Enhancing Commitment Machines

Michael Winikoff[1], Wei Liu[2], and James Harland[1]

[1] RMIT University, Melbourne, AUSTRALIA
{winikoff,jah}@cs.rmit.edu.au
[2] University of Western Australia, Perth, AUSTRALIA
wei@csse.uwa.edu.au

Abstract. Agent interaction protocols are usually specified in terms of permissible sequences of messages. This representation is, unfortunately, brittle and does not allow for flexibility and robustness. The *commitment machines* framework of Yolum and Singh aims to provide more flexibility and robustness by defining interactions in terms of the commitments of agents. In this paper we identify a number of areas where the commitment machines framework needs improvement and propose an improved version. In particular we improve the way in which commitments are discharged and the way in which pre-conditions are specified.

1 Introduction

Communications between software agents are typically regulated by interaction protocols. These include general communication protocols, such as the auction protocol and the contract net protocol, as well as more specific protocols such as the NetBill payment protocol [7, 8]. Traditional protocol representations such as Finite State Machines (FSM), Petri-Nets [3] and AUML sequence diagrams [1, 2] often specify protocols in terms of legal message sequences. Under such protocol specifications, agent interactions are pre-defined and predictable. The inevitable rigidity resulting from this prevents agents from taking opportunities and handling exceptions in a highly dynamic and uncertain multi-agent environment.

Yolum and Singh's Commitment Machines [7] (CMs henceforth) define an interaction protocol in terms of actions that change the state of the system, which consists of the state of the world as well as the *commitments* that agents have made to each other. It is a commitment made to an interaction partner which makes an agent perform its next action. In other words, an agent acts because it wants to comply with the protocol and provide the promised outcomes for another party. Actions not only change the values of state variables, but also may initiate new commitments and/or discharge existing commitments. In traditional protocol representations, agents are constrained to perform a pre-defined sequence of actions, whereas in CMs, an agent is able to reason about what action should be taken next in accordance with the dynamics of the environment and the management of its commitments in that environment. This fundamentally changes the process of specifying a protocol from a procedural approach (i.e. prescribing *how* an interaction is to be executed) to a declarative one (i.e. describing *what* interaction is to take place) [7].

Another advantage of the CM approach is that it provides a natural means of managing multi-agent interactions. Agent programming concepts are often discussed in the context of a single agent situated in an environment, discussing properties such as autonomy, pro-activeness, reactivity and social awareness. The CM approach enables pro-activeness and reactivity to be discussed in a multi-agent context.

CMs thus allow interactions between agents to be organized in a manner which is more flexible and robust than an approach based on pre-defined sequences. For example, in the NetBill protocol (discussed in section 2), a customer may wish to order goods without first receiving a quotation, or a merchant may be happy to send goods to a known reliable customer with less rigorous checking than normal.

In this paper we identify a number of areas where the Commitment Machine framework can be improved. Specifically, we show how the identification of undesirable states (such as omitting to provide a receipt, or receiving the goods before payment has been confirmed) can be incorporated into the design process in order to achieve acceptable outcomes for a wider variety of circumstances than is done in [7, 8]. We also show how certain anomalies in discharging commitments and in handling pre-conditions can be remedied.

The paper is organized as follows: in section 2 we introduce the commitment machine framework and a detailed example, both based on [7]. In section 3 we identify a number of anomalies and issues with the commitment machines framework and in section 4 we propose some improvements.

2 Background

We briefly introduce the commitment machines framework and the NetBill protocol. Both are based on the description in [7] and we refer the reader to [7, 8] for further details.

The key example used in [7] is the NetBill protocol [4]. In this protocol a customer buys a product from a merchant. To buy a desired product, the protocol begins with a customer (C) requesting a quote (message 1 in Figure 1) from the merchant (M), followed by the merchant sending the quote (message 2). If the customer accepts the quote (message 3), the merchant proceeds by sending the goods (message 4) and waits for the customer to pay by sending an electronic payment order (EPO). Note that it is assumed that the goods cannot be used until the merchant has sent the relevant decryption key, such as software downloaded from the internet, or sent on a CD. Once the customer has sent payment (via an EPO in message 5), the merchant will send the decryption key along with a receipt (message 6). This concludes the NetBill transaction.

As suggested by the name "commitment machine", a crucial concept is that of commitment. A (social) commitment is an undertaking by one agent (the *debtor*, x) to another agent (the *creditor*, y) to bring about a certain property p, written $\mathsf{C}(x, y, p)$. A commitment of the form $\mathsf{C}(x, y, p)$ is a *base-level* commitment. For example, in the NetBill protocol when the customer sends message 3 and then receives the goods, he or she has a commitment to pay the merchant, i.e. $\mathsf{C}(C, M, pay)$.

When a party is willing to commit only if certain conditions hold (such as another party making a corresponding commitment), a *conditional commitment* can be used.

