Flexible Conversations Using Social Commitments and a Performatives Hierarchy

Rob Kremer[1] and Roberto A. Flores[2]

[1] University of Calgary,
Department of Computer Science
Calgary, AB T2N 1N4, Canada
`kremer@cpsc.ucalgary.ca`
[2] Christopher Newport University,
Department of Physics, Computer Science & Engineering,
Newport News, VA 23606 USA
`flores@pcs.cnu.edu`

Abstract. In this research, we re-arrange FIPA's ACL performatives to form a subsumption lattice (ontology) and apply a theory of social commitments to achieve a simplified and observable model of agent behaviour. Using this model, we have implemented agent interaction through social commitments (or obligations) based solely on observation of messages passed between the agents (such observation is supported by the *cooperation domain* mechanism in our agent infrastructure system). Moreover, because the performatives are in a subsumption lattice, it is relatively easy for an observer to infer social commitment relationships even if the observer does not understand the details of messages or even the exact performatives used (so long as the observer has access to the performatives ontology).

Our social commitment model can be used in agent implementation to simplify the specification and observation of agent behaviour even if the agents themselves are *not implemented* using social commitments. This is accomplished through the use of *commitment operators* attached to the performatives (as policies) in the subsumption lattice.

In this work, we show how FIPA's performatives can be interpreted in a theory of social commitment to allow observable social behaviour and conformance to social norms.

1 Introduction

The FIPA standard SC00061G [8] has defined inter-agent messages in the envelope/letter pattern, where the "envelope" contains several standard fields which must be understood by all agents in the community, and the "letter" part may or may not be understood by other agents. FIPA further defines the contents of several envelope fields such as *performative* (the type of the communicative act), *sender, receiver, content, ontology, reply-with, in-reply-to, reply-by* and others.

We focus primarily on the performative field as the main means by which agents can choose their behaviour in reaction to a particular message.

Table 1. FIPA performatives

Performative	Description
accept-proposal	accepting a previous proposal
agree	agreeing to perform some action
cancel	inform another agent that it no longer need perform some action
call-for-proposal	call for proposals to perform an action
confirm	informs a given proposition is true
disconfirm	informs a given proposition is false
failure	an action was attempted but failed
inform	a given proposition is true
inform-if	inform whether a proposition is true
inform-ref	inform the object which corresponds to a descriptor
not-understood	did not understand what the receiver just did
propagate	pass a message on
propose	submit a proposal to perform an action
proxy	pass on an embedded message
query-if	asking whether a proposition is true
query-ref	asking for the object referred to
refuse	refusing to perform an action
reject-proposal	rejecting a proposal during negotiation
request	request to perform some action
request-when	request to perform an action when a proposition becomes true
request-whenever	request to perform an action each time proposition becomes true
subscribe	request to notify the value of a ref. whenever the object changes

Furthermore, we only focus on the behaviour relative to communication acts (speech acts) in conversation and do not delve into physical acts or domain-specific acts.

1.1 Performatives

The FIPA standard SC00037J [9] defines 22 "Communicative Act" names as values for the performative field (see Table 1).

In implementing our agent infrastructure, CASA [13], we have found that the FIPA performatives were very useful in that they include communicative acts that we would not have initially thought of ourselves. Unfortunately, it became obvious that they do not form a computationally useful set for our agents to decide on an action when they receive a message. When our agents used FIPA's flat classification, they had to switch behaviour in an ad-hoc manner for (almost) each of the 22 performatives. Our agents needed to perform a list of actions for each performative, and these actions were often duplicated among several of the performative behaviours. This lead to a complex and error-prone specification. Furthermore, we had to extend the list of performatives, and each of our agents had to constantly update the list.

We found that if we arrange the same performatives in a subsumption lattice (see Figure 1), we can succinctly glean the semantic information we need to clas-

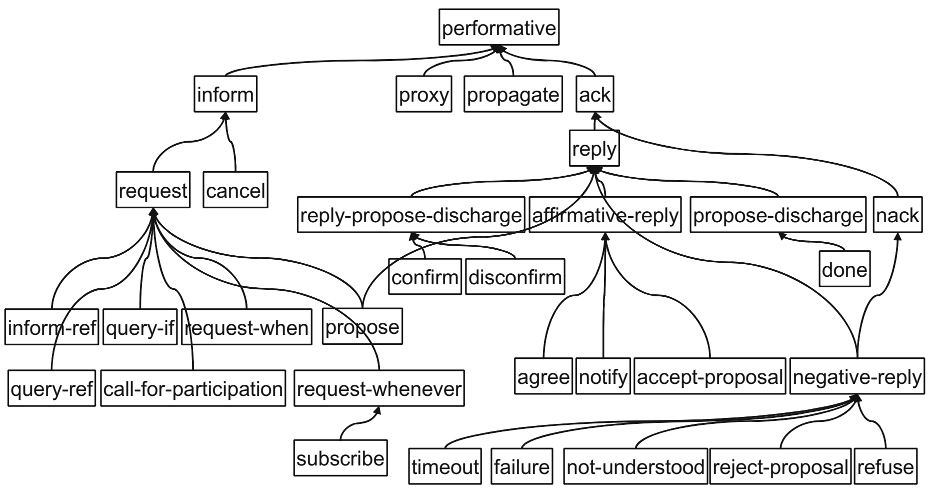


Fig. 1. The CASA performative subsumption lattice

sify the message and decide on a course of action. Because certain performatives are subtypes of others, we need only specify individual actions once for the parent performative type, and those actions are "inherited" by the child performative types. Thus, we eliminate the redundancies and simplify the specification significantly. We also eliminate the need for *every* agent to *always* be updated on the semantics of every extension to the performatives lattice: they can always interpret a new type of performative in terms of its parent type. (And for an agent to ask for the parentage of an unknown performative is a trivial operation which is supported by our infrastructure.)

