A Principled Modular Approach to Construct Flexible Conversation Protocols

Roberto A. Flores[1] and Robert C. Kremer[2]

[1] Laval University, Department of Informatics, Sainte-Foy, QC, G1K 7P4, Canada,
flores@damas.ift.ulaval.ca
[2] University of Calgary, Department of Computer Science, Calgary, AB, T2N 1N4, Canada
kremer@cpsc.ucalgary.ca

Abstract. Building conversation protocols has traditionally been an art more than a science, as their construction is often guided by designers' intuition rather than by a principled approach. In this paper we present a model for building conversation protocols using inference principles that allow the computational specification and verification of message sequencing and turn-taking. This model, which is based on the negotiation of social commitments, results in highly flexible protocols that support agent heterogeneity while abiding by software engineering practices. We exemplify the specification of protocols using the contract net protocol, a common interaction protocol from the multiagent literature.

1 Introduction

Traditionally, conversations in multiagent systems have been regulated through the use of conversation protocols. More often than not, system designers define these protocols according to the sequences of messages they intuitively believe are best to bring about the actions achieving the goals of their systems. Although such informal approaches free designers of methodological constraints, they reduce protocols to monolithic conversational units with no explicit state properties, a characteristic that limits their implementation in open environments [9][15] and their reuse throughout application domains [1][14]. At the heart of these concerns is the absence of formal principles to build protocols supporting sound software engineering practices (e.g., modularity) as well as designers' autonomy to build heterogeneous agents. Flexible protocols – defined in implementation-independent terms – are needed to achieve seamless interactions between agents programmed using dissimilar techniques and of various levels of sophistication and contextual responsiveness [9]. This versatility requires principles that could be programmed in offline analysis tools (to verify the correctness of protocols at design time) and could also be encoded in deliberative agents as rules (upon which they could infer their most appropriate conversational participation at runtime) [15].

We propose a model to build conversation protocols that fulfill these requirements. This model is based on the notion that conversation protocols aim at the orderly execution of actions, and that responsibilities to perform these actions are established through a series of negotiations to adopt social commitments. In particular, our proposal explicitly indicates the messages allowed (i.e., sequencing) and the agent expected to issue the next message (i.e., turn-taking) in all conversational states.

There have been several recent efforts to define conversation protocols using social commitments, particularly the approaches furthered in [8] and [15]. We share with these approaches the view that message sequencing is reflected through the properties afforded by the progression in the states of communicated commitments. However, these models differ from our proposal in that they fail to formally specify turn-taking as an emergent property of conversational states, and still rely on *ad-hoc* links to indicate the participant advancing the state of commitments at any point in a conversation. Instead, turn-taking in our model flows logically as agents dispose of their obligations (derived from the negotiation of social commitments) by performing both communicative and non-communicative actions. As we detail in this paper, these obligations indicate the types of messages that could be uttered (sequencing) as well as agents expected to issue the next message advancing a conversation (turn-taking), thus defining state properties on which the verification, compilation and execution of protocols can be based. Lastly, our model achieves this functionality while supporting software engineering principles through the modular composition of protocols from reusable components.

The structure of this paper is as follows: the next section is devoted to describing the elements in our model and their underlying principles. This section includes a detailed example of how conversation protocols are defined for simple one-action activities, and a brief explanation on how larger protocols involving several actions can be composed using simpler ones. A subsequent section reviews the characteristics afforded by our model, and discusses how these characteristics allow the modular and reusable composition of flexible protocols and their support for autonomy in open multiagent systems.

2 Modelling Conversations for Action

The notion of *social commitments* [2][11] has been advanced as a way to raise expectations about other agents' performances. Specifically, a social commitment can be defined as an engagement in which an agent is responsible relative to another agent for the performance of an action (independent of whether the agent responsible is also the performer of the action). In our model, social commitments are represented as a three-term predicate of the form

$$\forall\, d, c : \downarrow Agent;\ \ a : \downarrow Action \bullet SC(d, c, a)$$

where d and c are agent instances representing the debtor (the agent responsible for the satisfaction of the commitment) and the creditor (the agent on whose behalf the commitment is to be satisfied) of the commitment, and where a is the action that satisfies the commitment. Due to their extent, the description of actions requires a subsequent section of its own.

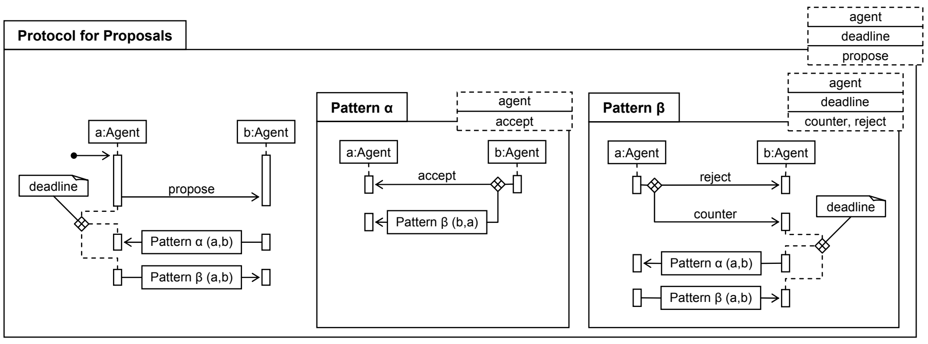


Fig. 1. Interaction diagram of the *protocol for proposals.*

2.1 The negotiation of social commitments

Communicative acts. Inspired by notions from the study of language use [3], we define message interactions as communicative acts from a speaker to an addressee conveying a collection of conversational tokens; and specify the following four tokens to support the negotiation of social commitments:

– *Propose:* to put forth the adoption or discard of a social commitment,
– *Accept:* to accept adopting or discharging a social commitment,
– *Reject:* to reject adopting or discharging a social commitment, and
– *Counter:* to reject a previous proposal while putting forth another proposal to be considered instead.

Lastly, we define a fifth token *Inform* to communicate data.

