, but also changes it. We capture this change in terms of modifications to the participants' commitments. Each utterance results in the creation, acceptance or declination of one or more commitments. The commitments associated with the language are *social* (as is the language itself), in that they affect all the parties that engage in communication [Winograd and Flores, 1986].

Social commitments are different from both internal and collective commitments. An internal commitment refers to the relation between an agent and a particular action, and a collective commitment refers to the relation between a group of agents committing to bring about a particular condition [Castelfranchi, 1995]. Social commitments, on the other hand, are commitments made from one agent to another agent to carry out a certain course of action [Singh, 1999].

Definition 1 A social commitment $C(x, y, G, p)$ relates a debtor x , a creditor y , and a condition p , in the scope of a context group G [Singh, 1999]. ■

When a social commitment of this form is created, x becomes responsible to y for satisfying p . In addition, there is an *emergence of right* [Conte and Castelfranchi, 1995]: y now has the right to make sure x carries out what he promised, and to protest if he doesn't.

The context group G is the organization within which the commitment exists. Making the context explicit enables us to control the evolution of the commitments and to accommodate compliance and resolution of disputes.

The condition p may involve relevant predicates and commitments, allowing the commitments to be nested or conditional. Contrary to commitments in databases, here the commitments are flexible, and can be revoked or modified. Almost always, the revocation or modification is constrained through *metacommitments*.

Definition 2 A commitment $c = C(x, y, G, p)$ is *base-level* if p does not refer to any other commitments; c is a *metacommitment* if p refers to a base-level commitment. ■

Viewing a commitment as an abstract data type, the creation and the manipulation of the commitments can be described using the following operations [Singh, 1999, Venkatraman and Singh, 1999]. Here, x, y, z denote agents, and c and c' denote commitments of the form $C(x, y, G, p)$.

1. $Create(x, c)$ establishes the commitment c . The *create* operation can only be performed by the debtor of the commitment.
2. $Discharge(x, c)$ 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. Thus, after the *discharge* operation the condition p starts to hold.
3. $Cancel(x, c)$ cancels the commitment c . Generally, cancellation of a commitment is followed by the creation of another commitment to compensate for the former one.
4. $Release(y, c)$ or $Release(G, c)$ releases the debtor from the commitment c . It can be performed either by the creditor or the context group, to mean that the debtor is no longer obliged to carry out his commitment.
5. $Assign(y, z, c)$ eliminates the commitment c , and creates a new commitment c' for which z is appointed as the new creditor.
6. $Delegate(x, z, c)$ eliminates the commitment c , and creates a new commitment c' in which the role of the debtor is transferred to z .

2.2 Event Calculus

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

Over the years, several variants of the event calculus have been proposed. The version of event calculus used here is the Simple Event Calculus (SEC), developed by Shanahan [1997]. It is based on first-order predicate calculus, with the addition of nine predicates to reason about the events. We now introduce these predicates and the axioms with which to integrate the predicates.

In the following, a, b, \dots refer to events, f, g, \dots refer to fluents; and t, t_1, t_2, \dots 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 fluent f holds after event a at time t .
2. *Terminates*(a, f, t) means that fluent f does not hold after event a at time t .
3. *Releases*(a, f, t) means that fluent f is not known to hold after event a at time t .
4. *Initially_P*(f) means that fluent f holds from time 0.
5. *Initially_N*(f) means that fluent f does not hold from time 0.
6. *Happens*(a, t_1, t_2) means that event a starts at time t_1 and ends at t_2 .
7. *HoldsAt*(f, t) means that the fluent f holds at time t .
8. *Clipped*(t_1, f, t_2) means that the fluent f is terminated between t_1 and t_2 .
9. *Declipped*(t_1, f, t_2) means that the fluent f is initiated between t_1 and t_2 .

Definition 3 We introduce a two argument *Happens* fluent to reason about events that start and end at the same time point. For simplicity, we will use this version of the *Happens*

fluent hereafter.

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

Example 3 Figure 2.1 shows that a fluent f holds at a time t_2 after it has been *initiated* but not *terminated*. \blacksquare

Domain Description:
 Initiates(a, f, t)
 Terminates(b, f, t)

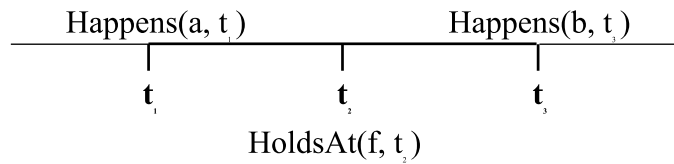


Figure 2.1: Holding of fluents

Based on the language of SEC, the following axioms are defined [Shanahan, 2000]:

Axiom 1 $\text{HoldsAt}(f, t) \leftarrow \text{Initially}_P(f) \wedge \neg \text{Clipped}(0, f, t) \blacksquare$

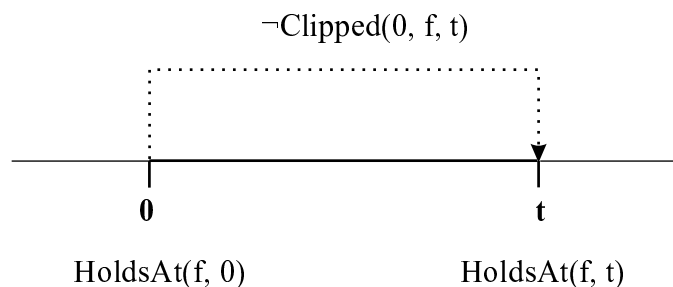


Figure 2.2: Axiom 1

As shown in Figure 2.2, all fluents that hold initially and are not terminated by any event from time 0 to time t continue to hold at time t .

Axiom 2 $HoldsAt(f, t_3) \leftarrow Happens(a, t_1, t_2) \wedge Initiates(a, f, t_1) \wedge (t_2 < t_3) \wedge \neg Clipped(t_1, f, t_3) \blacksquare$

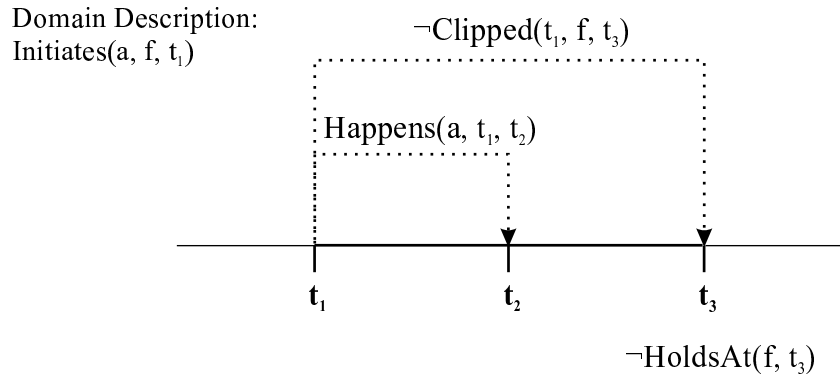


Figure 2.3: Axiom 2

As shown in Figure 2.3, after an event initiates a fluent, the fluent continues to hold if no other event that can terminate it occurs at a later time. These two axioms capture the fact that after a fluent begins to hold, an event that can terminate the fluent should occur in order to put an end to the holding of that fluent.

Axiom 3 $Clipped(t_1, f, t_4) \leftrightarrow \exists a, t_2, t_3 [Happens(a, t_2, t_3) \wedge (t_1 < t_3) \wedge (t_2 < t_4) \wedge (t_1 < t_2) \wedge (t_3 < t_4) \wedge [Terminates(a, f, t_2) \vee Releases(a, f, t_2)]] \blacksquare$

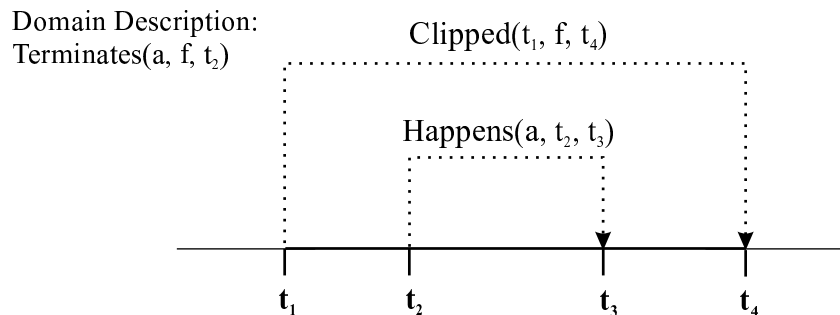


Figure 2.4: Axiom 3

Axiom 3 states that a fluent is said to be *clipped* if and only if an event occurs to terminate or release it.

The following two axioms represent the duals of Axiom 1 and Axiom 2, respectively.

Axiom 4 $\neg HoldsAt(f, t) \leftarrow Initially_N(f) \wedge \neg Declipped(0, f, t)$ ■

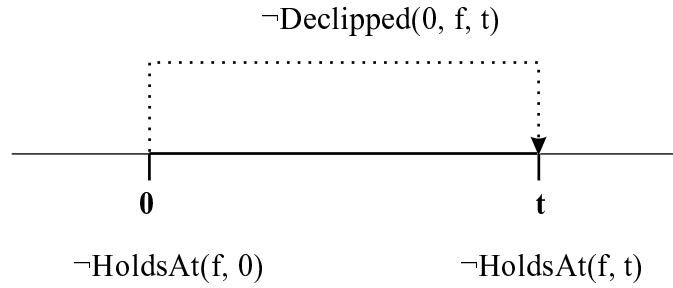


Figure 2.5: Axiom 4

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. This axiom is illustrated in Figure 2.5.