1.2 The CASA Architecture

The CASA architecture [13] is an experimental infrastructure on which agents can be implemented. CASA agents work by exchanging messages (via TCP/IP or by local method calls) which consist of key/value pairs. The keys in the messages are the various FIPA message field names, but may also include other, extended keys, as appropriate.

The CASA architecture is a general purpose agent agent environment, but defines several specialized agents (see Figure 2). CASA defines computational *areas* (usually a single computer), and each area has exactly one *Local Area Coordinator* (LAC) agent. A LAC agent is a registry of agents for its area, and is responsible to act as a "white pages directory" for its area, to run agents on behalf of agents in its or other areas, as well as to perform several other duties. Another important type of agent is a *Cooperation Domain* (CD), which acts like a "meeting room" for agents. Agents may *join* and then send messages to a CD which, by default, re-broadcasts the message to all of its members. CDs are particularly useful for third-party observers of agent conversations. These

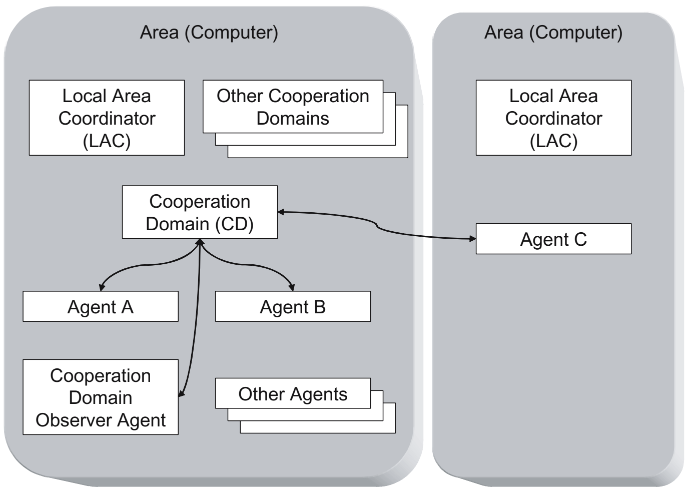

Area (Computer)
Local Area Coordinator (LAC)
Other Cooperation Domains
Cooperation Domain (CD)
Agent A
Agent B
Cooperation Domain Observer Agent
Other Agents
Area (Computer)
Local Area Coordinator (LAC)
Agent C

Fig. 2. The CASA architecture

observer agents can analyze agent behaviour on the behalf of the larger society of agents for various purposes such as analysis, possible sanctioning of rogue agents [11,16], or merely reporting unacceptable, malicious, or erroneous behaviour.

CASA is particularly concerned with agent behaviour and the observability of agents' behaviour. Unfortunately, the semantics behind FIPA's model is based on the BDI (Beliefs, Desired, Intensions) model, which has long been criticized as requiring “omniscient” knowledge of the internal workings of all agents in the environment [17]. Since the inner workings of agents is not typically available to an outside observer, the observer cannot predict expected behaviour of agents. Therefore, an observer has no formal bases on which to judge agent behaviour as “acceptable”, “harmful”, “malicious”, “useful”, etc. to the society of agents.

An alternate agent model is the commitment-based model [1]. Communicative acts between agents generate social commitments, which form a social “contract” among the agents. Assuming the communicative acts can be observed (as CASA is careful to support), an outside observer can infer social commitments among the observed agents. Our model is formally specified [3,4,5] and forms a clean formal basis on which an observer *can* decide whether or not a particular agent is fulfilling its social commitments, and therefore has a sound foundation on which to judge agents' behaviour.

2 Messages and Performatives

As stated in the introduction, we wish to simplify the specification of possible agent behaviour. As a step in that direction, we arrange our communicative acts, which we base on the FIPA standard, in a subsumption lattice of performatives as described in Figure 1. In the lattice, every child performative inherits the attributes of all of its ancestor performatives. In particular, we can associate *policies* with any performative, which will be inherited by all children of that performative. This is described in detail in Section 3.

Note that the performatives in Figure 1 are actually a superset of the performatives defined by FIPA. Some of the new performatives are classes of performative types which do not add any real semantic information to their children, but serve to enable our agents (and their observers) to more easily classify performatives into broader categories; thus allowing for more "superficial" specification where appropriate. For example, an observer, Carol, may note that an agent, Bob, sent a *request* to agent Alice, and that Alice replied with a *failure* performative. If Carol is tracking only social commitments, then she would not care if Alice had replied with a *failure*, a *non-understood*, a *reject-proposal*, a *refuse*, or some other descendent of *nack* and *reply*; in any of these cases, there is no social commitment entailed. Indeed, Carol need not understand the performative in the reply send by Alice, as long as she is aware (by looking it up in the appropriate ontology) that the performative in Alice's reply is subsumed by a *nack* (negative acknowledge).

Other extensions to the FIPA performatives include the addition of an *ack* (acknowledge) performative, which, in CASA, serves as an optional top-level method of checking receipt of messages. The use of *ack* will be be further explained in the light of social commitments in Section 3.

3 Commitments

We model agent communication as generating (or deleting) social commitments, thus allowing observation of the state of social commitments within a society of agents. More specifically, the performatives in agent communication acts (messages), are translated (by a set of *polices*) to a set of *social commitment operators*, which either add or delete a specific class of social commitments. We model a *social commitment* as the promise by a *debtor* agent to a *creditor* agent(s) to do some *action*:

$$(debtor, creditor, action)$$

and we model a *social commitment operator* as either an *add* or *delete* of a commitment (refer to [7] for a detailed view of the life cycle of commitments):

$$(add|delete, socialCommitment)$$

We have defined several *polices* (eg: *propose, accept, reject, counter*, and *inform*) [5] which can be *applied* to an agent's outgoing and incoming messages and set of social commitment operators:

$$apply\colon\ message \times \mathbf{P}policy \times ontology \to \mathbf{P}socialCommitmentOperator$$

Here, we mean that if we observe an agent's incoming or outgoing message, we can interpret it in the context of the agent's (or the society of agent's) policies and ontology. (The ontology is necessary to provide a semantics for the performatives.) Of course, not all the policies are *applicable* to a particular message;

Table 2. An informal description of the conversation policies as defined by Flores and Kremer. (The names of some of the policies have changed since the original work).

