Verification of Protocols for Automated Negotiation

Shamimabi Paurobally[1] and Jim Cunningham[2]

Abstract. This paper presents the verifying aspect of our work for verified, unambiguous and sharable protocols with desirable properties to facilitate automated negotiation. A protocol is represented as an abstract theory in a multi-modal meta-language, called ANML, thereby enabling the verification and proof of certain protocol correctness properties. Furthermore, such logical theories allow the use of AI techniques for reasoning about goal satisfaction. The two case studies discussed in this paper are two protocols proposed in FIPA AUML, [11], which can be shown to contain errors by using our framework. We provide improved representations in ANML and when possible in AUML.

1 INTRODUCTION

Negotiation is part of those wider dynamic processes whereby commercial goals are achieved by the parties to a contract. Overt negotiation, as deal making, is often suppressed by agreed rules of encounter, but it is rarely absent altogether. Automated negotiation has the potential to save both time and computational costs. Electronic agents may be able to find better deals in strategically complex environments without the drawbacks of human ego and prejudices. A key aspect in both real and automated negotiation is the rich, sometimes implicit, interaction between participants, using languages and protocols. Setting up a negotiation involves choosing a common language, ontology and protocol, [15].

Agents engage in conversations, which may be sequences of messages following a higher-level structure called a protocol, in order to perform certain tasks. A group of agents agree to follow this sequence when interacting with one another. Well-known protocols are English auction, Dutch auction, sealed-bid auction. Shared protocols and conversations facilitate heterogeneous applications to attain a common understanding among themselves. A common protocol thus ensures that all participants following it will coordinate meaningfully and can expect certain responses from others. Some researches have even taken the approach of a protocol-oriented view of ACL (agent communication language) semantics in order to standardise agent communication, [14]. We acknowledge the importance of protocols in addition to an ACL in a negotiation since a protocol allows rational behaviour for deriving paths of actions towards a goal.

Although conversations have become part of many multi-agent architectures, formalised conversation specification and implementation are needed for verification, goal-oriented reasoning and analysis of the properties of a protocol and hence a negotiation. The semantics of current specification methodologies have to meet these requirements in the case of agent interaction protocols. In a previous paper, [12], we presented an outline of our meta-language, ANML (Agent Negotiation Meta-Language), which is based on multi-modal propositional dynamic logic and which provides intuitive theories for negotiation. ANML can be used to represent and verify protocols and their properties and to reason about a negotiation through an abstract theory of a conversation. In addition, ANML is compatible with existing ACLs as we may define negotiation protocols using ANML where the messages being sent by an agent are part of an agent communication language like FIPA ACL, [3], or KQML, [7]. As an analogy with real-life conversations, ANML may be considered as a tool to construct and check the grammar in a conversation where the words being communicated may be in any language such as English, Japanese or Spanish.

In this paper, we use our framework to analyse two agent interaction protocols proposed in FIPA AUML, [11]. The next section gives an overview of our meta-language and of FIPA AUML. In section 3 we verify a simple AUML Request Protocol where at most three messages may be sent. We translate this protocol into ANML and show how even such a simple protocol is prone to error. We provide corrected versions of the protocol in both ANML and AUML. In section 5, we carry out similar verification on an iterated contract net protocol, initially proposed in AUML. We perform progressive corrections on given protocols by checking the possible states of a negotiation. AUML protocols depend on the semantics of the FIPA ACL performatives, which change according to the ongoing altercation about FIPA ACL. ANML on the other hand is a meta-language that can be made compatible with existing or emerging agent communication languages.

2 SPECIFYING PROTOCOLS

Traditionally, deterministic finite state machines have been used to specify protocols, ([10] and [14]). Other approaches for developing protocols include state charts [19], and petri-nets. For sophisticated interactions it is desirable to use formalisms with more support for concurrency and verification [8]. There is a need for tools for protocol specification, verification and sharing. In [12], we propose a meta-language, called ANML, for expressing protocols and for providing an abstract theory of a negotiation. ANML supports the specification of protocols such that they can be sharable, with desirable and testable properties and help to coordinate agents in their goal seeking activities.