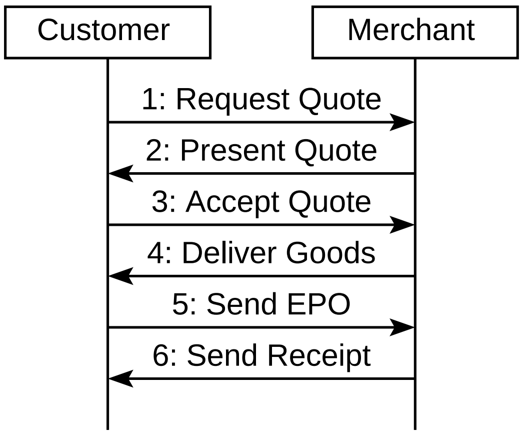

Fig. 1. Simplified Net Bill Protocol

A conditional commitment, denoted $\mathsf{CC}(x, y, p, q)$, indicates that agent x is committed to achieving q for agent y if p becomes true. A conditional commitment is latent – it doesn't commit x to do anything until p becomes true, at which point the conditional commitment is transformed to the base-level commitment $\mathsf{C}(x, y, q)$. For example, in the NetBill protocol the customer may insist on his or her commitment to pay being conditional on the goods being sent, which would be represented as $\mathsf{CC}(customer, merchant, goods, pay)$. Where the identity of the debtor and creditor are obvious from context we shall sometimes write $\mathsf{C}(p)$ in place of $\mathsf{C}(x, y, p)$ and $\mathsf{CC}(p \rightsquigarrow q)$ in place of $\mathsf{CC}(x, y, p, q)$.

Interactions are specified in the CM framework by defining the roles of the participants, the domain-specific fluents (i.e. boolean state variables), the (conditional) commitments that may arise during the interaction, and the rules for *initiates* and *terminates* which define the effects of (communicative) actions, and are used to regulate the choices of actions for the agents. The execution of a protocol is then driven by the commitments that are in place: the desire to fulfil these commitments generates an action or actions to achieve them, which in turn may create new commitments or discharge existing ones. The NetBill protocol as a CM can be found in figure 2.

A *state* in a CM is a triple $\langle F, CC, C \rangle$, where F is a set of fluents, CC is a set of conditional commitments and C is a set of base-level commitments.

A *final state* is a state that does not have undischarged base-level commitments. A final state may contain conditional commitments, since they are latent commitments that have not been activated. Formally, a state in a CM is a final state if $C = \emptyset$. Note that a final state in a CM is one where the interaction *may* end. However, it is also possible for interaction to continue from a final state.

A *protocol run* consists of a sequence of actions that results in a final state.

A commitment machine places constraints on the sequence of agent actions that constitute the interaction. For example, if an agent has a commitment, then it must at

Roles: M (merchant), C (customer)
Fluents:

– $request$ (the customer has requested a quote),
– $goods$ (the goods have been delivered to the customer),
– pay (the customer has paid),
– $receipt$ (the merchant has sent the receipt)

Commitments:

– $accept = \mathsf{CC}(C, M, goods, pay)$: a commitment by the customer (to the merchant) to pay once the goods have been delivered.
– $promiseGoods = \mathsf{CC}(M, C, accept, goods)$: a commitment by the merchant to send the goods if the customer accepts. Since $accept$ is itself a commitment this is a nested commitment: $promiseGoods = \mathsf{CC}(M, C, \mathsf{CC}(C, M, goods, pay), goods)$.
– $promiseReceipt = \mathsf{CC}(M, C, pay, receipt)$: a commitment by the merchant to send a receipt once the customer has paid.
– $offer = promiseGoods \wedge promiseReceipt$: an offer is a commitment by the merchant (a) to send the goods if the customer accepts the offer, and (b) to send a receipt after payment has been made.

Action Effects: the following (communicative) actions are defined:

– $sendRequest$: this action by the customer makes the fluent $request$ true.
– $sendQuote$: this action by the merchant creates the two commitments $promiseGoods$ and $promiseReceipt$ (i.e. $offer$) and terminates (makes false) the fluent $request$.
– $sendAccept$: this action by the customer creates the commitment $accept$.
– $sendGoods$: this action by the merchant makes the fluent $goods$ true and also creates the commitment $promiseReceipt$.
– $sendEPO$: this action by the customer makes the fluent pay true. This action is defined in [7] as having the pre-condition that the goods have been sent.
– $sendReceipt$: this action by the merchant makes the fluent $receipt$ true. This is defined in [7] as having the pre-condition that payment has been made.

Fig. 2. The NetBill Protocol as a Commitment Machine [7]

some point fulfil its commitment[3]. However, commitment machines do not dictate or require that agents perform particular actions.