Policy	**Description**
P-inform	commits the addressee to acknowledge
P-ack	releases informed agents of the commitment to acknowledge
P-request	commits the proposed agents to reply
P-counteroffer	commits addressees to reply
P-reply	releases proposed agents of the commitment to reply and releases counteroffered agents of the commitment to reply
P-agree	an acceptance realizes the shared uptake of proposed/counteroffered commitments
P-done	releases accepted agents of the commitment earlier agree

a matching function (see Section 4.1) is used to choose the subset of applicable policies. The applicable polices are then *executed* to produce the set of social commitment operators.

Furthermore, we can *commit* this set of social commitment operators to an existing set of social commitments:

$$\begin{aligned} &\mathit{commit}\colon \mathbf{P}\mathit{socialCommitment} \times \mathbf{P}\mathit{socialCommitmentOperator} \\ &\qquad\qquad\qquad\qquad \to \mathbf{P}\mathit{socialCommitment} \end{aligned}$$

Thus, it is easy to build up (or reduce) a set of social commitments based on observed messages. Note that this is just as easy for an individual agent to track its own social commitments (as in our implementation) or for a 3rd party observer to track all of the social commitments of a society of agents (as in Heard's study of sanctioning of rogue agents [11]).

Table 2 informally describes some of the fundamental polices we have defined so far. The policies are meant to be used by a community of agents as a description of "social norms". The policies are used to map our FIPA-based performatives to social commitments.

4 Using Social Commitments with Performatives

As already alluded to, we effectively use policies to annotate the performative lattice with social commitment operators to form expectations about agent behaviour (the "normative" behaviour of agents in a society of agents). Figure 3 illustrates some of the polices by describing the relationship between (part of) the performative lattice and commitments through policies and commitment operators. The performative lattice on the left, and the curved arrows originating on the performatives represent the policies that indicate the associated social commitment operators (center right column). The arrows originating in the commitment operators illustrate the type of the commitments' third parameter (an *action*) and terminate on the action subtype of the action. Since these particular policies are about conversational acts, all of these arrows (except the last two) terminate on subtypes of *communication-act*.

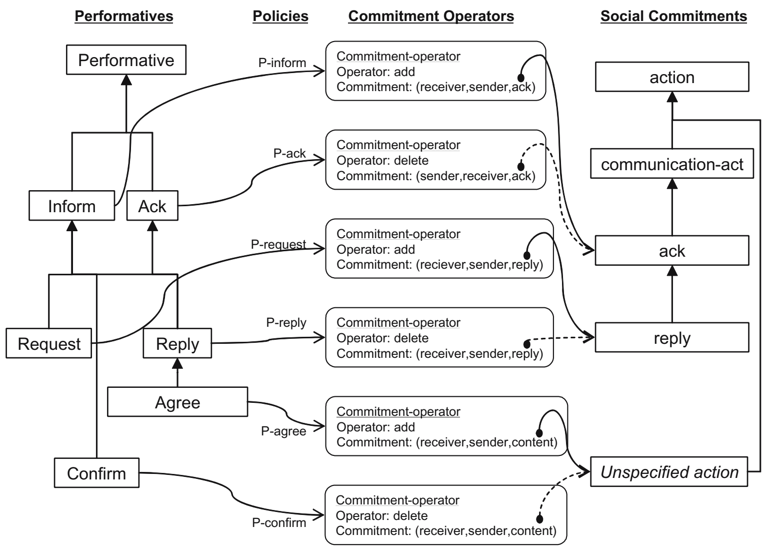


Fig. 3. Part of the CASA performative subsumption lattice together with their relationship via policies and performative operators to social commitments. The *policies* are labelled with the policy names from Table 2.

The curved arrows between the performatives and the social commitment operators in Figure 3 represent some of the policies described in [5] and informally described in Table 2. For example, the P-inform policy associated with the *inform* perforative would read "if Bob receives a message with an inform performative from Alice, then there exists a social commitment for Bob to send an acknowledgement to Alice ($\exists sc\colon socialCommitment, x\colon ack \bullet sc = (Bob, Alice, x)$)".

The reading of the *request* performative's P-request policy is a bit more complex. Because *request* is a subtype of *inform*, not only do we have to apply the P-request policy, but also the P-inform policy as well (and likewise up the lattice for every ancestor performative). So we would read the P-request policy as "if Bob receives a message with a *request* performative from Alice, then there exists a social commitment for Bob to send an acknowledgement to Alice and another social commitment for Bob to send a reply to Alice, ($\exists sc_1, sc_2 : socialCommitment, x_1\colon ack, x_2\colon reply \bullet sc_1 = (Bob, Alice, x_1) \wedge sc_2 = (Bob, Alice, x_2)$)".

This may seem somewhat redundant since a single conversational act (*request*) makes two (very similar) social commitments. But it makes sense and yields needed flexibility. If Alice were requesting Bob attend a meeting, Bob might not have his calendar with him, so might not be able to *reply* to Alice, but could *acknowledge* that he had received the request ("I'll check my calendar"). Alice would then know that Bob had received the request and the social commitment to acknowledge would be deleted (by policy *P-ack*), but the social commitment for Bob to reply to Alice would remain. Later, Bob would reply (affirmatively [*agree*] or negatively [by some reply that is subsumed by *nack*]), and that would remove the second social commitment (by policy P-reply). And that would end the conversation because there would exist no more conversational social commitments between the two. (Well, not quite: if Bob had replied affirmatively [using an *agree* performative], then Bob and Alice would uptake the social commitments for Bob to attend the meeting and to tell Alice about it [by policy P-agree] – but we will get into those details later in Section 5.)

