

# Implementing Flexible and Robust Agent Interactions using Distributed Commitment Machines

Michael Winikoff

RMIT University

Melbourne, AUSTRALIA

Phone: +61 3 9925 9651

Fax: +61 3 9662 1617

Email: [winikoff@cs.rmit.edu.au](mailto:winikoff@cs.rmit.edu.au)

July 19, 2006

## Abstract

This paper focuses on the question of designing flexible and robust interactions between entities (such as agents or web services). Traditional approaches to interaction design focus on the permissible sequences of messages, but this constrains the autonomy of the entities, and limits interaction flexibility and robustness. The key to allowing for more flexible and robust interactions is to move away from message-centric design, and instead to design interactions using concepts such as the *goals* of the interaction, or the *social commitments* of the entities. This paper focuses on the Commitment Machine (CM) framework of Yolum & Singh, a simple and elegant approach for interactions which is based on social commitments. However, the CM framework is centralised, which is not a realistic model for distributed systems. The contribution of this paper is four-fold: an alternative

formalisation of Commitment Machines is presented which is simpler than previous formalisations; Distributed Commitment Machines are defined as a more realistic model for interactions in distributed systems; the properties of Commitment Machines, both Distributed and centralised, are explored; and it is shown how the properties of Distributed Commitment Machines can be exploited to provide a simple implementation of CM-based interaction in distributed systems.

**Keywords:** Agent Interactions, Social Commitments, Commitment Machines, Interaction Design.

# 1 Introduction

Software agents are autonomous software entities which are situated in some environment, balance responding to changes in their environment (being reactive) with achieving their goals (being proactive), and interact with other entities (i.e. are social) [13]. If the environment in which the agents are deployed allows for failure then the additional attributes of *flexibility* and *robustness* become important. Flexibility is defined as having multiple ways of getting things done, and robustness is defined as detecting failure, and responding to it [10]. Flexibility and robustness are related: a flexible agent can use its flexibility to deal with failure. For example, if an attempt to achieve a goal fails, then the agent can try an alternative way of achieving the goal in question.

Although a range of agent platforms support the development of flexible and robust *individual* agents (see e.g. [1]), the development of flexible and robust agent *interactions* is not well supported.

This paper focuses on the question of how to design interactions that allow for flexibility and robustness.

Traditional approaches to defining interactions use interaction protocols which focus on the possible sequences of messages that can occur in an interaction. These interaction protocols can be defined using a range of notations such as Agent UML [6], Petri nets [11], and finite state machines (FSMs).

Unfortunately, designing agent interactions in terms of permissible sequences of messages constrains the autonomy of the agents participating in the interaction by forcing them to follow prescribed message sequences, and limits agents' ability to use their flexibility and robustness.

This is not just an abstract theoretical consideration: rigid (non-flexible) interactions do not deal with as wide a range of situations. Such situations include failure, but also alternative (valid) interactions. For example, suppose that an interaction protocol specifies that a purchasing agent sends a credit card number to a merchant and that this is followed by the merchant either accepting the card, in which case the interaction continues, or declining the card, in which case the inter-

action terminates. This simple interaction protocol fragment, which is typical of what one finds in interaction protocols, does not allow the agents to recover from failure (e.g. by sending an alternative credit card), nor does it allow for flexibility in the interaction (e.g. the merchant might accept payment after delivery, might accept alternative forms of payment, or a particular product might be free under certain circumstances in which case the protocol requires a credit card number for a payment of \$0!)

The key to allowing interactions to be flexible and robust is to move away from designing interactions by prescribing legal message sequences, and to instead design interactions in terms of higher-level concepts such as the goals of the interaction or the social commitments of the agents.

A number of approaches to developing flexible and robust interactions have been proposed. One approach is the work of Cheong and Winikoff [2, 3] where a pragmatic software engineering methodology is developed which designs interactions in terms of interaction goals and action maps (similar to UML activity diagrams). Another approach is the work of Kumar et al. [8, 9] which introduces the concept of an interaction *landmark* — roughly speaking intermediate states in the interaction — formalised using the joint intention logical framework.

There have also been a number of approaches that design interactions in terms of the *social commitments* of the participating entities [4, 5, 7, 15, 17]. This paper focuses on the Commitment Machine framework of Yolum and Singh [14, 16, 17], which is noteworthy for its elegance and simplicity. Recent work [15] has begun to focus on making the framework more practical by considering what criteria should be used when designing Commitment Machines to realise a given desired interaction. However, one key issue that needs to be addressed in order to allow Commitment Machines to be used as the basis for real implemented interactions is centralisation. Commitment Machines are centralised: there is a single state of the interaction (consisting of fluents and commitments) which is changed by the actions of agents. This is a simple and clean model, but it does not match with the reality where agents are distributed and there is no single synchronised interaction state.

This paper proposes *Distributed Commitment Machines* which provide a more complex, but more realistic, model for real implementations of flexible and robust interactions using the commitment machine framework. This paper also presents a new formulation of commitment machines that is simpler, more elegant, and which has various useful properties which are explored. These properties are used to show that under certain (reasonable) assumptions, the symmetry inherent in the dynamics of commitments can allow Distributed Commitment Machines to be implemented without needing to worry about race conditions.

In fact, although this paper talks about “agents” and “roles”, there is no need to make any assumptions about the internals of agents, and so this approach would work with agents, with web services, with communicating processes, etc. In fact, the remainder of the paper uses the terms “agent” and “role” to mean the same thing: an entity which participates in an interaction.

This paper is structured as follows. Section 2 begins with a brief review of Commitment Machines. A new formalisation of Commitment Machines is then presented in section 3 before proceeding to define Distributed Commitment Machines and to explore their properties (section 4). The paper then explains how the properties of DCMs can be used to simplify the implementation of DCM-based interactions in section 5. Section 6 then concludes.

## 2 Commitment Machines

A *Commitment Machine* (CM) [14, 16, 17] defines interaction in terms of a state which is changed by actions. Although originally formalised in terms of the event calculus, a state can be thought of informally as a set of formulae. The key contribution of the CM framework was to allow the set of formulae comprising the interaction’s state to contain *commitments* (both base-level and conditional), and to define the rules governing the behaviour of commitments over time.

Two types of (social) commitments are defined: base-level, and conditional. A base-level commitment<sup>1</sup>  $C(A, B, p)$  denotes that agent  $A$  (the debtor) is committed

---

<sup>1</sup>Capital letters ( $A, B$ ) are used to denote agents, and lower case letters ( $p, q, r$ ) are used to

to agent  $B$  (the creditor) to make the condition  $p$  true. A base-level commitment is *discharged* (deleted) when  $p$  becomes true. When the identities of the debtor and creditor are clear from context or are unimportant the base-level commitment is sometimes abbreviated to  $C(p)$ .

However, in some situations base-level commitments are not enough. It is sometimes important to be able to represent *conditional* commitments. For example, a merchant might commit to sending the goods once payment has been made. This is a conditional commitment: until payment is received there is no obligation on the merchant to send the goods. However, it is important to allow these sorts of commitments in interactions. Without this sort of conditional commitment the customer may not be willing to send payment, but once the conditional commitment is made the customer can send payment knowing that doing so will cause the merchant to become committed to sending the goods.

Formally, a conditional commitment,  $CC(A, B, p, q)$ , denotes that agent  $A$  (the debtor) is committed to agent  $B$  (the creditor) to make condition  $q$  true, conditional on  $p$  becoming true. More precisely, if  $p$  becomes true then the conditional commitment is replaced by the base-level commitment  $C(A, B, q)$ . When the identities of the debtor and creditor are clear from context or are unimportant the conditional commitment  $CC(A, B, p, q)$  is sometimes abbreviated to  $CC(p \rightsquigarrow q)$ , where the arrow emphasises the causal relationship between  $p$  becoming true and the resulting commitment to make  $q$  true.

A given interaction state is considered to be *final* if it has no base-level commitments. The intuition is that if there are outstanding base-level commitments then the interaction cannot end until they have been discharged. This requirement is not applied to conditional commitments because they are latent: unless the triggering condition is made true, there is no commitment to do anything.

An interaction is defined by specifying the roles involved, defining the possible formulae that the state can contain (both fluents and commitments), and for each action that an agent can perform defining the effects of the action on the state. An action's effects are defined in terms of the fluents and commitments that it adds  

---

denote conditions.

(initiates) and those that it deletes (terminates).

The commitment machine framework defines the effects of these additions on commitments. The original framework [14, 16, 17] specified three discharge rules<sup>2</sup>:

1. If the state contains a (base-level) commitment  $C(p)$  and  $p$  becomes true then delete the commitment.
2. If the state contains a (conditional) commitment  $CC(p \rightsquigarrow q)$  and  $q$  becomes true then delete the commitment.
3. If the state contains a (conditional) commitment  $CC(p \rightsquigarrow q)$  and  $p$  becomes true then delete  $CC(p \rightsquigarrow q)$  and add  $C(q)$ .