2.1 ANML

ANML, (Agent Negotiation Meta-Language), is a multi-modal, propositional dynamic logic-based meta-language for representing and reasoning about the states and processes of a negotiation. New or existing protocols can be expressed and extended in ANML so as to use its verification capability. ANML addresses much that can be standardised seamlessly without the danger of semantics mismatch in language and ontology between different agents. Our syntax is an adaptation of the program logic described in Goldblatt (1987), [6], where a process may be expressed in terms of its sub-processes. For example, a negotiation process can be decomposed into possibly non-atomic sub-processes like *browsing*, *bargaining* and *paying*. We associate an agent with processes by prefixing the process with an agent in the same way an object is suffixed by its methods e.g., *r:retailer.display* means *retailer r* executes the *display* process. Usually we omit the agent type and denote a joint process between two parties with '$\cup$' as in {*c* $\cup$ *r*}.*shopping*. A process may be decomposed into a sequence of sub-processes each possibly coupled with the agent or agents executing that sub-process. An

[1] ,[2] Imperial College, London SW7 2BZ, UK, email: {sp1,rjc}@doc.ic.ac.uk

action is an atomic process. The process denoted by *a*;*b* is composed of the sequence *a* followed by *b*, *a** denotes zero or more iterations and a^+, one or more iterations. A state test operator '?' allows sequential composition to follow only if successful. For example *c.browse*?;*c.choose* is the process *c.choose* if *c.browse* succeeds, otherwise it fails. Paurobally and Cunningham (2000), [12], covers ANML in more detail and specifies various negotiation protocols in ANML.

Throughout this paper, for the sake of conciseness, we use the terms state and current state as implicitly referring to the state and current state of a negotiation on a subject respectively. The state of a negotiation may have a hierarchical structure e.g. a state may contain sub-states. When a state holds, it is necessary that its parent states also hold e.g. when *offered* is true, then *open* is also true since *offered* is a sub-state of *open*. However it is not necessary for a sub-state to be true while its parent state holds. Hence a state conveys information about a negotiation. Partial information is available when we know that a parent state holds e.g. we know the negotiation is *closed*, but we do not know what sub-state of *closed* out of *rejected*, *agreed* or *timedout*. More specific information is obtained by knowing which sub-state is true. A current state means a state which is true while other non-parent states are false. The current state of a negotiation is produced by agents' actions.

2.2 FIPA AUML

Bauer et al. [1] has proposed AUML, (Agent Unified Modeling Language), as an extension of UML to define interaction protocols between agents. AUML is intended to be a graphical specification technique, which rely partly on FIPA ACL by using a subset of its communicative acts as messages. In sections 3 and 4, we show that AUML allows errors and ambiguities in protocols and does not scale easily to multi-agent systems.

An AUML Interaction Protocol (IP) diagram expresses a protocol in the form of a UML sequence diagram with extensions specific to AUML, as shown in Figure 2. Agents are assigned to roles, belong to classes and an IP diagram shows interactions between these agents along a timeline. A dashed box at the upper right-hand corner declares the protocol as a template specification for later instantiation. An arrow indicates an unnested, asynchronous communication while a diamond means a decision point that can result in zero or more communications being sent.

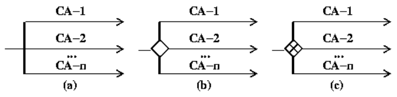
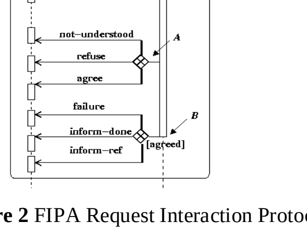


Figure 1. Extensions supporting concurrent threads

Figure 1 shows AUML notation for asynchronous messages. Figure 1(a) indicates that all threads CA-1 to CA-n are sent concurrently, 1(b) shows a decision point where zero or more messages may be sent and 1(c) indicates exactly one message may be sent.

3 A Request Protocol



Initiator, Participant, request, refuse*, not–understood*, agree, inform–done:inform*, failure*, inform–ref:inform*
FIPA–Request–Protocol
Initiator
Participant
request
not–understood
refuse
agree
A
failure
inform–done
inform–ref
B
[agreed]

Figure 2 FIPA Request Interaction Protocol,[4]

Figure 2 shows a Request IP, [4], in AUML where at most three messages can be exchanged. We show that errors exist in even such simple protocols. An initiator sends a request to another agent, a participant, who refuses, or replies that it does not understand or agrees to satisfy the request and later reports on its success. We represent this protocol in ANML and point out a number of errors in it, which lead to an ambiguous and inconsistent interaction and wrong beliefs. We finally give rectified versions of the protocol in both ANML and AUML.

3.1 Initial remarks

An agent may not have built-in knowledge about real life interactions, so a protocol has to be complete and make all possibilities in a specific situation explicit. Consider the case when either agent should be able to send some message at any time or when an event can occur at any point, e.g. any agent can reject, send a failure or a timeout can occur at any time. To represent such dynamic processes in an AUML diagram would require each interaction thread from each agent to show all possible messages, leading to cluttered diagrams. In Figure 2, an initiator cannot send a failure, reject or timeout message. In ANML, we just add axioms to express the effect of these processes from a parent state.