On the other hand, if Bob *did* have his calendar with him when Alice requested he attend the meeting, then does Bob have to send an acknowledgement to Alice, and *then* send a reply to Alice? That wouldn't be very efficient. Fortunately, Bob doesn't have to respond twice: If Bob immediately sends a *reply* to Alice, then the social commitment to reply will be removed (by policy P-reply), *and so will the social commitment to acknowledge.* Why? Because, by virtue of *reply* being subsumed by *ack*, the *reply* will generate two commitment operators

$$\begin{aligned}&\exists delReply, delAck\colon socialCommitmentOperator \bullet\\ &\quad \exists r\colon reply, a\colon ack \bullet\\ &\qquad delReply = (delete, (Bob, Alice, r)) \wedge delAck = (delete, (Bob, Alice, a))\end{aligned}$$

which will remove both of the commitments set up by the original request.

4.1 Implementation with Social Commitments

Thus, agents can be implemented by dealing with incoming messages by merely applying all the policies associated with the performative in the message and also those policies associated with all of the ancestors of the performatives in the message. These polices will either add or delete social commitments. It is important to note that this is also exactly what an observer does as well: The social commitments are in the context of the entire community of agents, so an observer's record of social commitments should always be consistent with (be a superset of) any observed agent's record of social commitments.

Agents do not have to be *implemented* using social commitments (as may have been implied by the previous paragraph). Observers can still use social commitments to formulate a model of agent behaviour regardless of *how* the agent is implemented. The policies merely form a codification of social norms. An agent that is not implemented using social commitments (who is well behaved) would still be regarded as not breaking any commitments by an observer using reasonable social commitment policies (like the ones in Table 2).

CASA implements its agents as either social commitment agents as listed above, or as *reactive* agents. Both kinds of agents use the same set of named policies, but the difference is that the policy *implementation* is different. When a social commitment-based agent "sees" an incoming or outgoing message, it merely applies it's policies to add or delete social commitments; later (during otherwise idle time) it will attempt to discharge any social commitments (for which it is the debtor) by executing them when it can. On the other hand, reactive agents will respond to a message immediately (without "thinking") whenever it "sees" an incoming message. Reactive agents do nothing in idle time, do nothing with outgoing messages, and don't keep a record of social commitments.

Both types of agents follow the same normative protocols, but the sequence of messages is usually quite different. For example, social commitment agents may easily and naturally choose to prioritize their tasks; reactive agents can't handle prioritized tasks easily.

4.2 Formal Model

It only remains to more formally describe how to apply social commitment operators to an agent's record of social commitments. If we assume an agent's record of social commitments is a set, SC, the operator op is applied as follows:

$$\begin{aligned}&\forall op\colon socialCommitmentOperator, sc\colon socialCommitment \bullet\\ &\quad op = (add, sc) \rightarrow SC' = SC \cup sc \wedge\\ &\quad op = (delete, sc) \rightarrow SC' = SC \backslash match(sc, SC)\end{aligned}$$

(In the above, we use SC$'$ to represent the value of SC *after* the operation has taken place, *á la* Z [2].) That is, an add operator just inserts a new social commitment into the record, and a delete operator just removes any matching social commitments from the record. The *match* function takes a commitment and a set of commitments and returns a subset of the second argument as follows:

$$\begin{aligned}&\forall sc : socialCommitment, SC\colon \mathbf{P} socialCommitment \bullet match(sc, SC) \equiv\\ &\quad \{i \in SC | sc.debtor = i.debtor \wedge sc.creditor = i.creditor \wedge\\ &\qquad typeOf(sc.action) \sqsubseteq typeOf(i.action)\}\end{aligned}$$

The reader may have noticed that there is no order specified on the application of several operators in response to a message, and, as a result, it is therefore possible that a delete operation may not remove any social commitments at all. In fact, this could be the case in Alice and Bob's meeting. If Bob were to reply to Alice (without first sending an acknowledgement) and the observer first applied the (delete,(Bob,Alice,reply)) operator, it would remove *both* the (Bob,Alice,reply) and the (Bob,Alice,ack) social commitments from the social commitments record. Then, when the observer applied the second operator, (delete,(Bob,Alice,ack)), there would be no change to the social commitments record. Our choice is not to worry about such null deletions, but other implementations may wish to avoid such empty applications either by applying the least specific deletions first if there is a subsumption relationship among operators, or by changing the match() function to only match on the most specific social commitment in the argument set.

Space limitations prohibit a detailed account of the formalization here, but a detailed formalization may be found in [3].