The protocol for proposals. It is one thing to define communicative acts and quite another to describe how they are used and what they can accomplish in conversations. To that end, we define a negotiations protocol that we call the *protocol for proposals* (*pfp*), which provides a flexible and unambiguous pattern of conversational turn-taking supporting the mutual adoption and discharge of social commitments. As shown in Figure 1, the protocol starts with a proposal from agent a to agent b. This message can be followed (before the expiration of a reply deadline) by the interaction patterns α or β. The interaction pattern α indicates that either agent b sends an accepting message to agent a, or that the interaction continues with pattern β (but with agents a and b's participatory roles inverted, that is, the role of the agent that in pattern α was agent a in pattern β will be agent b, and likewise for agent b). Interaction pattern β indicates that agent a sends a rejection or counterproposal message to agent b, in which case the interaction follows (before the expiration of a reply deadline) by either pattern α or pattern β. In brief, given a proposal from a, b could reply with an acceptance, rejection or counterproposal, or a could issue a rejection or counterproposal to its own proposal. All replies except a counterproposal terminate the instance of the protocol. In theory, a counterproposal can follow another counterproposal *ad infinitum*; in practice, however, successive counterproposals are limited by the reasoning, competence or endurance of interacting agents. Lastly, when an acceptance is issued, both a and b simultaneously apply the proposed (and now accepted) social commitment operation to their record of social commitments, which then affects their record of obligations according to whether they participate as performers in the committed actions.

We specify the *pfp* using three conversation policies:

- *Policy 1:* issuing a proposal (i.e., a message containing a *propose* token) creates an obligation in which the hearer replies with an acceptance, rejection or counterproposal (i.e., a message containing an *accept*, *reject* or *counter* token, respectively) with identical characteristics as those of the proposal.
- *Policy 2:* issuing a reply (i.e., an acceptance, rejection or counterproposal) discards an obligation where the speaker issues a reply.
- *Policy 3:* issuing a proposal followed by a matching acceptance (within proper reply time limits) results in the adoption or discard (whatever operation is involved) of a shared social commitment and the corresponding obligations.

In addition to these policies, we define another policy to raise the expectation that certain social commitments will not remain indefinitely once that they are adopted (thus indicating to the interacting agents that any obligations acquired by adopting the commitment will also be discharged). As such, we define the following additional policy

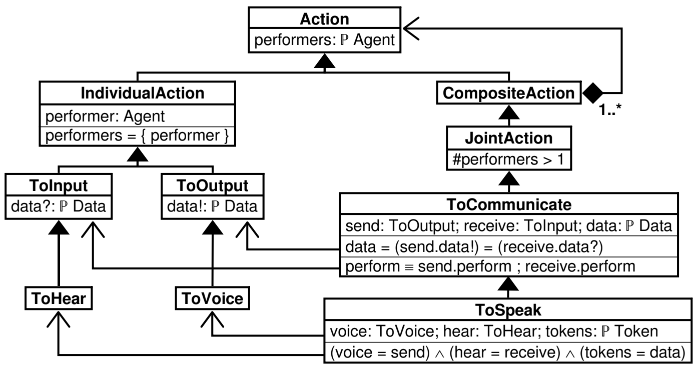
- *Policy 4:* once a dischargeable social commitment (a subtype of social commitment) has been adopted, there is an agent (usually the one committed to the action) that is responsible for proposing its discharge.

Although the negotiation of social commitments is the central piece upon which the model for conversations is founded, it is nevertheless a vehicle to an end: that of giving rise to social obligations to action. That is, by mutually agreeing to uptake social commitments, agents not only adopt the shared state of these commitments, but also uptake obligations to perform the involved actions (if and only if these agents are specified as the performers of the actions).[3]

2.2 Actions

Figure 2 shows the basic set of action definitions in our model. As indicated in this figure, actions are defined as either individual or composite, where an individual action is an atomic action performed by one agent, and a composite action is a collection of actions performed by one or more agents. Based on the latter, we (simplistically) define joint actions as actions in which there is more

[3] Whether agents abide to their obligations is a matter of study in rational social action (e.g.,[4]), where issues such as norms, trust, reliance, sanctions and reputation influence agents' willingness to comply.



Action
performers: ℙ Agent
IndividualAction
performer: Agent
performers = { performer }
CompositeAction
1..*
JointAction
#performers > 1
ToInput
data?: ℙ Data
ToOutput
data!: ℙ Data
ToCommunicate
send: ToOutput; receive: ToInput; data: ℙ Data
data = (send.data!) = (receive.data?)
perform ≡ send.perform ; receive.perform
ToHear
ToVoice
ToSpeak
voice: ToVoice; hear: ToHear; tokens: ℙ Token
(voice = send) ∧ (hear = receive) ∧ (tokens = data)

Fig. 2. Class diagram of basic actions.

than one performer. The individual actions *ToOutput* and *ToInput* are defined as actions that generate an output and receive an input, respectively. These actions are used in the joint action *ToCommunicate*, which specifies that the output of the *send* action equals the input of the *receive* action, and that the performance of the *send* action precedes that of the *receive* action. Lastly, *ToSpeak* is defined as an action inheriting from *ToCommunicate* in which the *send* and *receive* actions are specialized to the actions *voice* and *hear*, respectively, and where the communicated data is a set of conversational tokens. It is worth mentioning that these and all concepts in the model were defined using the Object-Z specification language [12], which is based on first-order predicate logic, set theory and object-oriented principles. Although in this paper we show a minimum of the model's formal definitions, a more comprehensive account can be found in [5] and [6].

2.3 Creating protocols: The Contract Net Protocol

The contract net protocol (*cnp*) [13] is a well-know mutual selection strategy in which a manager agent delegates the performance of actions to agents from a set of contender bidding agents. An instance of the protocol begins when a manager sends a request for proposals to agents that could potentially perform these actions. These agents, if willing, submit a proposal to the manager, who then evaluates submitted proposals to select the most suitable candidates to whom to delegate the actions. The protocol terminates when results are send from the delegated agents to the manager. In this section, we show how the principles of our model can be used to define such a conversation protocol between a manager and a bidder. We begin by defining the various data, actions, and agent roles, as well as the joint activity encompassing actions and participants (i.e., agents in specific roles). There are three actions involved in the *cnp*: offering to perform a set of actions (in the form of a proposal), evaluating a set of actions (in the form of an evaluation), and executing a set of actions (in the form of a contract). In this paper, each of these actions and corresponding agent roles are defined independently and then are assembled in a contract net activity, thus illustrating the modular features afforded by our model.