In AUML diagrams, splitting and merging of threads of interaction can be abbreviated to one thread along the same lifeline, where execution at these threads usually depends on the current state of the interaction and on previous messages. Conditions for executing a process at decision threads have to be defined, initialised and reset. In ANML, action-condition rules determine possible state transitions and ensure the current state of a negotiation is valid with respect to other states. In AUML protocols, the guards at a decision point are often informally treated or implicit, giving rise to misunderstandings. For example, in Figure 2, *agreed* has not been defined and initialised anywhere and is not shown to be related to the previous *agree* message. Therefore, a participant can refuse a *request* or send a *non-understood* message, but later may still send an *inform* message that it has successfully performed the request.

Although using the same timeline as on the Initiator's side in Figure 2 makes the graphical diagram more readable, this abbreviation cannot be used in more realistic protocols. Out of the 3 messages received by an Initiator in Figure 2, at least two of them are terminal messages ending the interaction. In practical conversations, at a decision point, there may be more than one non-terminal message that gives rise to various sequences of interactions.

3.2 Verification of the request protocol

A literal translation of Figure 2 along its timeline from AUML to ANML results in the following path of execution:

{*Initiator* ∪ *Participant*}. *FIPA-Request-Protocol = Initiator. request*; **(**one-of [*Participant.not-understood, Participant. refuse, Participant.agree*] ∧ (*agreed* → one-of ([*Participant.failure, Participant.inform-done, Participant.inform-ref*])) **)**

The *one-of* predicate returns *true* if and only if exactly one of the elements in its given list is *true* whereas *none-of* returns *true* if and only if none of the elements in its given list are *true*.

3.2.1 Errors in AUML protocol

As remarked before, the condition *agreed* at point B in Figure 2 has no meaning for an agent and is not shown to be related to the previous *agree* message. Even though a participant sends *not-understood* or *refuse* after a *request*, it may still send *inform-done* or *failure* or *inform-ref* later. It is thus not obvious that *not-understood* and *refuse* lead to terminal states. In ANML, we can treat *agreed* as a state which we initialise to *false* and set to *true* when Participant sends an *agree* message.

In the AUML diagram, the request protocol is a path from a *request* to success or failure messages. The preconditions and postconditions for an action are not well-defined and there are no constraints to stop an agent restarting a negotiation through a new request while in the middle of following the protocol. There is no facility for an agent to refer to a point in an interaction and to the effects of messages. All actions end up in anonymous points. What is the difference in the world between sending a not-understood, refuse or agree message since the state of the world or the effects of these actions are not given? How does an agent refer to the result of an interaction or differ between possible worlds as messages are sent? AUML protocols depend on the semantics of the FIPA ACL performatives.

According to the AUML notation, in Figure 2 there is no constraint against threads at *A* and *B* being executed concurrently. This leads to a number of contradictions with realistic interactions. A participant can send an *agree* or *not-understood* message while at the same time sending a *failure* or *inform-done*. Logically a participant should not be able to simultaneously send an agreement at point *A* and a failure at point *B*. In practice, [*agreed*] is initially *false* and depends on the decision at *A*. When messages at *A* and *B* are sent concurrently, even if a participant sends an *agree*, [*agreed*] is *false* and point *B* is not executed. Thus a participant does not execute a request even after agreeing to it and an initiator waits in vain for the result of an agreement. An agent may need time to complete a task after agreeing to it and cannot execute *A* and *B* with success concurrently.

3.2.2 Corrections in an ANML translation

We define the states of a negotiation after each possible message by assuming that an (agent's) action triggers a corresponding state e.g. an *agree* action triggers an *agreed* state. The states after *request, refuse, inform-done, not-understood, agree, failure, inform-ref* actions are *requested, refused, informed-done, not-understood, agreed, failed* and *informed-ref* respectively. A protocol is ambiguous if any of the states are undefined at any point. It is incorrect if a state that becomes *true* is not as required by the semantics of the protocol, e.g. in an auction a bid triggers the *rejected* state. We aim to obtain a complete protocol where all states are well-defined at all instances by analysing the truth values of the above states. The set of states is finite and bounded by the protocol. Before a negotiation, all the above states are initialised to *false*. To prevent an agent from restarting a negotiation whilst in the middle of one, we introduce the *interaction* state as the parent of all states; ¬*interaction* is *true* at the start of a negotiation and is the precondition for sending a request. We also express points *A* and *B* as a sequential path.