Subsequent work by Winikoff et al. [12] argued that these rules failed to give the desired outcome in certain situations and extended the rules by making them symmetrical, and by introducing a notion of implied commitments.

The original rules defined by Yolum and Singh were not symmetrical: adding a commitment  $C(p)$  followed by adding  $p$  gave a different result to making  $p$  true and then creating the commitment  $C(p)$ . This was fixed by modifying the commitment creation rules as follows. If an action's effects are defined to add the commitment  $C(p)$  but  $p$  is already true when the action is done, then the action does not add the commitment. Similarly, if an action's effects are defined to add  $CC(p \rightsquigarrow q)$  but  $q$  is already true then the action does not add the commitment. Finally, if the action's effects are defined to add  $CC(p \rightsquigarrow q)$  but  $p$  is already true (and  $q$  is not), then performing the action adds the commitment  $C(q)$ .

The notion of *implied* commitments is now considered. Suppose a merchant has the conditional commitment  $CC(CC(\text{goods} \rightsquigarrow \text{pay}) \rightsquigarrow \text{goods})$ , where “goods” holds in states where the goods have been sent to a customer, and “pay” holds in states where the customer has paid. The intention of this commitment is that if the

---

<sup>2</sup>Actually the framework also defined additional rules that allowed commitments to be released by the creditor, or transferred between roles; but these three rules are the key ones which capture the behaviour of commitments.

customer promises to pay once they've received the goods, then the merchant is willing to send the goods on the strength of the promise. However, consider what happens if instead of making the desired promise,  $CC(\text{goods} \rightsquigarrow \text{pay})$ , the customer instead simply goes ahead and pays. The original rules defined by Yolum and Singh state that nothing happens: the merchant's conditional commitment requires a promise from the customer, and that promise has not been made. This is counter-intuitive since paying is *stronger* than the promise: the promise is to commit to paying (under certain conditions). Actually paying is stronger than the promise to pay, and indeed if the customer had taken on the promise then paying would have discharged it. Winikoff et al. [12] argued that the condition for discharging a conditional commitment  $CC(p \rightsquigarrow q)$  and replacing it with  $C(q)$  should be not that  $p$  is true, but that  $p$  is *implied*. The difference being that commitments are defined as implied if they hold, or (roughly speaking) if they would be discharged were they to hold. So, in the example above, once the customer has paid, the promise  $CC(\text{goods} \rightsquigarrow \text{pay})$  is considered to be implied, and the merchant's commitment is discharged and replaced with the commitment to send the goods.

Given a definition of an interaction in terms of roles and the effects of actions, an *interaction run* is any sequence of actions that begins in a given state and ends in a final state.

Given a Commitment Machine definition it is possible to define a corresponding finite state machine (FSM) where each state corresponds to a distinct possible interaction state, and the arcs between states correspond to actions. Doing this can be useful to visualise the interaction space defined by a given Commitment Machine.

### **Example: Outrageous Haircuts**

A (very) simple example from [12] is now given. This example is used, rather than the traditional (and more complicated) NetBill example, because introducing Distributed Commitment Machines significantly enlarges the interaction space, and in order to be able to fit the complete interaction space onto a page a simpler

example is needed.

The example involves two friends (called “me” and “you”) who would like to negotiate with the outcome that both of them get an outrageous haircut for the last day of classes. Neither friend wants to be the only person with the haircut. The interaction state can contain the fluents *yourscut* and *minecut* (representing that you (respectively me) have had the haircut). The interaction state can also contain the commitments:

$$\begin{aligned} Ok &= CC(me, you, yourscut, minecut) \\ Dare &= CC(you, me, Ok, yourscut) \end{aligned}$$

There are four actions: *cutme* (done by me, making *minecut* true), *cutyou* (done by you, making *yourscut* true), *Offer* (done by you, creating the *Dare* commitment), and *Accept* (done by me, creating the *Ok* commitment). There are also some pre-conditions: *cutme* can only be done if the agent “me” has the commitment  $C(minecut)$ , and similarly *cutyou* can only be done if the agent “you” has the commitment  $C(yourscut)$ .

The Finite State Machine<sup>3</sup> (FSM) for the Haircut example can be found in Figure 1. Final states are shaded. The FSM shows that the interaction can begin with an *Offer* (or an *Accept*). The interaction can then stop (since states 2 and 6 are final), or it continue with an *Accept* (or an *Offer*). At this point the interaction cannot end since there is an outstanding commitment, so it must continue with *cutyou*, followed by *cutme*.

### 3 Commitment Machines Revisited

Winikoff et al. [12] have argued that the notion of *implied* commitments is important to correctly discharging commitments. A new formulation of commitment

---

<sup>3</sup>All finite state machines in this paper were software-generated: the nodes and connections were computed by an implementation of the axioms (available from <http://www.winikoff.net/CM>) and were then laid out by *graphviz* (<http://www.graphviz.org/>, <http://www.pixelglow.com/graphviz/>).

dynamics is now proposed, based on the realisation that in fact it is simpler to formalise directly the notion of implied commitments, and then derive the usual discharge rules indirectly through a more general normalisation process.

Consider a state as a set of formulae  $X$ , assuming that states do not contain any formulae with logical connectives, only fluents (atoms), and commitments (this restriction is termed “flatness”). Then inference rules are defined that specify when a given formulae  $c$  is *implied* by the set of formulae  $X$  (denoted  $X \vdash c$ ; and where  $X \not\vdash c$  is used to denote that  $c$  is not implied by  $X$ ).

The inference rules are presented in the style of sequent calculus: each rule allows one to infer its conclusion (below the line) from its premises (above the line). A proof is a tree with the final conclusion at the bottom and the leaves (instances of the  $Ax$  rule) at the top.

$$\begin{array}{c}
\frac{X \vdash p \quad X \vdash q}{X \vdash p \wedge q} \wedge \quad \frac{X \vdash p_i}{X \vdash p_1 \vee p_2} \vee \quad \frac{p \in X}{X \vdash p} Ax \\
\frac{X \vdash p}{X \vdash C(p)} C_1 \quad \frac{CC(q \rightsquigarrow p) \in X \quad X \vdash q}{X \vdash C(p)} C_2 \\
\frac{X \vdash q}{X \vdash CC(p \rightsquigarrow q)} CC_1 \quad \frac{X \vdash C(q)}{X \vdash CC(p \rightsquigarrow q)} CC_2
\end{array}$$

The rules for conjunction and disjunction are standard sequent calculus rules which simply specify that  $p \wedge q$  is implied by  $X$  if  $p$  is implied by  $X$  and  $q$  is implied by  $X$ , and that  $p \vee q$  is implied by  $X$  if one of  $p$  or  $q$  is implied by  $X$ . The axiom rule ( $Ax$ ) says that  $X$  implies any  $p$  where  $p \in X$ .

The remaining rules capture the meaning of commitments. For base-level commitments there are two rules. Rule  $C_1$  says that if  $p$  is implied by  $X$  then  $C(p)$  is also implied by  $X$ . Rule  $C_2$  says that if  $CC(q \rightsquigarrow p)$  is an element of  $X$  and  $q$  is implied by  $X$  then  $C(p)$  is implied by  $X$ . For conditional commitments there are two rules. Rule  $CC_1$  says that if  $q$  is implied by  $X$  then  $CC(p \rightsquigarrow q)$  is also implied by  $X$  (for any  $p$ ). Rule  $CC_2$  says that if  $C(q)$  is implied by  $X$  then  $CC(p \rightsquigarrow q)$  is implied by  $X$  (for any  $p$ ). These rules are based on the revised commitment machine axioms of Winikoff et al. [12, Fig. 7].

The inference below shows a derivation of  $X \vdash C(\text{yourscut})$  where  $X = \{\text{minecut}, \text{CC}(\text{CC}(\text{yourscut} \rightsquigarrow \text{minecut}) \rightsquigarrow \text{yourscut})\}$ .

$$\frac{\frac{\text{CC}(\text{CC}(\text{yourscut} \rightsquigarrow \text{minecut}) \rightsquigarrow \text{yourscut}) \in X}{X \vdash C(\text{yourscut})} \quad \frac{\frac{\overline{\text{minecut} \in X}}{X \vdash \text{minecut}} \text{Ax}}{X \vdash \text{CC}(\text{yourscut} \rightsquigarrow \text{minecut})} \text{CC}_1}{X \vdash C(\text{yourscut})} \text{C}_2$$

The properties of this formulation of commitment machines is now briefly explored, ultimately leading to the definition of a normal form, and a normalisation operator which is used to define the dynamics of commitments.