Defining joint actions. The initial peer interaction in the *cnp* is a request from the manager to a bidder to submit a proposal. Abstractly, this interaction can be seen as a request to offer to perform a set of actions.[4] This action is defined as

$$\begin{array}{|l}
\hline
\textit{ToOffer} \\
\textit{ToProduce} \\
\hline
\textit{input}, \textit{output} : \mathbb{P}\, \textit{ConstrainedAction} \\
\hline
(\textit{input} = \textit{in}) \wedge (\textit{output} = \textit{out}) \wedge \\
(\forall\, i, o : \mathbb{P} \downarrow\!\textit{Action} \mid i = \{a : \downarrow\!\textit{Action} \mid \exists\, i_1 : \textit{input} \bullet a = i_1.\textit{action}\} \wedge \\
\qquad\qquad o = \{a : \downarrow\!\textit{Action} \mid \exists\, o_1 : \textit{output} \bullet a = o_1.\textit{action}\} \bullet o \subseteq i) \\
\hline
\end{array}$$

where *ToOffer* is a class inheriting from the joint action *ToProduce* (not shown). *ToProduce* defines an individual action receiving an input and producing an output that is followed by a speaking action in which the producer informs the receiver of this output. *ToOffer* shows *input* and *output* as data sets of *ConstrainedAction* items, which are generically specified as an action schema with constraints (not shown), and that the actions in *output* are a subset of the actions in *input.* The action definitions to evaluate (*ToEvaluate*) and to execute actions (*ToExecute*) are defined along the same lines.

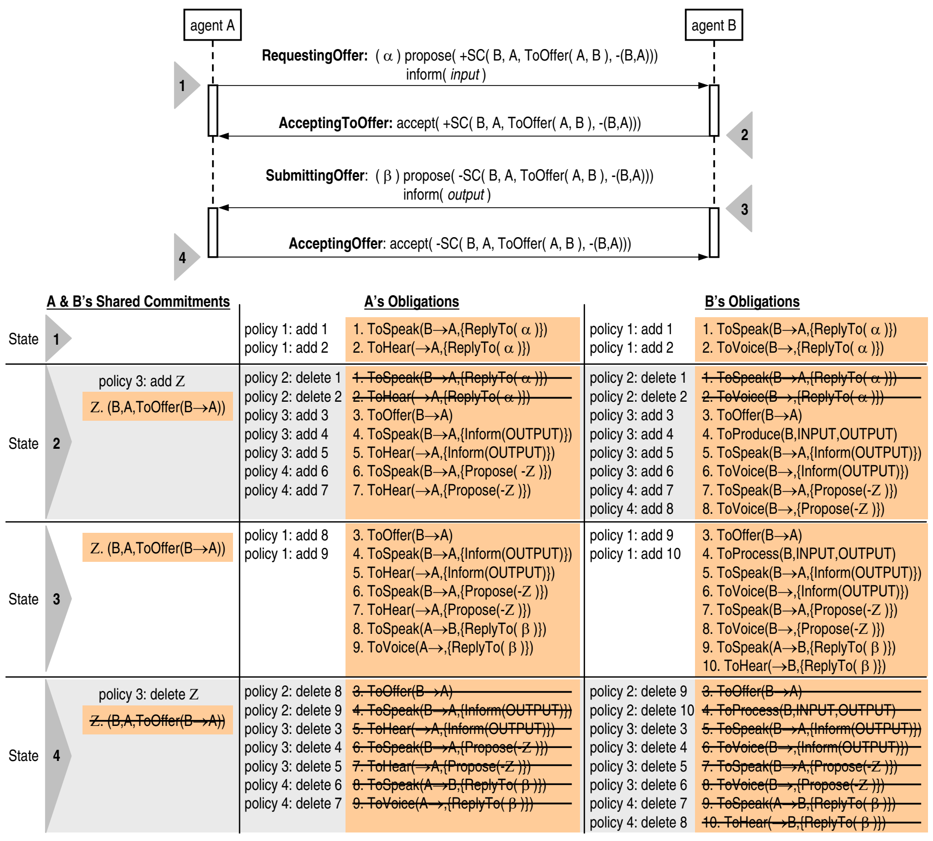
Defining agent roles. There are two agent roles involved in a *ToOffer* action: a producer and a receiver. Figure 3 shows the class *OfferRequester* that defines the receiver agent role. This class specifies *RequestingOffer* and *AcceptingOffer* as the utterances that agents of this type are allowed to issue (for simplicity, we omitted other *pfp*-allowed utterances, such as counterproposals and rejections). *RequestingOffer* is modelled as the uttering of a speaking act whose characteristics comply with the specifications in *ToProposeToAdoptOffering* and *ToInformConstrainedActions*, which indicate a message proposing to adopt a commitment to offer, and informing a set of action constraints, respectively. Similarly, *AcceptingOffer* models the uttering of a speaking act compliant with *ToAcceptToDischargeOffering*, which specifies a message accepting to discharge

[4] We see offering as an action whose results are a set of constrained actions that the agent could commit to perform. Just as with any other action (e.g., buying an appliance), results are expected to satisfy qualifying standards (e.g., the appliance satisfies functional and aesthetics requirements); likewise, the standard for results of offering is that the agent could commit to perform the indicated actions. We do not model this type of analysis, as we have purposefully abstracted from the specification of actions (including any standards of quality for their results) to concentrate only on the observable flow of information between agents.