Each commitment machine implicitly defines a corresponding Finite State Machine[4] (FSM) where the states of the FSM correspond to states of the CM and the transitions are defined by the effects of the actions. Figure 3 shows a (partial) view of the states and transitions corresponding to the CM defined in figure 2. Final states (those with no undischarged base-level commitments) are shaded and dotted lines depict actions that are intended to be prevented by pre-conditions (but see section 3.4). This figure is an extension of the figure given in [7, 8]. The table in figure 3 gives the fluents and commitments that hold in each state.

3 Properties of CMs

In this section we discuss various properties of CMs as presented in [7, 8] and identify a number of areas where we propose improvements to the CM framework.

3.1 Explicit labelling of undesirable states

The presentation in [7, 8] presents protocols as defining states (in terms of the commitments of the agents and the fluents that hold). A query is then given and the interpreter finds possible sequences of actions that lead to the requested state. For example, in [8] given the commitment machine defined in figure 2, the interpreter is asked to find sequences of actions that lead to a final state where goods have been received, payment has been made, and a receipt has been issued.

However, when designing interaction rules it is important to not only ensure that a desirable final state is possible, but also to ensure that undesirable states are not possible.

In this context when we talk about "desirable" and "undesirable" states we are talking from the perspective of the *designer* of the interaction, not from the perspective of an agent who will take part in the interaction. Roughly speaking, the designer should consider a state to be desirable if an agent desires it and no agents find it undesirable. A state should be considered undesirable if any agent finds it undesirable.

If an undesirable final state is determined to be possible then this can be fixed by either adding additional commitments so that the state is no longer final, or by adding pre-conditions so that the state can not be reached. It is *not* possible to fix undesirable final states by merely having the agents be aware of the undesirable state - if a state is undesirable to one agent, another agent may still perform an action that results in that state.

For example, in the NetBill protocol the desirable final states are those in which the goods have been delivered and paid for and a receipt has been given. Undesirable states are those where only one or two of these three conditions hold; it is clearly undesirable to have the goods without payment, to have paid for the goods without getting a receipt, to have a receipt without payment, or to have paid without the goods being delivered.

[3] Commitments can also be discharged in other ways than being fulfilled [7].

[4] Actually, a variation of finite state machines, since there is no defined initial state.

No.	State
1	-
2	$request$
3	M: $promiseReceipt \wedge promiseGoods$
4	M: $promiseReceipt \wedge \mathsf{C}(goods)$, C: $accept$
5	$goods$, M: $promiseReceipt$, C: $\mathsf{C}(pay)$
6	$goods, pay$, M: $\mathsf{C}(receipt)$
7	$goods, pay, receipt$
8	$goods$, M: $promiseReceipt$
9	C: $accept$
10	pay, M: $\mathsf{C}(receipt) \wedge promiseGoods$
11	$pay, receipt$, M: $promiseGoods$
12	$goods, receipt$
13	$goods$, M: $promiseReceipt$, C: $accept$

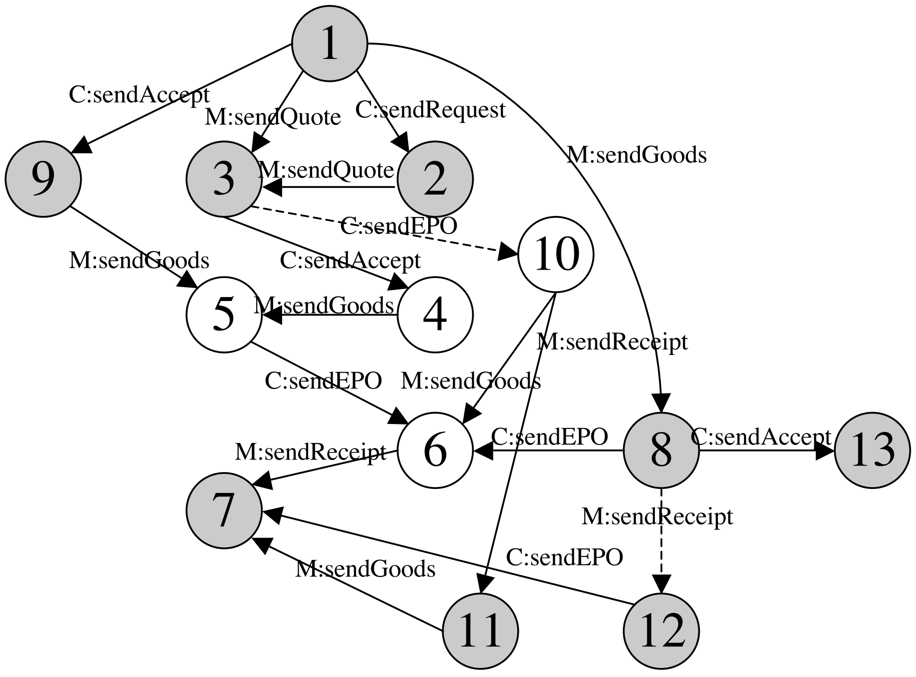

Fig. 3. Implied FSM for the NetBill CM (partial)

The final state where the goods have not been delivered, no payment has been made, and there is no receipt is acceptable, but not desirable (neutral). In figure 3 state 7 is desirable, states 8,11,12 and 13 are undesirable, and states 1,2,3 and 9 are neutral. Note that states 10, 11, 12 and 13 have been added to the machine discussed in [7, 8]. Note also that states 4,5,6 and 10 have undischarged commitments, and hence are not final states.