Axiom 5 $\neg HoldsAt(f, t_3) \leftarrow Happens(a, t_1, t_2) \wedge Terminates(a, f, t_1) \wedge (t_2 < t_3) \wedge \neg Declipped(t_1, f, t_3)$ ■

Domain Description:
Terminates(a, f, t₁)

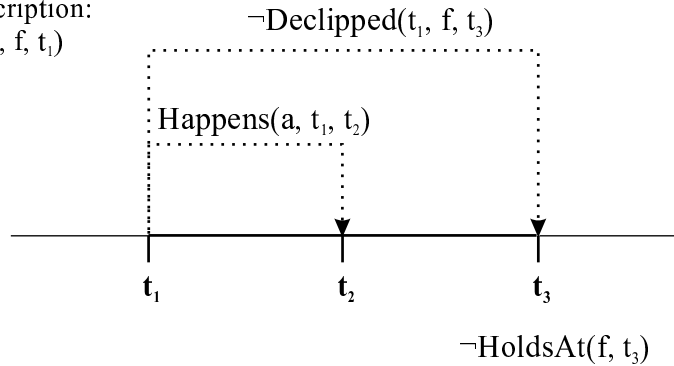


Figure 2.6: Axiom 5

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, as shown in Figure 2.6.

Axiom 6 $Declipped(t_1, f, t_4) \leftrightarrow \exists a, t_2, t_3 [Happens(a, t_2, t_3) \wedge (t_1 < t_3) \wedge (t_2 < t_4) \wedge (t_1 < t_2) \wedge (t_3 < t_4) \wedge [Initiates(a, f, t_2) \vee Releases(a, f, t_2)]]$ ■

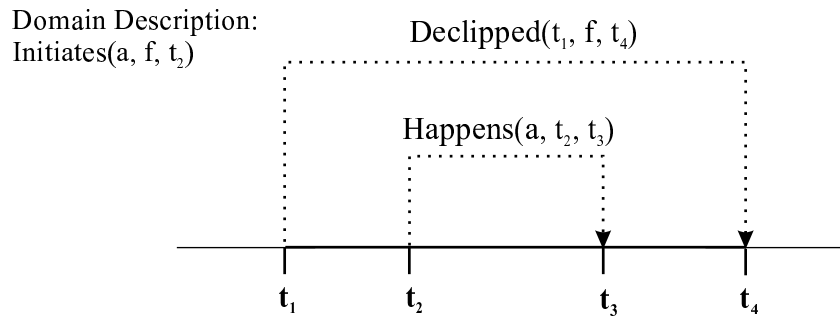


Figure 2.7: Axiom 6

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 2.7.

Axiom 7 $Happens(a, t_1, t_2) \rightarrow t_1 \leq t_2$ ■

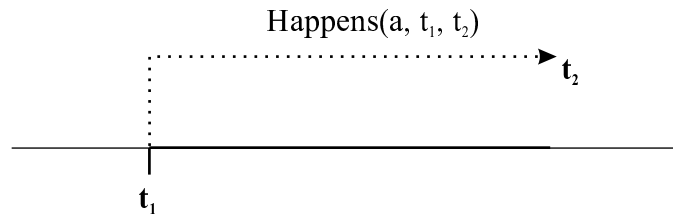


Figure 2.8: Axiom 7

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

2.3 Finite State Machines

One of the most common methods of specifying network protocols is through the use of the deterministic finite state machine (FSM) formalism. FSMs are also extensively used to specify e-commerce protocols [USE, 1998].

Definition 4 A deterministic finite state machine is a five-tuple, $(\mathbf{S}, \Sigma, s_0, \mathbf{Q}, \delta)$, where \mathbf{S} is the set of states, Σ is the input alphabet, $s_0 \in \mathbf{S}$ is the start state, $\mathbf{Q} \subseteq \mathbf{S}$ are the final states, and $\delta : \mathbf{S} \times \Sigma \mapsto \mathbf{S}$ is the transition function. ■

An FSM can be represented graphically by a labeled directed graph, where nodes represent the states and the arcs represent the transitions. In an e-commerce protocol specification, the alphabet of the FSM is given by the set of possible actions. The transition function is a means to coordinate the possible actions in the protocol.

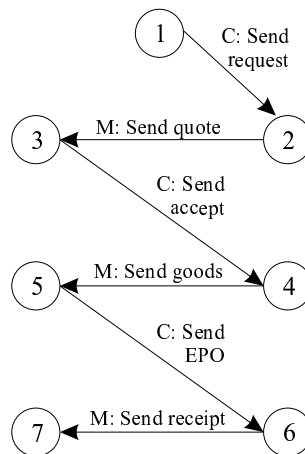


Figure 2.9: FSM representation of the NetBill Protocol

Example 4 Figure 2.9 shows the simplified version of the NetBill payment protocol as an FSM labeled with the actions of merchant agent M and customer agent C. ■

Chapter 3

Commitment Machines

We define a commitment machine (CM) conceptually as an FSM whose states and alphabet are given a semantic content that is defined in terms of commitments. A CM specifies the possible states an executing protocol can be in, the actions that are used for transition from one state to another, and the possible final states of the protocol. The meaning associated with each state specifies which commitments are in force in that particular state, and the meaning associated with each action defines how the commitments are affected by that action (thereby leading to a state change).

Unlike an FSM, the representation of a CM does not specify a starting state. The participants may start the protocol from a state where there are no commitments in force, or by accepting the commitments that are in force in that state. Again, unlike FSMs, the transitions between the states are not explicitly specified. The meanings of the states and the actors are logically represented. Based on the intrinsic meaning of actions, the new state that is reached by performing an action at a particular state can be logically inferred. Thus, instead of specifying the sequences of actions that can be performed, we simply specify meanings that can be reached and of these which are final (i.e., desirable).

By specifying an e-commerce protocol using a CM, we emphasize that the aim of executing a protocol is not merely to perform sequences of actions, but to reach a state that represents the result of performing these sequences of actions. With this in mind, we can come up with different paths that accomplish the same goal as the original path. Thus, a protocol can be reconfigured (enhanced or abbreviated) by finding alternative paths between states.

Example 5 We define the semantic content of each state in Figure 2.9 based on the participants' commitments:

- In state 3, having sent a quote to a customer, the merchant commits to delivering goods and sending a receipt afterwards, if the customer promises to pay.
- In state 4, having sent an accept to a merchant, the customer agrees to pay, but only if the merchant promises to send a receipt afterwards.
- In state 5, the merchant has fulfilled the first part of his promise by sending the goods.
- In state 6, the customer has discharged his commitment of sending the EPO.
- In state 7, the merchant has discharged his commitment of sending the receipt.

■

The CM specification of a protocol can be applied in two main ways.

Run time. A CM specification of a protocol gives the states and the effects of performing the various actions. Given a CM, an agent that can process logical formulae can compute the transitions between states. In this respect, the choice of actions is a planning problem for each agent. That is, from the possible final states, the agent first decides on the desired

final state, and then logically infers a path that will take it from the current state to the desired final state. Effectively, the agent interprets the CM directly at run time.

Compile time. To reduce the computation required at run time, a CM can be converted into an FSM representation at compile time. This transformation is based on systematically producing paths between pairs of meanings, and can be carried out in several ways using a search algorithm or a planner. The meanings in CM map to states in the FSM. The transitions between the states follow from the valid paths in the CM. The set of final states would be all the states that satisfy the ending constraints of the CM, namely, the states where no commitments are in force or where all metacommitments of a party have been honored. The set of actions remains the same. An agent can then use this FSM representation to execute the protocol at run time. We study compilation in detail below.

3.1 Formalization

Our formal language, \mathcal{P} , is based on the language of propositional logic with the addition of a commitment operator to represent commitments.

The following Backus-Naur Form (BNF) grammar with a distinguished start symbol *Protocol* gives the syntax of \mathcal{P} . In this grammar, *slant* typeface indicates nonterminals; \longrightarrow is a metasymbol of BNF specification; \ll and \gg delimit comments; $\{$ and $\}$ indicate that the enclosed item is repeated 0 or more times; the remaining symbols are terminals.

- $Protocol \longrightarrow \{Action\} \ll\text{set of actions}\gg$
- $Action \longrightarrow Token: L \ll\text{token is a label; } L \text{ is the associated meaning}\gg$
- $Commitment \longrightarrow C_x(L) \ll\text{simplified as explained below}\gg$
- $L \longrightarrow Commitment$

- $L \longrightarrow L \rightsquigarrow L$ «leads to, indicating a strict implication»
- $L \longrightarrow L \wedge L$ «conjunction»
- $L \longrightarrow \neg L$ «negation»
- $L \longrightarrow Prop$ «atomic propositions»

Since in the simplified version of the NetBill protocol there are only two roles, customer and merchant, we use a simplified notation to represent commitments. Here $C_x p$ means that x (which can be the customer or the merchant) is committed to the other party to carry out p . We also assume that the context group G is understood, so we don't represent it explicitly.

Definition 5 The strict implication, $p \rightsquigarrow q$, requires q to hold when p holds. Contrary to regular implication ($p \Rightarrow q$) which is true when p is false, the strict implication will be false if p is false. ■

Definition 6 A metacommitment is a commitment of the form $C_x(p \rightsquigarrow r)$. This expresses the conditional commitment where if the proposition p becomes true, then the debtor x will become committed to bringing about r . ■

Definition 7 The meaning of a state is given by any formula derivable from the nonterminal L . ■

Definition 8 $p \vdash q$ means that q can be logically derived from p . ■

Definition 9 $p \equiv q$ means that p and q are logically equivalent, that is, $p \vdash q$ and $q \vdash p$. ■

A minimal set of meanings is one in which all meanings are logically distinct. A set of final meanings is consistent if it is well-behaved with respect to logical consequence.