First a notion of subsumption between sets of formulae is defined:  $X \sqsubseteq Y$  if anything that  $X$  implies is also implied by  $Y$ . Two sets are defined to be logically equivalent (denoted  $X \equiv Y$ ) if they subsume each other.

**Definition 1 (Subsumption)**  $X \sqsubseteq Y \stackrel{\text{def}}{=} \forall x. (X \vdash x \Rightarrow Y \vdash x)$

**Definition 2 (Logical equivalence)**  $X \equiv Y \stackrel{\text{def}}{=} X \sqsubseteq Y \wedge Y \sqsubseteq X$

Clearly  $\sqsubseteq$  is reflexive and transitive, as well as monotonic ( $X \sqsubseteq (X \cup Y)$ ).

**Lemma 1** *If  $X \equiv Y$  then  $X \cup Z \equiv Y \cup Z$ .*

**Proof:** *Use induction over the structure of proofs. The rules  $C_1$ ,  $CC_1$ ,  $CC_2$ ,  $\wedge$  and  $\vee$  are simple, leaving as the two interesting cases the inference rules  $Ax$  and  $C_2$ .*

*For the  $Ax$  rule assume that  $X \equiv Y$  and seek to show that (without loss of generality) if  $X \cup Z \vdash p$  is provable using an application of the  $Ax$  rule (for an arbitrary formula  $p$ ), then  $Y \cup Z \vdash p$  is provable. From the  $Ax$  rule we know that  $p \in (X \cup Z)$ . If  $p \in X$  then we can conclude, using a different application of the  $Ax$  rule, that  $X \vdash p$ . Hence we have that  $Y \vdash p$  (because  $X \equiv Y$ ) and hence that  $Y \cup Z \vdash p$  (by monotonicity). If  $p \notin X$  then we must have  $p \in Z$  in which case we know that  $Z \vdash p$  and hence  $Y \cup Z \vdash p$ .*

*For the  $C_2$  rule we have that*

$$\frac{\text{CC}(q \rightsquigarrow p) \in (X \cup Z) \quad \begin{array}{c} \vdots \\ X \cup Z \vdash q \end{array}}{X \cup Z \vdash C(p)} \text{C}_2$$

By the induction hypothesis we have that  $Y \cup Z \vdash q$ . If  $CC(q \rightsquigarrow p) \in Z$  then we have that  $CC(q \rightsquigarrow p) \in (Y \cup Z)$  and hence that

$$\frac{CC(q \rightsquigarrow p) \in (Y \cup Z) \quad Y \cup Z \vdash q}{Y \cup Z \vdash C(p)} C_2$$

If  $CC(q \rightsquigarrow p) \notin Z$  then  $CC(q \rightsquigarrow p) \in X$  and hence  $X \vdash CC(q \rightsquigarrow p)$  and hence  $Y \vdash CC(q \rightsquigarrow p)$ . Now there are two possibilities. One possibility is that  $CC(q \rightsquigarrow p) \in Y$  in which case we can apply the  $C_2$  rule as above to show that  $Y \cup Z \vdash C(p)$ . The second possibility is that  $CC(q \rightsquigarrow p) \notin Y$ , in which case  $Y \vdash CC(q \rightsquigarrow p)$  must be inferred using either  $CC_1$  or  $CC_2$ . If  $CC_2$  was used then we know that  $Y \vdash C(p)$  must be provable, as desired. If  $CC_1$  was used then we know that  $Y \vdash p$  must be provable, and hence  $Y \vdash C(p)$  can be proven using rule  $C_1$ .

However, subsumption is not anti-symmetric, since, for example,  $\{p, C(p)\} \sqsubseteq \{p\}$  and  $\{p\} \sqsubseteq \{p, C(p)\}$ . However, by defining an appropriate *normal form* anti-symmetry can be recovered. This is a strong property which allows one to conclude that two (normal form) sets are equal if they are logically equivalent.

A set of formulae  $X$  is defined to be in *normal form* if the following conditions hold<sup>4</sup>:

- if  $C(p) \in X$  then  $X \not\vdash p$
- if  $CC(p \rightsquigarrow q) \in X$  then  $X \not\vdash p$  and  $X \not\vdash q$  and  $X \not\vdash C(q)$

Clearly, not all formulae sets are in normal form, but it is possible for any set of formulae  $X$  to be transformed to an equivalent set of formulae which is in normal form, denoted  $\mathcal{N}(X)$ .

**Definition 3**  $\mathcal{N}(X)$  is obtained by performing the following non-deterministic process which considers all commitments in  $X$  in an unspecified order, removing violations of the normal form condition.

---

<sup>4</sup>Recall that  $X$  can only contain fluents and commitments.

- if  $C(p) \in X$  and  $X \vdash p$  then remove  $C(p)$
- if  $CC(p \rightsquigarrow q) \in X$  then
  - if  $X \not\vdash q$  and  $X \vdash p$  then remove  $CC(p \rightsquigarrow q)$  and add  $C(q)$ .
  - otherwise, if either  $X \vdash q$  or  $X \vdash C(q)$  then remove  $CC(p \rightsquigarrow q)$ .

The astute reader might be wondering whether it is possible for the normalisation process to produce different results depending on the order in which commitments are considered. The fact that this is not possible is a corollary of theorem 1 below.

However, before proceeding to the main theorem — which shows that for normal form sets logical equivalence implies equality — two basic properties of normalisation are first established: firstly, that the process indeed does produce a set that is in normal form, and secondly, that the resulting set is logically equivalent to the original set (i.e.  $X \equiv \mathcal{N}(X)$ ).

**Lemma 2**  $\mathcal{N}(X)$  is in normal form.

**Proof:** *The normalisation process removes all violations of the normal form conditions.*

**Lemma 3**  $X \equiv \mathcal{N}(X)$

**Proof:** *The elements of  $X$  which are removed are redundant — they can be inferred. For example, if  $X \vdash p$  then by inference rule  $C_1$  we can infer  $X \vdash C(p)$  without requiring that  $C(p) \in X$ . Similarly, if  $X \vdash q$  or  $X \vdash C(q)$  then we can infer that  $X \vdash CC(p \rightsquigarrow q)$  without requiring that  $CC(p \rightsquigarrow q) \in X$ <sup>5</sup>. In the final case, if  $X \vdash p$  and  $X \not\vdash q$  then the original set has  $CC(p \rightsquigarrow q) \in X$ , and the normal form set has  $C(q) \in \mathcal{N}(X)$ . To show that  $X \equiv \mathcal{N}(X)$  in this case we observe that  $X \vdash C(q)$  using  $C_2$  and that  $\mathcal{N}(X) \vdash CC(p \rightsquigarrow q)$  using  $CC_1$ .*

---

<sup>5</sup>In fact, strictly speaking we also need to note that removing  $C(p)$  cannot affect the proof of  $X \vdash p$  (and similarly for  $CC(p \rightsquigarrow q)$ ) and the proofs of  $X \vdash q$  and  $X \vdash C(q)$ .

**Lemma 4**  $\mathcal{N}(X) = \mathcal{N}(\mathcal{N}(X))$

**Proof:** *None of the conditions of the normalisation process are applicable to a normal form set, so the second normalisation process will have no effect.*

**Theorem 1** *If  $X$  and  $Y$  are in normal form then  $X \equiv Y \iff X = Y$*

**Proof:** *Consider a proof of  $X \vdash p$ , then since  $X \equiv Y$  there is a corresponding proof of  $Y \vdash p$ . Considering the possible structure of the proof, it is argued that the two proofs must have precisely the same structure, leading up to the same applications of axiom rules, thus forcing the presence of the same formulae in both  $X$  and  $Y$ . Since this applies to any formula  $p$ , it follows that  $X = Y$ .*

*If  $p$  is a fluent then the only rule applicable is  $Ax$ , and in both proofs we have that  $p \in X$  and  $p \in Y$ .*

*If  $p$  is a (base level) commitment,  $C(q)$ , then we have three applicable rules:  $Ax$ ,  $C_1$  and  $C_2$ . Now, in fact  $C_2$  is not applicable because we know that  $X$  and  $Y$  are in normal form. We then consider without loss of generality the case where the proof of  $X$  uses  $C_1$  and the proof of  $Y$  uses  $Ax$ . We then have*

$$\frac{X \vdash q}{X \vdash C(q)} C_1 \quad \frac{C(q) \in Y}{Y \vdash C(q)} Ax$$

*since  $Y$  is in normal form and  $C(q) \in Y$  we have that  $Y \not\vdash q$  and hence — since  $Y \equiv X$  — that  $X \not\vdash q$  which means that this case cannot occur. Hence the proofs either both use  $Ax$  or both use  $C_1$ . In the former case  $C(q)$  must be an element of both  $X$  and  $Y$ , in the latter case it must be an element of neither (since the sets are in normal form and we have that  $X \vdash q$  and  $Y \vdash q$ ).*

*Finally, consider the case where  $p$  is a conditional commitment  $CC(r \rightsquigarrow q)$ . There are now three distinct cases (with three additional symmetrical cases with analogous arguments):*

1. *The proof of  $X \vdash p$  begins with  $Ax$  and the proof of  $Y \vdash p$  begins with*

*$CC_1$ :*

$$\frac{CC(r \rightsquigarrow q) \in X}{X \vdash CC(r \rightsquigarrow q)} Ax \quad \frac{Y \vdash q}{Y \vdash CC(r \rightsquigarrow q)} CC_1$$

Since  $X$  is in normal form we must have  $X \not\vdash q$  and hence  $Y \not\vdash q$  so this case cannot occur.