$$
\begin{array}{l}
\textit{OfferRequester} \\
\hline
\upharpoonright(\textit{RequestingOffer}, \textit{AcceptingOffer}) \\
\textit{Agent} \\
\\
\quad \textit{ToProposeToAdoptOffering} \\
\quad a? : \textit{ToOffer};\ s! : \textit{ToSpeak} \\
\quad \hline
\quad (s!.\textit{speaker} = a?.\textit{receiver} = \textit{self}) \land (s!.\textit{addressee} = a?.\textit{producer}) \land \\
\quad (\exists\, p : \textit{Propose} \mid p = \textit{ProposeToAdoptOffering}(a?) \land \\
\quad \qquad p.\textit{reply}.\textit{until} \geq \textit{now} \bullet p \in s!.\textit{tokens}) \\
\\
\quad \textit{ToAcceptToDischargeOffering} \\
\quad a? : \textit{ToOffer};\ s! : \textit{ToSpeak} \\
\quad \hline
\quad (s!.\textit{speaker} = a?.\textit{receiver} = \textit{self}) \land (s!.\textit{addressee} = a?.\textit{producer}) \land \\
\quad (\exists\, a : \textit{Accept} \mid a = \textit{AcceptToDischargeOffering}(a?) \bullet a \in s!.\textit{tokens}) \\
\\
\quad \textit{ToInformConstrainedActions} \\
\quad \textit{input}? : \mathbb{P}\,\textit{ConstrainedAction};\ a? : \textit{ToOffer};\ s! : \textit{ToSpeak} \\
\quad \hline
\quad (a?.\textit{input} = \textit{input}?) \land \\
\quad (\exists\, i : \textit{Inform} \mid i = \textit{InformConstrainedActions}(\textit{input}?) \bullet i \in s!.\textit{tokens}) \\
\\
\textit{RequestingOffer} \mathrel{\widehat=} \textit{ToProposeToAdoptOffering} \bigwedge \\
\qquad \textit{ToInformConstrainedActions} \mathbin{⨟} \textit{SendUtterance} \\
\textit{AcceptingOffer} \mathrel{\widehat=} \big[\, a? : \textit{ToOffer} \mid (a?.\textit{receiver} = \textit{self}) \land \\
\qquad \textit{ReplyToProposeToDischargeOffering}(a?.\textit{receiver}, \\
\qquad\qquad\qquad a?.\textit{producer}, a?) \in \operatorname{dom} \textit{obligations} \,\big] \bullet \\
\qquad \textit{ToAcceptToDischargeOffering} \mathbin{⨟} \textit{SendUtterance}
\end{array}
$$

Fig. 3. Definition of the *OfferRequester* agent role.

the commitment to offer, and defines as a precondition that the speaker has an obligation to reply to a proposal to discharge the commitment to offer (as specified in *ReplyToProposeToDischargeOffering*, which is not shown). The counterpart agent role *OfferProvider* (not shown) is defined similarly. It specifies the utterances *AcceptingToOffer*, *RejectingToOffer* and *SubmittingOffer*, where the first accepts and the second rejects to adopt a commitment to offer (if and only if there is the obligation to reply to a proposal to add a commitment to offer), and where the third informs a set of actions and proposes to discharge a commitment to offer (if and only if there is the obligation to propose to discharge this commitment). A fourth utterance *AcceptingandSubmittingOffer* is defined as a shortcut for cases in which an acceptance to adopt and a proposal to discharge committing to offer (along with the resulting set of actions) are


agent A
agent B
RequestingOffer: (α) propose(+SC(B, A, ToOffer(A, B), -(B,A)))
inform(*input*)
AcceptingToOffer: accept(+SC(B, A, ToOffer(A, B), -(B,A)))
SubmittingOffer: (β) propose(-SC(B, A, ToOffer(A, B), -(B,A)))
inform(*output*)
AcceptingOffer: accept(-SC(B, A, ToOffer(A, B), -(B,A)))
1
2
3
4

	A & B's Shared Commitments	A's Obligations		B's Obligations	
State 1		policy 1: add 1 policy 1: add 2	1. ToSpeak(B→A,{ReplyTo(α)}) 2. ToHear(→A,{ReplyTo(α)})	policy 1: add 1 policy 1: add 2	1. ToSpeak(B→A,{ReplyTo(α)}) 2. ToVoice(B→,{ReplyTo(α)})
State 2	policy 3: add Z Z. (B,A,ToOffer(B→A))	policy 2: delete 1 policy 2: delete 2 policy 3: add 3 policy 3: add 4 policy 3: add 5 policy 4: add 6 policy 4: add 7	~~1. ToSpeak(B→A,{ReplyTo(α)})~~ ~~2. ToHear(→A,{ReplyTo(α)})~~ 3. ToOffer(B→A) 4. ToSpeak(B→A,{Inform(OUTPUT)}) 5. ToHear(→A,{Inform(OUTPUT)}) 6. ToSpeak(B→A,{Propose(-Z)}) 7. ToHear(→A,{Propose(-Z)})	policy 2: delete 1 policy 2: delete 2 policy 3: add 3 policy 3: add 4 policy 3: add 5 policy 3: add 6 policy 4: add 7 policy 4: add 8	~~1. ToSpeak(B→A,{ReplyTo(α)})~~ ~~2. ToVoice(B→,{ReplyTo(α)})~~ 3. ToOffer(B→A) 4. ToProduce(B,INPUT,OUTPUT) 5. ToSpeak(B→A,{Inform(OUTPUT)}) 6. ToVoice(B→,{Inform(OUTPUT)}) 7. ToSpeak(B→A,{Propose(-Z)}) 8. ToVoice(B→,{Propose(-Z)})
State 3	Z. (B,A,ToOffer(B→A))	policy 1: add 8 policy 1: add 9	3. ToOffer(B→A) 4. ToSpeak(B→A,{Inform(OUTPUT)}) 5. ToHear(→A,{Inform(OUTPUT)}) 6. ToSpeak(B→A,{Propose(-Z)}) 7. ToHear(→A,{Propose(-Z)}) 8. ToSpeak(A→B,{ReplyTo(β)}) 9. ToVoice(A→,{ReplyTo(β)})	policy 1: add 9 policy 1: add 10	3. ToOffer(B→A) 4. ToProcess(B,INPUT,OUTPUT) 5. ToSpeak(B→A,{Inform(OUTPUT)}) 6. ToVoice(B→,{Inform(OUTPUT)}) 7. ToSpeak(B→A,{Propose(-Z)}) 8. ToVoice(B→,{Propose(-Z)}) 9. ToSpeak(A→B,{ReplyTo(β)}) 10. ToHear(→B,{ReplyTo(β)})
State 4	policy 3: delete Z ~~Z. (B,A,ToOffer(B→A))~~	policy 2: delete 8 policy 2: delete 9 policy 3: delete 3 policy 3: delete 4 policy 3: delete 5 policy 4: delete 6 policy 4: delete 7	~~3. ToOffer(B→A)~~ ~~4. ToSpeak(B→A,{Inform(OUTPUT)})~~ ~~5. ToHear(→A,{Inform(OUTPUT)})~~ ~~6. ToSpeak(B→A,{Propose(-Z)})~~ ~~7. ToHear(→A,{Propose(-Z)})~~ ~~8. ToSpeak(A→B,{ReplyTo(β)})~~ ~~9. ToVoice(A→,{ReplyTo(β)})~~	policy 2: delete 9 policy 2: delete 10 policy 3: delete 3 policy 3: delete 4 policy 3: delete 5 policy 3: delete 6 policy 4: delete 7 policy 4: delete 8	~~3. ToOffer(B→A)~~ ~~4. ToProcess(B,INPUT,OUTPUT)~~ ~~5. ToSpeak(B→A,{Inform(OUTPUT)})~~ ~~6. ToVoice(B→,{Inform(OUTPUT)})~~ ~~7. ToSpeak(B→A,{Propose(-Z)})~~ ~~8. ToVoice(B→,{Propose(-Z)})~~ ~~9. ToSpeak(A→B,{ReplyTo(β)})~~ ~~10. ToHear(→B,{ReplyTo(β)})~~