5 An Example

As a more formal example, we repeat the example of Bob and Alice's meeting using the more formal framework and tracking the conversation through to the end (signaled by there being no more social commitments left from the conversation). Figure 4 shows an interaction diagram of the conversation: Alice first

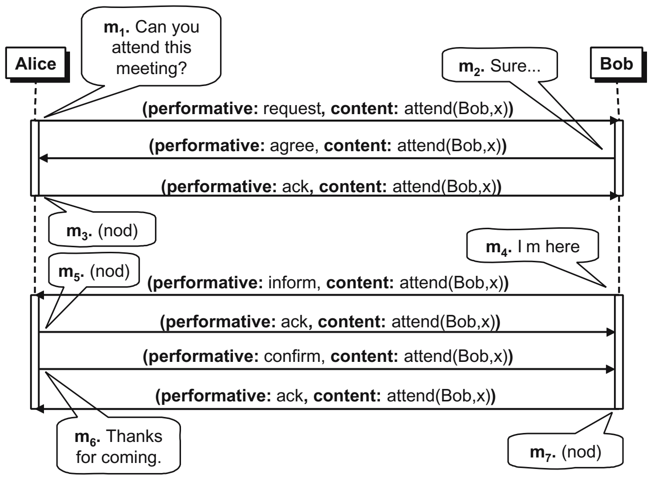

Alice
Bob
m1. Can you attend this meeting?
m2. Sure...
(performative: request, content: attend(Bob,x))
(performative: agree, content: attend(Bob,x))
(performative: ack, content: attend(Bob,x))
m3. (nod)
m4. I m here
m5. (nod)
(performative: inform, content: attend(Bob,x))
(performative: ack, content: attend(Bob,x))
(performative: confirm, content: attend(Bob,x))
(performative: ack, content: attend(Bob,x))
m6. Thanks for coming.
m7. (nod)

Fig. 4. Alice and Bob's conversation about a meeting

asks Bob to attend a meeting, "x" [1], by sending a message to Bob with a *request* performative and a contents describing the request, $(attend(Bob, x))$. Bob immediately confirms his acceptance to attend the meeting, by sending a message back to Alice with an agree performative and the same descriptive content. Alice acknowledges by sending an ack message back to Bob.

Later, Bob sends another message to Alice, informing him that the predicate, $attend(Bob, x)$, is true, that he is currently attending the meeting. Alice acknowledges. Alice then responds by sending a message to Bob with a confirm-complete performative, and the same contents. Bob acknowledges.

Does Alice and Bob's conversation conform to the social norms implied by the policies? Figure 5 describes the conversation in terms of the messages, policies, social commitment operators, and the constantly changing set of social commitments held by both Bob and Alice, and that would be held by an observer listening to the conversation.

Each row in Figure 5 represents the same message passing between the conversational participants as the corresponding cartoon balloons in Figure 4. In row m_1, Alice sends a message with a request performative to Bob and containing the content predicate $attend(Bob, x)$. Then Bob, Alice, and the observer can look up *request* in the policies in Figure 3 and see that there are two applicable policies (by searching up the lattice from the *request* node) representing policy P-inform and P-propose. To apply these policies, we need only apply the operators, which are $(add, (receiver, sender, reply))$ and $(add, (receiver, sender, ack))$. So we add these two social commitments to our set of social commitments.

Note that we have a slight notational difficulty here. We need to contextualize the reply and the *ack* social commitments with *what* to reply/acknoweldge to. In

[1] The meeting is normally described by an expression, but we omit the details here for the sake of brevity.

Message					Policy	Operator	Social Commitments
Id	performative	sender	rec r	content			
m_1	request	Alice	Bob	attend(Bob, x)	P-request P-inform	(add,(Bob, Alice, reply(m_1))) (add,(Bob, Alice, ack(m_1)))	(Bob, Alice, reply(m_1)) (Bob, Alice, ack(m_1))
m_2	agree	Bob	Alice	attend(Bob, x)	P-reply P-ack P-agree P-inform	(delete,(Bob, Alice, reply(m_1))) (delete,(Bob, Alice, ack(m_1))) (add,(Bob, Alice, attend(Bob,x))) (add,(Bob, Alice, p-d(Bob,x))) (add,(Alice, Bob, ack(m_2)))	~~(Bob, Alice, reply(m_1))~~ ~~(Bob, Alice, ack(m_1))~~ (Bob, Alice, attend(Bob,x)) (Bob, Alice,p-d(Bob,x)) (Alice, Bob, ack(m_2))
m_3	ack	Alice	Bob	attend(Bob, x)	 P-ack	 (delete,(Alice, Bob, ack(m_2)))	(Bob, Alice, attend(Bob,x)) (Bob, Alice,p-d(Bob,x)) ~~(Alice, Bob, ack(m_2))~~
m_4	propose- discharge	Bob	Alice	attend(Bob, x)	 P-prop-dis P-inform	 (delete,(Bob,Alice,p-d(Bob,x)) (add,(Alice,Bob,r-p-d(Bob,x)) (add,(Alice, Bob, ack(m_4)))	(Bob, Alice, attend(Bob,x)) ~~(Bob, Alice,p-d(Bob,x))~~ (Alice,Bob,r-p-d(Bob,x) (Alice, Bob, ack(m_4))
m_5	ack	Alice	Bob	attend(Bob, x)	 P-ack	 (delete,(Alice, Bob, ack(m_4)))	(Bob, Alice, attend(Bob,x)) (Alice,Bob,r-p-d(Bob,x) ~~(Alice, Bob, ack(m_4))~~
m_6	confirm	Alice	Bob	attend(Bob, x)	P-confirm P-reply-p-d P-inform	(delete,(Bob, Alice, attend(Bob,x)) (delete, (Alice,Bob,r-p-d(Bob,x)) (add,(Bob, Alice, ack(m_6)))	~~(Bob, Alice,~~ ~~attend(Bob,x))~~ ~~(Alice,Bob,r-p-d(Bob,x)~~ (Bob, Alice, ack(m_6))
m_7	ack	Bob	Alice	attend(Bob, x)	P-ack	(delete,(Bob, Alice, ack(m_6)))	~~(Bob, Alice, ack(m_6))~~

Fig. 5. Alice and Bob's conversation about a meeting

the software, this is just done by attaching a copy of the message, which allows us to take advantage of FIPA's *reply-with* field and unambiguously mark the message as specifically in the context of the original inform/request message. However, here, we use the notation "$reply(message_i)$" to succinctly show the same thing.