2. The proof of  $X \vdash p$  begins with  $Ax$  and the proof of  $Y \vdash p$  begins with  $CC_2$ :

$$\frac{CC(r \rightsquigarrow q) \in X}{X \vdash CC(r \rightsquigarrow q)} Ax \quad \frac{Y \vdash C(q)}{Y \vdash CC(r \rightsquigarrow q)} CC_2$$

But because  $X$  is in normal form it cannot have both  $CC(r \rightsquigarrow q) \in X$  and  $X \vdash C(q)$  (which is implied by  $Y \vdash C(q)$ ), so this case cannot occur.

3. The proof of  $X \vdash p$  begins with  $CC_1$  and the proof of  $Y \vdash p$  begins with  $CC_2$ :

$$\frac{X \vdash q}{X \vdash CC(r \rightsquigarrow q)} CC_1 \quad \frac{Y \vdash C(q)}{Y \vdash CC(r \rightsquigarrow q)} CC_2$$

Now, since we have  $X \vdash q$  we have that  $Y \vdash q$ , and hence that the proof of  $Y \vdash C(q)$  must be the conclusion of a  $C_1$  — if it were the conclusion of an  $Ax$  rule then we would have  $C(q) \in Y$  and  $Y \vdash q$  which is not possible for a normal form set. Hence we have the following situation where the proofs, although not identical, are effectively equivalent, and can be easily made identical by replacing the proof involving  $CC_2$  with a simpler one simply using  $CC_1$ .

$$\frac{X \vdash q}{X \vdash CC(r \rightsquigarrow q)} CC_1 \quad \frac{\frac{Y \vdash q}{Y \vdash C(q)} C_1}{Y \vdash CC(r \rightsquigarrow q)} CC_2$$

Thus, it has been shown that if  $X \equiv Y$  and both  $X$  and  $Y$  are in normal form, then it must be the case that  $X = Y$ .

**Corollary:**  $\mathcal{N}(X)$  is well-defined, i.e. for a given set  $X$  the normalisation process will produce a single set  $Y = \mathcal{N}(X)$  regardless of the order in which the elements of  $X$  are considered by the normalisation process.

A normalisation process has been defined and it has been shown that although in general logical equivalence does not imply equality, for two sets,  $X$  and  $Y$ , in

normal form, if  $X$  and  $Y$  are logically equivalent ( $X \equiv Y$ ) then they must be equal.

The next question to be considered is how the effects of adding a formula to a set can be defined using the inference rules and normalisation process presented.

Adding a new formula  $p$  to a set  $X$  has the following three effects:

1. Some commitments (e.g.  $C(p)$ ) will be deleted as a result of adding  $p$ .
2. Additional (base-level) commitments are added where  $p$  causes conditional commitments to become base-level commitments.
3. The formulae  $p$  itself is added to the state<sup>6</sup>.

The result of these three effects is the new state resulting from adding  $p$ , which is denoted as  $X^{+p}$ .

How can this — seemingly fairly complex — set of changes be formalised using the inference rules and normalisation process?

The first key realisation is that  $X^{+p}$  and  $X \cup \{p\}$  are logically equivalent ( $X^{+p} \equiv X \cup \{p\}$ ), but not necessarily equal. For example, consider  $X = \{C(p \rightsquigarrow q), C(p)\}$  then  $X^{+p}$  should intuitively be  $\{C(q), p\}$ , and  $X \cup \{p\}$  is just  $\{C(p \rightsquigarrow q), C(p), p\}$ . Observe now that for any element  $r \in X^{+p}$  it must also be true that  $X \cup \{p\} \vdash r$ , because the first two effects of adding  $p$  to  $X$  do not affect logical equivalence with respect to  $X \cup \{p\}$ . Conversely, for any element  $r \in (X \cup \{p\})$  it must be true that  $X^{+p} \vdash r$ . Specifically, it can be shown that  $X^{+p} \vdash C(p \rightsquigarrow q)$  (using rule  $CC_2$ ), that  $X^{+p} \vdash C(p)$  (using rule  $C_1$ ), and that  $X \cup \{p\} \vdash C(q)$  (using rule  $C_2$ ), i.e. that  $X^{+p} \equiv X \cup \{p\}$ .

The second key realisation is that actually deleting the formulae that should be removed, adding new base-level commitments, and adding the correct variant of  $p$  is precisely what is done by the normalisation process. Thus  $X^{+p}$  is defined as a two-step process: adding  $p$ , and then normalising the result.

**Definition 4**  $X^{+p} \stackrel{def}{=} \mathcal{N}(X \cup \{p\})$

<sup>6</sup>If the rules of [12] are used then in some cases a variant of  $p$  will be added, for example if  $p = CC(r \rightsquigarrow q)$  and  $X \vdash r$  then  $C(q)$  will be added, not  $p$ .

Now consider the effects of removing a formula  $p$ . This is simpler: removing a formula simply removes it (if it is present). There are no effects on other formulae.

**Definition 5**  $X^{-p} = X \setminus \{p\}$

These definitions are simpler than those given by Winikoff et al. — for instance, compare the code in appendix A with that given in [12] — and are also more direct, since they define directly a way of computing the set of formulae resulting from adding a formula.

## 4 Distributed Commitment Machines

In this section the issue of CMs being centralised is addressed by defining *Distributed Commitment Machines* (DCMs) as a more realistic model of distributed agent interaction.

The basic idea is to have multiple copies of the state (one copy per agent), and to introduce messages between agents: whenever an agent performs an action it updates its own state and sends a message to other agents. When an agent receives a message it processes it by updating its own state with the effects of the action.

For example, starting with an empty state, suppose that the agent “you” performs the *Offer* action, creating the *Dare* commitment. Then you update your own state by adding the *Dare* commitment and send the agent “me” a message indicating that the *Offer* action has been done. When the agent “me” receives and processes this message it updates its state by adding the *Dare* commitment.

Separating the states of the different agents and introducing message processing as a separate step from performing an action, allows DCMs to model situations where one agent performs an action in parallel with another agent performing another action, and thus each agent perceives a different ordering of the actions.

The following sequence of events shows such a situation where two agents disagree on the order in which actions were performed: from the perspective of one agent the actions occurred in a given order, but from the perspective of the other agent the actions occurred in the opposite order.

1. Agent  $A$  (me) does action  $cutme$
2. Agent  $B$  (you) does action  $cutyou$
3. Agent  $A$  (me) processes message to do  $cutyou$ , thus agent  $A$  (me) has seen  $cutme$  followed by  $cutyou$ .
4. Agent  $B$  (you) processes message to do  $cutme$ , thus agent  $B$  (you) has seen  $cutyou$  followed by  $cutme$ .

A given DCM is defined similarly to a CM: one defines the roles that interact, the possible fluents and commitments that can occur, and the effects of the actions. For DCMs a new piece of information is introduced: each action specifies which roles, other than the role that performs the action, should be informed when the action is performed. A reasonable default is to inform all other roles.

Formally a Distributed Commitment Machine consists of:

- A set of roles ( $R$ )
- A set of possible fluents ( $F$ )
- A set of possible commitments ( $C$ )
- A set of possible actions  $A$  where for each action  $a \in A$  the following are defined:
  - A role  $r_a \in R$  which performs that action.
  - A set of addition effects  $E_a^+$  and deletion effects  $E_a^-$ , both of which are sets of formulae ( $E_a^+ \subset (F \cup C)$  and  $E_a^- \subset (F \cup C)$ ). The addition and deletion effects must be disjoint ( $E_a^+ \cap E_a^- = \emptyset$ ). Each commitment that is added or deleted must be a commitment made by  $r_a$  (e.g.  $C(r_a, r', p)$ ).
  - A set of roles  $R_I \subseteq (R \setminus r_a)$  which are informed when the action is performed.

The local state of a role  $r$  is given by the tuple  $S_r \stackrel{\text{def}}{=} \langle X, Q \rangle$  where  $X$  is a set of formulae (fluents and/or commitments), and  $Q$  (which is new) is a queue (sequence) of incoming messages that have not yet been processed by  $r$ .