Theory 1 is an axiomatisation in ANML of the request protocol by giving the relation between parent and sub-states and action-condition rules for state transitions. We use multi-modal operators in ANML to represent axioms for state transitions. $s_1 \leftrightarrow [X.a]s_2$ can be read as state s_1 of negotiation holds if and only if at this state, an agent *X* performing action *a* always leads to next state s_2. $s(X)$ denotes the state *s* produced by an action from agent *X*. Axioms between states ensure that only the current state is enabled while other non-parent states remain *false*. Now we can differentiate between worlds by referring to the current state of a negotiation. We can identify the state and result of a negotiation at any point.

We specify two more states *open* and *closed*, as sub-states of *interaction*, to explicitly convey non-terminal and terminal states respectively. We also allow a *timeout* event to occur at any time, leading to a *timedout* state. We can prove that the final protocol, given by Theory 1, leads to well-defined states at all times. We can further incorporate our corrections into a new AUML diagram and extended statecharts given in Figure 3. Let *P* denote a participant, *I* denote an initiator and *X* as any agent.

interaction ↔ one-of ([*open*, *closed*])
open ↔ one-of ([*requested*(*X*), *agreed*(*X*)])
closed ↔ one-of ([*not-understood*(*X*), *refused*(*X*), *failed*(*X*), *informed-done*(*X*), *informed-ref*(*X*), *timedout*])
¬*interaction* ↔ none-of ([*open*, *closed*])
¬*open* ↔ none-of ([*requested*(*X*), *agreed*(*X*)])
¬*closed* ↔ none-of ([*not-understood*(*X*), *refused*(*X*), *failed*(*X*), *informed-done*(*X*), *informed-ref*(*X*)])
¬*interaction* ↔ [*I.request*] *requested*(*I*)
requested(*I*) ↔ [*P. not-understood*]*not-understood*(*P*) ∨ [*P.refuse*] *refused*(*P*) ∨ [*P.agree*]*agreed*(*P*)
agreed(*P*) ↔ [*P.failure*]*failed*(*P*) ∨ [*P.inform-done*] *inform-done*(*P*) ∨ [*P.inform-ref*]*informed-ref*(*P*)
open ↔ [*timeout*] *timedout*

Theory 1. Logical theory for request protocol

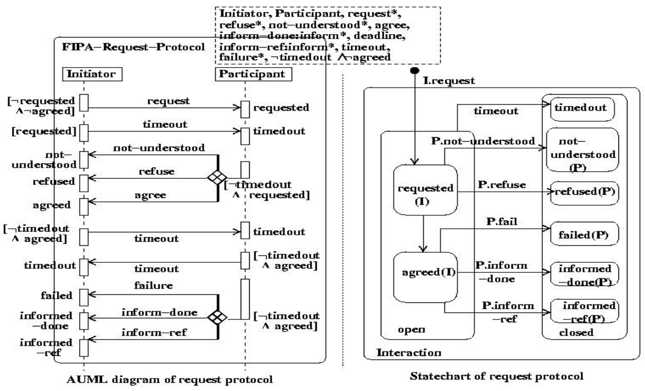


Figure 3. Suggested AUML and statechart diagrams for a request IP

4 ITERATED CONTRACT NET PROTOCOL

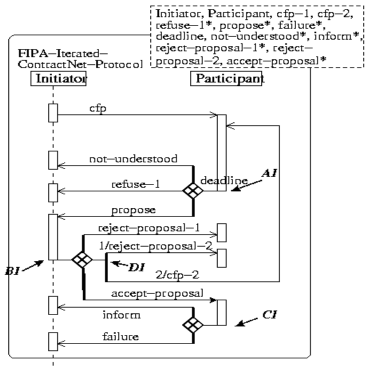

Initiator, Participant, cfp−1, cfp−2, refuse−1*, propose*, failure*, deadline, not−understood*, inform*, reject−proposal−1*, reject−proposal−2, accept−proposal*
FIPA−Iterated− ContractNet−Protocol
Initiator
Participant
cfp
not−understood
refuse−1
deadline
A1
propose
reject−proposal−1
1/reject−proposal−2
D1
2/cfp−2
B1
accept−proposal
inform
C1
failure

Figure 4. FIPA Iterated Contract Net Interaction Protocol, [5]

In the FIPA contract net protocol, a manager solicits proposals from other agents by issuing a call for proposals (*cfp*). Agents receiving a *cfp* may reply with *propose*, *not-understood* or *refuse-1* before a deadline. After the deadline, the manager sends an *accept-proposal* to selected agents and a *reject-proposal* to others. When a contractor has completed its task, it sends a completion message to the manager. The FIPA iterated contract net IP, [5], is an extension of the contract net protocol to allow multi-round iterative bidding. After a first call for proposals, a manager may accept one or more of the bids, rejecting the others, or may repeat the process by issuing a revised *cfp* and rejecting all the proposals. The process terminates when the manager refuses all proposals and does not issue a new *cfp* or accepts one or more of the bids or the contractors all refuse to bid and the manager does not issue a new cfp.