Fig. 4. Interaction diagram and states of obligations and commitments.

issued in one message. This utterance is defined with the same preconditions as *AcceptingToOffer* and *RejectingToOffer*. Similarly, the agent roles *EvaluationRequester*, *EvaluationProvider*, *ExecutionRequester* and *ExecutionProvider* define utterances advancing conversations for the actions to evaluate and to execute actions, respectively.

The dynamics of interaction. Preconditions indicate whether an utterance could be issued at any given state in a conversation. In the case of the agent roles in the offering action, all utterances are defined with preconditions except for the requester's *RequestingOffer*. The absence of preconditions is not necessarily an indicator that an utterance could initiate a conversation; a precondition for this utterance could have been, for example, that a commitment with the same characteristics has not already been adopted. State transitions in conversations occur when an utterance makes one or more of the *pfp* conversation policies alter the obligations of agents.

In this section, we illustrate the dynamics of inference in our model, in which utterances cause transitions to new states (indicating obligations to speak) that enable utterances that cause advances to further states and obligations.[5] Figure 4 shows a conversation to offer between agents A and B. The initial assumption in this example is that A (the *OfferRequester*) and B (the *OfferProvider*) do not have any obligations enabling utterances other than A's *RequestingOffer*. Once uttered, this proposal (labelled α in the figure) triggers policy 1 (the uttering of a proposal creates obligations to reply to it), which establishes obligations where B performs the voicing and A performs the hearing in a speaking joint action replying to α (shown in state 1 as A and B's obligations 1 and 2). By modifying A and B's obligations, this policy effectively changed the state of their conversation, enabling some utterances (B's *AcceptingToOffer*, *Rejecting-ToOffer* and *AcceptingandSubmittingOffer*) while disabling others (in this case, none). Since these enabled utterances have the characteristics that make them adequate replies to α, it is expected that B will utter one of them to satisfy his obligations. We show the case in which B utters *AcceptingToOffer*, causing a transition to state 2 through the triggering of the following policies: policy 2 (once a reply is uttered it discharges obligations to reply), which discharges A and B's obligations 1 and 2; policy 3 (given a proposal and a matching acceptance, agents apply the negotiated commitment operation), which results in the adoption of social commitment Z, A's obligations 3 to 5 (indicating that she jointly participates with B in a *ToOffer* and a *ToSpeak* joint actions in which she performs a *ToHear* individual action) and B's obligations 3 to 6 (indicating that he participates with A in the same *ToOffer* and *ToSpeak* joint actions although he performs a *ToProcess* and a *ToVoice* individual actions); and policy 4 (adopting a dischargeable commitment creates obligations to propose its discharge), which creates A's obligations 6 and 7, and B's obligations 7 and 8, which indicate that B is ought to propose to A discharging commitment Z. Reaching this state enables B's utterance *SendingOffer* while disabling all others (except the initial request, which is always enabled). State 3 shows the resulting set of commitments and obligations after B's issuing of a *SubmittingOffer* message that proposes to discharge commitment Z (labelled β) and informs an offer. This utterance triggers policy 1, which results in A's obligations 8 and 9 and B's obligations 9 and 10 indicating that these agents are the voicer and hearer, respectively, in a speaking action replying to proposal β. The only enabled utterance in this state is A's *AcceptingOffer*. As shown in state 4, this utterance triggers policy 2, which discharges obligations to reply to β (A's obligations 8 and 9, B's obligations 9 and 10); policy 3, which discharges commitment Z and corresponding obligations (A's obligations 3 to 5; B's obligations 3 to 6); and policy 4, which discharges obligations to propose discharging Z (A's obligations 6 and 7; B's obligations 7 and 8). At this point, no further obligations to speak remain thus signaling the end of the conversation.

[5] Readers interested in a formal account of the dynamics of conversations can refer to [5], where a segment of a conversation similar to the one hereby presented is analyzed through strict logical proofs.