Definition 10 A set \mathbf{M} of meanings is minimal if and only if the following conditions hold:

- $(\forall m_i, m_j \in \mathbf{M}: m_i \equiv m_j \Rightarrow m_i = m_j)$
- $\text{true} \in \mathbf{M}$

■

Definition 11 A set of final meanings \mathbf{F} is consistent with respect to a set of meanings \mathbf{M} if and only if any meaning that is stronger than a final meaning is also final. That is,

$$(\forall m_i \in \mathbf{F}, m_j \in \mathbf{M}: (m_j \vdash m_i) \Rightarrow m_j \in \mathbf{F}) \blacksquare$$

Definition 12 Actions are represented as a two-tuple, where the first element is the token (name) of the action, and the second element is the effect of the action. Formally, $\langle a, e \rangle$ is an action, if a is the token and e is the effect of the action. ■

Definition 13 A CM is a three-tuple $M = \langle \mathbf{M}, \Delta, \mathbf{F} \rangle$, where \mathbf{M} is a minimal set of meanings, Δ is a set of actions defined in terms of commitments, and $\mathbf{F} \subseteq \mathbf{M}$ is a consistent set of final meanings. ■

Definition 14 A meaning changes from meaning q to meaning r under action $\langle a, e \rangle$ if and only if after applying the effect e of action a on q , r can be logically derived. Formally, $q \models_{\langle a, e \rangle} r \equiv_{def} q \wedge e \vdash r$ ■

Thus, deriving the resulting meaning from a given meaning and action involves computing the logical consequence (that is, the \vdash relation). For the propositional part of the language this is as usual. For commitments and metacommitments, we now present some important rules for reasoning about their consequences. These rules capture the operational semantics of our approach.

Reasoning Rule 1 A commitment $C_x p$ ceases to exist when the proposition p becomes true. ■

Reasoning Rule 2 A metacommitment $C_x(p \rightsquigarrow r)$ ceases to exist when the proposition p becomes true, but a new base-level commitment $C_x r$ is created to capture the fact that x has to satisfy the original metacommitment by bringing about the proposition r . ■

Reasoning Rule 3 A metacommitment $C_x(p \rightsquigarrow r)$ ceases to exist when the proposition r holds (before p is initiated), and no additional commitments are created. ■

The following example illustrates the applicability of these reasoning rules.

Example 6 Consider the metacommitment $C_m(\textit{pay} \rightsquigarrow \textit{receipt})$, which denotes the commitment that the merchant is willing to send a receipt if the customer pays. After the creation of this metacommitment the following scenarios may take place:

- The customer pays, making the proposition *pay* true. In this case, the metacommitment is terminated and a new commitment, $C_m \textit{receipt}$, is created. When the merchant actually sends the receipt, i.e., when the proposition *receipt* becomes true, then the commitment $C_m \textit{receipt}$ is discharged.
- Before the customer pays, the merchant sends the receipt, making the proposition *receipt* true. In this case, the metacommitment is terminated, but no other commitment is created since the customer did not commit to paying in the first place.

■

We now define the NetBill protocol formally as a CM.

Example 7 Following Example 5, the messages and the states can be given a content based on the following definitions:

- **Atomic Propositions**

- *request*: an atomic proposition meaning that the customer has requested a quote.
- *goods*: an atomic proposition meaning that the merchant has delivered the goods.
- *pay*: an atomic proposition meaning that the customer has paid the agreed amount.
- *receipt*: an atomic proposition meaning that the merchant has delivered the receipt.
- **Abbreviations for the Metacommitments**
 - *accept*: an abbreviation for $C_c(\text{goods} \rightsquigarrow \text{pay})$ meaning that the customer is willing to pay if he receives the goods.
 - *promiseGoods*: an abbreviation for $C_m(\text{accept} \rightsquigarrow \text{goods})$ meaning that the merchant is willing to send the goods if the customer promises to pay.
 - *promiseReceipt*: an abbreviation for $C_m(\text{pay} \rightsquigarrow \text{receipt})$ meaning that the merchant is willing to send the receipt if the customer pays.
 - *offer*: an abbreviation for $(\text{promiseGoods} \wedge \text{promiseReceipt})$

For simplicity, we place the content of an action in the state that results from it, and since each action can be performed by only one party, we do not specify the performers explicitly.

- **Meanings (M):**
 1. true
 2. *request* \ll The customer has requested a quote. \gg
 3. *offer* \ll The merchant has made an offer. \gg

4. $C_m \text{goods} \wedge \text{accept} \wedge \text{promiseReceipt}$ «The customer has accepted the merchant's offer.»»
5. $\text{goods} \wedge C_c \text{pay} \wedge \text{promiseReceipt}$ «The merchant has delivered the goods, and therefore the customer is committed to paying and the merchant is willing to send the receipt after the customer pays.»»
6. $\text{goods} \wedge \text{pay} \wedge C_m \text{receipt}$ «The merchant has delivered the goods, the customer has paid, and the merchant is committed to sending the receipt.»»
7. $\text{goods} \wedge \text{pay} \wedge \text{receipt}$ «The merchant has delivered the goods and the receipt, and the customer has paid.»»

- **Actions (Δ)**

1. $\langle \text{sendRequest: request} \rangle$ «sending a request for quote»»
2. $\langle \text{sendQuote: offer} \rangle$ «sending a quote»»
3. $\langle \text{sendAccept: accept} \rangle$ «sending an accept»»
4. $\langle \text{sendGoods: goods} \wedge \text{promiseReceipt} \rangle$ «delivering the goods»»
5. $\langle \text{sendEpo: pay} \rangle$ «sending an EPO»»
6. $\langle \text{sendReceipt: receipt} \rangle$ «sending the receipt»»

- **Final meanings (F)**

1. request
2. offer
3. $\text{goods} \wedge \text{pay} \wedge \text{receipt}$

■

3.2 Compiling CMs to FSMs

We now study the requirements of compiling a CM into an FSM. For an agent to be able to directly execute the resulting FSM, we seek an FSM that is deterministic and with no irrelevant transitions. Given a CM, the states of the FSM can be generated from the meanings of the CM. However, the execution of the FSM follows the usual regime of state-transition-state without regard to the formulas that exist in CM meanings on which the FSM states are based.

To infer the valid transitions in the FSM, we consider the possible entailments between the meanings. To ensure determinism and efficiency, we constrain the allowed transitions with two major restrictions.

Restriction 1 $\langle m_i, a, m_j \rangle \in \delta$ entails that $m_i \not\vdash m_j$

If the source meaning already entails the content captured in the target meaning, no transition from the source to the target is necessary. Entailments of this form ensure that the meaning that will be reached is not already captured at the current state, and that the FSM has no transitions that do not add to the content. ■

Restriction 2 $\langle m_i, a, m_j \rangle \in \delta$ entails that $(\forall m_k \in \mathbf{M}: m_i \models_{\langle a, e \rangle} m_k \Rightarrow m_j \vdash m_k)$

This is to ensure that if applying an action at a particular meaning entails several possible meanings, then the transition will end in a state that contains the maximal information. Later we will restrict our CMs so that a maximal meaning always exists. ■

Example 8 Let us assume that the meanings depicted in Figure 3.1 constitute the meaning set \mathbf{M} of a CM. Among these four meanings, \mathbf{F} contains only m_2 . Let the action set Δ contain two actions: *sendaccept* which initiates *accept*, and *sendoffer* which initiates *offer*.

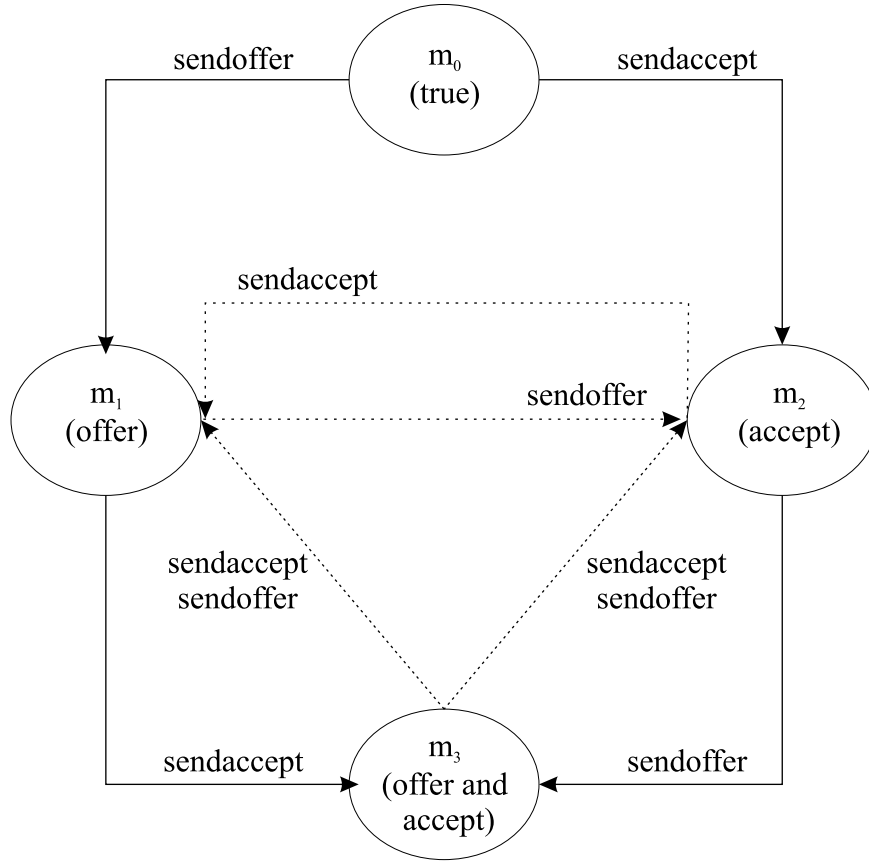


Figure 3.1: Example transition

We now look at some possible entailments and demonstrate which of these entailments result in a valid transition converting a CM to an FSM.