The m_2 row of Figure 5 shows Bob immediately agreeing to go to the meeting. (He could have acknowledged receipt of the message first, which would have deleted the $(Bob, Alice, ack(m_1))$ commitment.) He replied with an *agree* performative, which isn't listed in Figure 3, but is a subtype of *affirmative-reply* (see Figure 1). Looking up the policies for *affirmative-reply* in Figure 3 shows that four policies are applicable (representing policies P-reply, P-ack, P-agree, and P-inform). These four policies can be applied in any order, but all sequences will yield the same end result (although intermediate results may differ). Applying these policies in the order given, $(delete, (Bob, Alice, reply(m_1)))$ will delete *both* social commitments $(Bob, Alice, reply(m_1))$ and $(Bob, Alice, ack(m_1))$.

$(delete, (Bob, Alice, ack(m_1)))$ will find nothing to delete (because the "target" has just been deleted), but this is fine. The $(add, (Bob, Alice, attend(Bob, x)))$ operator is parameterized with the action predicate in the contents of the m_2 message, and adds the $(Bob, Alice, attend(Bob, x))$ social commitment to the set of social commitments. Finally, the $(add, (Alice, Bob, ack(m_2)))$ operator adds the required commitment for Bob to acknowledge.

The m_3 row shows Bob acknowledging the previous *agree* message, and removing the social commitment for that acknowledgement.

Time passes, and the meeting commences. In row m_4, Bob informs Alice that he has fulfilled his commitment, $(Bob, Alice, attend(Bob, x))$, to attend the meeting, which invokes two policies, P-inform and P-propose-discharge. This message does *not* remove the $(Bob, Alice, attend(Bob, x))$ commitment. Intuitively, this is because Alice has not yet confirmed that Bob has attended the meeting and has satisfactorily fulfilled his commitment. If Alice were an agent that could sense her environment, and could "see" that Bob were in attendance, Bob would not have to send this message and we wouldn't have to include rows m_4 and m_5 in the table.

Row m_5 shows Alice acknowledging Bob's inform.

In row m_6, Alice has "seen" that Bob is in attendance at the meeting and sends a message with the confirm performative. This invokes three policies (P-done, P-reply-propose-discharge, and P-inform) which delete Bob's outstanding commitments to attend the meeting and to tell Alice about it and adds a commitment for Bob to acknowledge the confirm message.

Finally, in row m_7, Bob acknowledges Alice's last message, which removes the last of the social commitments. There being no more social commitments left, the conversation is over.

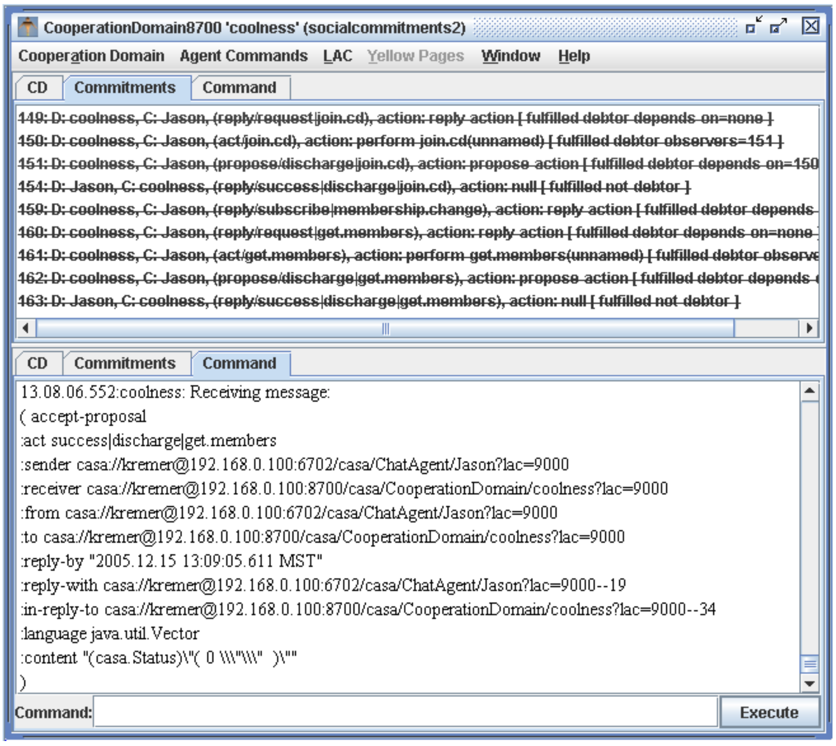
Just so the reader is not left with the impression that this work only applies to hypothetical human examples, we include a snapshot of the CASA system in the process of an actual agent conversation (see Figure 6). Here, we show a Cooperation Domain that has just fulfilled its obligations in three distinct (and interleaved) conversations: a request-to-join-CD conversation, a request-to-subscribe (to be updated on membership changes) conversation, and a request-for-a-membership-list conversation. The upper pane in the snapshot shows the recently discharged social commitments. The lower pane shows the message just received from another agent (called Jason) acknowledging successful completion of the get-members request (a *reply* performative).

5.1 Variations: Flexibility and Efficiency

As already mentioned, if Alice could sense her environment, she could notice on her own that Bob was attending the meeting, and messages m_4 and m_5 (rows m_4 and m_5 in Figure 5) could be omitted. If this were the case, and Bob sent the inform message anyway, the conversation would still not be harmed. The number of the messages in the conversation would drop from 7 to 5.

Our protocols, as defined in Table 2 and Figure 3, call for every message to be acknowledged. This is an option in our system, and can easily be "turned off" by merely deleting the policies in Figure 3 associated with P-inform and P-ack. If we do remove the P-inform policy, then messages m_3, m_5 and m_7 disappear and the number of messages drops from 7 to 4.