The global state of the interaction for a DCM is a collection of states, one for each role:  $\mathcal{S} \stackrel{\text{def}}{=} \{(r_1 \mapsto \langle X_{r_1}, Q_{r_1} \rangle), \dots, (r_n \mapsto \langle X_{r_n}, Q_{r_n} \rangle)\}$ .

There are two types of state transitions that can occur in a DCM: a role can perform an action, or a role can process a message. When a role performs an action  $a$ , its own state is updated immediately, in the usual way for a commitment machine. In addition, for each role  $r \in R_I$  a message indicating that  $a$  was performed is added to  $r$ 's message queue  $Q$ . Formally a role performing an action is defined as  $\mathcal{S}^a$ :

$$\begin{aligned} \mathcal{S}^a \stackrel{\text{def}}{=} & \{(r_i \mapsto \langle X_{r_i}, Q_{r_i} \rangle) \mid (r_i \mapsto \langle X_{r_i}, Q_{r_i} \rangle) \in \mathcal{S} \wedge r_i \notin R_I \wedge r_i \neq r_a\} \\ & \cup \{(r_i \mapsto \langle X_{r_i}, (Q_{r_i} \oplus a) \rangle) \mid (r_i \mapsto \langle X_{r_i}, Q_{r_i} \rangle) \in \mathcal{S} \wedge r_i \in R_I\} \\ & \cup \{(r_a \mapsto \langle \mathcal{N}((X_{r_a} \cup E_a^+) \setminus E_a^-), Q_{r_a} \rangle)\} \end{aligned}$$

Where  $Q \oplus a$  denotes adding  $a$  to the message queue  $Q$ , and where  $(r_a \mapsto \langle X_{r_a}, Q_{r_a} \rangle) \in \mathcal{S}$ .

When a role processes a message in its message queue, the message is removed from its queue and that role's state is updated as if the action were performed. Formally a role  $r$  processing a message  $a$  is defined as  $\mathcal{S}_r^a$ :

$$\begin{aligned} \mathcal{S}_r^a \stackrel{\text{def}}{=} & \{(r_i \mapsto \langle X_{r_i}, Q_{r_i} \rangle) \mid (r_i \mapsto \langle X_{r_i}, Q_{r_i} \rangle) \in \mathcal{S} \wedge r_i \neq r\} \\ & \cup \{(r \mapsto \langle \mathcal{N}((X_r \cup E_a^+) \setminus E_a^-), (Q_r \ominus a) \rangle)\} \end{aligned}$$

Where  $(r \mapsto \langle X_r, Q_r \rangle) \in \mathcal{S}$ , and  $a$  is the most recent message in  $Q_r$ , and  $Q_r \ominus a$  denotes the result of removing  $a$  from  $Q_r$ .

A final state in a DCM is one where (a) for each role its state ( $X$ ) is final, that is contains no base-level commitments, and (b) for each role the message queue ( $Q$ ) is empty.

Having defined Distributed Commitment Machines, the remainder of this section explores their properties. This exploration begins by considering properties

of the Outrageous Haircut example, and then generalises the observed properties by showing formally that certain observed properties do, in fact, apply to all interactions.

The FSM corresponding to the Distributed Commitment Machine for the Outrageous Haircut example can be found in Figure 2. Clearly, it is much more complex than the CM version (in Figure 1): it has 54 states, of which 4 are final, and 164 transitions.

One of the key properties of DCMs which distinguishes them from CMs is that they allow for the perceptions of agents to differ regarding the ordering of actions. For example, agent  $A$  may believe that its action  $a_1$  was done before action  $a_2$ ; whereas agent  $B$  may believe that its action  $a_2$  was done before  $a_1$ . An interesting observation is that for the Outrageous Haircut DCM, whenever there are two actions that are done in one order by one agent, and in the other order by the other agent, then the resulting state is the same. In other words, actions commute: starting in state  $X$  the result of doing action  $a$  followed by action  $b$  is the same as the result of doing  $b$  first followed by  $a$ :

$$\begin{array}{ccc} X & \xrightarrow{b} & Y_1 \\ \downarrow a & & a \downarrow \\ Y_2 & \xrightarrow{b} & Z \end{array}$$

A related observation is that whenever the message queues of both agents are empty (i.e. all messages have been processed), then the states of the agents are identical.

Another observation about the Outrageous Haircut DCM concerns actions that have no effects. For (non-distributed) commitment machines actions that have no effect are suppressed. However, for DCMs such actions are not suppressed because even though an action  $a$  might have no effect on the state, it will always have the effect of sending a message to the other role, so the resulting global state is different. In the FSM this appears as pairs of states where there is an arc from state  $A$  to state  $B$  labelled *send- $X$*  and an arc from  $B$  to  $A$  labelled *receive- $X$* .

By suppressing actions which have no *local* effect (i.e. an effect on the state)

a much simpler FSM is obtained, which is depicted in Figure 3. This FSM has 14 states and 18 transitions. Since this FSM is the expanded version of a Distributed Commitment Machine, each state is a mapping from a role to that role’s state, where a role’s state comprises a set of formulae, and a message queue. The details of the states for the FSM can be found in Figure 4.

It can be clearly seen that preventing actions which have no local effect results in a much simpler FSM. However, a justification for preventing these actions is needed. The justification is that for the Outrageous Haircut example, in all cases, if an action  $a$  performed by a role has no effect on that role’s state, then when the other role processes the message and performs  $a$ , then  $a$  also has no effect on the other role’s state.

A final observation is that although there are loops in the FSM of Figure 2, the desired final state (state number 25 in Figure 2) can be reached from any other state.

So, to summarise, for the Outrageous Haircut example:

- Actions commute: doing  $a$  followed by  $b$  yields the same result as  $b$  followed by  $a$ .
- Whenever both agents have empty message queues they also have identical states.
- When an action  $a$  has no effect on its performer’s state, then it also has no effect on the other agent’s local state. This justifies preventing actions that have no local effects, and yields a much simpler interaction space.
- The final state is reachable from any other state.

These properties are now generalised. It transpires that, except for the last property, they hold for *all* monotonic (see definition 6) DCMs, not just for this example.

First consider the commutativity of actions. Recall that  $X^{+p}$  has been defined as the state resulting from adding  $p$  to state  $X$ . This notation is extended by writing  $X^{+p;+q}$  to denote the state resulting from adding  $p$  to state  $X$  and then adding  $q$

to the result (i.e.  $X^{+p;+q} = X^{+p+q}$ ). Similarly,  $X^{-p;-q}$  denotes the state resulting from deleting  $p$  from state  $X$  and then deleting  $q$  from the resulting state. Given actions  $a$  and  $b$  the notation  $X^a$  denotes the state resulting from performing the action in state  $X$ , and  $X^{a;b}$  denotes the state resulting from performing the action  $a$  in state  $X$  and then performing the action  $b$  in the resulting state<sup>7</sup>. Recall also that the effect of performing an action is to add all of the elements in its addition effects set  $E_a^+$  and to delete all of the elements in its deletion effects set  $E_a^-$ .

**Theorem 2**  $X^{+p;+q} = X^{+q;+p}$ , i.e. addition is commutative.

**Proof:** We know that  $X^{+p} \equiv X \cup \{p\}$ . We also have that  $X^{+p;+q} = X^{+p+q} \equiv X^{+p} \cup \{q\}$ . By lemma 1 this is logically equivalent to  $(X \cup \{p\}) \cup \{q\}$  which is clearly equivalent to  $(X \cup \{q\}) \cup \{p\} \equiv X^{+q} \cup \{p\} \equiv X^{+q;+p}$ . Hence  $X^{+p;+q} \equiv X^{+q;+p}$ . Since both sides are in normal form (by definition 3) it follows from theorem 1 that they are equal.

**Theorem 3**  $X^{-p;-q} = X^{-q;-p}$ , i.e. deletion is commutative.

**Proof:**  $X^{-p;-q} = X^{-p} \setminus \{q\} = (X \setminus \{p\}) \setminus \{q\} = X \setminus (\{p\} \cup \{q\}) = (X \setminus \{q\}) \setminus \{p\} = X^{-q;-p}$ .

However, although two deletions commute and two additions commute, an addition and a deletion do not, in general, commute. For example, consider  $X = \{p\}$ , then  $X^{-p;+C(p)} = \{C(p)\}$  but  $X^{+C(p);-p} = \{\}$ . Formally, is it *not* a theorem that  $X^{-p;+q} = X^{+q;-p}$ .

Although actions in general do not commute, they do commute under the assumption that the DCM is *monotonic*.

**Definition 6 (Monotonic DCM)** A DCM is monotonic if for each action the deletion effects set is empty ( $E_a^- = \emptyset$ ).