Manager

$$\upharpoonright(\mathit{RequestingBid}, \mathit{AcceptingToEvaluate}, \mathit{AwardingContract},$$
$$\mathit{AcceptingToEvaluateandAwardingContract}, \mathit{RejectingBid},$$
$$\mathit{AcceptingToEvaluateandRejectingBid}, \mathit{AcceptingResults})$$

$$\mathit{OfferRequester}\ \mathit{EvaluationProvider}\ \mathit{ExecutionRequester}$$

$$\mathit{RequestingBid} \mathrel{\widehat=} \mathit{RequestingOffer}$$

$$\begin{array}{l}
\mathit{AcceptingToEvaluate} \mathrel{\widehat=} \big[\, a? : \mathit{ToOffer};\ e? : \mathit{ToEvaluate} \mid \\
\quad (a?.\mathit{receiver} = e?.\mathit{producer} = \mathit{self}) \wedge (a?.\mathit{producer} = e?.\mathit{receiver}) \wedge \\
\quad \{\mathit{ReplyToProposeToDischargeOffering}(a?.\mathit{receiver}, a?.\mathit{producer}, a?), \\
\quad\ \ \mathit{ReplyToProposeToAdoptEvaluating}(e?.\mathit{producer}, \\
\qquad\qquad\qquad e?.\mathit{receiver}, e?)\} \subseteq \operatorname{dom} \mathit{obligations} \,\big] \bullet \\
\quad \mathit{ToAcceptToDischargeOffering} \bigwedge \\
\quad \mathit{ToAcceptToAdoptEvaluating} \ \S\ \mathit{SendUtterance}
\end{array}$$

$$\begin{array}{l}
\mathit{AwardingContract} \mathrel{\widehat=} \big[\, e? : \mathit{ToEvaluate};\ x? : \mathit{ToExecute} \mid \\
\quad (e?.\mathit{producer} = x?.\mathit{receiver} = \mathit{self}) \wedge (e?.\mathit{receiver} = x?.\mathit{producer}) \wedge \\
\quad \mathit{SpeakToProposeToDischargeEvaluating}(e?.\mathit{producer}, \\
\qquad\qquad\qquad e?.\mathit{receiver}, e?) \in \operatorname{dom} \mathit{obligations} \,\big] \bullet \\
\quad \mathit{ToProposeToDischargeEvaluating} \bigwedge \mathit{ToInformEvaluation} \bigwedge \\
\quad \mathit{ToProposeToAdoptExecuting} \bigwedge \mathit{ToInformConstrainedActions} \ \S \\
\quad \mathit{SendUtterance}
\end{array}$$

Definition *AcceptingToEvaluateandAwardingContract* omitted.
Definition *AcceptingToEvaluateandRejectingBid* omitted.

$$\mathit{RejectingBid} \mathrel{\widehat=} \mathit{SubmittingEvaluation}$$

Fig. 5. Partial definition of the *Manager* agent role.

Defining joint activities. Joint activities are the components where actions and agent roles are integrated. In this example, we define a contract net joint activity in which a manager and a bidder interact. Figure 5 shows the definition of the agent role *Manager*, which inherits from the agent roles *OfferRequester*, *EvaluationProvider* and *ExecutionRequester*. This agent role defines several utterances. *RequestingBid* is defined as *OfferRequester*'s utterance *RequestingOffer*. *AcceptingToEvaluate* is an utterance accepting the discharge of a commitment to offer, and accepting the adoption of a commitment to evaluate; this can be uttered only when there are obligations to reply to a proposal to discharge a commitment to offer, and a proposal to adopt a commitment to evaluate. *AwardingContract* is an utterance proposing the discharge of and the informing of the results of evaluating, and proposing to adopt the execution of informed actions; this can be uttered only when there is an obligation to propose the discharge of a commitment to evaluate. *AcceptingToEvaluateandAwardingContract* (not shown) is an utterance encompassing *AcceptingToEvaluate* and *Awarding-Contract*. *RejectingBid* is *EvaluationProvider*'s utterance *SubmittingEvaluation*. *AcceptingToEvaluateandRejectingBid* (not shown) is an utterance encompassing *AcceptingToEvaluate* and *RejectingBid*. Finally, *AcceptingResults* is an utterance inherited unchanged from agent role *ExecutionRequester*. The agent role *Bidder* (not shown) is defined in a similar vein as in *Manager*.

The joint activity *ContractNet*, which encompasses the aforementioned definitions, is shown in Figure 6. This class defines *manager* and *bidder* as the participants, *requirements*, *bid*, *evaluation* and *contract* as the data sets, and *bidding*, *evaluating* and *executing* as the actions in the activity. This class also specifies the roles that the *manager* and *bidder* play in each of these actions, the relationship between the data sets and the input and output variables in the actions, and that these actions are the only actions in the activity. Lastly, a protocol (which is also shown in Figure 7) is defined as the sequential arrangement of utterances that participants could issue according to their role types. Although not shown in this paper, these sequences conform to the strict dependency between utterances and obligations that give way to state transitions such as the one exemplified through the interaction in the previous section.

2.4 Flexibility

We study flexibility in terms of the number of unique message sequences that could be supported by a protocol. In the case of our model, flexibility is mainly afforded by the ability of agents to issue counterproposals. As such, to calculate the number of unique message sequences, we define $\kappa : \mathbb{N}_1$ as the maximum number of consecutive counterproposals that could be issued in a *pfp* instance. For example, a κ of 1 indicates that any counterproposal is automatically rejected; a κ of 2 indicates that the second consecutive counterproposal is automatically rejected, and so on. Since it effectively limits the maximum number of messages in a sequence, we call κ the depth of a *pfp* instance. Using κ, we define the number of unique message sequences ending in an acceptance and ending in a rejection (denoted by A and R, respectively, in Formula 1) for any *pfp* instance.

$$A = \sum_{i=1}^{\kappa} 2^{i-1} \qquad R = 2^{\kappa} + \sum_{i=1}^{\kappa} 2^{i} \tag{1}$$

Conversations are composed of more than one consecutive *pfp* sequence. As such, we define $\eta : \mathbb{N}_1$ as the number of consecutive *pfp* instances in a protocol. Using η, we can calculate the number of consecutive *pfp* instances ending in acceptances (denoted by N in Formula 2), and the total number of unique message sequences in a series of η consecutive *pfp* instances (denoted by T in Formula 3).