To illustrate why we need to identify and avoid undesirable states we consider an alternative protocol which seems quite reasonable. This protocol differs from the one presented in [7, 8] in that we remove the axiom:

$$Initiates(sendGoods, promiseReceipt, t)$$

This axiom is not needed in the "normal" expected sequence of actions (depicted in figure 1) and it is quite possible that a naïve protocol designer would leave it out of an initial protocol specification.

Now suppose that the customer is desperate for the goods[5] and begins the interaction with $sendAccept$. The merchant replies to the $sendAccept$ with $sendGoods$. At this point in the interaction the customer's acceptance commitment $\mathsf{CC}(good \rightsquigarrow pay)$ becomes a commitment to pay - $\mathsf{C}(pay)$ - since the goods have been received. The customer then fulfils their obligation by paying. At this point we are in a final state - there are no remaining commitments - and goods have been received and payment made. However, this state is an undesirable one because the customer has not received a receipt.

The important point is that the omission of the *Initiates* rule is detected by checking whether undesirable (final) states are reachable, rather than by only checking whether desirable ones can be reached. If we had simply taken the variant protocol and asked for sequences which result in goods being delivered along with payment and a receipt then the problem would not have been noticed. In other words, the undesirable states can be used as a check on the interaction rules, which in this case results in the problem being easily found.

3.2 Failure to discharge conditional commitments

There are anomalies in the rules that govern the discharge of conditional commitments. These anomalies can, in certain situations, result in conditional commitments not being discharged when, intuitively, they ought to be.

Consider the following sequence of steps:

1. The customer asks for a quote
2. The merchant replies with a quote. At this point the merchant has promised to send the goods if the customer accepts, and has promised to send a receipt if the customer pays.
3. The customer, misunderstanding the protocol perhaps, decides to accept but sends payment instead of an acceptance.

[5] Or has interacted with the merchant in the past and hence does not need to obtain a quote.

At this point the merchant becomes committed to sending a receipt, which it does, resulting in the following final state:

– fluents: $pay, receipt$
– commitments of merchant: $\mathsf{CC}(\mathsf{CC}(goods \rightsquigarrow pay) \rightsquigarrow goods)$

The crucial point here is that this is a final state and the merchant is not committed to sending the goods. The reason is that in order for $\mathsf{CC}(\mathsf{CC}(goods \rightsquigarrow pay) \rightsquigarrow goods)$ to become $\mathsf{C}(goods)$ the commitment $\mathsf{CC}(goods \rightsquigarrow pay)$ must hold: it is not enough according to the formal framework for pay to hold. This is counter-intuitive because pay is stronger than $\mathsf{CC}(goods \rightsquigarrow pay)$ in that it discharges the commitment. The formal framework does recognise this, but only at the top level – the reasoning process that discharges $\mathsf{CC}(goods \rightsquigarrow pay)$ when pay becomes true is not applied to nested commitments.

3.3 Commitment discharge is not symmetrical

The axiom/postulate defining the conditions when a commitment (or conditional commitment) is discharged says that the commitment is discharged when it already exists and its condition is brought about by an event.

A problem with this is that it is possible to create a commitment $\mathsf{C}(p)$ when p already holds. This commitment will not be discharged unless an event takes place subsequently which re-initiates p.

For example, consider the following sequence:

1. The customer sends an accept. The customer has now committed to paying if the goods are received ($\mathsf{CC}(goods \rightsquigarrow pay)$)
2. The merchant sends the goods. Since the goods have been sent, the customer now is committed to paying ($\mathsf{C}(pay)$).

However, lets consider what happens if the two steps occur in the reverse order:

1. The merchant sends the goods to the customer[6]
2. The customer sends an accept.

What is the resulting state? When sending the acceptance the customer initiates the conditional commitment to pay if the goods are received. This conditional commitment, however, does *not* become a commitment to pay even though the goods have already been sent. Consequently, the resulting state has no base-level commitments and so is an (undesirable) final state (state 13 in figure 3).

3.4 Pre-condition mechanism does not prevent action

A standard view of actions that goes back to STRIPS is that an action definition contains a pre-condition and a post-condition. The formalization of actions in the CM framework uses these, but the way in which pre-conditions are handled has a slight problem.