The Outrageous Haircut example is monotonic. The Net Bill example [16] is almost monotonic: the one violation is that the *sendQuote* action deletes the fluent *request* (which was added by the *sendRequest* action). This can be easily

---

<sup>7</sup>This assumes that both actions can be performed.

fixed by having *sendRequest* create a commitment in the merchant to respond<sup>8</sup> ( $C(\textit{responded})$ ) and having the *sendQuote* action also create the fluent *responded*.

**Theorem 4** *If a DCM is monotonic, then  $X^{a;b} = X^{b;a}$ , where  $b$  and  $a$  are actions.*

**Proof:** *Since the DCM is monotonic, the effects of  $a$  and  $b$  are to add formulae to the state, but formulae addition is commutative, so the addition of formulae can be rearranged without changing the end result. Specifically we can rearrange an addition sequence where the formulae added by performing  $a$  come before those of  $b$ , into a sequence where the formulae added by performing  $b$  come before those formulae added by performing  $a$ .*

A corollary to this theorem is that states in a monotonic DCM remain synchronised.

**Theorem 5** *In a monotonic DCM, if all agents have empty message queues, then the states of all agents will be equal.*

**Proof:** *Suppose that all agents start with the same state (which is true at the start of the interaction), then once all messages have been processed each agent will have performed the same actions, albeit not necessarily in the same order. Theorem 4 guarantees that the state resulting from performing a set of actions does not depend on the order in which the actions are performed, and thus the agents all have the same resulting state.*

The next observation about the Outrageous Haircut DCM was that actions with no effect on the local state have no effect on the states of other agents. The following theorem shows that this observation holds for all monotonic DCMs.

**Theorem 6** *If an action in a monotonic DCM has no effect on the local state then it will have no effect on the state of other agents.*

**Proof:** *(by contradiction) Suppose that beginning in state  $X$  there is an action  $a$  which results in  $X$  (i.e. action  $a$  has no local effect). Then in order for the action*

---

<sup>8</sup>This weakens the assumption that actions can only create commitments by the performer of the action.

*a* to have an effect in another agent's state (which starts in the same state,  $X$ , due to theorem 5) there must be an action  $b$  which the other agent performs before  $a$  such that the end result of both  $a; b$  and  $b; a$  is  $Z$  (due to commutativity), but where  $a$  has an effect if it comes after  $b$  (i.e.  $Y \neq Z$ ):

$$\begin{array}{ccc} X & \xrightarrow{b} & Y \\ \downarrow a & & a \downarrow \\ X & \xrightarrow{b} & Z \end{array}$$

However, we have that  $X \xrightarrow{ab} Z$ , and that  $X \xrightarrow{a} X$ , hence  $X \xrightarrow{b} Z$  so  $Y = Z$  giving a contradiction.

This theorem justifies the suppression of actions that have no local effect in a monotonic DCM.

The final property of the Outrageous Haircut example was that the desired final state was always reachable. Clearly, this does not hold for all interactions. For example, consider an interaction with two actions: the first action has effect  $+p$ , the second action, which has the precondition  $\neg p$ , has effect  $+r$ , and the desired final state has  $r$ . Performing the first action leads to a dead-end from which the desired final state cannot be reached.

## 5 Implementing Agent Interaction with DCMs

This section discusses how the properties of DCMs are used to simplify their implementation. A key simplification is that actions that have no effect on the agent's local state are disallowed. This is justified by theorem 6. A second key simplification is that there is no need to worry about race conditions. Thus it is not necessary to ensure that agents wait for each other, or that only one agent is "active" at any given point in time.

Given these two simplifications, the implementation of a DCM is as follows:

- Each agent has an associated CM module which contains the state (fluents

and commitments) local to that agent and updates the CM with the effects of actions (see Figure 5)

- At each execution cycle the CM gives the agent the state and a list of enabled actions (those actions that (a) have a true pre-condition; and where (b) executing the action has an effect on the state, i.e.  $\mathcal{N}(X \cup E_a^+) \neq X$ ).
- The agent then provides the CM with a selected action from the list, or the special “do-nothing” action *wait*.
- The agent’s local CM updates the state by performing the action ( $X_{\text{new}} := \mathcal{N}(X_{\text{old}} \cup E_a^+)$ ), since in a monotonic DCM  $E_a^- = \emptyset$ ) and sends a message to the relevant roles (those in the action’s  $R_I$ ). If the agent selected the *wait* action then the state is unchanged and no messages are sent.
- When the agent’s CM module receives a message from another agent’s CM module it updates the state with the effects of the relevant action.

Note that the only synchronisation required is that within a single CM module changing the state due to an agent’s action and due to a received message does not occur at the same time. There is no need for any synchronisation or coordination between agents.

The sequence of events below is one possible execution of the Outrageous Haircut example. The numbers in the left column correspond to the states in Figures 3 and 4. The next two columns show the local state for each of the two agents where the state is given as a number (referring to the states given in Figure 1) and a message queue. The final column gives the change that leads from one row to the next. This change is either an action (e.g. “you : offer”) or processing a message (e.g. “me  $\leftarrow$  offer”). For example, the first row shows that both agents “me” and “you” start in state 1, and that agent “you” performs the *offer* action, resulting in row 2 where agent “me” has a message, and agent “you” is in state 2 of Figure 1, i.e. has the conditional commitment  $CC(\text{you}, \text{me}, CC(\text{me}, \text{you}, \text{cutyou}, \text{cutme}), \text{cutyou})$ . Similarly, the third row is the result of agent “me” performing the accept action

which takes it from state 1 of Figure 1 to state 6, i.e. from having no commitments to having the commitment  $CC(me, you, cutyou, cutme)$ .

DCM	Me	You	Change
1	1, $\langle \rangle$	1, $\langle \rangle$	
2	1, $\langle offer \rangle$	2, $\langle \rangle$	you : offer
3	6, $\langle offer \rangle$	2, $\langle accept \rangle$	me : accept
4	3, $\langle \rangle$	2, $\langle accept \rangle$	me $\leftarrow$ offer
5	3, $\langle \rangle$	3, $\langle \rangle$	you $\leftarrow$ accept
6	3, $\langle cutyou \rangle$	4, $\langle \rangle$	you : cutyou
7	4, $\langle \rangle$	4, $\langle \rangle$	me $\leftarrow$ cutyou
8	5, $\langle \rangle$	4, $\langle cutme \rangle$	me : cutme
9	5, $\langle \rangle$	5, $\langle \rangle$	you $\leftarrow$ cutme

The following shows another possible execution where agent “you” cuts its hair before the agent “me” realises that an Offer has been made, i.e. agent “me” processes the message *offer* after agent “you” does the action *cutyou*.

DCM	Me	You	Change
1	1, $\langle \rangle$	1, $\langle \rangle$	
13	6, $\langle \rangle$	1, $\langle accept \rangle$	me : accept
3	6, $\langle offer \rangle$	2, $\langle accept \rangle$	you : offer
10	6, $\langle offer \rangle$	3, $\langle \rangle$	you $\leftarrow$ accept
11	6, $\langle offer, cutyou \rangle$	4, $\langle \rangle$	you : cutyou
6	3, $\langle cutyou \rangle$	4, $\langle \rangle$	me $\leftarrow$ offer
7	4, $\langle \rangle$	4, $\langle \rangle$	me $\leftarrow$ cutyou
8	5, $\langle \rangle$	4, $\langle cutme \rangle$	me : cutme
9	5, $\langle \rangle$	5, $\langle \rangle$	you $\leftarrow$ cutme

## 6 Conclusion

A new formalisation of CMs has been presented. This formalisation is based on the notion of implied commitments and includes a general normalisation operator. As can be seen from the code in appendix A, this formalisation is simpler than previous formalisations such as [12].

This new formalisation was used as the basis for defining Distributed Commitment Machines (DCMs). These are more realistic, since they do not assume a centralised state, and are thus a more suitable basis for implementation. Properties of DCMs were explored, and a number of strong symmetry properties were proven, including the lack of race conditions in monotonic DCMs. These properties rely on symmetries in the handling of commitments, and thus require the axioms of Winikoff et al. [12]: they do *not* hold for the original CM framework of Yolum and Singh.

Finally, an approach for implementing agent interaction using DCMs was described. This approach exploits the symmetry properties to simplify the implementation. In particular, there is no need to worry about race conditions or synchronisation between agents due to the commutativity properties of monotonic DCMs.

Although this paper advances the state of the art for commitment-based interaction, there are a number of outstanding issues. What this paper has presented is a theoretical framework which may be simpler, and may address the centralisation problem, but is still just a theoretical framework. For this approach to be used in practice requires that the approach be made pragmatic. Two key missing pieces are a methodology for designing commitment-based interactions, and a guide to mapping an interaction designed as a commitment machine to an implementation.