$$N = \mathbb{A}^{\eta-1} * A \qquad \mathbb{A} = \frac{3A+1}{2} \tag{2}$$

$$T = R_1 + \mathbb{A}_1(R_2 + \mathbb{A}_2(...R_{\eta-1} + \mathbb{A}_{\eta-1}(R_\eta + A_\eta)...)) \tag{3}$$

$$
\begin{array}{l}
\textit{ContractNet} \\
\hline
\textit{JointActivity} \\
\hline
\textit{manager} : \downarrow \textit{Manager};\ \textit{bidder} : \downarrow \textit{Bidder};\ \textit{requirements}, \textit{bid} : \mathbb{P}_1\ \textit{ConstrainedAction} \\
\textit{evaluation} : \mathbb{P}_1\ \textit{EvaluationItem};\ \textit{contract} : \mathbb{P}_1\ \textit{ConstrainedAction} \\
\textit{bidding} : \textit{ToOffer};\ \textit{evaluating} : \textit{ToEvaluate};\ \textit{executing} : \textit{ToExecute} \\
\hline
(\textit{manager} = \textit{bidding.receiver} = \textit{evaluating.producer} = \textit{executing.receiver}) \land \\
(\textit{bidder} = \textit{bidding.producer} = \textit{evaluating.receiver} = \textit{executing.producer}) \land \\
(\textit{requirements} = \textit{bidding.input}) \land (\textit{bid} = \textit{bidding.output} = \textit{evaluating.input}) \land \\
(\textit{evaluation} = \textit{evaluating.output}) \land (\textit{contract} = \textit{executing.input}) \land \\
(\forall e, c : \mathbb{P} \downarrow \textit{Action} \mid e = \{a : \downarrow \textit{Action} \mid \exists e_1 : \textit{evaluation} \bullet a = e_1.\textit{action}\} \land \\
\qquad\qquad\qquad\qquad c = \{a : \downarrow \textit{Action} \mid \exists c_1 : \textit{contract} \bullet a = c_1.\textit{action}\} \bullet c \subseteq e) \land \\
(\forall a_1, a_2, a_3 : \downarrow \textit{JointAction} \mid a_1 = \textit{bidding} \land a_2 = \textit{evaluating} \land a_3 = \textit{executing} \\
\ \bullet \textit{actions} = \{a_1, a_2, a_3\}) \\
\hline
\textit{Protocol} \mathrel{\widehat{=}} [\, m? : \downarrow \textit{Manager};\ b? : \downarrow \textit{Bidder} \mid m? = \textit{manager} \land b? = \textit{bidder} \,] \bullet \\
m?.\textit{RequestingBid} \fatsemi (b?.\textit{RejectingToBid}\ [] \\
(((b?.\textit{AcceptingToBid} \fatsemi b?.\textit{SubmittingBid})\ [] \\
\quad b?.\textit{AcceptingToBidandSubmittingBid}) \fatsemi \\
\quad ((((m?.\textit{AcceptingToEvaluate} \fatsemi m?.\textit{AwardingContract})\ [] \\
\qquad m?.\textit{AcceptingToEvaluateandAwardingContract}) \fatsemi \\
\qquad (((b?.\textit{AcceptingAward} \fatsemi b?.\textit{SubmittingResults})\ [] \\
\qquad\quad b?.\textit{AcceptingAwardandSubmittingResults}) \fatsemi m?.\textit{AcceptingResults}))\ [] \\
\quad (((m?.\textit{AcceptingToEvaluate} \fatsemi m?.\textit{RejectingBid})\ [] \\
\qquad m?.\textit{AcceptingToEvaluateandRejectingBid}) \fatsemi b?.\textit{AcceptingBidRejection}))))
\end{array}
$$

Fig. 6. Definition of the *Contract Net* joint activity.

We made several assumptions to simplify our evaluation of flexibility[6]: a) a successful protocol is one that follows a sequence of *pfp* instances where each instance terminates with an acceptance; b) interacting agents alternate as initiators of *pfp* instances (e.g., the manager's proposal to adopt committing to submit a bid is followed by the bidder's proposal to discharge the commitment to submit a bid); and c) in cases where the agent issuing an acceptance is the same that is expected to issue the proposal in the subsequent *pfp* instance, this agent has the option of uttering these messages separately or as one message (e.g., the manager's acceptance to commit to evaluate a bid and his proposal to discharge this commitment could be uttered as one message). Given the latter assumption, we redefine the number of possible accepting messages in a non-terminating *pfp* instance (i.e., all *pfp* instances i in a sequence of η *pfp* instances,

[6] Although these assumptions reduce the number of possible sequences supported by the *pfp*, it facilitates calculating sequences in protocols abiding to them.

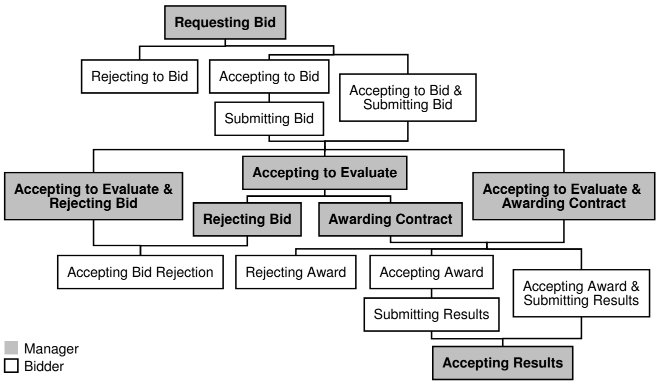


Fig. 7. Protocol in the *Contract Net* joint activity.

where $i : \mathbb{N}_1 \bullet i \leq \eta$) as $\mathbb{A}$ in Formula 2. The rationale for this formula is that there are A possible acceptances in a *pfp* instance of depth κ, where $\frac{A-1}{2}$ of these acceptances could be issued by the agent that uttered the initial proposal, and where the remaining $\frac{A+1}{2}$ are acceptances that could be issued by the agent who was initially proposed. Given our assumption that the agent who was proposed will utter the next *pfp*'s proposal indicates that he could either issue $\frac{A+1}{2}$ acceptances followed by a separate proposal, or that he could issue $\frac{A+1}{2}$ messages encompassing both an acceptance an a proposal. Thus, the possible number of unique message sequences terminating in an acceptance for a non-terminating *pfp* instance is the number of sequences in which the proposing agent utters the acceptance, plus the number of sequences in which the proposed agent utters a message containing only the acceptance, plus the number of sequences in which this same agent utters the acceptance along with the subsequent proposal.