- $\langle m_0, \text{sendoffer}, m_1 \rangle : \text{true} \models_{\text{sendoffer}} \text{offer}$
 $\langle m_0, \text{sendaccept}, m_2 \rangle : \text{true} \models_{\text{sendaccept}} \text{accept}$
- Since neither restriction applies to these entailments, both transitions are accepted.
- $\langle m_1, \text{sendoffer}, m_1 \rangle : \text{offer} \models_{\text{sendoffer}} \text{offer}$
 $\langle m_1, \text{sendaccept}, m_1 \rangle : \text{offer} \models_{\text{sendaccept}} \text{offer}$

These two transitions will not be allowed due to Restriction 1, which does not allow transitions where the starting state already entails the end state.

$$\langle m_1, \text{sendaccept}, m_2 \rangle : \text{offer} \models_{\text{sendaccept}} \text{accept}$$

$$\langle m_1, \text{sendaccept}, m_3 \rangle : \text{offer} \models_{\text{sendaccept}} (\text{offer} \wedge \text{accept})$$

Based on Restriction 2, only the latter transition will be allowed.

- $\langle m_2, \text{sendaccept}, m_2 \rangle : \text{accept} \models_{\text{sendaccept}} \text{accept}$

$$\langle m_2, \text{sendoffer}, m_2 \rangle : \text{accept} \models_{\text{sendoffer}} \text{accept}$$

Again, based on Restriction 1, these two transitions will not be allowed.

$$\langle m_2, \text{sendoffer}, m_1 \rangle : \text{accept} \models_{\text{sendoffer}} \text{offer}$$

$$\langle m_2, \text{sendoffer}, m_3 \rangle : \text{accept} \models_{\text{sendoffer}} (\text{offer} \wedge \text{accept})$$

The only transition that will be allowed will again be the latter, since it yields more information than the former.

- $\langle m_3, \text{sendaccept}, m_2 \rangle : (\text{offer} \wedge \text{accept}) \models_{\text{sendaccept}} \text{accept}$

$$\langle m_3, \text{sendaccept}, m_3 \rangle : (\text{offer} \wedge \text{accept}) \models_{\text{sendaccept}} (\text{offer} \wedge \text{accept})$$

$$\langle m_3, \text{sendaccept}, m_1 \rangle : (\text{offer} \wedge \text{accept}) \models_{\text{sendaccept}} \text{offer}$$

$$\langle m_3, \text{sendoffer}, m_2 \rangle : (\text{offer} \wedge \text{accept}) \models_{\text{sendaccept}} \text{accept}$$

$$\langle m_3, \text{sendoffer}, m_3 \rangle : (\text{offer} \wedge \text{accept}) \models_{\text{sendaccept}} (\text{offer} \wedge \text{accept})$$

$$\langle m_3, \text{sendoffer}, m_1 \rangle : (\text{offer} \wedge \text{accept}) \models_{\text{sendaccept}} \text{offer}$$

It is easy to see that no transitions can start from m_3 , since m_3 already entails all the meanings associated with the other states.

In Figure 3.1, the solid lines show the allowed transitions and the dashed lines show the other possible entailments that are not allowed. ■

3.2.1 Compilation Formalized

This section shows how a CM can be formally compiled into an FSM, and establishes soundness and completeness results regarding our compilation procedure. The compilation can be realized in an automatic tool. Recall that a CM allows multiple starting states, whereas an FSM allows only one. In the compilation below, we show how an FSM can be constructed after choosing a start state for the CM, namely, `true`.

Definition 15 Let $X = \langle \mathbf{M}, \Delta, \mathbf{F} \rangle$ be a CM, and $Y = \langle \mathbf{S}, \Sigma, s_0, \mathbf{Q}, \delta \rangle$ be an FSM.

X is *compiled* into Y if the following conditions hold.

- $\mathbf{S} = \mathbf{M}$
- $\Sigma = \{a : \langle a, e \rangle \in \Delta\}$
- $s_0 = \text{true}$
- $\mathbf{Q} = \mathbf{F}$
- $\delta = \{\langle m_i, a, m_j \rangle : m_i, m_j \in \mathbf{M}, \langle a, e \rangle \in \Delta \text{ and } (m_i \models_{\langle a, e \rangle} m_j, m_i \not\models m_j \text{ and } (\forall m_k \in \mathbf{M}: m_i \models_{\langle a, e \rangle} m_k \Rightarrow m_j \vdash m_k))\}$

■

Notice that the definition of δ given above incorporates Restriction 1 and Restriction 2. The correctness of a compilation must be based on the computations that can result from it. Therefore, we formalize the notion of a computation, how a computation may be generated by a CM, and how a computation may be realized by an FSM. There are two main aspects of correctness. *Soundness* means that only allowed computations are realized. *Completeness* means that all allowed computations can be realized.

Definition 16 $\tau = m_0 a_0 m_1 a_1 \dots a_{f-1} m_f$ is a computation. Intuitively, the action labels, $a_0, a_1 \dots a_{f-1}$, describe the externally visible part of the computation because these are the action seen by other agents. ■

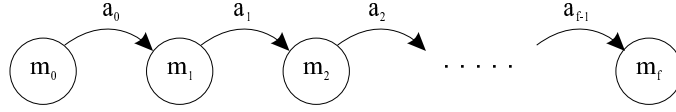


Figure 3.2: A sample computation

For the following definitions and theorems, let \mathbf{I} be the set of indices, $\{0, 1, \dots, (f-1)\}$, and \mathbf{J} be the set of indices, $\{0, 1, \dots, f\}$.

Definition 17 $\tau = m_0 a_0 m_1 \dots m_f$ is generated by a CM $X = \langle \mathbf{M}, \Delta, \mathbf{F} \rangle$ if and only if $m_f \in F$, and $(\forall i \in \mathbf{I} : m_i \in \mathbf{M} \text{ and } (\exists e_i, \langle a_i, e_i \rangle \in \Delta : m_i \models_{\langle a_i, e_i \rangle} m_{i+1}))$. ■

Definition 18 $\tau = m_0 a_0 m_1 \dots m_f$ is realized by an FSM $Y = \langle \mathbf{S}, \Sigma, s_0, \mathbf{Q}, \delta \rangle$ if and only if $m_0 = s_0, m_f \in \mathbf{Q}$, and $(\forall i \in \mathbf{I}, \langle m_i, a_i, m_{i+1} \rangle \in \delta)$. ■

Theorem 1 establishes that a compiled FSM can be directly executed by an agent with a single thread.

Theorem 1 An FSM produced by compiling a CM is deterministic.

Proof. Let $X = \langle \mathbf{M}, \Delta, \mathbf{F} \rangle$ be a CM compiled into $Y = \langle \mathbf{S}, \Sigma, s_0, \mathbf{Q}, \delta \rangle$ and let $\langle m_i, a, m_j \rangle \in \delta$ and $\langle m_i, a, m_k \rangle \in \delta$ be two transitions of Y . From Restriction 2, we know $m_j \vdash m_k$ and $m_k \vdash m_j$, that is, $m_j \equiv m_k$. Then, by Definition 10, $m_j = m_k$. ■

Theorem 2 establishes the soundness of our compilation method. Effectively, it states that the compiled FSM won't produce a computation that was not allowed by the original CM.

Theorem 2 Let $X = \langle \mathbf{M}, \Delta, \mathbf{F} \rangle$ be compiled into $Y = \langle \mathbf{S}, \Sigma, s_0, \mathbf{Q}, \delta \rangle$. Then any computation realized by Y is generated by X .

Proof. Let $\tau = m_0 a_0 m_1 \dots m_f$ be a computation generated by Y . Since $\mathbf{S} = \mathbf{M}$,

$(\forall i \in \mathbf{I}, m_i \in \mathbf{M})$, and $m_f \in \mathbf{F}$ by compilation of X into Y . Consider the i th transition in τ , $\langle m_i, a_i, m_{i+1} \rangle \in \delta$. This implies, $(\exists e_i : \langle a_i, e_i \rangle \in \Delta \text{ and } m_i \models_{\langle a_i, e_i \rangle} m_{i+1})$. ■

Interestingly, with the general definition of a CM, our compilation is not complete. For one, a computation that is generated by a CM may begin from any arbitrary meaning. Also, there may be no transitions in the FSM corresponding to some transition in the CM. Restriction 2 forces the transitions to yield a meaning that carries maximal information among the possible meanings. It might be the case that a transition emanating from a source state entails several meanings, none of which entails the rest. In this case, no meaning has the maximal information, and the transition is not included in δ . In order to enforce that there is always a meaning with the most information, we need to ensure that the meaning set \mathbf{M} of the CM is closed under antecedence.

Definition 19 A complete CM is a CM whose meaning set \mathbf{M} and final meaning set \mathbf{F} are closed under antecedence. Formally, $(\forall R \subseteq \mathbf{M}: (\forall m_i \in R : (\exists m_k \in \mathbf{M}: m_k \vdash m_i)))$ ■

Definition 20 A conjunctive CM is a CM whose meaning set \mathbf{M} is closed under conjunction. Formally, $(\forall R \subseteq \mathbf{M}: (\exists m_k \in \mathbf{M}: m_k \equiv \bigwedge_{m_i \in R} m_i))$ ■

Lemma 1 Every conjunctive CM is complete. ■

The main idea underlying a CM execution is that instead of specifying protocols in terms of legal sequences of actions, a CM specifies them in terms of possible meanings to reach. Thus, two computations may follow different sequences of actions but still achieve the same meaning.