The obvious error in Figure 4 is the participant's missing timeline. Figure 4 is translated along its timelines into the three axioms in ANML in Theory 2. Let *P* denote the Participant and *I* an Initiator.

{*I*:*Initiator* ∪ *P*:*Participant*}.*FIPA-IteratedContractNet-Protocol* = *I.cfp*; {*I* ∪ *P*}.*contractNetProtocol*
{*I* ∪ *P*}.contractNetProtocol = **(***deadline* ↔ one-of ([*P.not-understood*, *P.refuse-1*, *P.propose*])**)** ; one-of([*I.reject-proposal-1*, {*I* ∪ *P*}.*accepting*, **(***I.reject-proposal-2* ∧ (*I.cfp-2*; {*I* ∪*P*}.*contractNetProtocol* **)**])

$\{I \cup P\}$*.accepting* = *I.accept-proposal*; one-of([*P.failure*, *P.inform*])

Theory 2. First ANML version of iterated contract net protocol

4.1 Verifying the iterated contract net protocol

Most of the errors mentioned in section 3 also occur in Figure 4 and assuming that we have resolved them, we discuss here additional errors and rectification, while mentioning in brackets which axioms in Theory 3 correct that error. For example, a recurrent error is the lack of conditions or their incorrect setting at decision points. In Figure 4, even after a participant responds with a *not-understood* or *refuse-1* message at *A1* after a *cfp*, the interaction can continue and an initiator can send an *accept-proposal* at *B1*. The threads of interaction for both agents lying along one timeline require constraints at the decision points *A1*, *B1* and *C1*. Theory 3 gives our logical theory for this protocol after several revisions for ensuring all states are well-defined and as expected. We refer to points *A1*, *B1*, *C1* and *D1* in Figure 4.

4.1.1 Deadlines

In theory 2 and Figure 4, a participant must wait for the deadline (at point *A1*) to occur in order to respond to a *cfp*. For example, if the deadline is set to 4 minutes then at exactly 4 minutes and 0 seconds, all agents must send their responses. This is not feasible and in practice, a participant is allowed to respond before a deadline. Deadlines that are given a ground value such as 9:00 am become unattainable when reiterating cfps. Figure 4 also does not cover the case where a deadline passes without a participant sending any messages. Axioms 26 and 20 respectively specify a timeout event and the possible actions *before* a deadline.

4.1.2 Incompleteness

Point *B1* has no deadline or timeout for a manager to accept or reject proposals, neither does a participant have any deadline or timeout at *C1*. In fact, a number of events specified in the English description of the protocol are not portrayed in Figure 4. The case when all the contractors refuse to bid and the initiator does not issue a new *cfp* is not shown. Axioms 10 and 22 capture this. The negotiation may also not terminate, ending up in infinite call for proposals, even if all agents send a refusal. A logical theory of a protocol allows us to derive its properties such as termination and decidability, [13]. In Figure 4, contrary to the English description, an initiator cannot both accept some proposals and reject others since point *B1* is an exclusive-or decision where exactly one of the interaction threads can be performed. Axioms 6 and 23 allow *rejected-proposal-1*(*C*) and *proposal-accepted*(*Z*) to coexist.

4.1.3 Scaling to multi-agent systems

The contract net protocol is a multi-agent interaction. Yet Figure 4 and theory 2 show an interaction between only two agents. They do not portray how an interaction in a multi-agent system depends on the responses of specific agents and their dynamic roles. How does an agent infer at point *B*, whether only one agent is sent an acceptance or some agents, e.g. *X,Y,Z*, are sent acceptance.

For example after *A1* in Figure 4, out of *n* agents, *m* agents send a proposal and *p* agents send a refusal. What is the relation between an agent among these *m* agents and one from the *p agents*? An initiator must send acceptances and rejections to these *m* agents and not to those (*n-m*) agents that did not send a proposal. The identity of these *m* and (*n-m*) agents must be known so that each agent receives and sends the appropriate message. Likewise only those agents whose proposals have been accepted should be able to send the results of executing a *cfp* at *C1*, but the AUML protocol does not stop other agents from doing so. There is a lack of guards and lack of information about which agent can perform which action. Figure 4 does not associate the identity of an agent to the messages being sent and it is not known which agents' proposals are accepted and which ones are rejected.