[6] As discussed in [8, example 2], this may be a sensible strategy if the goods are cheap to copy - e.g. software.

Pre-conditions in a CM are defined by putting conditions on the action effect definitions. For example, in [7] the effects of the $sendEPO$ action are defined using the clause[7]

$$Initiates(sendEPO, pay, t) \leftarrow HoldsAt(goods, t)$$

A standard reading in line with traditional pre-conditions would be that "payment can only be sent (by the $sendEPO$ action) when the goods have already been delivered[8]". However, what this formalization actually does is limit the *effects* of $sendEPO$ rather than the action itself. In the event calculus this does not prevent the event $sendEPO$ from occurring if $goods$ is false, it merely means that if the event $sendEPO$ occurs without $goods$ being true then the fluent pay does not become true as a result of $sendEPO$.

This is a fairly subtle difference but it does have one significant implication: if we consider agents that use an implementation of commitment machines to reason about what actions to perform, then, for example, a customer agent who has not received the goods is not prevented from executing the $sendEPO$ action. Although the reasoning module will, in this case, believe that the effects of payment have not taken place, if the $sendEPO$ action is executed resulting in credit card details being sent, then in the real world the action's execution *will* have resulted in the undesired effect of payment.

3.5 Communication mode assumptions not clear

The state space defined by the available events (actions) includes sequences of events where an event representing an action by an agent (e.g. the merchant) is followed by an event representing another action by the same agent. This may not be desirable, if the intention is to define interactions where a message from M to C can only be followed by a response from C to M.

The point here is that in the CM framework, there is no explicit specification of how the conversation should be carried out between the two parties, i.e. whether it should follow a synchronous mode or an asynchronous mode. Were the synchronous communication mode clearly specified, the action $sendReceipt$ by the Merchant would have been prevented in state 8 as the actors for the incoming and outgoing arc are the same.

However, there are situations where consecutive actions from the same agent *are* desirable. A typical CM state that may result in multiple actions from the same agent (or simultaneous actions from multiple agents) would have more than one base level commitment. See Section 4 for an example of a state with multiple base level commitments (state 10 in figure 5).

We do not address this issue in this paper; we will return to it in subsequent work.

[7] Notation has been slightly changed. The actual clause in [7] is: $Initiates(sendEPO(i, m), pay(m), t) \leftarrow HoldsAt(goods(i), t)$.

[8] This reading may seem contrary to the standard meaning of implication, but it is correct: in the context of the event calculus $Initiates(sendEPO, pay, t)$ means that whenever $sendEPO$ occurs, the fluent pay becomes true. The causality between $sendEPO$ and pay is *not* captured by the implication, but by the predicate $Initiates$. The implication places a condition on when the causality holds. Since there is only a single $Initiates$ clause, the clause above specifies that the causality only occurs if $HoldsAt(goods, t)$ is true when $sendEPO$ occurs.

4 Proposed extended CM model

In this section we propose an extended CM model which addresses some of the concerns discussed in the previous section.

4.1 Labelling undesirable states

This isn't a change to the model so much as an extension and a change to how it is used (the methodology). As part of developing the commitment machine the designer indicates which states are undesirable (bad), which are desirable (good) and which are acceptable but not desirable (neutral). Indicating the desirability of states can be done by specifying conditions.

The indication of good/bad states is specific to a particular interaction and the preferences of the parties involved. For example, in [8, example 2] where the goods are cheap to copy, the merchant may not consider state 8 in figure 3 to be a bad state.

The desirability of states, particularly of those states that are undesirable, is then used to perform safety checking.

4.2 Issues with commitment discharge

We now present a revised axiomatisation that remedies both anomalies associated with commitment discharge (sections 3.2 and 3.3). We first consider the issue discussed in section 3.2. Our proposed solution involves treating certain commitments as being "implied". For example, if pay is true, then any commitment of the form $\mathsf{CC}(X \rightsquigarrow pay)$ that occurs as a condition can be treated as having implicitly held (and been discharged).

We introduce predicates $Implied$ and $Subsumes$ which capture when a commitment (base or conditional) holds implicitly or is subsumed by a condition. These are used in the rules that govern commitment dynamics. When checking whether a condition p holds, we also check whether it is implied or subsumes[9].