By combining both strategies in the previous two paragraphs, we can reduce the number of messages in the conversation from 7 to 3. The resulting conversation appears in Figure 7.



~~149: D: coolness, C: Jason, (reply/request|join.cd), action: reply action [fulfilled debtor depends on=none]~~
~~150: D: coolness, C: Jason, (act/join.cd), action: perform join.cd(unnamed) [fulfilled debtor observers=151]~~
~~151: D: coolness, C: Jason, (propose/discharge|join.cd), action: propose action [fulfilled debtor depends on=150~~
~~154: D: Jason, C: coolness, (reply/success|discharge|join.cd), action: null [fulfilled not debtor]~~
~~159: D: coolness, C: Jason, (reply/subscribe|membership.change), action: reply action [fulfilled debtor depends~~
~~160: D: coolness, C: Jason, (reply/request|get.members), action: reply action [fulfilled debtor depends on=none~~
~~161: D: coolness, C: Jason, (act/get.members), action: perform get.members(unnamed) [fulfilled debtor observe~~
~~162: D: coolness, C: Jason, (propose/discharge|get.members), action: propose action [fulfilled debtor depends~~
~~163: D: Jason, C: coolness, (reply/success|discharge|get.members), action: null [fulfilled not debtor]~~

```
13.08.06.552:coolness: Receiving message:
( accept-proposal
:act success|discharge|get.members
:sender casa://kremer@192.168.0.100:6702/casa/ChatAgent/Jason?lac=9000
:receiver casa://kremer@192.168.0.100:8700/casa/CooperationDomain/coolness?lac=9000
:from casa://kremer@192.168.0.100:6702/casa/ChatAgent/Jason?lac=9000
:to casa://kremer@192.168.0.100:8700/casa/CooperationDomain/coolness?lac=9000
:reply-by "2005.12.15 13:09:05.611 MST"
:reply-with casa://kremer@192.168.0.100:6702/casa/ChatAgent/Jason?lac=9000--19
:in-reply-to casa://kremer@192.168.0.100:8700/casa/CooperationDomain/coolness?lac=9000--34
:language java.util.Vector
:content "(casa.Status)\"( 0 \\\"\\\"  )\""
)
```

Fig. 6. A CASA CD conversing with another agent requesting to join the CD

5.2 Implementation Considerations

Figure 8 shows the conversational "schema" that arises from the polices involved in a typical *request* conversation, like the one between Alice and Bob or between the CD and another agent in Section 5. This figure is from the viewpoint of the actual implementation in CASA. The heavy vertical lines represent the two agents over time. The heavy horizontal arrows indicate messages, and the reader will no doubt notice that there are *eight* messages exchanged in this seemingly simple conversation. The reader should not be put off by this: this is only the worst case, and we have shown how this conversation can be dramatically simplified (optimized) earlier in this section. CASA can do this optimization.

Each of Figure 8's messages are labelled above with their possible performatives and their supertype sublattice. Arrows emerging from the performative names represent the applicable policies and social commitment operators (solid indicates *add*, and dashed indicates *delete*). The policy arrows terminate on shared (underscored) and private (grayed) social commitments. Some interesting details of the theory and implementation are shown in this diagram that aren't explicit elsewhere in this paper:

The lighter-colored (non-underscored) private social commitments in the figure form the method we use to attach agent executable code (usually a method

Message					Policy	Operator	Social Commitments
Id	performative	sender	rec r	content			
m_1	request	Alice	Bob	attend(Bob, x)	P-request	(add,(Bob, Alice, reply(m_1)))	(Bob, Alice, reply(m_1))
m_2	done	Bob	Alice	attend(Bob, x)	P-reply P-prop-dis	(delete,(Bob, Alice, reply(m_1))) (add,(Alice,Bob,r-p-d(Bob,x))	~~(Bob, Alice, reply(m_1))~~ (Alice,Bob,r-p-d(Bob,x))
m_6	confirm	Alice	Bob	attend(Bob, x)	P-reply-p-d	(delete,(Alice,Bob,r-p-d(Bob,x))	~~(Alice,Bob,r-p-d(Bob,x))~~

Fig. 7. Alice and Bob's conversation about a meeting, without Bob's inform to Alice, and without policy P-inform

call) to the policies: one needs to reference some bit of the agent's code to "wake" the agent to a particular event. These private social commitments are always bound to an *inform*, but are usually referenced from some subtype of of *inform á la* the *template method* design pattern [10]. These template references are represented in the figure by the light-colored curved arrows among the performatives in the sub-lattices at center.

The curved arrow on the extreme left and right of the diagram connecting social commitments are *dependencies* between social commitments. This is a powerful concept that is easily implemented by the *observer* design pattern [10], and arises naturally in the system. For example, naturally, one needs to actually *perform* an action before proposing to discharge it.

6 Related Work

Conversations and commitments have been studied in argumentation [19], where the evolution of conversations is motivated by the commitments they imply, and which are not necessarily made explicit. Others have looked into the mechanics of conversations using operations advancing the state of commitments, which is a view independent of the intentional motives behind their advancement [6,12,14,15,18]. We share these views, and aim at identifying public elements binding the evolution of conversations.

7 Conclusion

The main contribution of this paper is to show how the FIPA performatives can be mapped onto a social commitment theory framework to allow observable social behaviour. "Rules" (or policies), like those described in this paper, act as a codification of social norms, so can be easily used by an observer to judge whether an agent is well behaved relative to the social norms. Social commitments, and the ontology of performatives can be used to implement agents, but agents do not have be to implemented as social commitment-style agents to be observed and monitored by an observing agent using commitments as described here.