Figure 4 shows only two timelines. Illustrating a complete and reasonable interaction between more than 2 or 3 agents in AUML would be hard and would result in an illegible diagram. In addition, AUML does not keep a parameter to record which agent sent which message to whom. What if participants wish to send messages between themselves as in a forum instead of just between an initiator and a participant? For a multilateral IP in AUML, one would need to show all the timelines of all the agents in all their roles. This has to be specified beforehand and the number of agents fixed, preventing open interactions. Auctions are popular forms of negotiation allowing dynamic entries and thus cannot be realistically specified in AUML. Hence AUML does not capture correctly multi-agent and dynamic negotiations with an agent changing roles dynamically.

Even if a group of (*n*+1) agents follows Figure 4 as a protocol, an initiator must keep *n* instances of the same negotiation for *n* participants, giving rise to complexity, concurrency and coordination problems. In an online auction, an auctioneer following an AUML protocol would need an auction instance for each possible (registered) bidder, instead of only keeping track of the state of the auction with those agents which sent bids. When the participants are allowed to know how the initiator responds to others, then each participant would need *n* instances of the interaction for each other participant. If an interaction allows communication between participants, then the *n* instances are related to each other leading to a complexity of m^n for *m* states.

In ANML we pass an agent as a parameter to the actions and states related to it. We use finite set theory to manage groups of agents and parameterise actions and states with a group of agents. The sets of agents in each state is initialised to empty and an agent is added to an appropriate set according to the message exchange and the resulting state. For example, just after a *cfp*, *not-understood*({}) means that no agent has sent a *not-understood* message. When agents *ann* and *bob* send a *not-understood*, they are added to the set to give *not-understood*({*ann*,*bob*}).

4.1.4 Iterating call for proposals (*cfps*)

After a participant responds with a *refuse-1* or a *not-understood* at *A1*, it is unclear in Figure 4 whether that participant is sent further *cfps* or it is written off for the rest of the interaction. In practice all agents should be sent revised *cfps*, since an agent which previously sent a refusal may later be able to execute a revised *cfp*. Figure 4 presupposes that only those participants that previously sent a proposal are eventually sent a revised *cfp*, as the interaction thread *D1* is a response to *propose*. In axioms (7) and (9), we create a sub-state of *open* called *on-hold* for those agents which sent a refusal or *not-understood* in response to a *cfp*. Axioms (22) and (24) allow revised *cfps* to be sent to all agents and reinitialise the set of agents in the *on-hold* state.

Point *D1* is a concurrent interaction thread where an initiator simultaneously rejects the proposals of a number of agents and sends a new *cfp* to them. Confusion may arise when several messages are sent or received simultaneously. A participant may receive a revised *cfp* followed by a rejection because of delays in the communication channel. In this case, the participant's beliefs become inconsistent with the other agents. An initiator may also have difficulty in distinguishing between delayed proposals from previous *cfps*. *D1* should not be a concurrent thread and an initiator should first send a *reject-proposal-2* to some agents followed by a *cfp-2* to all agents. We also need to re-initialise the conditions, states and sets of agents when reiterating a *cfp* as in axioms (22) and (24) and the rules between states. It is unclear how to do this initialisation in AUML, even if guarding conditions were given. The first *cfp* is an entry point into a negotiation and must be distinguished from revised *cfps*. (Axiom 19).

4.1.5 Making Proposals

The AUML protocol supposes that as soon as an initiator receives a proposal, it cannot accept other messages and must perform the decisions at point *B1*. This behaviour is at odds with the English definition of the protocol where an initiator waits for a number of proposals before deciding on which ones to accept. We solve this in axiom(8) by letting *proposed*(*Y*) be a sub-state of *cfped*(*X*).

In Figure 4, in order to proceed to point *B1*, at least one participant must send a proposal. An initiator cannot send a revised call for proposal unless he/she has received at least one proposal. This can lead to a deadlock because all participants may refuse to bid and an initiator cannot revise its *cfp* from the protocol. The interaction does not terminate when no participants make a proposal. Axiom 22 allows a timeout and sending a revised *cfp* in case of no proposals.

4.1.6 Terminal states

In AUML, diagrams abbreviating different threads of interaction onto a single timeline obscures the terminal actions and points for where a negotiation can end. It is not obvious in Figure 4 that an interaction ends when an initiator refuses all proposals and does not issue a new *cfp*. Contrary to figure 2, in figure 4 a negotiation is not closed if some agents send *not-understood* or *refuse-1* messages, since there can be revised *cfps*. In ANML, both *open* and *closed* states, their sub-states and the processes leading to them are obvious. States *failed*(*F*) and *informed*(*E*) may both be true at a point since some participants will send a successful completion while others a failure.