In order to make commitment discharge symmetrical (section 3.3) we also decouple intended causation from actual causation: instead of stating that an action initiates a commitment (e.g. $Initiates(sendGoods, promiseReceipt, t)$), we state that the action is *intended* to cause the initiation of the commitment (e.g. $Causes(sendGoods, promiseReceipt)$). The rules in figure 4 link the two notions by defining $Initiates$ in terms of $Causes$. When p is a fluent (not a commitment) then an event $Initiates$ the fluent p exactly when it $Causes$ it. However, for a base level commitment $\mathsf{C}(p)$ even though $Causes(e, \mathsf{C}(p))$, the event e will not make $\mathsf{C}(p)$ true if p already holds. Similarly, for $Causes(e, \mathsf{CC}(p \rightsquigarrow q))$, if p holds then e will create $\mathsf{C}(q)$, not $\mathsf{CC}(p \rightsquigarrow q)$, and if q holds then e will have no effect. The rules in figure 4 realise these cases.

We then have the following action effect rules for the NetBill CM (the roles, fluents and commitments remain unchanged):

[9] $Implies(p, t)$ checks whether p is implied at time t and is used to check whether a condition (implicitly) holds at the current time. $Subsumes(p, p')$ checks whether p subsumes p' and is used to check whether an event would cause a condition to (implicitly) hold.

$$Implied(p,t) \leftarrow HoldsAt(p,t)$$
$$Implied(\mathsf{C}(x,y,p),t) \leftarrow Implied(p,t)$$
$$Implied(\mathsf{CC}(x,y,p,q),t) \leftarrow Implied(q,t)$$

$$Subsumes(p,p)$$
$$Subsumes(p,\mathsf{C}(x,y,p')) \leftarrow Subsumes(p,p')$$
$$Subsumes(p,\mathsf{CC}(x,y,q,p')) \leftarrow Subsumes(p,p')$$

$$Happens(e,t) \leftarrow AgentTry(a,e,t) \wedge Precond(e,p) \wedge HoldsAt(p,t)$$

$$Initiates(e,p,t) \leftarrow Happens(e,t) \wedge Causes(e,p) \wedge isFluent(p)$$
$$Initiates(e,\mathsf{C}(x,y,p),t) \leftarrow Causes(e,\mathsf{C}(x,y,p)) \wedge Happens(e,t) \wedge \neg Implied(p,t)$$
$$\begin{aligned}&Initiates(e,\mathsf{C}(x,y,p),t) \leftarrow Causes(e,\mathsf{CC}(x,y,q,p)) \wedge Happens(e,t) \wedge Implied(q,t) \wedge \\ &\qquad \neg Implied(p,t)\end{aligned}$$
$$\begin{aligned}&Initiates(e,\mathsf{CC}(x,y,p,q),t) \leftarrow Causes(e,\mathsf{CC}(x,y,p,q)) \wedge Happens(e,t) \wedge \\ &\qquad \neg Implied(q,t) \wedge \neg Implied(p,t)\end{aligned}$$
$$\begin{aligned}&Initiates(e,\mathsf{C}(x,y,q),t) \leftarrow HoldsAt(\mathsf{CC}(x,y,p,q),t) \wedge Happens(e,t) \wedge \\ &\qquad Initiates(e,p',t) \wedge Subsumes(p',p)\end{aligned}$$
$$\begin{aligned}&Terminates(e,\mathsf{C}(x,y,p),t) \leftarrow Implied(\mathsf{C}(x,y,p),t) \wedge Happens(e,t) \wedge Initiates(e,p',t) \\ &\qquad \wedge Subsumes(p',p)\end{aligned}$$
$$\begin{aligned}&Terminates(e,\mathsf{CC}(x,y,p,q),t) \leftarrow Implied(\mathsf{CC}(x,y,p,q),t) \wedge Happens(e,t) \wedge \\ &\qquad Initiates(e,q',t) \wedge Subsumes(q',q)\end{aligned}$$
$$\begin{aligned}&Terminates(e,\mathsf{CC}(x,y,p,q),t) \leftarrow Implied(\mathsf{CC}(x,y,p,q),t) \wedge Happens(e,t) \wedge \\ &\qquad Initiates(e,p',t) \wedge Subsumes(p',p)\end{aligned}$$

Fig. 4. Revised Commitment Machine Framework

$Causes(sendRequest, request)$
$Causes(sendQuote, \mathit{offer})$
$Causes(sendAccept, accept)$
$Causes(sendGoods, goods)$
$Causes(sendGoods, promiseReceipt)$
$Causes(sendEPO, pay)$
$Causes(sendReceipt, receipt)$
$Terminates(sendQuote, request, t)$

We now explain how the revised axiomatisation and rules address the two commitment discharge anomalies. Let us begin with the first anomaly (section 3.2). Consider the following sequence of steps:

1. The customer asks for a quote
2. The merchant replies with a quote. At this point the merchant has promised to send the goods if the customer accepts, and has promised to send a receipt if the customer pays.
3. The customer, misunderstanding the protocol perhaps, decides to accept but sends payment instead of an acceptance.