Definition 21 A computation $\tau = m_0 a_0 m_1 \dots m_f$ is *semantically equivalent* to computation $\tau' = m'_0 a_0 m'_1 \dots m'_f$ if and only if $m_0 \equiv m'_0$ and $m_f \equiv m'_f$. ■

Definition 22 A computation $\tau' = m'_0 a_0 m'_1 \dots m'_f$ is *semantically superior* to computation $\tau = m_0 a_0 m_1 \dots m_f$ if and only if $(\forall i \in \mathbf{J}: m'_i \vdash m_i)$. ■

Definition 23 A computation $\tau' = m'_0 a_0 m'_1 \dots m'_f$ is the *semantically strongest* computation on the action sequence a_0, a_1, \dots, a_{f-1} if and only if for all computations $\tau = m_0 a_0 m_1 \dots m_f$ generated by X (that involve the given action sequence), τ' is semantically superior to τ . ■

Procedure 1 Let $\tau = m_0 a_0 m_1 \dots m_f$ be a computation generated by a complete CM $X = \langle \mathbf{M}, \Delta, \mathbf{F} \rangle$. We construct the computation $\tau' = m'_0 a'_0 m'_1 \dots m'_f$ in which $m'_0 = m_0$ and m'_{i+1} is the strongest state that follows m'_i and $\langle a_i, e_i \rangle$. By the definition of complete CM, we know that such an m'_{i+1} exists. ■

Lemma 2 Let $X = \langle \mathbf{M}, \Delta, \mathbf{F} \rangle$ be a complete CM and let $\tau = m_0 a_0 m_1 \dots m_f$ be a computation generated by X . Then Procedure 1 on τ yields a computation $\tau' = m'_0 a'_0 m'_1 \dots m'_f$ which is the semantically strongest computation on a_0, a_1, \dots, a_{f-1} .

Proof. In Procedure 1 after each action a_i at state m'_i , τ' will move to a new state m'_{i+1} , such that m'_{i+1} is the strongest state. Since each m'_i is the strongest state, τ'_i is the strongest computation for the given sequence of actions. ■

Definition 24 The computations with no consecutively repeated states are called *efficient computations*. ■

Procedure 2 Let $\tau = m_0 a_0 m_1 \dots m_f$ be a computation generated by CM $X = \langle \mathbf{M}, \Delta, \mathbf{F} \rangle$. We produce the computation $\tau' = m'_0 a'_0 m'_1 \dots m'_f$ that does not have any equivalent consecutive states. That is, we start by copying m_0 to τ' . Iteratively, we check whether applying a_i from state m_i move the computation to a new meaning m_{i+1} . If that is the case we move both a_i and m_{i+1} to τ' . Otherwise, we continue the iteration with a_{i+1} . ■

Lemma 3 Procedure 2 preserves semantic equivalence of computations and provides efficient computation.

Proof. Procedure 2 removes redundant transitions from a computation. Any transition that results in a new meaning is kept. Thus, the semantic equivalence is preserved. Since Procedure 2 removes consecutively repeated states, the computation is efficient. ■

Lemma 4 If the τ given as input to Procedure 2 is semantically strongest, the τ' produced by Procedure 2 is also semantically strongest. ■

Lemma 5 Let $X = \langle \mathbf{M}, \Delta, \mathbf{F} \rangle$ be a complete CM compiled into an FSM

$Y = \langle \mathbf{S}, \Sigma, s_0, \mathbf{Q}, \delta \rangle$. Let $\tau = m_0 a_0 m_1 \dots m_f$ be a computation generated by X , such that τ is efficient, semantically strongest, and begins from true. Then τ can be realized by Y .

Proof. By compilation, $s_0 = \text{true}$. Since τ is efficient, we know there are no consecutively repeated states, thus no transition will be disallowed by Restriction 1. Also, since τ is semantically strongest, each state in τ will be the strongest possible state. Recall that Restriction 2 enforces this requirement on transitions of FSMs. Since Restriction 1 is never exercised, and Restriction 2 does not cause deviation from the flow of τ , τ can be realized by Y . ■

Theorem 3 Let $X = \langle \mathbf{M}, \Delta, \mathbf{F} \rangle$ be a complete CM compiled into an FSM $Y = \langle \mathbf{S}, \Sigma, s_0, \mathbf{Q}, \delta \rangle$. Then for any computation that is generated by X and begins from true, there exists an efficient, semantically strongest computation realized by Y .

Proof. Let $\tau = m_0 a_0 m_1 \dots m_f$ be a computation generated by X . Let $\tau' = m_0 a_0 m'_1 \dots m'_f$ be a computation produced by Procedure 1 when given τ as input. By Lemma 2, we know τ' is semantically strongest. Now, if we give τ' as input to Procedure 2, this yields an efficient (Lemma 3) and the semantically strongest computation (Lemma 4), which can be realized by Y (Lemma 5). ■

3.2.2 Configuration

One of the main advantages of specifying e-commerce protocols using CMs is their ease of configuration. By adding new meanings to the meaning set, \mathbf{M} , and appropriately, to the final meaning set, \mathbf{F} , the protocol can be enhanced to allow more computations. Similarly, by reducing the meanings set \mathbf{M} and \mathbf{F} the possible computations can be restricted. The following example depicts how an existing CM specification can be enhanced.

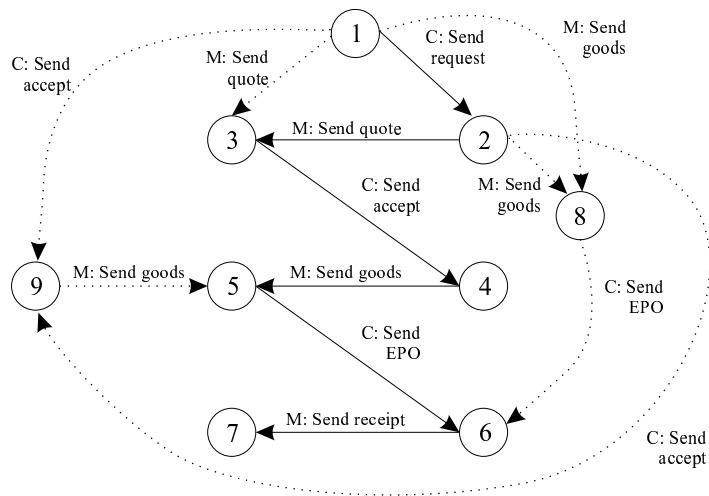


Figure 3.3: Enhanced CM

Example 9 We now reconsider the shortcomings of Example 2. In order to remedy these shortcomings, we introduce two new meanings, *accept* and $(goods \wedge promiseReceipt)$. The first meaning denotes the state that the customer is accepting any price, and the second meaning denotes the state where the merchant has delivered the goods and willing to send the receipt if the customer pays. We now add these two new meanings to the meaning set, \mathbf{M} , and the final meaning set, \mathbf{F} , of Example 7, the previous specification of our protocol. Figure 3.3 shows the enhanced FSM that can be derived from the CM specification of the NetBill protocol.

- The merchant can now start the protocol by sending a quote to a customer. As was the case in Example 7, by performing this action the merchant creates a metacommithment in state 3, namely that he is willing to send the goods and the receipt if the customer agrees to pay.
- By sending an accept message without prior conversation about the quote, the customer commits to paying if the merchant makes an offer. Thus, we move from a state where no commitments are made (state 1), to a state where the customer is making an offer (state 9). If the merchant does not make an offer, then the protocol ends at this point. On the other hand, if the merchant makes an offer by sending the goods, then both parties have to carry out their commitments, so the protocol moves to state 5.
- By sending the goods without an explicit accept message, the merchant makes an offer to send the receipt if the customer agrees to pay. Thus, we move from a state where no commitments are made (state 1), to a state where the merchant is making an offer (state 8). If the customer does not accept the offer, then the protocol ends at this point. Conversely, if the customer actually sends an accept message, then both parties need to fulfill their commitments, so the protocol moves to state 5.

Compared to the original version of the protocol in Figure 2.9, we have introduced two new states, state 8 and state 9, for the new added meanings.

- State 8: $\text{goods} \wedge \text{promiseReceipt}$
- State 9: accept

In addition to the above states, we have added new transitions from state 1 to states 3, 8, and 9; from state 2 to states 8 and 9; from state 8 to state 6 and from state 9 to state 5. The new transitions are shown with dashed lines in Figure 3.3. ■

3.3 Reasoning with CMs

Although the formal descriptions of CMs and FSMs are quite different from each other, we can use the same pictorial representation for both. Thus, we use the graphical representation in Figure 2.9 to reason about the CM specification.

We now address some of the inadequacies of the traditional approaches and show how our approach handles these situations if used at run time.

3.3.1 Opportunism

Depending on the circumstances, a participant may choose to start a protocol from the middle to take advantage of an opportunity. An opportunity in this context corresponds to a commitment-based meaning of a state that provides some convenience to the participant. By starting a protocol in this fashion, the agent accepts the commitments that are in force in that particular state. Similarly, during the execution of a protocol an agent may be able to jump across several steps at once.

3.3.2 Negotiation

During the execution of a protocol, exceptions can arise because a participant cannot fulfill his commitment for whatever reason. It is often not feasible and sometimes impossible, to predict these exceptions before they arise. Agents that are participating in a protocol should be able to handle such exceptions successfully.

If an agent can not carry out one of its commitments, then the participants need to negotiate and come to a new agreement. This may result in the ending conditions of the protocol to be changed, forcing the agents to end a protocol at a different ending state than initially identified.

3.3.3 Composition

Protocols may be combined into larger protocols, as long as one protocol yields a commitment state that another protocol can accommodate. In Section 1.2, we abstracted out the banking procedures to simplify the NetBill protocol, although there is a convention used between the merchant and the bank to deal with the EPO processing. The participants of a protocol should be able to participate in side protocols, provided that they can return to the point where they left off in the main protocol.