4.1.7 A logical theory of the iterated Contract Net Protocol

Theory 3 is the ANML logical theory for an iterated contract net protocol after several iterations to ensure that the states of a negotiation are well-defined and as expected. We express a multi-agent interaction between an Initiator, *I*, and *n* other participants. Let *I*, *P* and *X* denote single agents and *Y*, *Z*, *A*, *B*, *C*, *D*, *E*, and *F* denote sets of agents. Axioms (4) to (13) represent the relation between states and sub-states. Axioms (14) to (18) ensure that messages are sent to the right agents or group of agents, so that an agent does not receive contradictory or unintended messages. Axioms (19) to (26) are action-condition rules for state transitions.

$$\textit{interaction} \leftrightarrow \text{one-of}\,([\textit{open}, \textit{closed}]) \tag{4}$$

$$\textit{open} \leftrightarrow \text{one-of}\,([\textit{cfped}(X), \textit{pending-accomplishment}\,(Y)]) \tag{5}$$

$$\textit{pending-accomplishment}\,(Y) \leftrightarrow (\textit{rejected-proposal-1}(C) \vee \textit{proposal-accepted}(Y\text{-}C)) \tag{6}$$

$$\textit{on-hold} \rightarrow \textit{cfped}(X) \tag{7}$$

$$\textit{proposed}(Y) \rightarrow \textit{cfped}(X) \tag{8}$$

$$\textit{on-hold} \leftrightarrow \textit{not-understood}(A) \vee \textit{refused-1}(B) \tag{9}$$

$$\textit{closed} \leftrightarrow \text{one-of}\,([\,(\textit{failed}(E) \vee \textit{informed}(F)), \textit{rejected}, \textit{timedout}, \textit{rejected-proposal-2}(D)]) \tag{10}$$

$$\neg\textit{interaction} \leftrightarrow \text{none-of}\,([\textit{open}, \textit{closed}]) \tag{11}$$

$$\neg\textit{open} \leftrightarrow \text{none-of}\,([\textit{cfped}(X), \textit{pending-accomplishment}\,(Y)]) \tag{12}$$

$$\neg\textit{closed} \leftrightarrow \text{none-of}\,([\,\textit{failed}(E), \textit{informed}(F), \textit{rejected}, \textit{timedout}, \textit{rejected-proposal-2}(D)]) \tag{13}$$

$$(\textit{not-understood}(A) \wedge \textit{refused-1}(B) \wedge \textit{proposed}(Y)) \rightarrow A \cap B \cap Y = \{\} \tag{14}$$

$$(\textit{rejected-proposal-1}(C) \wedge \textit{proposal-accepted}(Z) \wedge \textit{proposed}(Y)) \rightarrow ((Z = Y - C) \wedge C \cap Z = \{\}) \tag{15}$$

$$\textit{failed}(E) \wedge \textit{informed}(F) \wedge \textit{proposal-accepted}(Z) \rightarrow (F = Z - E) \tag{16}$$

$$\textit{failed}(E) \wedge \textit{informed}(F) \rightarrow (F \cap E = \{\}) \tag{17}$$

$$\textit{rejected-proposal-2}(C) \rightarrow \textit{proposed}(C) \tag{18}$$

$$\neg\textit{interaction} \leftrightarrow [I.\textit{initial-cfp}]\,(\textit{cfped}(I) \wedge \textit{not-understood}(\{\}) \wedge \textit{refused-1}(\{\})) \tag{19}$$

$$(\textit{cfped}(I) \wedge \textit{not-understood}(A) \wedge \textit{refused-1}(B) \wedge \neg\textit{proposed}(Y)) \leftrightarrow \text{one-of}\,([\,[P.\,\textit{not-understood}]\textit{not-understood}(A \cup P), [P.\textit{refuse-1}]\textit{refused-1}(B \cup P), [P.\textit{propose}]\textit{proposed}(\{P\})]) \tag{20}$$

$$(\textit{cfped}(I) \wedge \textit{not-understood}(A) \wedge \textit{refused-1}(B) \wedge \textit{proposed}(Y)) \leftrightarrow \text{one-of}([[P.\,\textit{not-understood}]\textit{not-understood}(A \cup P), [P.\textit{refuse-1}]\textit{refused-1}(B \cup P), [P.\textit{propose}]\textit{proposed}(\{Y \cup P\})]]) \tag{21}$$

$$(\textit{on-hold} \wedge \neg\textit{proposed}(Y) \wedge \textit{cfped}(I)) \leftrightarrow [\textit{timeout}\,;\, I.\textit{cfp}]\,(\textit{cfped}(I) \wedge \textit{not-understood}(\{\}) \wedge \textit{refused-1}(\{\})) \tag{22}$$

$$\textit{proposed}(Y) \leftrightarrow ([I.\textit{reject-proposal-1}]\,\textit{rejected-proposal-1}(C) \wedge [I.\textit{accept-proposal}]\,\textit{proposal-accepted}(Z) \wedge Z = Y - C) \vee ([I.\textit{reject-proposal-2}]\,\textit{rejected-proposal-2}(Y) \tag{23}$$