A methodology is needed to guide the software designer in the development of the interaction. It should provide a simple process that assists in determining what commitments are needed and when they should be created. Ultimately, the application of the methodology should result in a commitment machine that will realise the desired interaction, allowing the desired goals of the interaction to be

achieved in a range of ways, while preventing the interaction from reaching undesired states. It is also desirable for there to be tool support that assists the designer in developing the interaction and in assuring that it is correct. A specific piece of advice that can be drawn from the work of this paper is that one should strive to create monotonic DCMs.

Once an interaction has been designed it needs to be implemented, and there is a need for guidance as to how this should be done. In section 5 an outline of one approach for implementing commitment-based interactions was sketched, but clearly more work is needed to make implementation of commitment-based interactions easy and routine.

Compared with other approaches for developing more flexible interactions (discussed in section 1), commitment machines are, in the author's opinion, not (yet) a pragmatic and usable approach. However, *none* of the other commitment-based approaches, nor the landmark-based approach, provide a design methodology, and thus they cannot, at this point in time, be seen as a pragmatic approach for designing flexible interactions. On the other hand, although the work on *Hermes* [2, 3] is clearly pragmatic, it does not have clear conceptual foundations, nor semantics for its concepts.

In addition to the two key areas for future work — methodologies and implementation techniques — there are a number of smaller, but nevertheless important, areas for future work.

One of these concerns eliminating redundant messages. When an agent performs an action it sends messages to all agents in  $R_I$ . It is possible that some of these messages in fact are not needed for the interaction to work. For example, if a Barber agent is added to the Outrageous Haircut example, where the Barber agent receives a request to cut hair from either agent “me” or “you” and does the haircut, then the Barber does not need to know about the negotiations or commitments leading up to the request for a haircut. It should be possible to perform analysis which detects this and automatically derives a simpler local CM for the Barber agent that does not require the Barber agent to be sent unnecessary messages.

A second area concerns the restriction to monotonic DCMs. Clearly, it would

be desirable to identify more precise conditions that allow DCMs to remain race-condition-free without requiring them to be monotonic.

**Acknowledgements** The author would like to acknowledge the support of Agent Oriented Software Pty. Ltd. and of the Australian Research Council (under grant LP0453486, *Advanced Software Engineering Support for Intelligent Agents*).

## References

- [1] Rafael H. Bordini, Mehdi Dastani, Jürgen Dix, and Amal El Fallah Seghrouchni, ed., *Multi-Agent Programming: Languages, Platforms and Applications*, Springer, 2005. ISBN 0-387-24568-5.
- [2] Christopher Cheong and Michael Winikoff, Hermes: Designing goal-oriented agent interactions, in: *Agent-Oriented Software Engineering VI*, J. Müller and F. Zambonelli, ed., Springer, LNCS 3950, 2006, 16–27.
- [3] Christopher Cheong and Michael Winikoff, Improving flexibility and robustness in agent interactions: Extending Prometheus with Hermes, in: *Software Engineering for Multi-Agent Systems IV: Research Issues and Practical Applications*, A. Garcia, R. Choren, C. Lucena, P. Giorgini, T. Holvoet and A. Romanovsky, ed., Springer, LNCS 3914, 2006, 189–206.
- [4] Roberto A. Flores and Robert C. Kremer, A pragmatic approach to build conversation protocols using social commitments, in: *Autonomous Agents and Multi-Agent Systems (AAMAS)*, N. R. Jennings, C. Sierra, L. Sonenberg, and M. Tambe, ed., ACM Press, 2004, 1242–1243.
- [5] Nicoletta Fornara and Marco Colombetti, Operational specification of a commitment-based agent communication language, in: *Autonomous Agents and Multi-Agent Systems*, C. Castelfranchi and W. Lewis Johnson, ed., ACM Press, 2002, 535–542.

- [6] Marc-Philippe Huget and James Odell, Representing agent interaction protocols with agent UML, in: *Agent Oriented Software Engineering V*, J. Odell, P. Giorgini, and J. P. Müller, ed., Springer, LNCS 3382, 2005, 16–30.
- [7] Rob Kremer and Roberto Flores, Using a performative subsumption lattice to support commitment-based conversations, in: *Autonomous Agents and Multi-Agent Systems (AAMAS)*, F. Dignum, V. Dignum, S. Koenig, S. Kraus, M. P. Singh, and M. Wooldridge, ed., ACM Press, 2005, 114–121.
- [8] Sanjeev Kumar and Philip R. Cohen, STAPLE: An agent programming language based on the joint intention theory, in: *Autonomous Agents and Multi-Agent Systems (AAMAS)*, N. R. Jennings, C. Sierra, L. Sonenberg, and M. Tambe, ed., ACM Press, 2004, 1390–1391.
- [9] Sanjeev Kumar, Marcus J. Huber, and Philip R. Cohen, Representing and executing protocols as joint actions, in: *Autonomous Agents and Multi-Agent Systems*, C. Castelfranchi and W. Lewis Johnson, ed., ACM Press, 2002, 543–550.
- [10] Lin Padgham and Michael Winikoff, *Developing Intelligent Agent Systems: A Practical Guide*, John Wiley and Sons, 2004. ISBN 0-470-86120-7.
- [11] Wolfgang Reisig, *Petri Nets: An Introduction*, Springer-Verlag, 1985. ISBN 0-387-13723-8.
- [12] Michael Winikoff, Wei Liu, and James Harland, Enhancing commitment machines, in: *Declarative Agent Languages and Technologies II*, J. Leite, A. Omicini, P. Torroni, and P. Yolum, ed., Springer, LNAI 3476, 2004, 198–220.
- [13] Michael Wooldridge, *An Introduction to MultiAgent Systems*, John Wiley & Sons, 2002. ISBN 0 47149691X.

- [14] P. Yolum and M.P. Singh, Commitment machines, in: *Agent Theories, Architectures, and Languages (ATAL)*, J-J. Ch. Meyer and M. Tambe, ed., Springer, LNCS 2333, 2002, 235–247.
- [15] Pinar Yolum, Towards design tools for protocol development, in: *Autonomous Agents and Multi-Agent Systems (AAMAS)*, F. Dignum, V. Dignum, S. Koenig, S. Kraus, M. P. Singh, and M. Wooldridge, ed., ACM Press, 2005, 99–105.
- [16] Pinar Yolum and Munindar P. Singh, Flexible protocol specification and execution: Applying event calculus planning using commitments, in: *Autonomous Agents and Multi-Agent Systems*, C. Castelfranchi and W. Lewis Johnson, ed., ACM Press, 2002, 527–534.
- [17] Pinar 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)*, 42(1-3) (2004), 227–253.

## Author Biography

Dr. Michael Winikoff is a senior lecturer at RMIT University. His research interests concern notations for specifying and constructing software. Dr. Winikoff received his PhD in 1997 from the University of Melbourne in the area of logic programming with linear logic. He then spent a year and a half as a research fellow at the University of Melbourne working on animating formal specifications. After a stint in the USA (working for the Institute for Software Research) he returned to Australia to join RMIT University's agent group. In addition to his academic experience Dr. Winikoff has also worked as a consultant for Agent Oriented Software, developers of the JACK agent platform, and has worked as a software developer in industry. These days he is perhaps best known — with Professor Lin Padgham — as the developer of the *Prometheus* methodology and author of the book *Developing Intelligent Agent Systems: A Practical Guide*, published by Wiley and Sons in 2004.

## A Implementation

The Prolog code below implements the inference rules presented in section 3 and the normalisation process. The code can be obtained from <http://www.winikoff.net/CM>.