Example 10 If we consider the banking operations as a separate protocol, we can see that the banking protocol may be combined with our protocol through state 6. Recall that at state 6, the customer has discharged his commitment of sending an EPO, but the merchant still has to send the receipt. The merchant at this state can participate in a side protocol with a bank to verify that the EPO has actually cleared. After taking part in this side protocol, the merchant will come back to state 6 to send the receipt to the customer. Since the commitments of the main protocol are preserved in the side protocol, the composition of the two protocols yields a valid protocol. ■

3.3.4 Factoring and Regrouping

A protocol can be thought of a composition of several subprotocols that are combined as described in the last subsection. A subprotocol can be substituted for another subprotocol if the substituted subprotocol can provide the same content as the replaced subprotocol, that is, if the same commitments are in force at the start and at the end as in the original subprotocol. The factoring of protocols provides a natural means to practice different subprotocols for different operations in a protocol.

Example 11 If we think about the steps of request-quote-accept as a subprotocol, we may be able to use a more sophisticated negotiation protocol as a subprotocol. As long as both subprotocols provide the same commitments with respect to the validity of the quoted price, the factoring can be carried out. ■

A protocol specification that allows the above characteristics can drastically improve the flexibility and thus the performance of the protocols. At this point, it is important to restate what we mean by flexibility. Although we want the agents to practice a flexible protocol, we still want to preserve an ordering that will allow only meaningful conversations. For example, a merchant should not send a quote after sending the goods, or the customer should not start the conversation by sending an EPO. More importantly, flexibility can be introduced only to the point where the intended meanings of the actions are not violated. The interactions among the parties can be given a meaning based on their commitments to each other. When we allow more flexible interactions, we need to ensure that the original commitments are in force or that they are altered by mutual agreement. The execution of a protocol is then minimally constrained only to satisfy those metacommitments. This is a major advantage over low-level representations, which require specific execution sequences and provide no basis for deciding on the correct state independent of the execution sequence.

Chapter 4

Commitments in Event Calculus

The event calculus provides a natural way to reason about commitments. Commitments, as emphasized before, result from communicative actions. The event calculus is based on the idea of events initiating and terminating properties in the world. Thus, if we represent commitments as properties, the event calculus provides a foundation for modeling 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.

4.1 Representing Commitments in Event Calculus

The main difference between an obligation and a commitment is that commitments are more flexible. Unlike an obligation, a commitment may be resolved even without bringing about the action committed to. In what follows, we present a formal account of the operations that can be performed to create and manipulate commitments.

Commitments can be represented in simple event calculus (SEC) as a fluent. The operations on commitments that were defined conceptually in Section 2.1 can be formalized in SEC as follows:

1. *Create*(x, c) establishes the commitment c .

$$\textit{Create}(x, \mathbf{C}(x, y, G, p)) : \exists e \{ \textit{Happens}(e, t) \wedge \textit{Initiates}(e, \mathbf{C}(x, y, G, p), t) \}$$

2. *Discharge*(x, c) resolves the commitment c .

$$\textit{Discharge}(x, \mathbf{C}(x, y, G, p)) : \exists e \{ \textit{Happens}(e, t) \wedge \textit{Initiates}(e, p, t) \}$$

3. *Cancel*(x, c) cancels the commitment c .

$$\textit{Cancel}(x, \mathbf{C}(x, y, G, p)) : \exists e \{ \textit{Happens}(e, t) \wedge \textit{Terminates}(e, \mathbf{C}(x, y, G, p), t) \}$$

4. *Release*(G, c) or *Release*(y, c) releases the debtor from the commitment c .

$$\textit{Release}(y, \mathbf{C}(x, y, G, p)) : \exists e \{ \textit{Happens}(e, t) \wedge \textit{Terminates}(e, \mathbf{C}(x, y, G, p), t) \}$$

5. *Assign*(y, z, c) eliminates the commitment c , and creates a new commitment c' for which z is appointed as the new creditor.

$$\textit{Assign}(y, \mathbf{C}(x, y, G, p)) : \exists e \{ \textit{Happens}(e, t) \wedge \textit{Terminates}(e, \mathbf{C}(x, y, G, p), t) \wedge \textit{Initiates}(e, \mathbf{C}(x, z, G, p), t) \}$$

6. *Delegate*(x, z, c) eliminates the commitment c , and creates a new commitment c' in which the role of the debtor is transferred to z .

$$\textit{Delegate}(x, \mathbf{C}(x, y, G, p)) : \exists e \{ \textit{Happens}(e, t) \wedge \textit{Terminates}(e, \mathbf{C}(x, y, G, p), t) \wedge \textit{Initiates}(e, \mathbf{C}(z, y, G, p), t) \}$$

These six operations are used in creating and manipulating commitments. Recall that in Section 3.1 we described our reasoning rules informally. The following three postulates formalize those reasoning rules.

Postulate 1 formalizes Reasoning Rule 1, that is, a commitment is no longer in force if the condition committed to holds.

Postulate 1 $Terminates(e, C(x, y, G, p), t) \leftarrow HoldsAt(C(x, y, G, p), t) \wedge Happens(e, t) \wedge Initiates(e, p, t)$ ■

For the condition p to hold, an event must occur to initiate it. In Postulate 1, when the event e occurs at time t , it initiates the property p , and therefore the commitment $C(x, y, G, p)$ can be terminated.

In order to represent metacommitments in SEC, we introduce a fluent $meta(p, q)$ where q may refer to other commitments. Recall that in Chapter 3, we introduced the strict implication among meanings ($p \rightsquigarrow q$). We use the fluent $meta(p, q)$ for the same purpose. Thus, the interpretation of $meta(p, q)$ is the same as the interpretation of ($p \rightsquigarrow q$), in that it requires q to hold when p holds and the fluent itself to be false if p is false. Using this fluent, we represent a metacommitment in SEC by $C(x, y, G, meta(p, q))$ meaning that the debtor of x becomes committed to bring about q when p is true.

Postulates 2 and 3 represent Reasoning Rules 2 and 3, respectively.

Postulate 2 $Initiates(e, C(x, y, G, q), t) \wedge Terminates(e, C(x, y, G, meta(p, q)), t) \leftarrow HoldsAt(C(x, y, G, meta(p, q)), t) \wedge Happens(e, t) \wedge Initiates(e, p, t)$ ■

Recall that in Reasoning Rule 2, when p becomes true, then the original metacommitment ceases to exist but a new base-level commitment is created, since the debtor x is now committed to bring about q . In Postulate 2, when the event e occurs, it initiates p , which results in the termination of the original metacommitment, and the initiation of $C(x, y, G, q)$.

Postulate 3 $Terminates(e, C(x, y, G, meta(p, q)), t) \leftarrow HoldsAt(C(x, y, G, meta(p, q)), t) \wedge Happens(e, t) \wedge Initiates(e, q, t)$ ■

Postulate 3 follows Reasoning Rule 3. Thus, when an event e that can initiate q occurs, q starts to hold and the original metacommitment is terminated. Since the creditor y has not committed to anything, no additional commitments are created.

Having represented our reasoning mechanism about commitments, we now look at representation of protocols in SEC.

4.2 Representing Protocols in Event Calculus

To represent a protocol in SEC, we need to represent the flow of execution within the protocol. In SEC, 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* clauses.

Definition 25 A *protocol description* is a set of *Initiates* and *Terminates* clauses that define the preconditions under which properties pertaining to the protocol are initiated and terminated. ■

Given a protocol description, a computation that is generated by executing the protocol constitutes a protocol run. In SEC, any computation is a result of execution of events at different timepoints.

Definition 26 A *protocol run* is a set of *Happens* predicates along with an ordering of the timepoints referred to in the predicates. ■

Definition 27 A protocol run is *complete* if all the base-level commitments that have been created are resolved as formalized by Postulate 4. ■

Postulate 4 $\exists e\{\text{Happens}(e, t) \wedge \text{Terminates}(e, C(x, y, G, p), t)\} \leftarrow$
 $\text{HoldsAt}(C(x, y, G, p), t)$ ■

If a protocol run is not complete, that is, if there is an open base-level commitment after the execution of the protocol, we know that a participant has not fulfilled its commitment. This signals a violation of the protocol. Although we don't investigate the issue of protocol compliance by agents in this work, incomplete protocol runs provide evidence to identify non-compliant agents.

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

Example 12 The actions (messages) of Figure 2.9 can be given a content based on the following definitions.

- **Roles and context:**

- *MR* represents the merchant.
- *CT* represents the customer.
- *NB* represents the context group for the NetBill protocol.

- **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*.

- **Metacommitments:**

- $accept(i, m)$: an abbreviation for $C(CT, MR, NB, meta(goods(i), pay(i)))$ meaning that the customer is willing to pay if he receives the goods.
- $promiseGoods(i, m)$: an abbreviation for $C(MR, CT, NB, meta(accept(i, m), goods(i)))$ 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 $C(MR, CT, NB, meta(pay(m), receipt(i)))$ meaning that the merchant is willing to send the receipt if the customer pays the agreed-upon amount.
- $offer(i, m)$: an abbreviation for $(promiseGoods(i, m) \wedge promiseReceipt(i, m))$
- **Definition of Initiates in the NetBill Protocol:**
 - $Initiates(sendRequest(i), request(i), t)$
 - $Initiates(sendQuote(i, m), promiseGoods(i, m), t)$
 - $Initiates(sendQuote(i, m), promiseReceipt(i, m), t)$
 - $Initiates(sendAccept(i, m), accept(i, m), t)$
 - $Initiates(sendGoods(i, m), goods(i), t)$
 - $Initiates(sendGoods(i, m), promiseReceipt(i, m), t) \leftarrow \neg holdsAt(promiseReceipt(i, m), t)$
 - $Initiates(sendEPO(i, m), pay(m), t) \leftarrow HoldsAt(goods(i), t)$
 - $Initiates(sendReceipt(i, m), receipt(i), t) \leftarrow HoldsAt(pay(m), t)$
- **Definition of terminates:**
 - $Terminates(sendQuote(i, m), request(i)) \leftarrow HoldsAt(request(i), t)$

■

Having provided the protocol description, we now look at an example protocol run.