These formulas help us calculate the number of unique message sequences in the *cnp*, and evaluate these results against a comparable *ad-hoc* protocol, e.g., the FIPA contract net protocol [7]. In brief, this latter protocol specifies an initiator (a manager) issuing a call for proposals to participants (bidders), who can either refuse (thus terminating the protocol) or propose to the initiator. This propose can then be followed by a rejection (terminating the protocol) or acceptance from the initiator, and then by a failure or an inform (indicating success), any of which signals the termination of the protocol (in fact, there could be an inform indicating results or an inform indicating successful execution; given the minimal semantic differences between these messages, we hereby consider them as one). In the case of our *cnp* definition, which has $\eta = 4$, and assuming a minimum flexibility κ=1 (i.e., no counterproposals allowed), we derive (from Formula 2) that there exist 8 unique sequences leading to a successful termination, and (from Formula 3) that there are 68 unique sequences in total. If one counterproposal was allowed (κ=2), these numbers would increase to 375 and 1,935 sequences, respectively. These results contrast with the 4 possible unique sequences in the FIPA contract net protocol, of which only one leads to a successful termination. In addition, we achieve this flexibility without significant message overhead: while the number of messages for a successful termination in the FIPA request is 4, in our model is a minimum of 5. From the aforementioned, we could conclude that our model for conversations effectively supports more flexible conversations, without a significant overhead, than those afforded by *ad-hoc* conversations protocols.

3 Conclusions

It has been mentioned in recent years that the key desirable criteria for models for conversation protocols is their support for agent heterogeneity (e.g., [9][15]) and the definition of principles for their modular composition and re-use (e.g., [1][14]). In this paper we presented a model for conversations that supports this criteria by specifying declarative principles that allow building protocols in a modular manner. On the one hand, this model supports heterogeneity by defining protocols through social commitment messages yielding obligations influencing action performances. In particular, by focusing on the properties of messages rather than on the implementation of actions, it allows designers to implement agents using their programming technique of choice; by featuring an inferential definition of sequencing and turn-taking, it allows protocols whose correctness could be verified at design time and then hard-coded in agents using a procedural language or programmed as inference rules in deliberative agents (whom then could verify compliance to the protocol at runtime); by supporting counterproposals it allows flexible protocols that sophisticated agents can use to exploit context-dependent circumstances, while also allowing interactions with less-able agents that only follow message sequences leading directly to successful terminations (and whom may not pursue or immediately reject counterproposals). In addition, the principles in the model are compliant to meaningful, verifiable and declarative criteria allowing for the compilation (at design time) and execution (at runtime) of protocols, as defined in [15]. These principles also support agent autonomy by negotiating rather than imposing the uptake of communicated commitments. On the other hand, our model supports compositional principles by defining protocols through modular components defined in a formal object-oriented specification language. In particular, the model supports the five criteria for modularity defined in [10]: it allows decomposability by structuring protocols in terms of loosely coupled components such as obligations, joint actions, agent roles, and so on; it allows composability by reusing such components to build new protocols through inheritance and aggregation; it allows understandability by abstracting the details of message, action and role definitions in the specification of protocols; it allows continuity by enabling the refinement and extension of protocols with minimal impact on their structural components; and lastly, it allows protection by defining inference principles upon which the verification of and compliance to protocols can be rigorously analyzed both at design and runtime.

4 Acknowledgements

We are thankful for support from the National Science and Engineering Research Council (NSERC) of Canada, and the Alberta Software Engineering Research Consortium (ASERC). We are also grateful to the anonymous reviewers for their valuable suggestions and comments.

References

1. S. Bussmann, N.R. Jennings, and M.J. Wooldridge. Re-use of interaction protocols for decision-oriented applications. In P. Ciancarini and M. Wooldridge, editors, *Proceedings of the 3rd International Workshop on Agent-Oriented Software Engineering*, pages 51–62, July 2002.
2. 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.
3. H.H. Clark. *Using language.* Cambridge University Press, 1996.
4. R. Conte and C. Castelfranchi. *Cognitive and Social Action.* University College London Press, 1995.
5. R.A. Flores. *Modelling agent conversations for action.* PhD thesis, Department of Computer Science, University of Calgary, June 2002.
6. R.A. Flores and R.C. Kremer. To commit or not to commit: Modelling agent conversations for action. *Computational Intelligence*, 18(2):120–173, 2003.
7. Foundation for Intelligent Physical Agents (FIPA). FIPA interaction protocol specifications. http://www.fipa.org/repository/ips.php3, October 2003.
8. N. Fornara and M. Colombetti. Defining interaction protocols using a commitment-based agent communication language. In J.S. Rosenschein, T. Sandholm, M.J. Wooldridge, and M. Yokoo, editors, *Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 520–527, Melbourne, Australia, July 2003.
9. M. Greaves, H. Holmback, and J.M. Bradshaw. What is a conversation policy? In M. Greaves and J.M. Bradshaw, editors, *Proceedings of the Workshop on Specifying and Implementing Conversation Policies*, pages 1–9, Seattle, WA, 1999.
10. B. Meyer. *Object-Oriented Software Construction.* Prentice Hall, second edition, 1997.
11. M.P. Singh. Social and psychological commitments in multiagent systems. In *AAAI Fall Symposium on Knowledge and Action at Social and Organizational Levels*, Monterey, California, November 1991.
12. G. Smith. *The Object-Z Specification Language.* Kluwer Publishers, 2000.
13. R.G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, 29(12):1104–1113, 1980.
14. B. Vitteau and M.-P. Huget. Modularity in interaction protocols. In F. Dignum, editor, *Advances in Agent Communication*, volume 2922 of *Lecture Notes in Artificial Intelligence*, pages 291–309. Springer Verlag, 2004.
15. P. Yolum and M.P. Singh. Flexible protocol specification and execution: Applying event calculus planning using commitments. In C. Castelfranchi and W.L. Johnson, editors, *Proceedings of the 1st International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 527–534, Bologna, Italy, July 2002.