Unlike previously, the payment causes the merchant to become committed to sending the goods (as well as a receipt). Through the postulate $Implied(\mathsf{CC}(x, y, p, q), t) \leftarrow Implied(q, t)$, the fact that the pay fluent holds indicates that the conditional commitment $\mathsf{CC}(goods \rightsquigarrow pay)$ *implicitly* holds[10] at the same time. This implied conditional commitment discharges the $promiseGoods$ ($\mathsf{CC}(\mathsf{CC}(goods \rightsquigarrow pay) \rightsquigarrow goods)$) conditional commitment and creates the base level commitment $\mathsf{C}(goods)$. Once the commitments $\mathsf{C}(goods)$ and $\mathsf{C}(receipt)$ are discharged we are in a desirable final state.

Consider now the second anomaly (section 3.3). Using the new predicate $Causes$, a conditional commitment is resolved to a base level commitment if the premise is already true using the clause

$$\begin{aligned} Initiates(e, \mathsf{C}(x, y, p), t) \leftarrow\ & Causes(e, \mathsf{CC}(x, y, q, p)) \wedge Happens(e, t) \\ & \wedge Implied(q, t) \wedge \neg Implied(p, t) \end{aligned}$$

Consider the transition from state 8 to state 13. Because $Causes(sendAccept, accept)$ and $accept$ is $\mathsf{CC}(goods \rightsquigarrow pay)$ and $Implied(goods, t)$ and $\neg Implied(pay, t)$, the actual commitment initiated is then the base level commitment $\mathsf{C}(pay)$, which makes state 13 no longer final.

Figure 5 shows (part of) the state machine implicitly defined by the revised NetBill protocol and CM axiomatisation. The differences are in states 10, 11 and 13. Whereas previously state 10 had pay, $\mathsf{C}(receipt)$ and $promiseGoods$, now it has pay, $\mathsf{C}(receipt)$ and $\mathsf{C}(goods)$. As a result state 11 now includes a commitment to send the goods and is no longer a final state. State 13, which previously had $goods$, $promiseReceipt$ and $accept$ now has $goods$, $\mathsf{C}(pay)$ and $promiseReceipt$ and is no longer a final state. As before, final states are shaded. Also, dotted lines indicate actions that are affected by pre-conditions.

[10] More precisely, it could be considered to hold: there is no actual commitment, because it has been discharged, since pay is true.

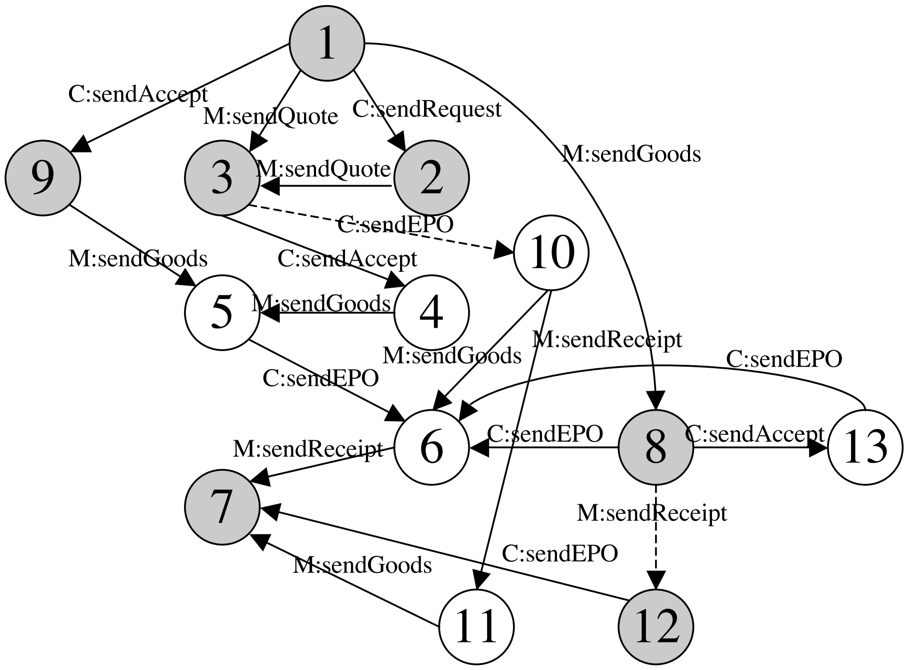

Fig. 5. Revised Transitions in example (partial)

4.3 Issues with pre-conditions

As discussed in section 3.4 trying to capture pre-conditions by adding conditions to $Initiates$ clauses does not work.

Our proposed solution is to extend the agents with a proper notion of pre-condition that specifies when actions should not be performable (as opposed to preventing the effects of the action from being caused). In the NetBill example we have the pre-conditions $Precond(sendEPO, goods)$ and $Precond(sendReceipt, pay)$.