$$\textit{rejected-proposal-2}(Y) \leftrightarrow [I.\textit{cfp}]\,(\textit{cfped}(I) \wedge \textit{not-understood}(\{\}) \wedge \textit{refused-1}(\{\})) \tag{24}$$

$$\textit{proposal-accepted}(Z) \leftrightarrow [P.\textit{inform}]\textit{informed}(E) \vee [P.\textit{failure}]\textit{failed}(F) \wedge E = Z - F \tag{25}$$

$$\textit{open} \rightarrow [\textit{reject}]\,\textit{rejected} \vee [\textit{timeout}]\,\textit{timedout} \tag{26}$$

Theory 3. Final ANML protocol

5 CONCLUSION

We have shown how to represent and verify existing protocols using a meta-language, ANML, to obtain an improved and intuitive theory of a protocol. We can also convert a logical theory of, for example a bilateral negotiation, [12], from ANML into an AUML diagram, a statechart or a petrinet. However the AUML diagram is relatively complicated since, unlike the two simple case studies analysed here, more than one non-terminal messages are possible at any one decision point. Also expressing all conditions and internal events in AUML increases the complexity of the diagram.

A logical theory for a protocol enables us to verify functional correctness and to address properties such as termination, fairness, safety and liveness, [13]. We can use the research on joint intention, [2], to facilitate issues of concurrency, nesting and consistency. An abstract theory for a protocol allows us to use AI techniques for reasoning and planning about the sets of possible paths towards a goal. In a forthcoming paper, we use epistemic logic to analyse the knowledge of an agent and consistency in the group's knowledge about the current state of a negotiation as it progresses.

REFERENCES

[1] Bauer, B., Müller, J., Odell, J. [2001] "Agent UML: A Formalism for Specifying Multiagent Interaction". *Agent-Oriented Software Eng.*, P. Ciancarini and M. Wooldridge eds., Springer, Berlin, pp. 91-103.

[2] Cohen, P. R. and Levesque, H. J. [1991]. "Teamwork". *Nous*, 35, 25(4):487—512.

[3] FIPA. [1997]. Agent Communication Language. Technical report. Foundation for Intelligent Physical Agents. http://www.fipa.org

[4] FIPA [2001]. "FIPA Request Interaction Protocol". http://www.fipa.org/specs/fipa00026/XC00026F.html

[5] FIPA [2001]. "FIPA Iterated Contract Net Interaction Protocol". http://www.fipa.org/specs/fipa00030/XC00030F.html

[6] Goldblatt, R. [1987]. Logics of Time and Computation, *Lecture Notes, Center for the Study of Language and Information 1987.*

[7] Labrou, Y. and Finin, T. [1997]. "Semantics for an agent communication language". *The fourth Int. workshop on Agent Theories, Architectures and Languages.* Rhode Island, USA.

[8] Labrou, Y. [2001]. "Standardising Agent Communication". *In Proc. of 9th ECCAI Advanced Course and EASSS 2001.* Springer, LNAI 2086.

[9] Moore, S. [1999]. "On conversation policies and the need for exceptions". *Workshop on Specifying and Implementing Conversation Policies, Third Int. Conf. on Autonomous Agents.* pages 19-28.

[10] Nodine M. and Unruh A. [1998]. "Facilitating open communication in agent systems: the InfoSleuth infrastructure". *In Proc. of the Fourth Int. Workshop on Agent Theories, Architectures, and Languages,* 1997. MCC-INSL-113-98.

[11] Odell, J., Parunak, H.V.D., Bauer B. [2001]. "Representing Agent Interaction Protocols in UML". *Agent-Oriented Software Eng.*, P. Ciancarini and M. Wooldridge eds., Springer Berlin, pp. 121–140.

[12] Paurobally, S. and Cunningham, J. [2000]. "Specifying the Processes and States of Negotiation". *In Agent Mediated Electronic Commerce. The European AgentLink Perspective*. Springer. pages 61-77.

[13] Paurobally, S. and Cunningham, J. Safety and Liveness of Negotiation Protocols. AISB2002. Convention on AI and the Simulation of Behaviour. London. ISBN 1902956299

[14] Pitt, J., and Mamdani, A. [1999]. Communication protocols in multi-agent systems". *In the Workshop on Specifying and Implementing Conversation Policies, Third Int. Conf. on Autonomous Agents.*

[15] Rosenschein, J. S. and Zlotkin, G. [1998]. "Rules of Encounter. Designing Conventions for Automated Negotiation among Computers". MIT Press. ISBN 0262181592.