Example 13 The protocol run shown in Figure 1.4 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. $t_2 < t_3$

Now we look at how the commitments among the participants evolve in the given protocol run.

1. $HoldsAt(C(MR, CT, NB, meta(pay(m), receipt(i))), t_1) \wedge HoldsAt(goods(i), t_1)$

When the goods are sent at time t_1 , the fluent $goods(i)$ is initiated. Furthermore, following the protocol definition in Example 12, the metacommitment $C(MR, CT, NB, meta(pay(m), receipt(i)))$ is created. So now, the goods have been delivered, and the merchant is willing to send the receipt if the customer pays.

2. $HoldsAt(C(MR, CT, NB, receipt(i)), t_2) \wedge HoldsAt(goods(i), t_2) \wedge HoldsAt(pay(i), t_2)$

By sending the EPO at time t_2 , the customer *initiates* the fluent $pay(m)$. By Postulate 2, this ends the metacommitment —sl $C(MR, CT, NB, meta(pay(m), receipt(i)))$ and creates the commitment $C(MR, CT, NB, receipt(i))$. Since no event occurred to terminate $goods(i)$ it continues to hold.

3. $HoldsAt(goods(i), t_3) \wedge HoldsAt(pay(i), t_3) \wedge HoldsAt(receipt(i), t_3)$

At time t_3 the third fact is applicable, which initiates the fluent $receipt(i)$. Following Postulate 1, this terminates the commitment $C(MR, CT, NB, receipt(i))$. Thus, we reach the state where the merchant has delivered the goods and the receipt, and the customer has paid.

■

4.3 Planning

Planning is the construction of plans for automatic or semiautomatic execution by an agent. We consider plans that would lead from an initial state to a final state. In planning, the operators are defined to allow transitions between states. Traditionally, states are described as lists of fluents, each interpreted as holding in the given state. Operators are specified in a fashion where the precondition for an operator to be applied and the effect of applying the operator are defined. The effect of an operator can be captured through the addition and removal of fluents from the representation of the source state in which the operator is applied; this is the well-known STRIPS approach.

Planning is carried out by either progression or regression planners. Progression planners start with the initial state and apply all possible operators until the goal state is reached; such planners suffer from high branching factor. Regression planners, on the other hand, start with the goal state and search backwards to get to the initial state [Russell and Norvig, 1995].

4.3.1 Planning in Event Calculus

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. The goal state is represented by conjunctive *HoldsAt* clauses. Operators are the events in the domain. The domain definition containing *Initiates* and *Terminates* clauses naturally specifies the preconditions and effects of the actions. Having the initial states, the goal states and the domain description, an event calculus planner can generate a narrative that contains *Happens* clauses, and an ordering of time points of the events that occur [Shanahan, 2000].

The form of logical reasoning that is commonly used in building event calculus planners is abduction. Abduction, unlike deduction, is based on an application of implications in reverse. In other words, given two well-formed formulae, $A \Rightarrow B$ and B , A can be assumed via abduction as long as it is consistent with the rest of the knowledge base [Rich and Knight, 1991]. One such abductive event calculus planner is due to Shanahan [2000]; we use it here to demonstrate how a protocol can be defined based on the preconditions and the effects of actions, and how possible paths can be generated between an initial state and a goal state.

4.3.2 Generating Paths

Given a protocol description, planning can be used to generate all possible runs of the protocol. We now present a sample SEC program that can be successfully interpreted by Shanahan's SEC planner.

Figure 4.1 gives the clauses to set up the initial state of the protocol. As we have

explained in Section 2.2, the initial state is represented with *Initially_P* and *Initially_N* predicates, to denote the positive and negative fluents, respectively. In the planner, though, only one predicate, *Initially*, has been implemented. In order to indicate negative fluents, a new predicate *neg* has been introduced.

```
axiom(initially(neg(goods(I, D))), []).
axiom(initially(neg(pay(M, D))), []).
axiom(initially(neg(receipt(I, D))), []).
axiom(initially(neg(c(receipt(I, D)))), []).
axiom(initially(neg(c(goods(I)))), []).
axiom(initially(neg(c(pay(M, D)))), []).
axiom(initially(neg(accept(I, M, D))), []).
axiom(initially(neg(request(I, D))), []).
axiom(initially(neg(offer(I, M, D))), []).
axiom(initially(neg(promisereceipt(I, M, D))), []).
```

Figure 4.1: Description of an initial state

Figure 4.2 specifies the first part of the protocol description that is, the *Initiates* clauses. 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 fluents used here are the same as the ones used in Example 12, except 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. The reasoning rules explained before are manually compiled here.

Figure 4.3 specifies the second part of the protocol description, that is, the *Terminates* clauses. Again, the axioms have the same format as in Figure 4.2, where the first argument is the *Terminates* clause and the second argument is the set of preconditions.

Figure 4.4 simply gives the set of actions in the program through the clause *executable*.

When the clauses of Figures 4.1, 4.2, 4.3, and 4.4 are appended, they form a complete SEC program. We invoke the planner and this program from a Prolog interpreter. In order

```

axiom(initiates(sendrequest(I, D),request(I, D),T),
      [holds_at(neg(goods(I, D)), T),
       holds_at(neg(request(I, D)), T)]).
axiom(initiates(sendquote(I, M, D),offer(I, M, D),T),
      [holds_at(neg(goods(I, D)), T),
       holds_at(neg(offer(I, M, D)), T),
       holds_at(neg(accept(I, M, D)), T)]).
axiom(initiates(sendgoods(I, M, D),goods(I, D),T),
      [holds_at(c(goods(I, D)), T),
       holds_at(neg(goods(I, D)), T)]).
axiom(initiates(sendgoods(I, M, D),c(pay(M, D)),T),
      [holds_at(accept(I, M, D), T),
       holds_at(neg(goods(I, D)), T)]).
axiom(initiates(sendgoods(I, M, D),c(pay(M, D)),T),
      [holds_at(c(goods(I, D)), T),
       holds_at(neg(goods(I, D)), T)]).
axiom(initiates(sendgoods(I, M, D),goods(I, D),T),
      [holds_at(accept(I, M, D), T),
       holds_at(neg(goods(I, D)), T)]).
axiom(initiates(sendgoods(I, M, D),goods(I, D),T),
      [holds_at(offer(I, M, D),T),
       holds_at(neg(goods(I, D)), T)]).
axiom(initiates(sendgoods(I, M, D),goods(I, D),T),
      [holds_at(neg(goods(I, D)), T)]).
axiom(initiates(sendgoods(I, M, D),
               promisereceipt(I, M, D),T),
      [holds_at(neg(goods(I, D)), T)]).
axiom(initiates(senddepo(M, D),pay(M, D),T),
      [holds_at(goods(I, D), T), holds_at(neg(pay(M, D)), T)]).
axiom(initiates(senddepo(M, D), c(receipt(I, D)),T),
      [holds_at(promisereceipt(I, M, D), T),
       holds_at(neg(pay(M, D)), T)]).
axiom(initiates(sendaccept(I, M, D),c(goods(I, D)),T),
      [holds_at(offer(I, M, D), T)]).
axiom(initiates(sendaccept(I, M, D),
               promisereceipt(I, M, D),T),
      [holds_at(offer(I, M, D), T)]).
axiom(initiates(sendaccept(I, M, D),accept(I, M, D),T),
      [holds_at(neg(offer(I, M, D)), T)]).
axiom(initiates(sendreceipt(I, D),receipt(I, D),T),
      [holds_at(pay(M, D), T)]).

```

Figure 4.2: Description of *Initiates* clauses

```

axiom(terminates(sendgoods(I, M, D),c(goods(I, D)),T),
      [holds_at(c(goods(I, D)), T)]).
axiom(terminates(sendepo(M, D),c(pay(M, D)),T),
      [holds_at(c(pay(M, D)), T)]).
axiom(terminates(sendreceipt(I, D),c(receipt(I, D)),T),
      [holds_at(c(receipt(I, D)), T)]).
axiom(terminates(sendepo(M, D),promisereceipt(I, M, D),T),
      [holds_at(promisereceipt(I, M, D), T)]).
axiom(terminates(sendquote(I, M, D),request(I, D),T),[]).
axiom(terminates(sendgoods(I, M, D),accept(I, M, D),T),
      [holds_at(accept(I, M, D), T)]).
axiom(terminates(sendaccept(I, M, D),offer(I, M, D),T),
      [holds_at(offer(I, M, D), T)]).

```

Figure 4.3: Description of *Terminates* clauses

```

abducible(dummy).
executable(sendrequest(I, D)).
executable(sendquote(I, M, D)).
executable(sendgoods(I, M, D)).
executable(sendaccept(I, M, D)).
executable(sendepo(M, D)).
executable(sendreceipt(I, D)).

```

Figure 4.4: Description of executable events

to interpret the code, we provide the description of the final state; Figure 4.5 shows such an example. The final state depicted in Figure 4.5 means that goods (software) have been sent with transaction identifier 51. This yields the result shown in Figure 4.6, where each **R** is a possible protocol run starting from the initial state depicted in Figure 4.2, and ending in the final state depicted in Figure 4.5.

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

Figure 4.5: Description of a final state

As described before, each protocol run consists of *Happens* clauses and an ordering of time points. The last argument in each *Happens* clause denotes a timepoint at which the event happens. The ordering of these time points are then shown with the *before* clauses.

```
R = [[happens(sendquote(software,H816,51),t191),
        happens(sendaccept(software,H816,51),t190),
        happens(sendgoods(software,H601,51),t189)],
      [before(t191,t),before(t191,t189),before(t191,t190),
        before(t190,t),before(t190,t189),before(t189,t)]] ;

R = [[happens(sendaccept(software,H601,51),t193),
        happens(sendgoods(software,H601,51),t192)],
      [before(t193,t),before(t193,t192),before(t192,t)]] ;
```

Figure 4.6: Protocol runs

Chapter 5

Conclusions

Given the open, dynamic nature of e-commerce, employing an agent-based approach to model and enact the trading is natural. Agents can provide the autonomy and heterogeneity required from participants in e-commerce settings. However, in order to fully exploit their capabilities, agents should be provided with a communication framework that allows them to interact flexibly. Thus, protocols they follow should be flexible enough to accommodate the varying interactions that can take place among agents.