We then need to de-couple an agent wanting to perform an action from the action actually occuring. This can be done by using a new predicate $AgentTry(a, e, t)$ to indicate that an agent a wants to perform an action e at time t. If the pre-conditions of the action e hold[11] at time t then this will imply that the event e happens.

$$Happens(e, t) \leftarrow AgentTry(a, e, t) \wedge Precond(e, p) \wedge HoldsAt(p, t)$$

Note that the definition of the interaction cannot prevent an agent from performing an action (any more than it can force an agent to honour its commmitments). However, it can specify when an action should not be performed, and detect violations, in the same way that violations of commitments are detected.

5 Conclusion

We analyzed the reasoning process of commitment machines and identified several anomalies in the current reasoning mechanism. We then indicated how these anomalies could be remedied, giving detailed rules for fixing the anomalies involving commitment discharge and pre-conditions.

Since the main contribution of this paper is a technical extension of [7] we do not perform a detailed literature review: discussion of how CMs relate to other approches can be found in [7].

There are a number of areas for future work including extending the CM framework to deal with protocols involving open numbers of participants ($1 - N$) such as auction protocols.

One area where we believe that commitment machines could be simplified concerns pre-conditions. In a sense pre-conditions and commitments are dual: the former state that a certain action must not be performed (under the prescribed conditions) whereas the latter state that a certain state must be brought about. It may be that the commitment machines framework could be simplified by merging the two concepts into a more generalised form of commitment. Specifically, pre-conditions could be replaced by commitments to *avoid* certain actions. These avoidance commitments, might be better termed *prohibitions*. A prohibition of the form $\mathsf{P}(x, a)$ would state that agent x is prohibited from performing action a. A *conditional* prohibition of the form $\mathsf{CP}(x, a, p)$ would state that agent x is *prohibited* from performing action a if p holds. For example, a merchant could have a conditional prohibition against sending a receipt if payment

[11] This assumes that p does not involve commitments. If it does then replace $HoldsAt(p, t)$ with $Implied(p, t)$.

has not been made: $\mathsf{CP}(M, sendReceipt, \neg pay)$. Prohibitions are more flexible than pre-conditions in that they can vary over time.

Another area for future work would be applying our changes to the presentation of commitment machines in [6, 5]. Whereas the presentation of commitment machines in [7, 8] uses the event calculus to formalise commitment machines, the presentation of [6, 5] defines a process for compiling a commitment machine to a finite state machine.

Finally, the reasoning that each agent performs when deciding which action to do needs to be specified in more detail. The reasoning could resemble a form of game playing where an agent wants to ensure that states that it considers undesirable cannot be reached by other agents' actions while trying to achieve states that it considers desirable.

Acknowledgments

We would like to acknowledge the support of Agent Oriented Software and of the Australian Research Council under grant LP0218928. We would also like to thank the anonymous reviewers and Min Xu for their comments.

References

1. Marc-Philippe Huget, James Odell, Øystein Haugen, Mariam "Misty" Nodine, Stephen Cranefield, Renato Levy, and Lin Padgham. Fipa modeling: Interaction diagrams. On *www.auml.org* under "Working Documents", 2003. FIPA Working Draft (version 2003-07-02).
2. J. Odell, H. Parunak, and B. Bauer. Extending UML for agents. In *Proceedings of the Agent-Oriented Information Systems Workshop at the 17th National conference on Artificial Intelligence*, 2000.
3. Wolfgang Reisig. *Petri Nets: An Introduction*. EATCS Monographs on Theoretical Computer Science. Springer-Verlag, 1985. ISBN 0-387-13723-8.
4. Marvin A. Sirbu. Credits and debits on the internet. In Michael N. Huhns and Munindar P. Singh, editors, *Readings in Agents*, pages 299–305. Morgan Kaufman, 1998. (Reprinted from *IEEE Spectrum*, 1997).
5. P. Yolum and M.P. Singh. Synthesizing finite state machines for communication protocols. Technical Report TR-2001-06, North Carolina State University, 2001. Available from *http://www.csc.ncsu.edu/research/tech-reports/README.html*.
6. P. Yolum and M.P. Singh. Commitment machines. In John-Jules Ch. Meyer and Milind Tambe, editors, *Agent Theories, Architectures, and Languages (ATAL)*, volume 2333 of *Lecture Notes in Computer Science*, pages 235–247. Springer, 2002.
7. Pınar Yolum and Munindar P. Singh. Flexible protocol specification and execution: Applying event calculus planning using commitments. In *Proceedings of the 1st Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 527–534, July 2002.
8. Pınar Yolum and Munindar P. Singh. Reasoning about commitments in the event calculus: An approach for specifying and executing protocols. *Annals of Mathematics and Artificial Intelligence (AMAI), Special Issue on Computational Logic in Multi-Agent Systems*, To appear (2004). Available from *http://www.csc.ncsu.edu/faculty/mpsingh/papers/mas/amai-03-events.pdf*.