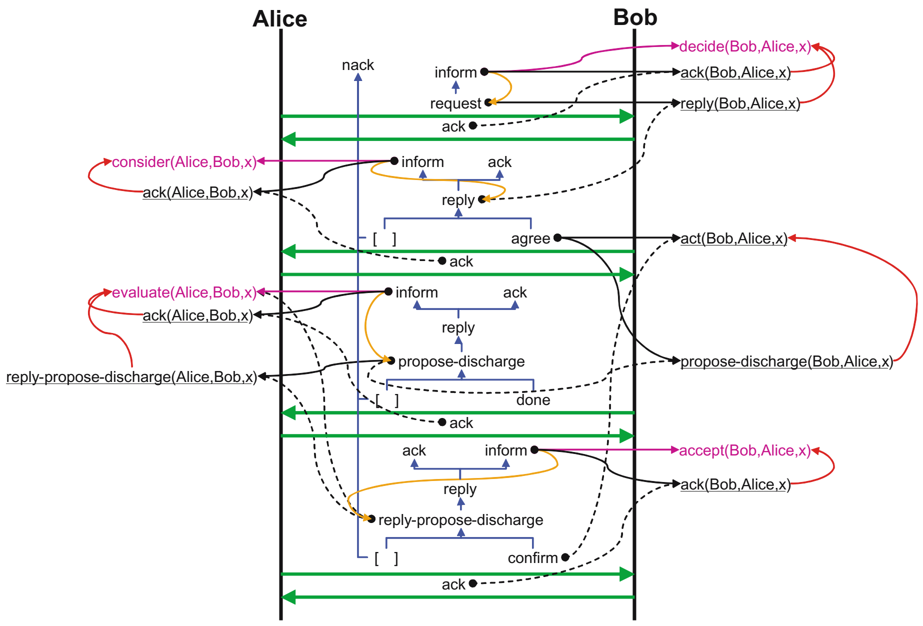


Fig. 8. An implementation view of the policies associated with a typical client-server *request* conversation

Acknowledgments

The authors thank the Canadian Natural Science and Engineering Research Council (NSERC) for their support.

References

1. C. Castelfranchi. Commitments: From individual intentions to groups and organizations. In *Proceedings of the First International Conference on Multi-Agent Systems*, pages 41–48, San Francisco, CA, June 1995.
2. A. Diller. *Z: An Introduction to Formal Methods.* John Wiley & Sons, Inc., Sussex, England, 1990.
3. R.A. Flores. *Modelling agent conversations for action.* Ph.D. thesis, Department of Computer Science, University of Calgary, June 2002.
4. R.A. Flores and R. Kremer. Formal conversations for the contract net protocol. In V. Marik, O. Stepankova, H. Krautwurmova, and M. Luck, editors, *Multi-Agent Systems and Applications II*, volume 2322 of *Lecture Notes in Artificial Intelligence*, pages 169–179. Springer Verlag, 2002.
5. R.A. Flores and R. Kremer. To commit or not to commit: Modelling agent conversations for action. *Computational Intelligence*, 18(2):120–173, 2003.
6. R.A. Flores and R. Kremer. Principled approach to construct complex conversation protocols. In A.Y. Tawfik, and S.D. Goodwin, editors, *Advances in Artificial Intelligence*, volume 3060 of *Lecture Notes in Artificial Intelligence*, pages 1–15. Springer Verlag, 2004.

7. R.A. Flores, P. Pasquier, and B. Chaib-draa. Conversational semantic sustained by social commitments. In F. Dignum, and R. van Eijk, editors, *Autonomous Agents and Multi-Agent Systems*. To appear.
8. Foundation for Intelligent Physical Agents (FIPA). FIPA ACL message structure specification. document number SC00061G, FIPA TC communication. http://www.fipa.org/specs/fipa00061/SC00061G.html, Dec. 2003.
9. Foundation for Intelligent Physical Agents (FIPA). FIPA communicative act library specification. document number SC00037J, FIPA TC communication. http://www.fipa.org/specs/fipa00037/SC00037J.html, Dec. 2003.
10. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series. Addison-Wesley, Reading, Mass., 1994.
11. J. Heard and R. Kremer. Detecting broken social commitments. In this volume.
12. S. Khan and Y. Lesperance. On the semantics of conditional commitments. In this volume.
13. R. Kremer, R.A. Flores, and C. LaFournie. *Advances in Agent Communication*, chapter A Performative Type Hierarchy and Other Interesting Considerations in the Design of the CASA Agent Architecture. In F. Dignum, and R. van Eijk, and M-P. Huget, editors, *Advances in Agent Communication*, volume 2922 of *Lecture Notes in Computer Science*, pages 59–74. Springer Verlag, 2004. Available: http://sern.ucalgary.ca/ kremer/papers/-AdvancesInAgentCommunication_KremerFloresLaFournie.pdf.
14. A. Mallya and M. Singh. Introducing Preferences into Commitment Protocols. In this volume.
15. P. Pasquier, M. Bergeron, and B. Chaib-draa. Diagal: A generic ACL for open systems. In M.-P. Gleizes, A. Omicini, and F. Zambonelli, editors, *ESAW*, volume 3451 of *Lecture Notes in Artificial Intelligence*, pp 139–152. Springer Verlag, 2004.
16. P. Pasquier, R.A. Flores, and B. Chaib-draa. Modelling flexible social commitments and their enforcement. In M.-P. Gleizes, A. Omicini, and F. Zambonelli, editors, *ESAW*, volume 3451 of *Lecture Notes in Artificial Intelligence*, pages 153–165. Springer Verlag, 2004.
17. M. Singh. Agent communication languages: Rethinking the principles. *IEEE Computer*, 31(12):40–47, 1998.
18. M. Verdicchio and M. Colombetti. A commitment-based Communicative Act Library. In this volume.
19. D. Walton and E. Krabbe. *Commitment in Dialogue: Basic Concepts of Interpersonal Reasoning*. State University of New York Press, 1995.