Traditionally, communication protocols are specified by defining the allowed orders in which communicative acts may take place, but no more. This holds for protocol formalisms such as FSMs, formal grammars, Petri Nets, and so on. Some of these formalisms are quite powerful, but they are used only to specify allowed actions. The actions are just labels, and the states, if explicit, do not capture the conceptual meaning of a protocol.

This research proposes a novel formalism, the CFSM, to specify and execute the protocols used in e-commerce. CFSMs provide flexibility by capturing the semantic content of the actions in a protocol, rather than capturing only their sequencing. The meaning of actions and states are represented via social commitments. By specifying communication protocols using commitments, we can analyze the interactions among participants through

the intrinsic meaning of those interactions. By walking through an example, we showed how we can define a CFSM specification, and how we can reason about the protocols and enhance them to achieve flexible execution. We also showed how this specification can be applied at run time or compile time, depending on the computational borders or agent architectures.

In adopting the language-as-action perspective, we can leverage the event calculus to reason about actions and commitments. The event calculus provides an elegant way to represent the change of the world through the actions in a protocol. Using the event calculus, we have shown how commitments and the operations to create and manipulate them can be formalized. Further, we have formalized the reasoning rules associated with commitments. Based on this formal grounding, e-commerce protocols can be specified rigorously yet flexibly.

5.1 Literature

There is a substantial body of literature on agent communication languages (ACLs) and their semantics. The Foundation for Intelligent Physical Agents (FIPA) has been standardizing an ACL along with a formal semantics based on a logic of beliefs and intentions. FIPA also includes interaction protocols, which are characterized purely operationally. Labrou and Finin [1998] describe a grammar for constructing conversations or protocols. This is a formal grammar but it only describes the sequencing of tokens. Labrou and Finin also give a belief and intention based semantics for Knowledge Query Manipulation Language (KQML). There is thus a major disconnect between the FIPA ACL semantics and protocol operationalization, and Labrou and Finin's ACL semantics and protocol execution. By contrast, in our approach the semantics given to the messages is directly operationalized. In

our approach, the tokens are chosen for each protocol. We don't attempt to give a semantics for tokens like *inform* that can apply in all possible protocols. More recently, Moore [2000] has developed an ACL, FLBC, for business communication that attempts to remedy the potential problems with KQML, but FLBC has not been applied as widely as KQML.

Commitments have been studied before [Castelfranchi, 1995, Gasser, 1998], but were not used for protocol specification as we have done here. Verharen [1997] develops a contract specification language, CoLa, to specify transactions and contracts. Verharen's approach benefits from commitments in expressing actions, but it treats commitments as obligations, and does not allow manipulation of commitments as in our approach. Further, Verharen only considers base-level commitments, without capturing conditional commitments as we have done through metacommitments.

Barbuceanu and Fox [1995] develop a language, COOL, for describing coordination among agents. Their approach is based on modelling conversations through FSMs, where the states denote the possible states a conversation can be in, and the transitions represent the flow of the conversation through message exchange. They try to handle exceptions through *error recovery rules*. They allow *interruption of conversations*, which is similar to our definition of *composition* of protocols. They give content to messages based on speech acts, but they only allow pre-defined transitions, disallowing dynamic transitions based on the meaning of messages.

Dignum and van Linder [1997] propose a framework for social agents based on dynamic logic, in which they distinguish messages based on speech acts. In addition to employing commitments, they use *directions* and *declarations* to denote the semantic content of messages. Furthermore, they employ an authority relation between agents to decide on the success of directions and declarations. In our work, we assumed peer-to-peer interactions, in that we do not consider the interactions based on different authority among agents.

But it would be a simple matter to handle various authority relations, e.g., to require that requests from one agent are always honored by the other.

Enabling flexible communication among agents is a two-pronged issue. In one direction, the protocols used should be flexible enough to accommodate the dynamic nature of interactions. This is what we have tried to develop in this research. The other direction is to develop agent architectures that can benefit from the flexibility of such protocols. In this direction, Haddadi [1998] develops a formal semantics based on beliefs, desires and intentions of agents. She describes the *means-ends reasoning* process of agents, through the concepts of *chance*, *choice* and *commitment*. Her description of *chance* is similar to our description of *opportunity*, but her treatment of commitments varies from our treatment, in that she focuses on formation of commitments whereas we deal with the utilization of commitments.

Event calculus has been theoretically studied but has not been used for modelling commitments and communication protocols as we have done here. Shanahan [1996] uses event calculus in robotics for coordinating robot actions. We were able to adapt Shanahan's planner to apply to reasoning about commitments. Cervesato et al. [1997] study event calculus theory and have developed modal variants of the calculus.

5.2 Future Directions

The approach we develop here is based on the widely-used finite state machine formalism. It would be worthwhile to incorporate commitments into other formalisms, such as Petri Nets, to evaluate how much these formalisms would benefit from commitments.

In our current approach, the commitments that are associated with actions are specified

manually. Another direction would be attempting to automate the definitions of commitments in a given protocol.

Although we believe in the importance of context group in commitments, this study has not investigated how commitments would be modified based on varying context group. Yet another direction would be extending the reasoning rules we have developed here, to handle the transformation of commitments as the context group changes. These are all interesting topics, which we defer to future research.

Bibliography

Mihai Barbuceanu and Mark S. Fox. COOL: A language for describing coordination in multi agent systems. In *Proceedings of the International Conference on Multiagent Systems*, pages 17–24, 1995.

Cristiano Castelfranchi. Commitments: From individual intentions to groups and organizations. In *Proceedings of the International Conference on Multiagent Systems*, pages 41–48, 1995.

Iliano Cervesato, Massimo Franceschet, and Angelo Montanari. Modal event calculi with preconditions. In R. Morris and L. Khatib, editors, *Fourth International Workshop on Temporal Representation and Reasoning — TIME'97*, pages 38–45, Daytona Beach, FL, 1997. IEEE Computer Society Press.

Rosaria Conte and Cristiano Castelfranchi. *Cognitive and Social Action*. UCL Press, London, 1995.

Frank Dignum and Bernd van Linder. Modelling social agents: Communication as action. In *Intelligent Agents III: Agent Theories, Architectures, and Languages*, pages 205–218. Springer-Verlag, 1997.

- Michael Fisher and Michael Wooldridge. Specifying and executing protocols for cooperative action. 6(1):37–65, 1997.
- Les Gasser. Social conceptions of knowledge and action: DAI foundations and open systems semantics. In *Huhns and Singh [1998]*, pages 389–404. 1998. (Reprinted from *Artificial Intelligence, 1991*).
- Afsaneh Haddadi. Towards a pragmatic theory of interactions. In *Huhns and Singh [1998]*, pages 443–449. 1998. (Reprinted from *Proceedings of the International Conference on Multiagent Systems, 1995*).
- Michael N. Huhns and Munindar P. Singh, editors. *Readings in Agents*. Morgan Kaufmann, San Francisco, 1998.
- Steven O. Kimbrough and Ronald M. Lee. On formal aspects of electronic (or digital) commerce: Examples of research issues and challenges. In *Proceedings of the 29th Annual Hawaii International Conference on System Sciences*, pages 319–328, 1996.
- Robert Kowalski and Marik Sergot. Logic-based calculus of events. *New Generation Computing*, 4(1):67–95, 1986.
- Yannis Labrou and Tim Finin. Semantics and conversations for an agent communication language. In *Huhns and Singh [1998]*, pages 235–242. 1998. (Reprinted from *Proceedings of the International Joint Conference on Artificial Intelligence, 1997*).
- Scott A. Moore. KQML & FLBC: Constrasting agent communication languages. *International Journal of Electronic Commerce*, 2000. To appear.
- Elaine Rich and Kevin Knight. *Artificial Intelligence*. McGraw-Hill, New Jersey, 2nd edition, 1991.

- Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Singapore, 1995.
- Murray Shanahan. Robotics and the commonsense informatic situation. In *Proceedings of the European Conference on Artificial Intelligence*, pages 684–688, 1996.
- Murray Shanahan. *Solving the Frame Problem: A Mathematical Investigation of the Common Sense Law of Inertia*. MIT Press, Cambridge, 1997.
- Murray Shanahan. An abductive event calculus planner. *Journal of Logic Programming*, 44:207–239, 2000.
- Munindar P. Singh. An ontology for commitments in multiagent systems: Toward a unification of normative concepts. *Artificial Intelligence and Law*, 7:97–113, 1999.
- Marvin A. Sirbu. Credits and debits on the internet. In *Huhns and Singh [1998]*, pages 299–305. 1998. (Reprinted from *IEEE Spectrum*, 1997).
- Usenix ecommerce '98: The third usenix workshop on electronic commerce, 1998. Held September 1998 in Boston, MA.
- Mahadevan Venkatraman and Munindar P. Singh. Verifying compliance with commitment protocols: Enabling open web-based multiagent systems. *Autonomous Agents and Multi-Agent Systems*, 2(3):217–236, 1999.
- Egon M. Verharen. *A Language-Action Perspective on the Design of Cooperative Information Agents*. Catholic University, Tilburg, Holland, 1997.
- Terry Winograd and Fernando Flores. *Understanding Computers and Cognition: A New Foundation for Design*. Ablex Publishing, Norwood, New Jersey, 1986.