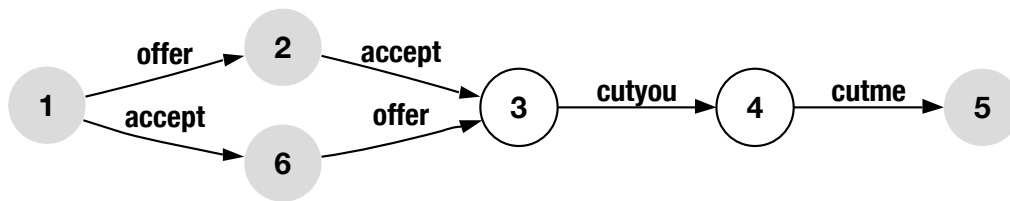
`implies(X,P)` is true if the formula `P` is implied by state (formula list) `X`.

```
implies(X,P) :- member(P,X).
implies(X,c(P)) :- implies(X,P).
implies(X,c(P)) :- member(cc(Q,P),X), implies(X,Q).
implies(X,cc(_,Q)) :- implies(X,Q).
implies(X,cc(_,Q)) :- implies(X,c(Q)).
```

`norm(A,B)`: `B` is the normalised form of `A`.

```
norm(A,B) :- norm(A,A,B).
```

```
norm(_, [], []).
norm(S, [c(P)|X], R) :- implies(S,P), !, norm(S,X,R).
norm(S, [cc(P,Q)|X], R) :-
    not implies(S,Q), implies(S,P), !,
    R=[c(Q)|R1], norm(S,X,R1).
norm(S, [cc(P,Q)|X], R) :-
    (implies(S,Q) ; implies(S,c(Q))), !, norm(S,X,R).
norm(S, [P|X], [P|R]) :- norm(S,X,R).
```



- 1 =  $\emptyset$
- 2 =  $\{CC(\text{you}, \text{me}, CC(\text{me}, \text{you}, \text{cutyou}, \text{cutme}), \text{cutyou})\}$
- 3 =  $\{CC(\text{me}, \text{you}, \text{cutyou}, \text{cutme}), C(\text{you}, \text{me}, \text{cutyou})\}$
- 4 =  $\{\text{cutyou}, C(\text{me}, \text{you}, \text{cutme})\}$
- 5 =  $\{\text{cutyou}, \text{cutme}\}$
- 6 =  $\{CC(\text{me}, \text{you}, \text{cutyou}, \text{cutme})\}$

Figure 1: FSM for non-distributed haircut example, final states are shaded



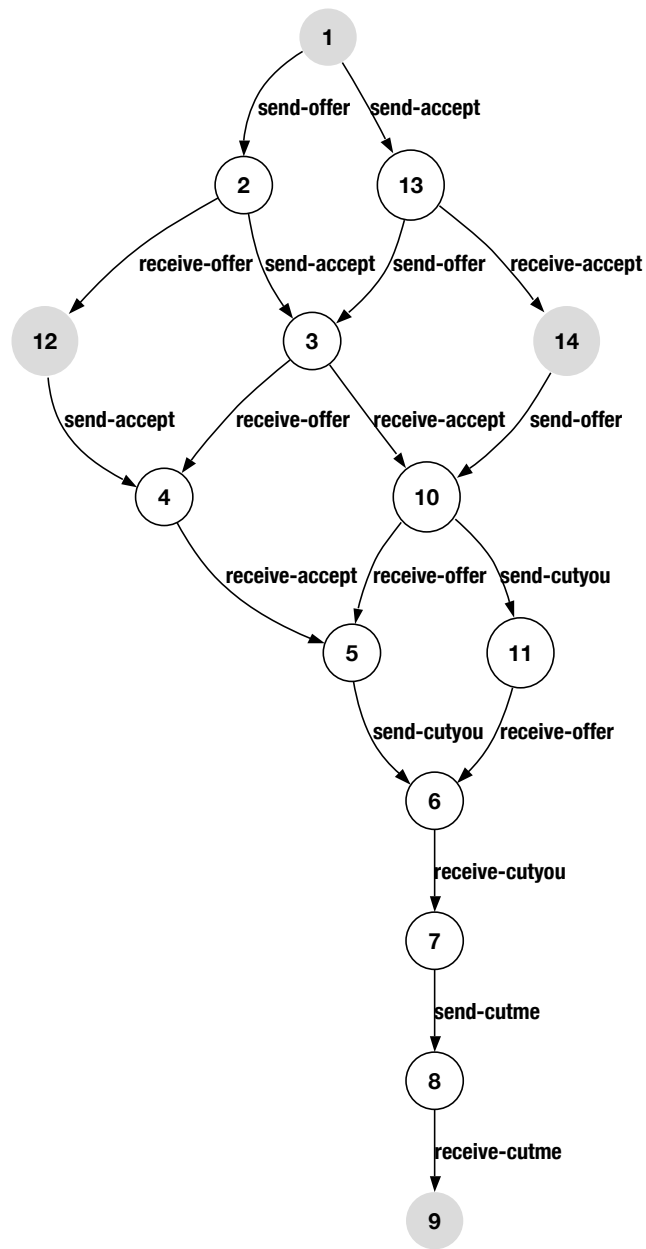


Figure 3: Simpler FSM resulting from preventing actions that have no local effect, final states are shaded

1.  $\{me \mapsto \langle \{\}, \langle \rangle \rangle,$   
 $you \mapsto \langle \{\}, \langle \rangle \rangle\}$
2.  $\{me \mapsto \langle \{\}, \langle offer \rangle \rangle,$   
 $you \mapsto \langle \{CC(you, me, CC(me, you, cutyou, cutme), cutyou)\}, \langle \rangle \rangle\}$
3.  $\{me \mapsto \langle \{CC(me, you, cutyou, cutme)\}, \langle offer \rangle \rangle,$   
 $you \mapsto \langle \{CC(you, me, CC(me, you, cutyou, cutme), cutyou)\}, \langle accept \rangle \rangle\}$
4.  $\{me \mapsto \langle \{CC(me, you, cutyou, cutme), C(you, me, cutyou)\}, \langle \rangle \rangle,$   
 $you \mapsto \langle \{CC(you, me, CC(me, you, cutyou, cutme), cutyou)\}, \langle accept \rangle \rangle\}$
5.  $\{me \mapsto \langle \{CC(me, you, cutyou, cutme), C(you, me, cutyou)\}, \langle \rangle \rangle,$   
 $you \mapsto \langle \{CC(me, you, cutyou, cutme), C(you, me, cutyou)\}, \langle \rangle \rangle\}$
6.  $\{me \mapsto \langle \{CC(me, you, cutyou, cutme), C(you, me, cutyou)\}, \langle cutyou \rangle \rangle,$   
 $you \mapsto \langle \{cutyou, C(me, you, cutme)\}, \langle \rangle \rangle\}$
7.  $\{me \mapsto \langle \{cutyou, C(me, you, cutme)\}, \langle \rangle \rangle,$   
 $you \mapsto \langle \{cutyou, C(me, you, cutme)\}, \langle \rangle \rangle\}$
8.  $\{me \mapsto \langle \{cutyou, cutme\}, \langle \rangle \rangle,$   
 $you \mapsto \langle \{cutyou, C(me, you, cutme)\}, \langle cutme \rangle \rangle\}$
9.  $\{me \mapsto \langle \{cutyou, cutme\}, \langle \rangle \rangle,$   
 $you \mapsto \langle \{cutyou, cutme\}, \langle \rangle \rangle\}$
10.  $\{me \mapsto \langle \{CC(me, you, cutyou, cutme)\}, \langle offer \rangle \rangle,$   
 $you \mapsto \langle \{CC(me, you, cutyou, cutme), C(you, me, cutyou)\}, \langle \rangle \rangle\}$
11.  $\{me \mapsto \langle \{CC(me, you, cutyou, cutme)\}, \langle offer, cutyou \rangle \rangle,$   
 $you \mapsto \langle \{cutyou, C(me, you, cutme)\}, \langle \rangle \rangle\}$
12.  $\{me \mapsto \langle \{CC(you, me, CC(me, you, cutyou, cutme), cutyou)\}, \langle \rangle \rangle,$   
 $you \mapsto \langle \{CC(you, me, CC(me, you, cutyou, cutme), cutyou)\}, \langle \rangle \rangle\}$
13.  $\{me \mapsto \langle \{CC(me, you, cutyou, cutme)\}, \langle \rangle \rangle,$   
 $you \mapsto \langle \{\}, \langle accept \rangle \rangle\}$
14.  $\{me \mapsto \langle \{CC(me, you, cutyou, cutme)\}, \langle \rangle \rangle,$   
 $you \mapsto \langle \{CC(me, you, cutyou, cutme)\}, \langle \rangle \rangle\}$

Figure 4: Details on the states in the FSM of Figure 3

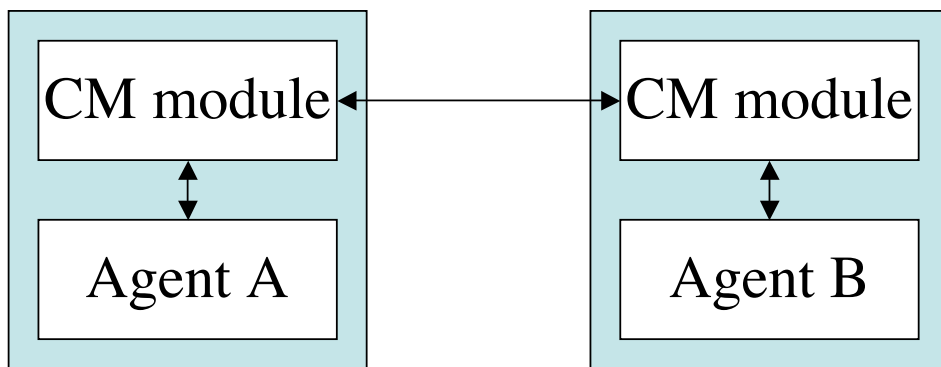


Figure 5: Execution Model (showing two agents)